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 significantly improve designer's productivity 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
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU
- 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 the HW
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, 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)

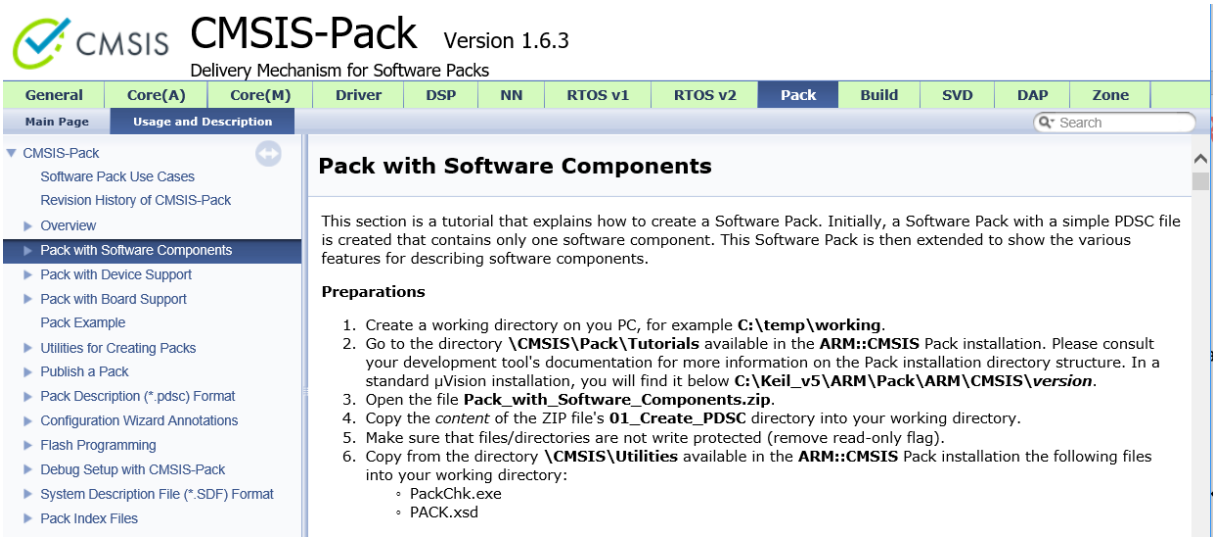
It includes:

- 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 upgrade 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 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

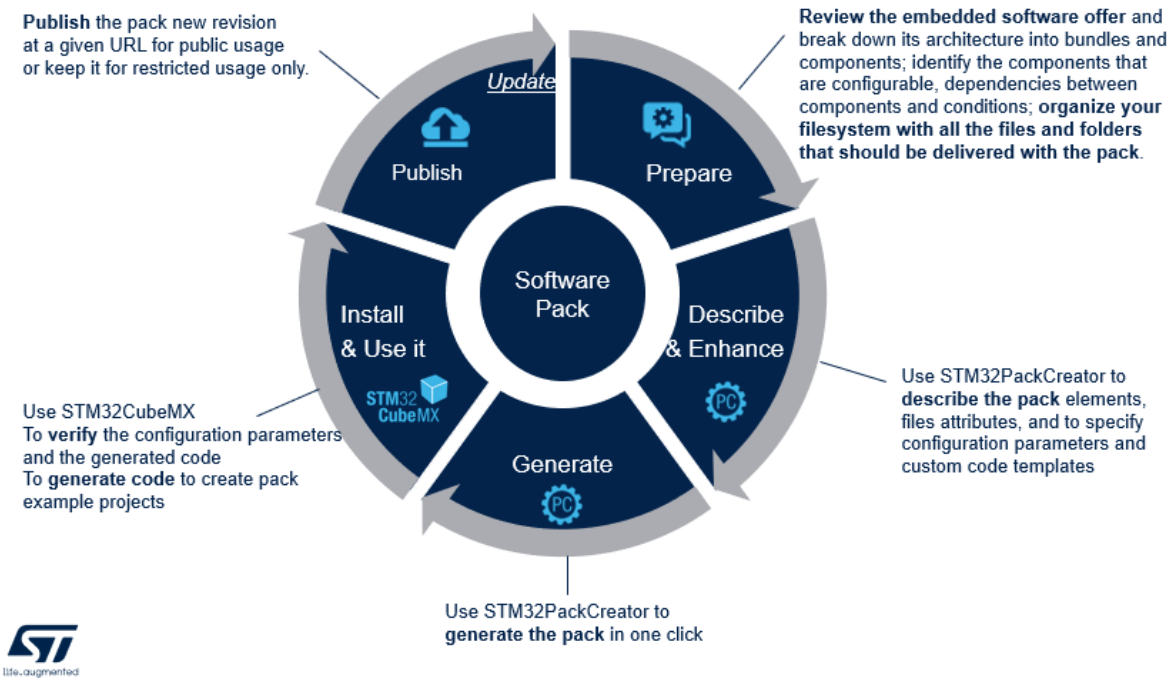
According to the CMSIS-Pack standard, pack owners are responsible for their pack hosting and maintenance. Refer to Figure 2.

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, 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 needs 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

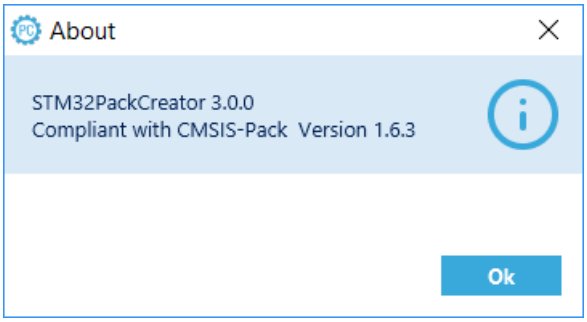
Compatibility with STM32CubeMX

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

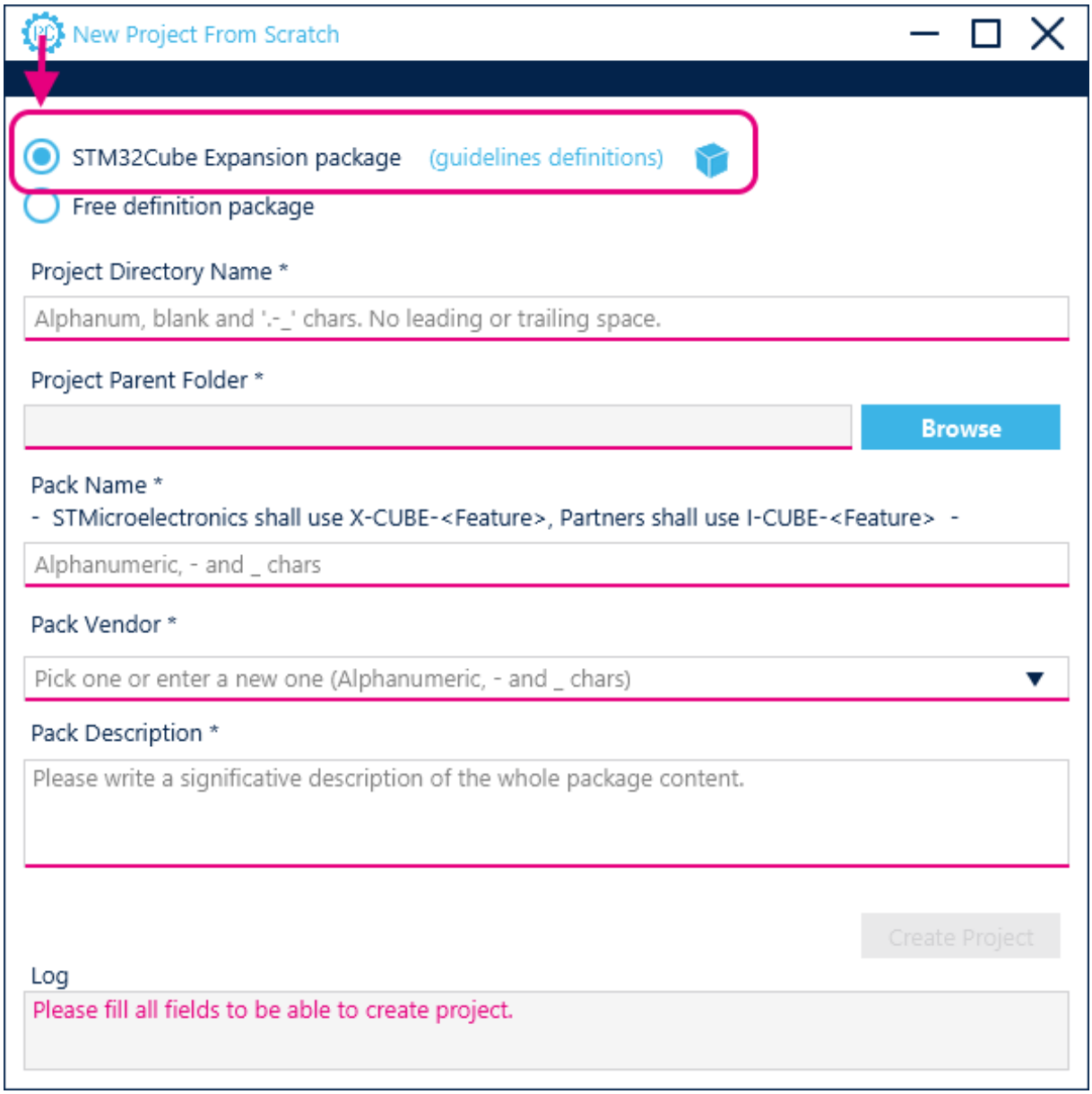


Compliance with STM32Cube Expansion rules

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**



4.4 System requirements

Supported operating systems and architectures are listed below:

- Windows® 8.x: 64-bit (x64)
- Windows® 10: 64-bit (x64)
- Linux® (tested on Red Hat®, Fedora®, and Ubuntu®, 64 bits)
- macOS® 64-bit (x64) (tested on version OS X® El Capitan and Sierra)

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 Liberica 1.8.0_265 from BellSoft.

Note: 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 1.8_45 (known limitation with 1.8_251).
- Version 11 is supported.
- 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™, no-cost solution based on OpenJDK. The JDK installer is recommended.

STM32PackCreator operation is not guaranteed with other JRE™s.

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

Red Hat® is a registered trademark of Red Hat, Inc.

Fedora® is a trademark of Red Hat, Inc.

Ubuntu® is a registered trademark of Canonical Ltd.

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

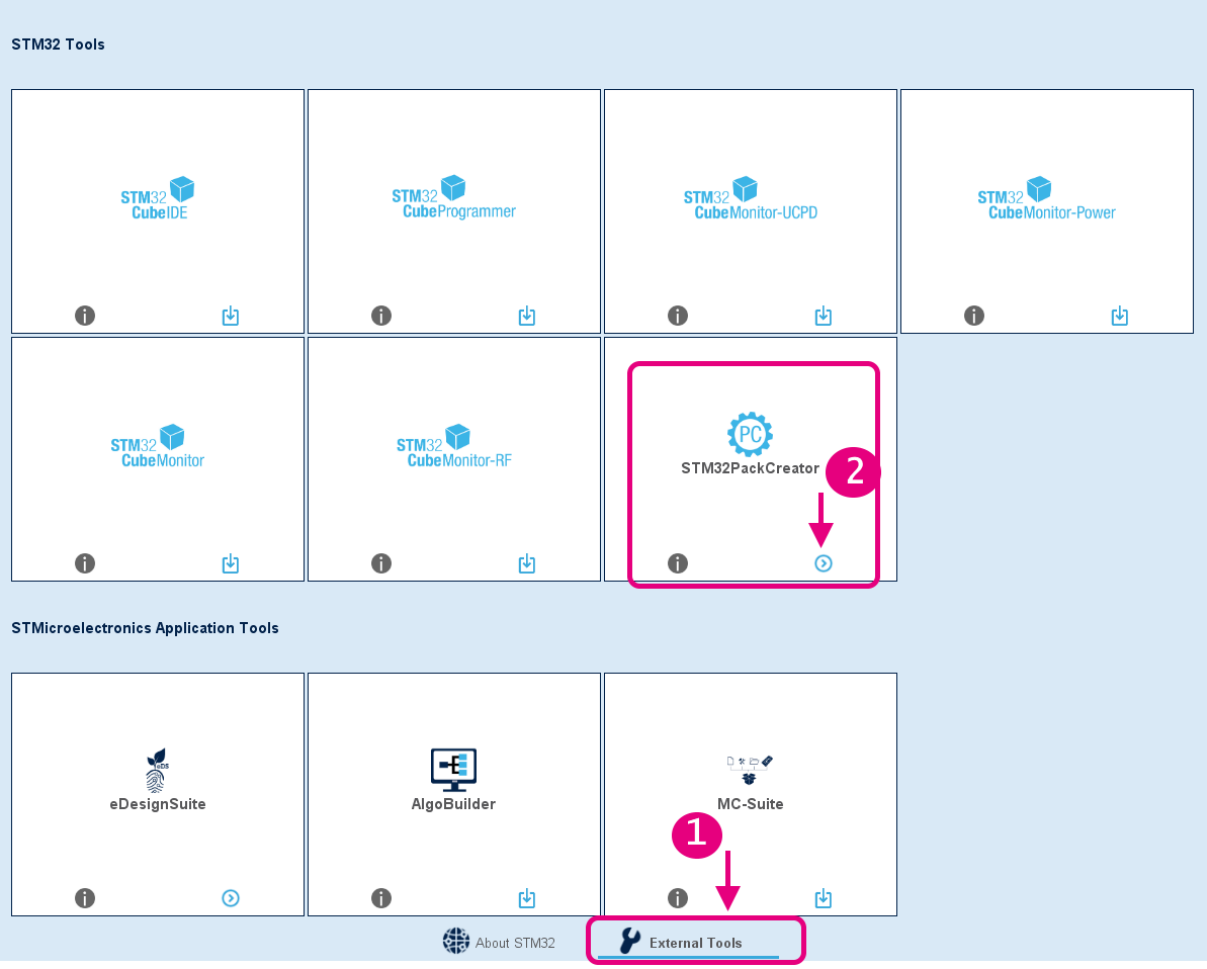
Amazon and Corretto are trademarks of Amazon in the United States and/or other countries.

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 STM32CubeMX installation path, in the `Utilities` folder.

To launch STM32PackCreator:

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

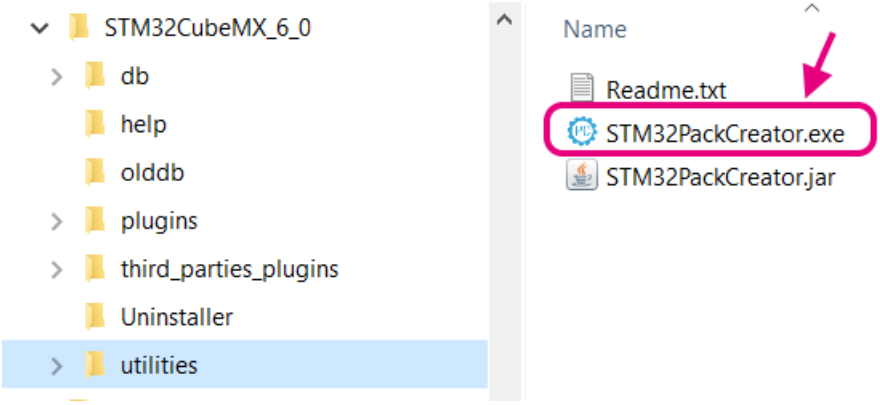
- On Linux and macOS®, use the following command from a terminal window:

  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 switch immediately to that cloned project.
[**Close Project**]	It closes the project and switch 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

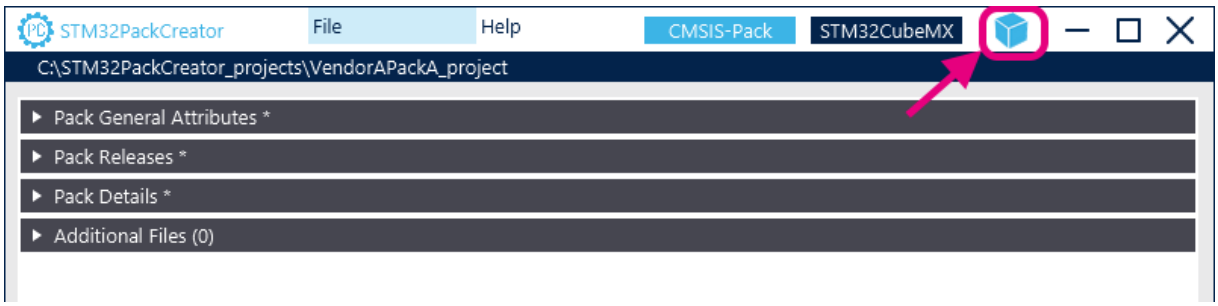
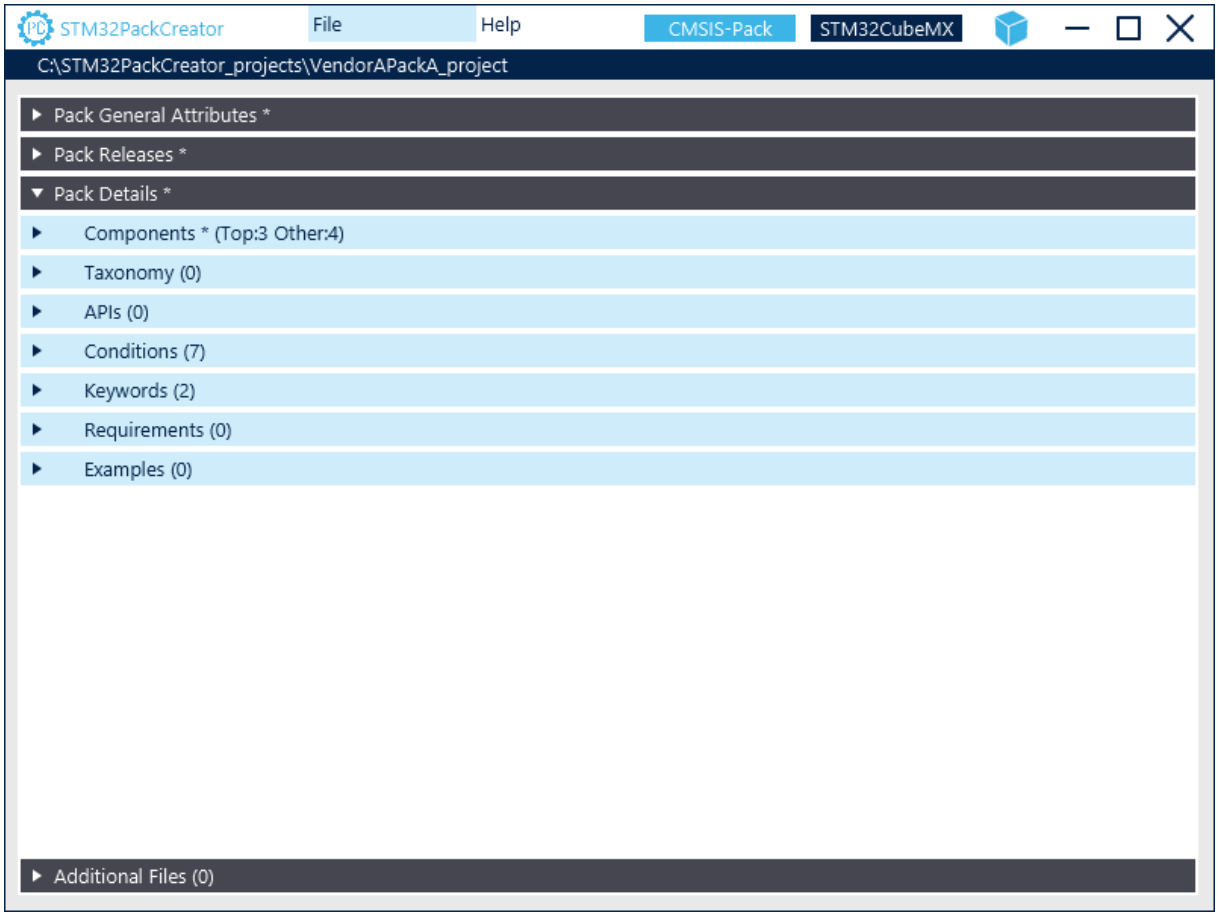


Figure 8. CMSIS-Pack view



It consists of the following entries. The different elements are described in the CMSIS-Pack standard. Tooltips are accessible by clicking the icon and 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, 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 path 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 to generate pack specific code using the STM32CubeMX code generation feature.

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

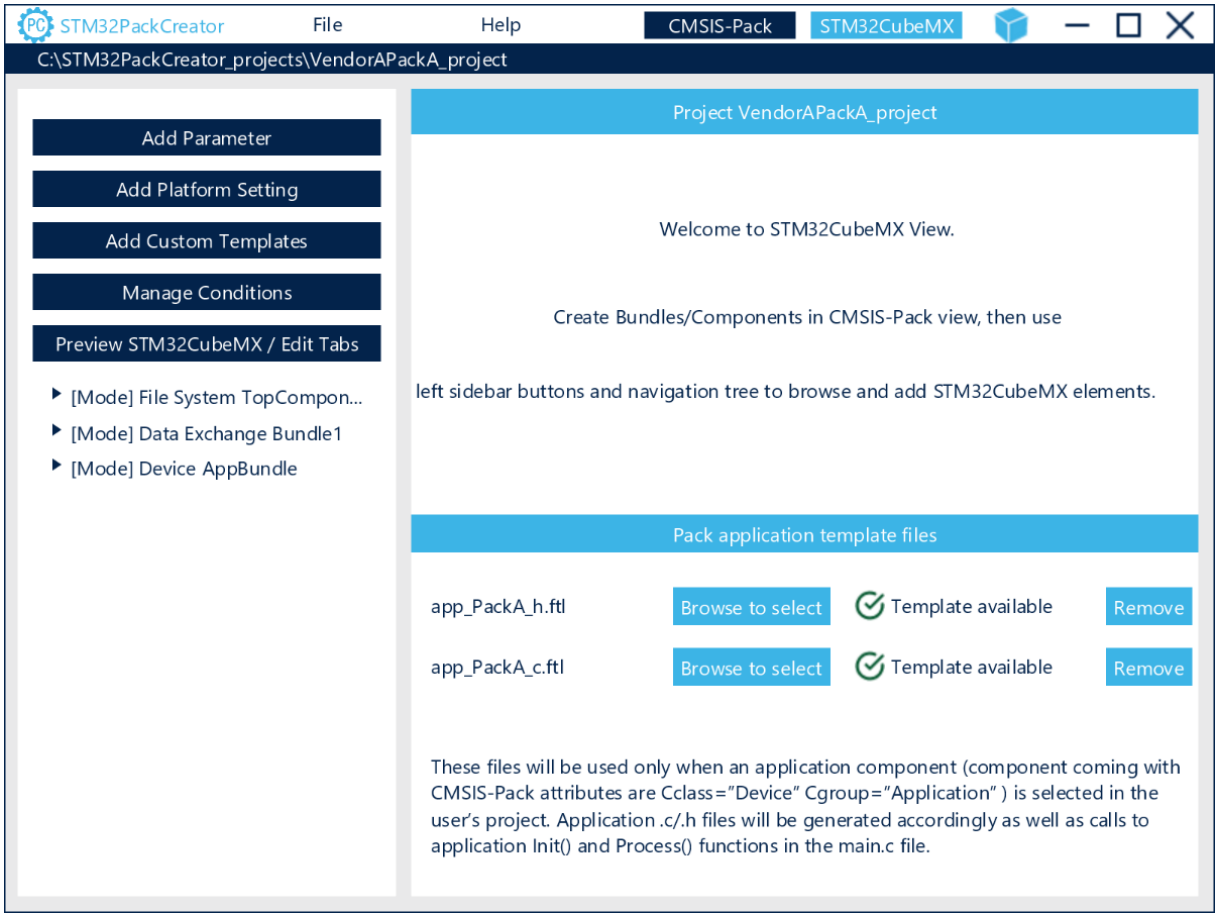


Table 4. **STM32CubeMX view - Main features**

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

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 they 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:

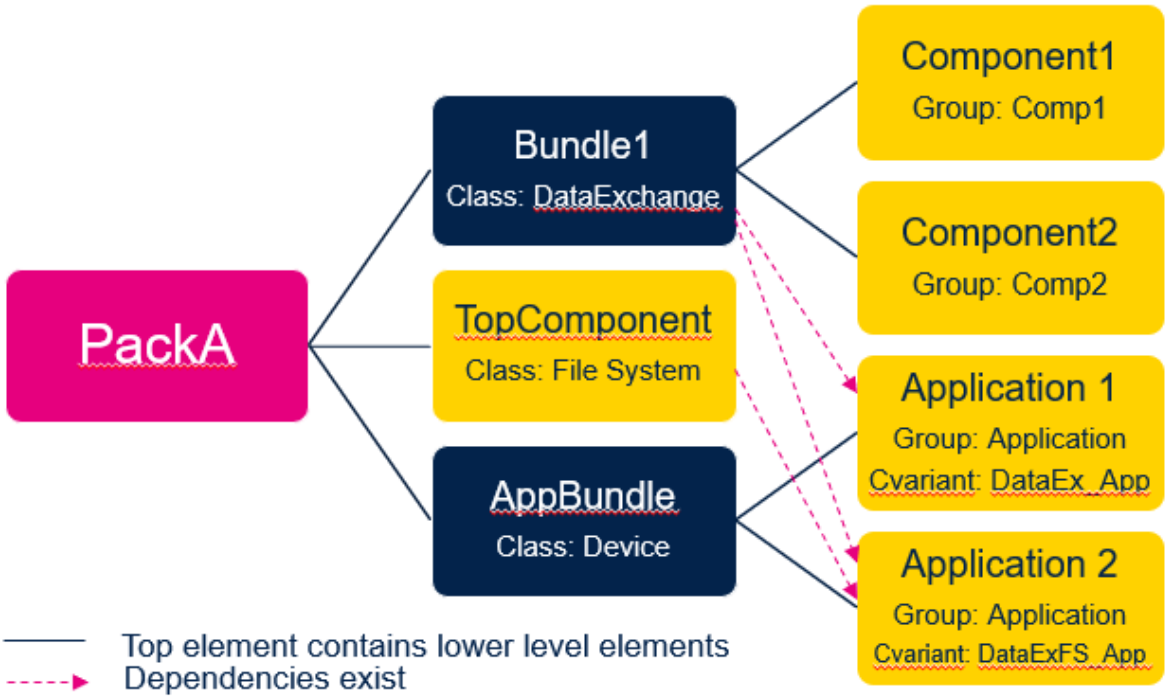
- Source, Header, Documentation files
  - Which components/bundles they must belong to.
- Freemarker template files (*) used for custom code generation by STM32CubeMX.
- Additional files (*)
  - Files that are present in the pack but not referenced in the `.pdsc` file, such as STM32 HAL driver files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

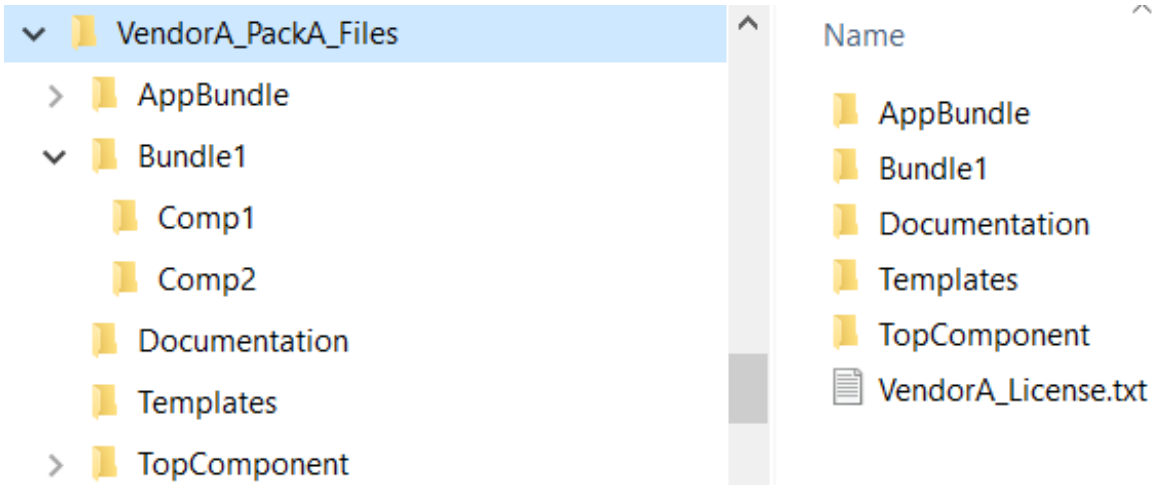


The pack comes with:

- 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

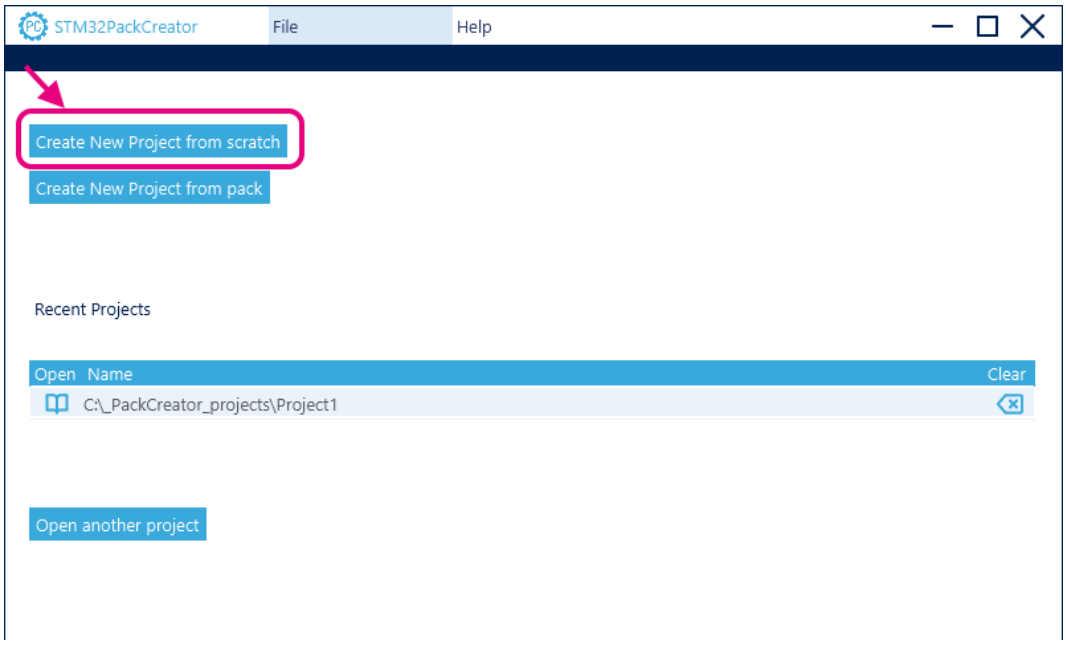
A folder is ready with all the files to be packaged in the pack.

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



5.2 Create a new project from scratch

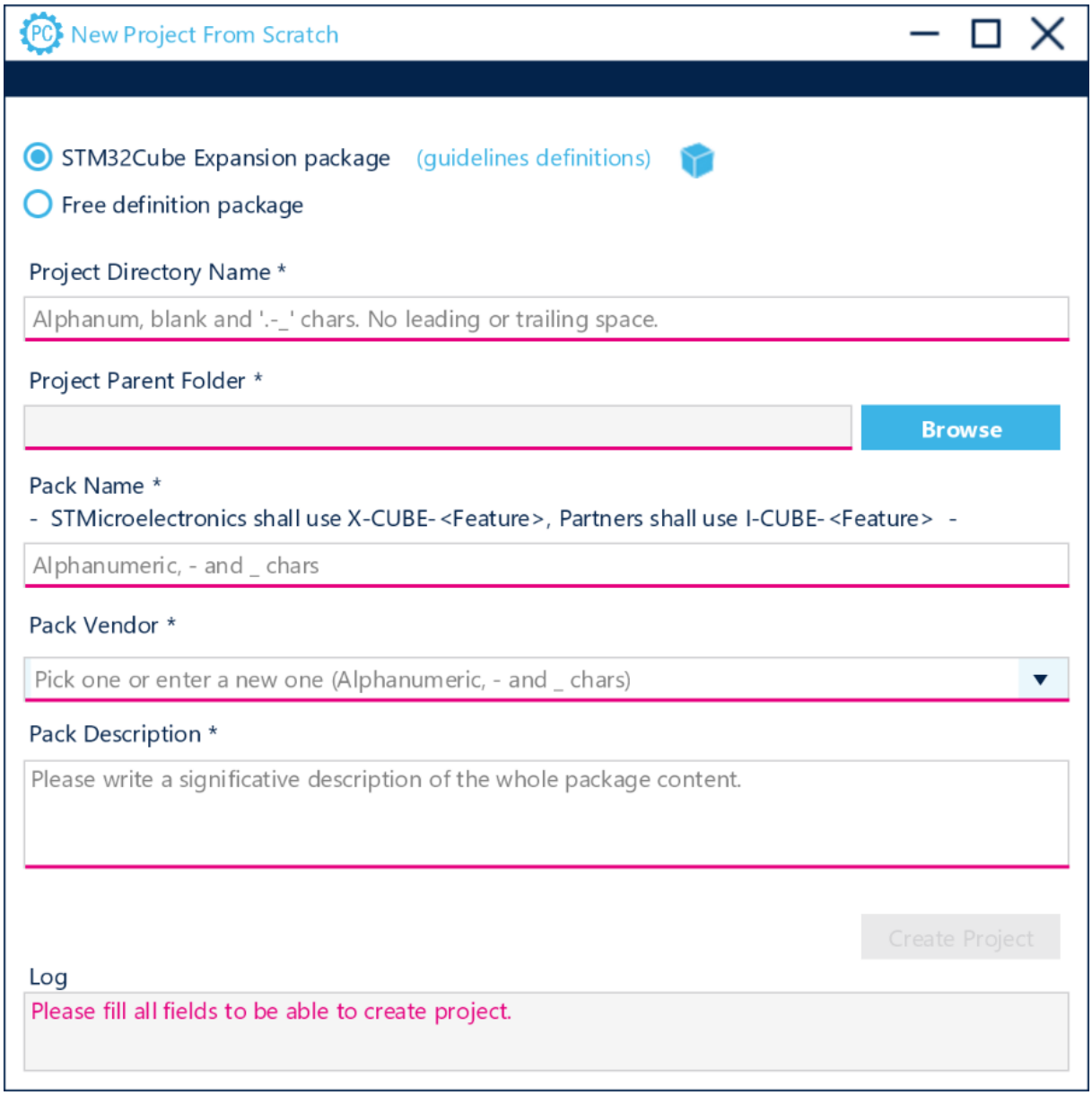
Figure 12. Create a new project from scratch



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.

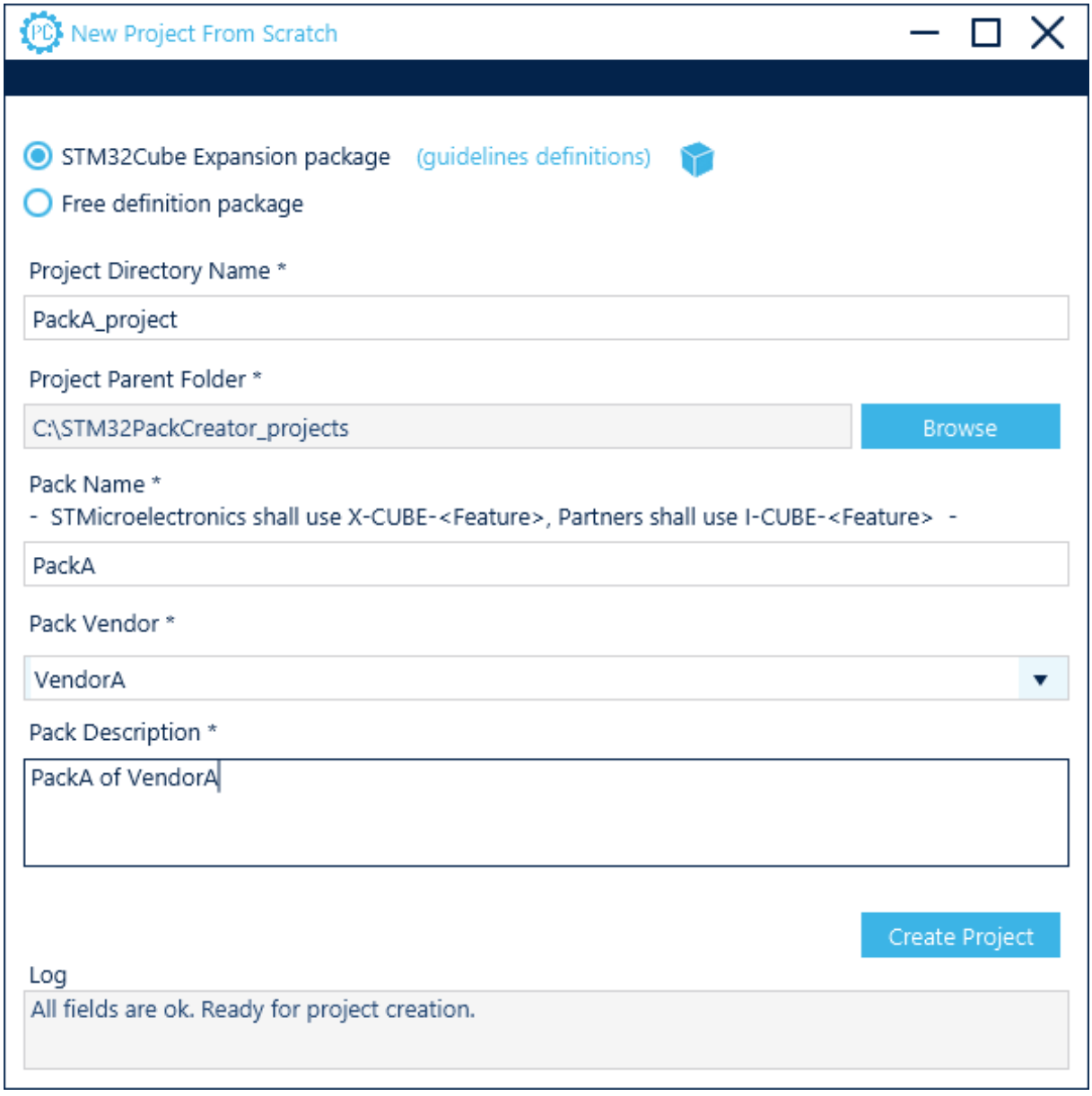
Figure 13. New project from scratch 1/2



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

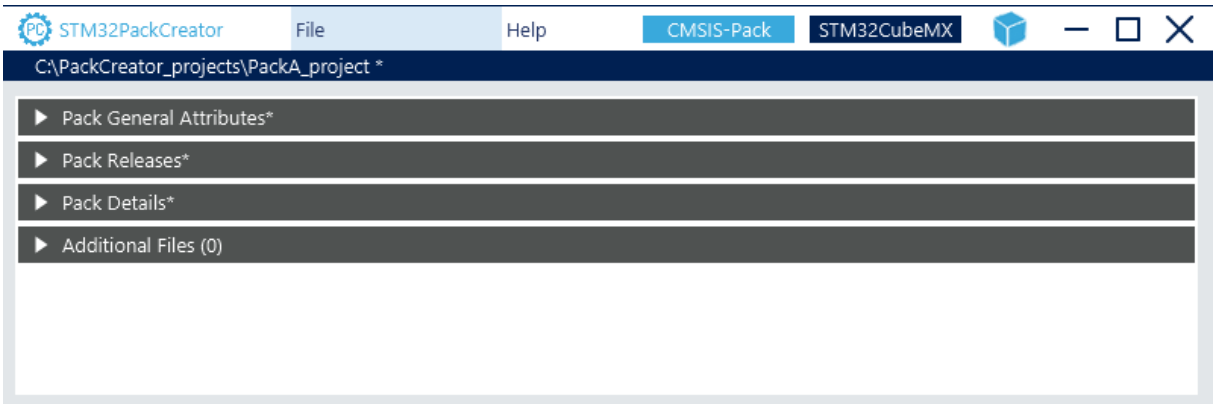
5. Fill in project details:
   a. Project folder name
   b. Project parent folder
   c. Pack name
   d. Vendor name
   e. Pack description

Figure 14. New project from scratch 2/2



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 attributes 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

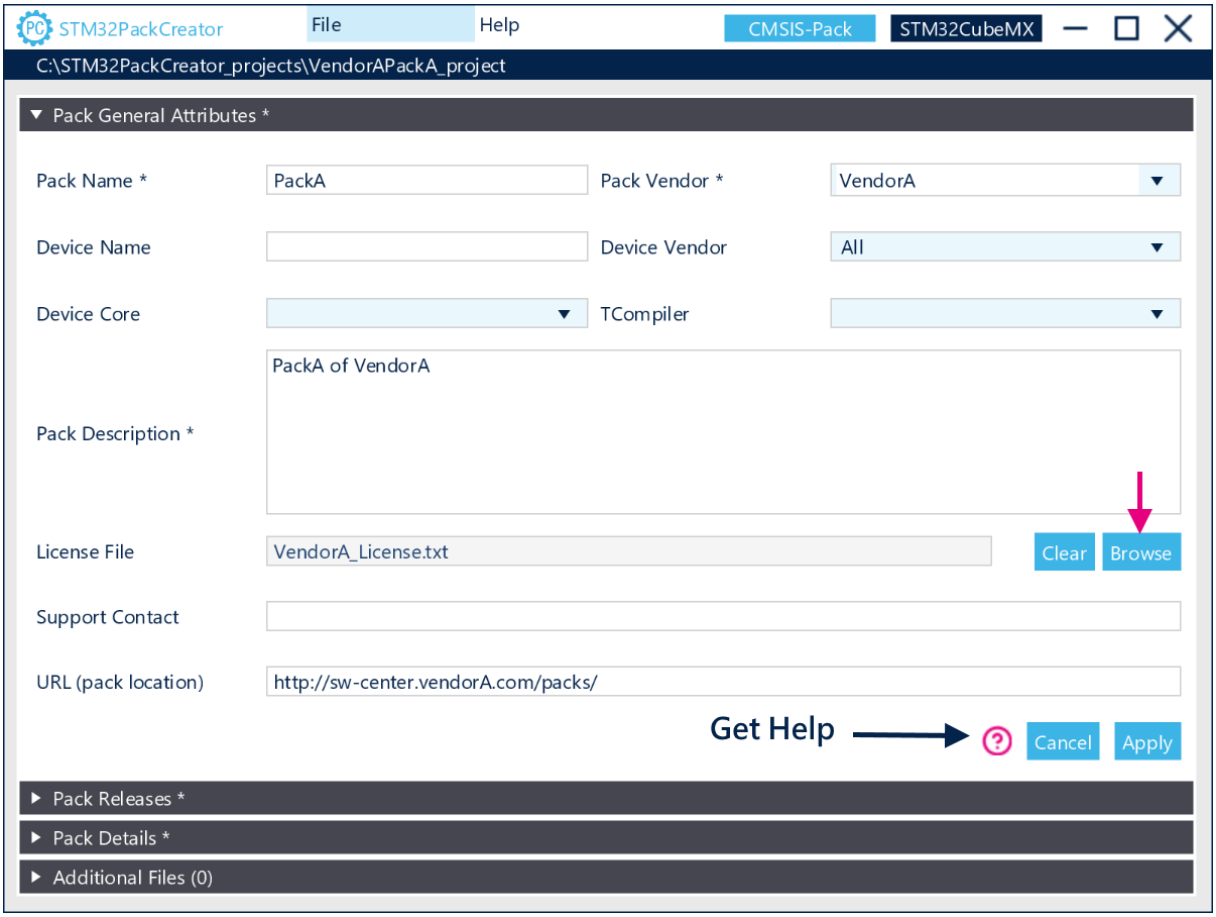
Click the left arrow to expand the Pack General Attributes section.

Pack license

STM32CubeExpansion packs must come with a license file

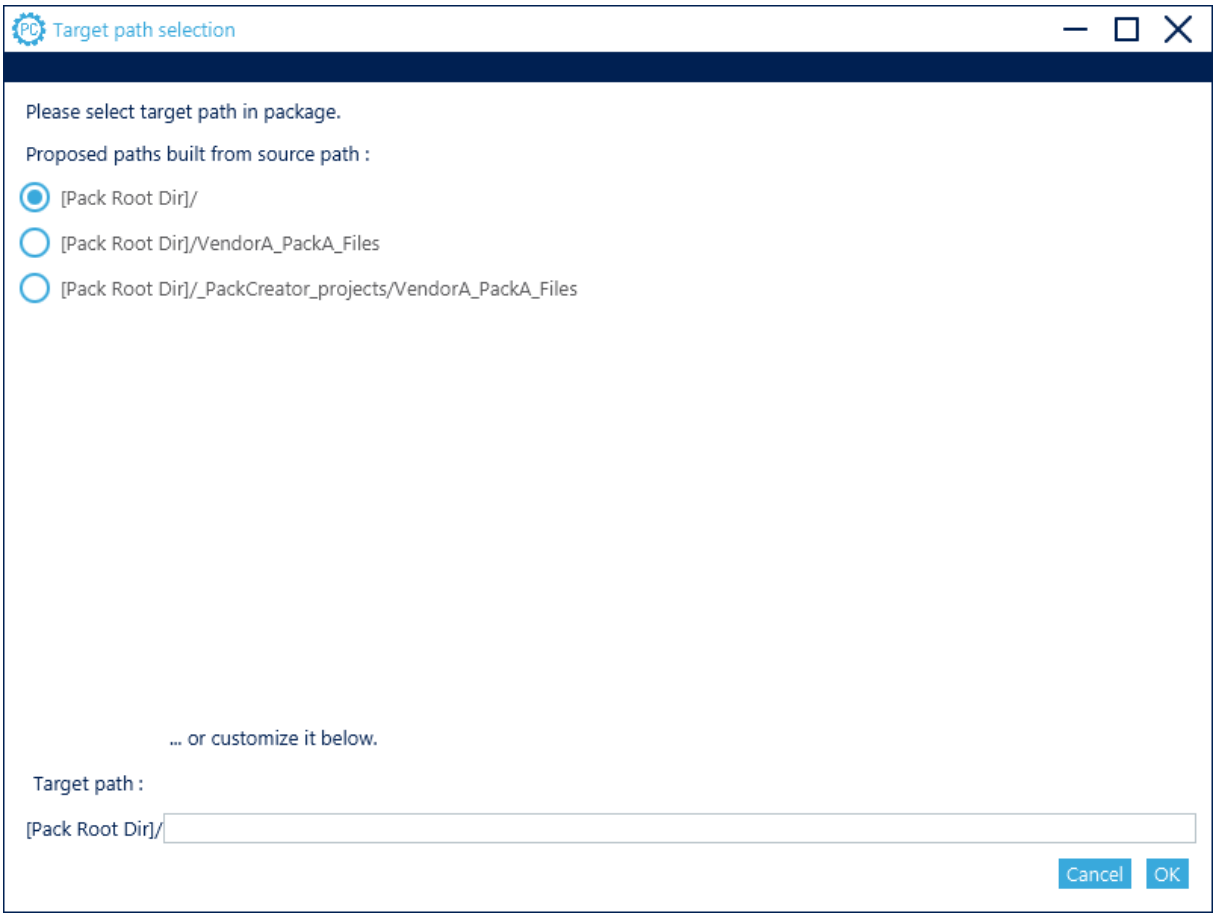
1. Click on browse and select the pack license. Refer to Figure 16.

Figure 16. Selecting the license file



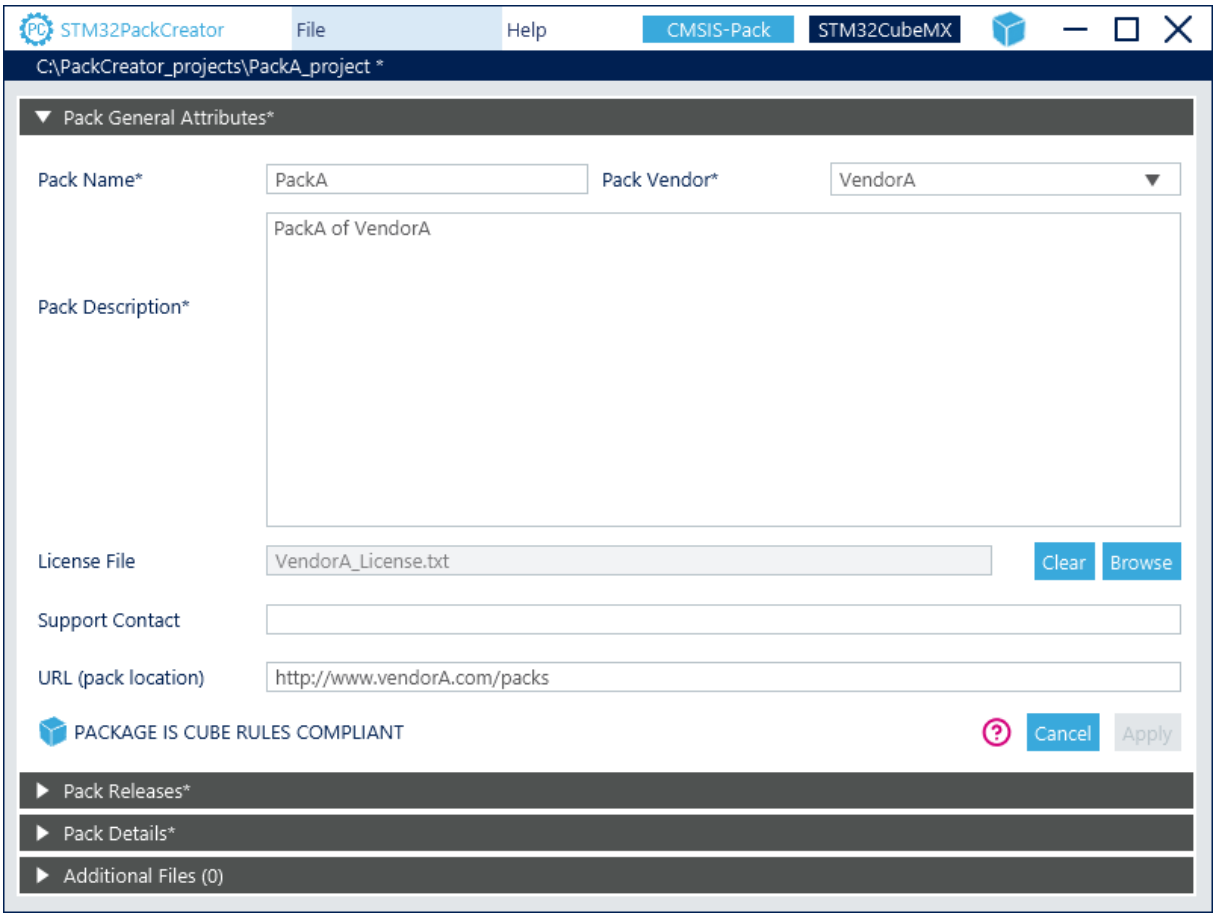
2. Select the destination folder to be the pack root directory. Refer to Figure 17.

Figure 17. Selecting the pack folder for the license file



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



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.

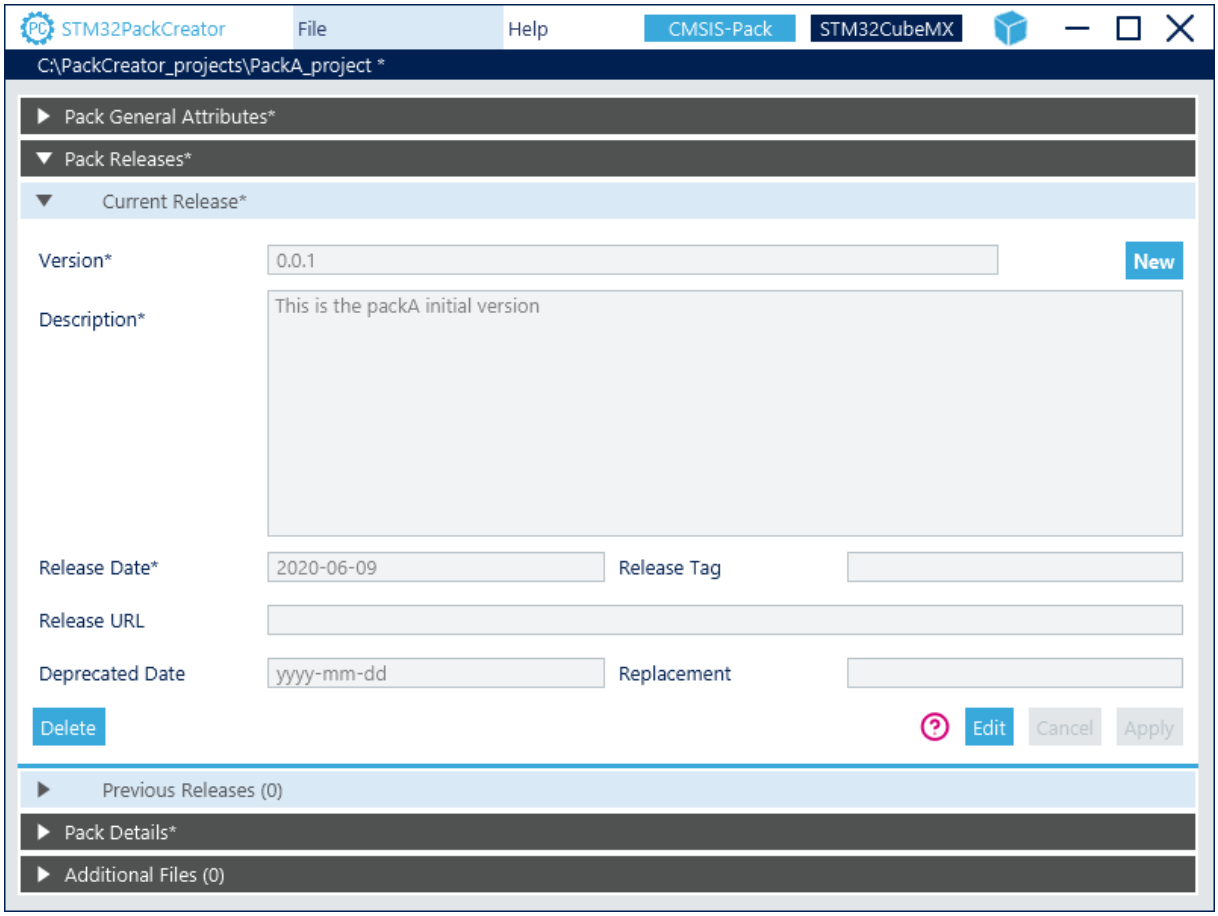
Initial release

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



Other actions

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

Important information

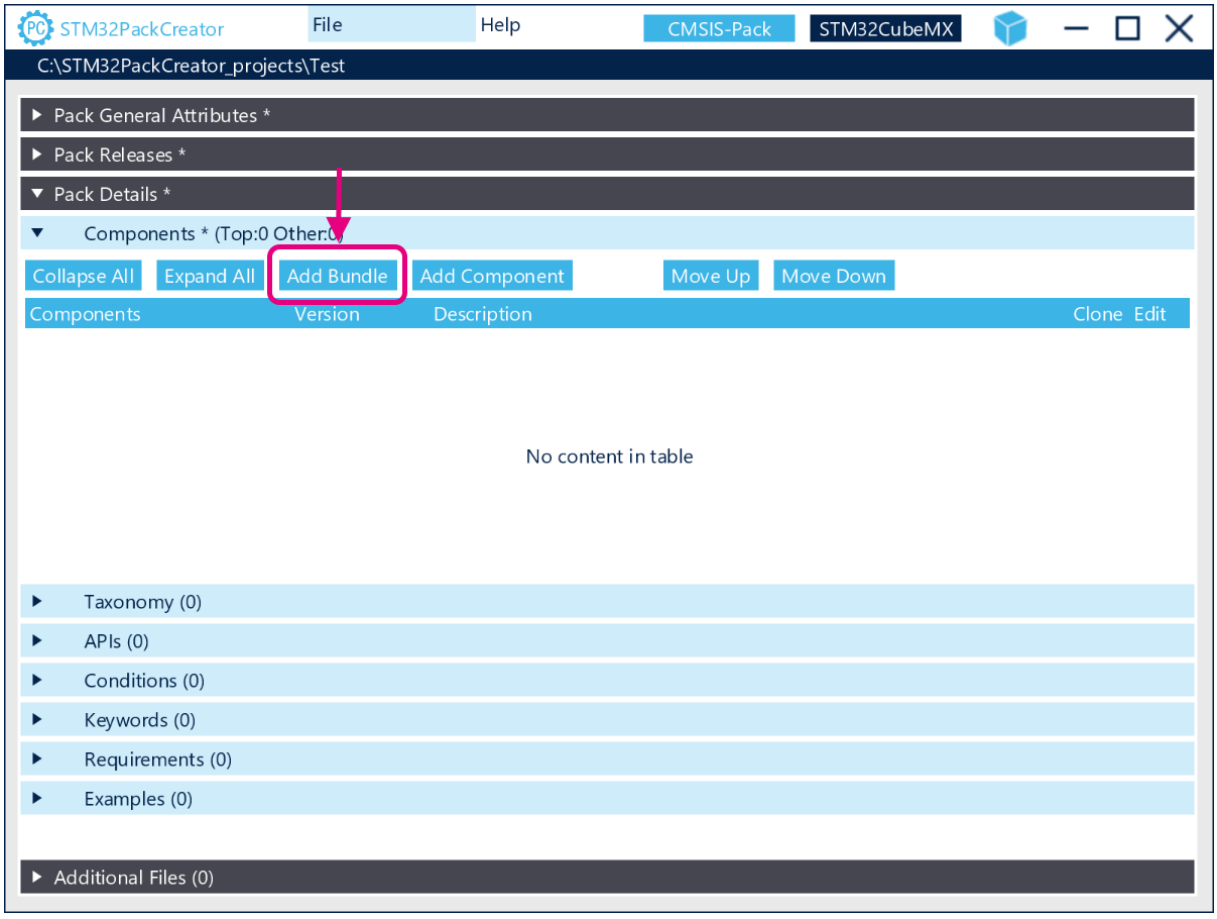
When dealing with official releases, do not delete the release history. According to the CMSIS-Pack standard, each new revision of the pack must come with the full release history. All previous release entries must be kept.

5.3.4 Create a bundle with two components

The first bundle comes with two components inside.

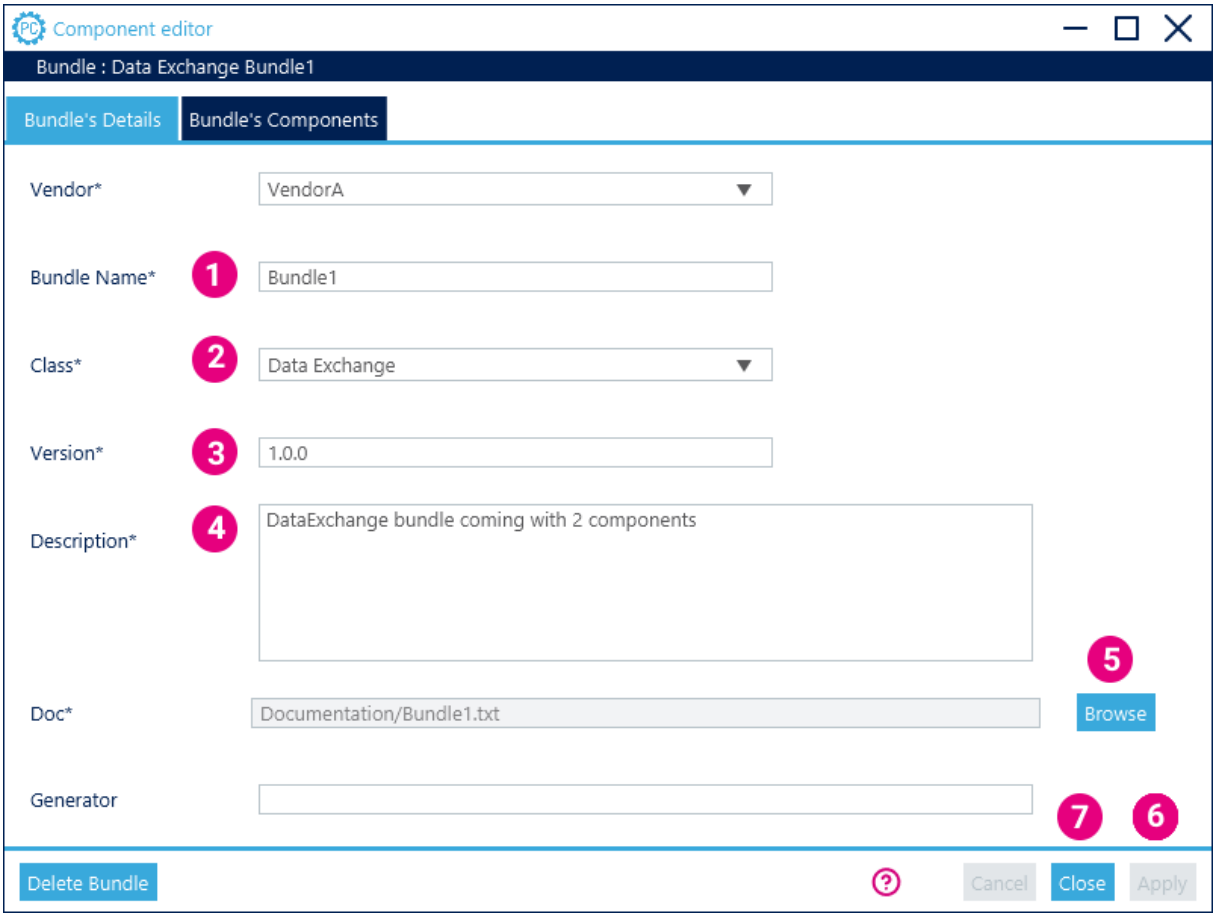
1. Click the left arrow to expand the Pack Details section.
2. Click `Add Bundle` to open the `Component editor` window.

Figure 20. Adding a bundle



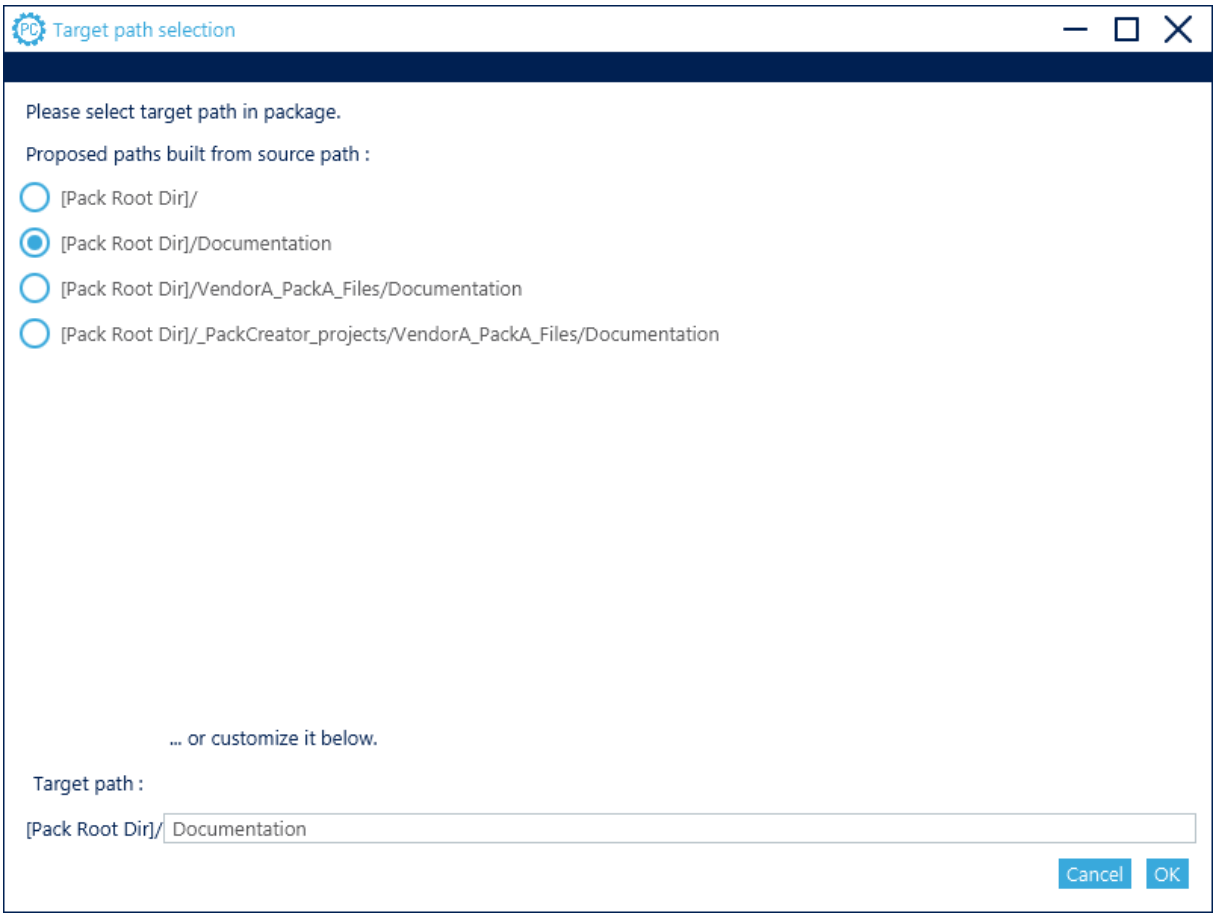
3. Fill in the bundle details, click `Apply`, and close the window.

Figure 21. Filling bundle details



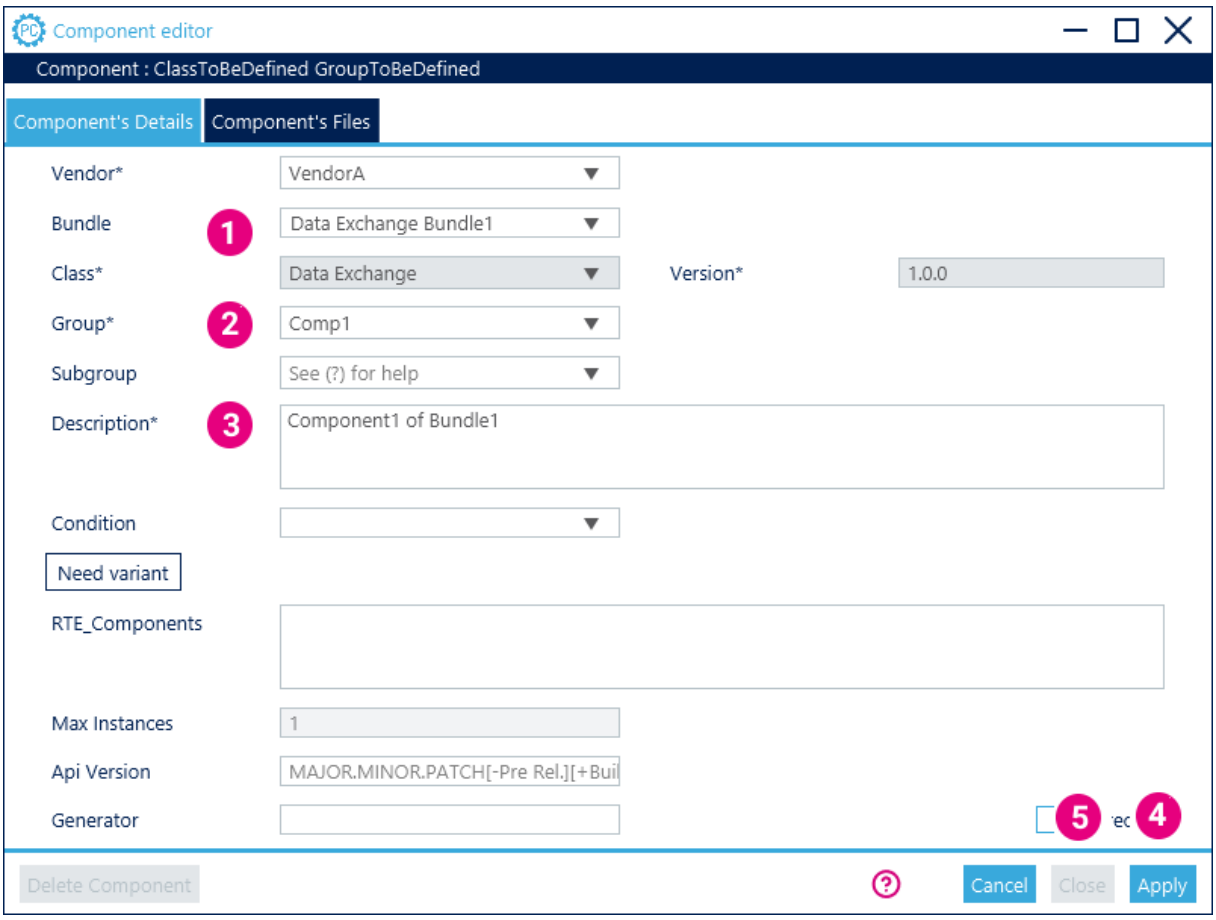
For step 5, select the Bundle documentation file and set its target path.

Figure 22. Specifying pack folder for bundle documentation



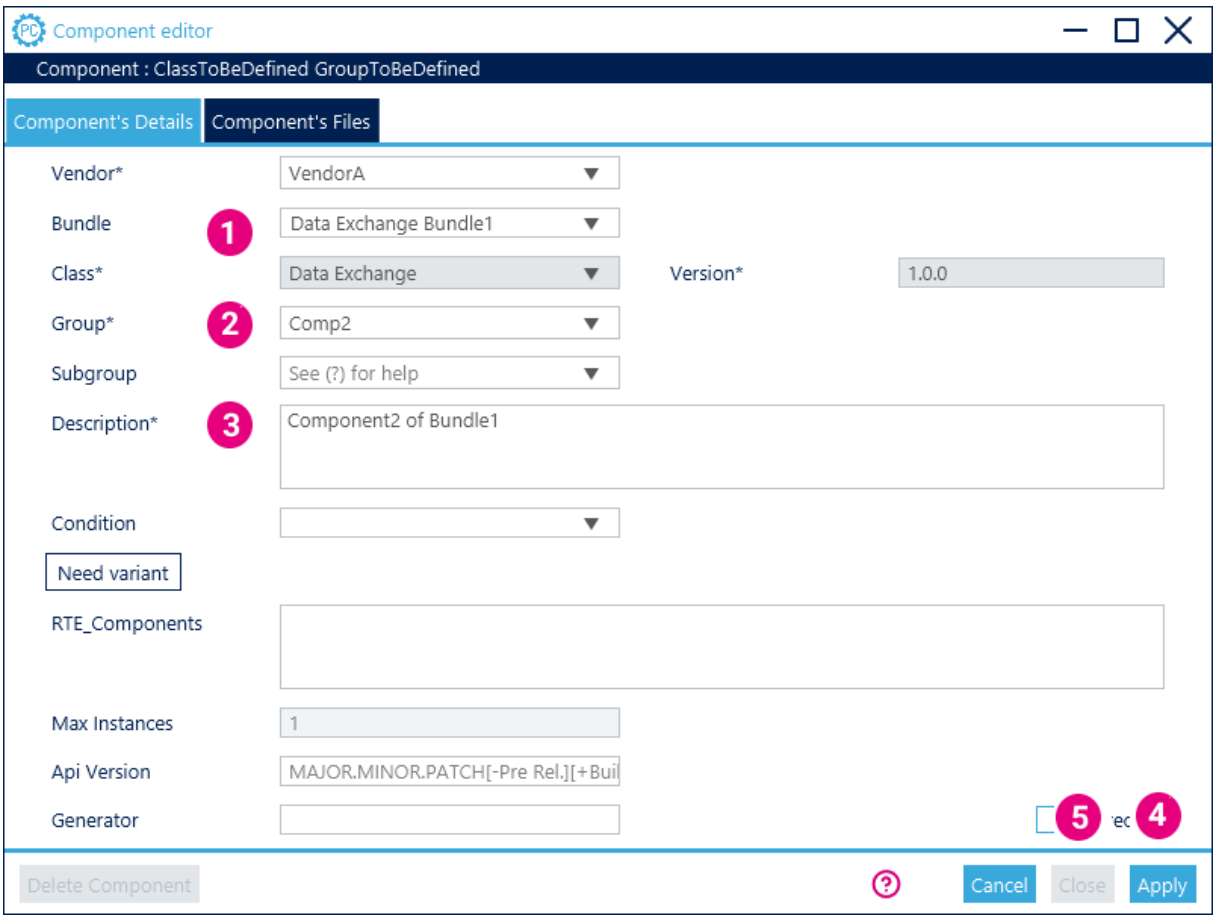
4. Create the first component for the bundle.
   Back to the Pack Details section, click `Add Component` and fill in the component details, then click `Apply` and close.

Figure 23. Creating component1 for Bundle1



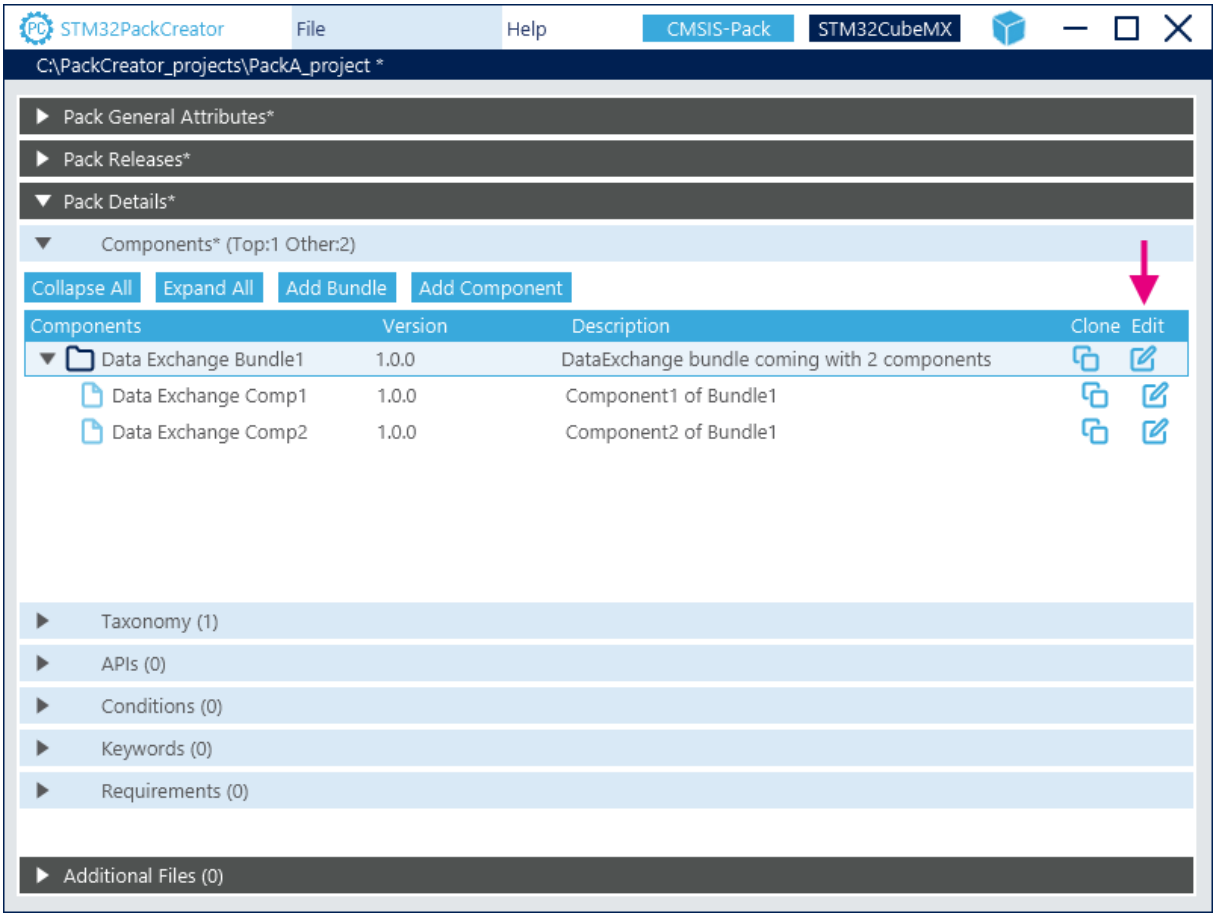
5. Create a second component for the bundle.
   Back to the Pack Details section, click `Add Component` and fill in the component details, then click `Apply` and close.

Figure 24. Creating component2 for Bundle1



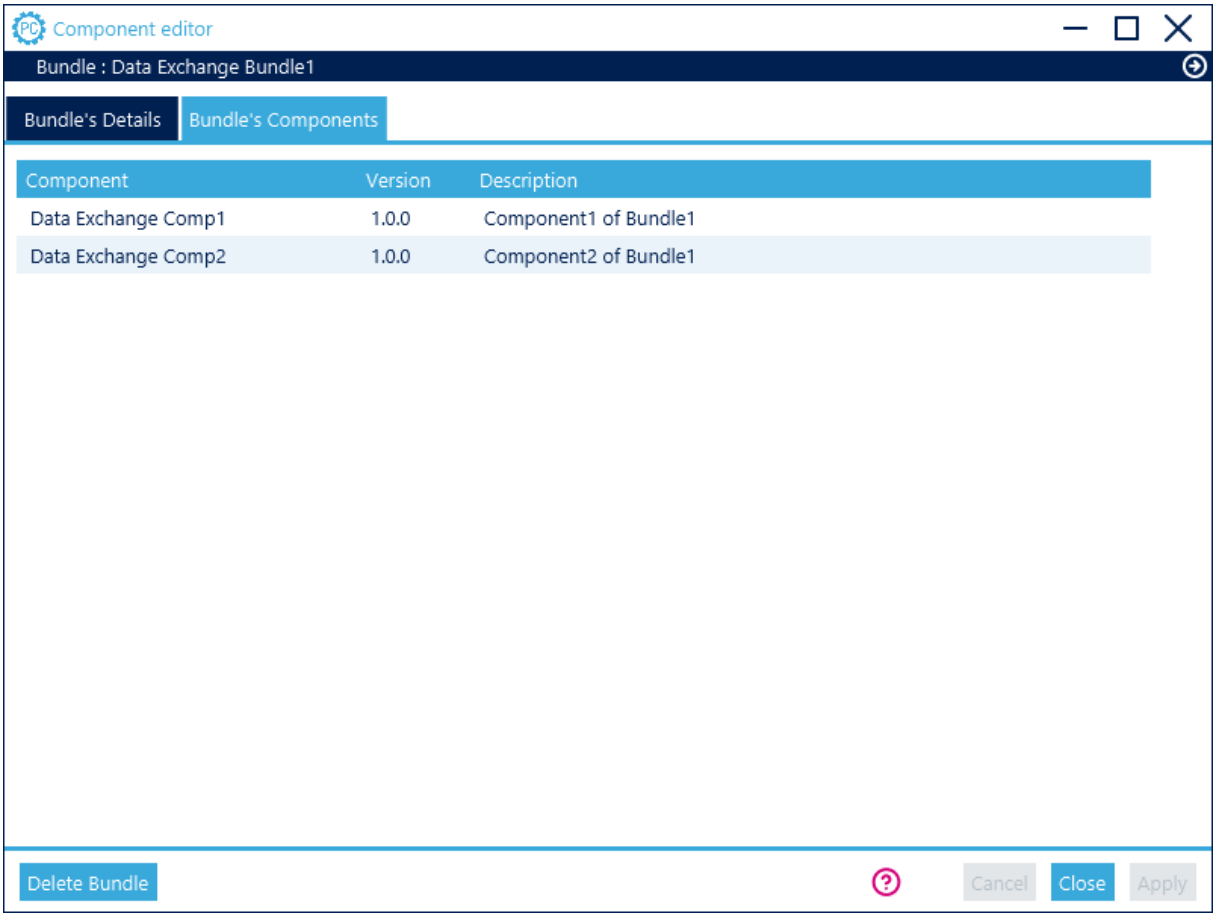
6. Verify the list of components are shown on the bundle's component tab
Back to the Pack Details section, click the edit icon on the bundle line to open the editor window.

Figure 25. Pack specified with one bundle and two components



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

Figure 26. Bundle's components tab view



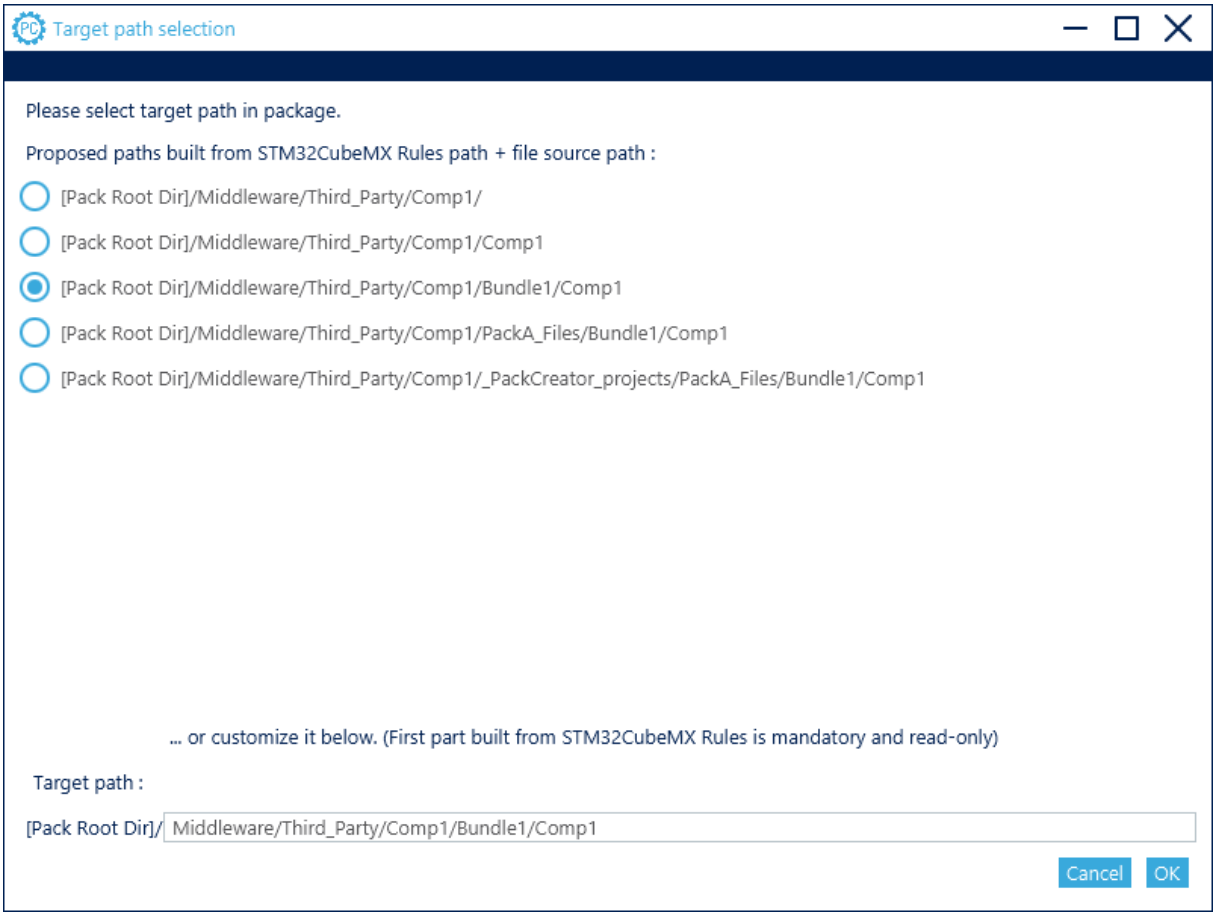
7. Add component files to Component 1

   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.

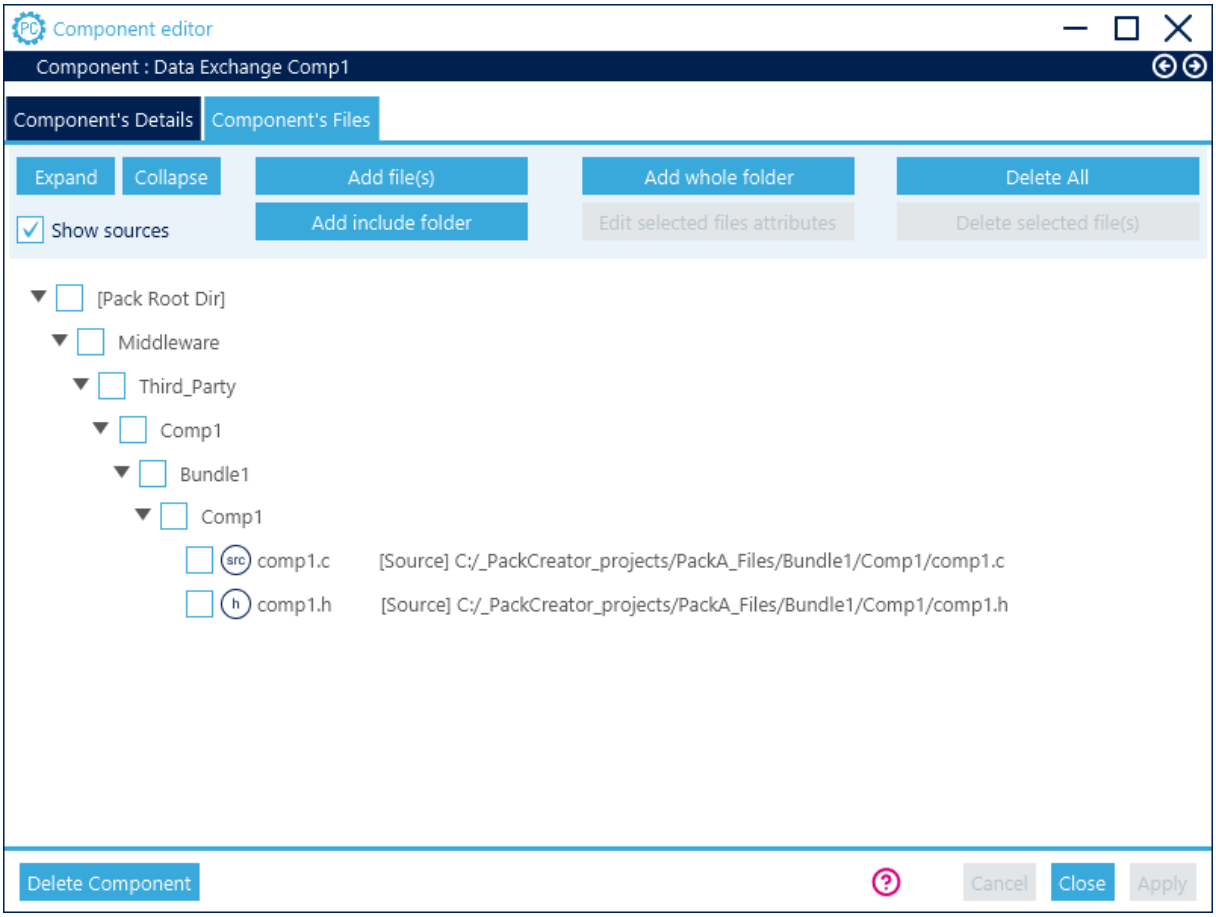
   Then, select the path in the pack where the selected files may be found.

Figure 27. Selecting the pack folder for the component1 files



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

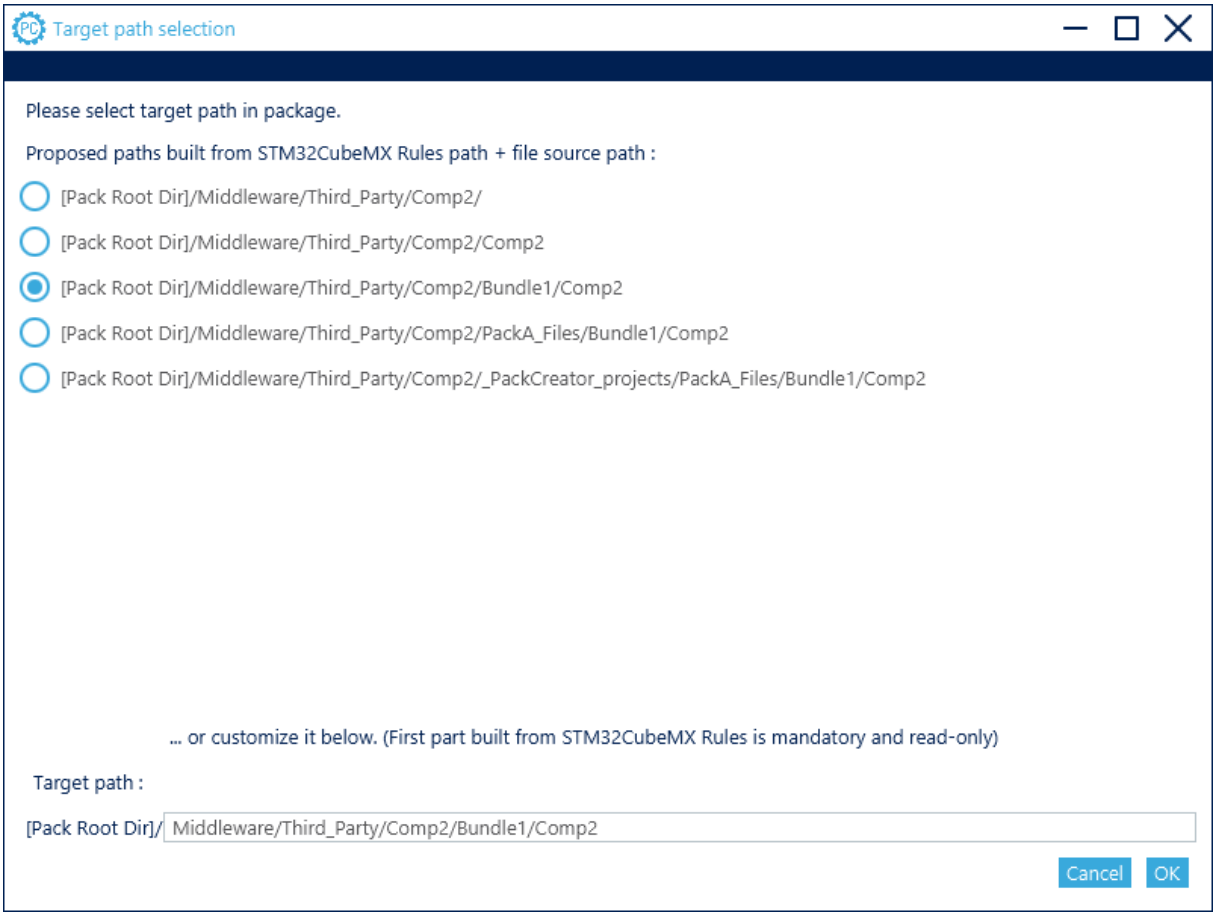
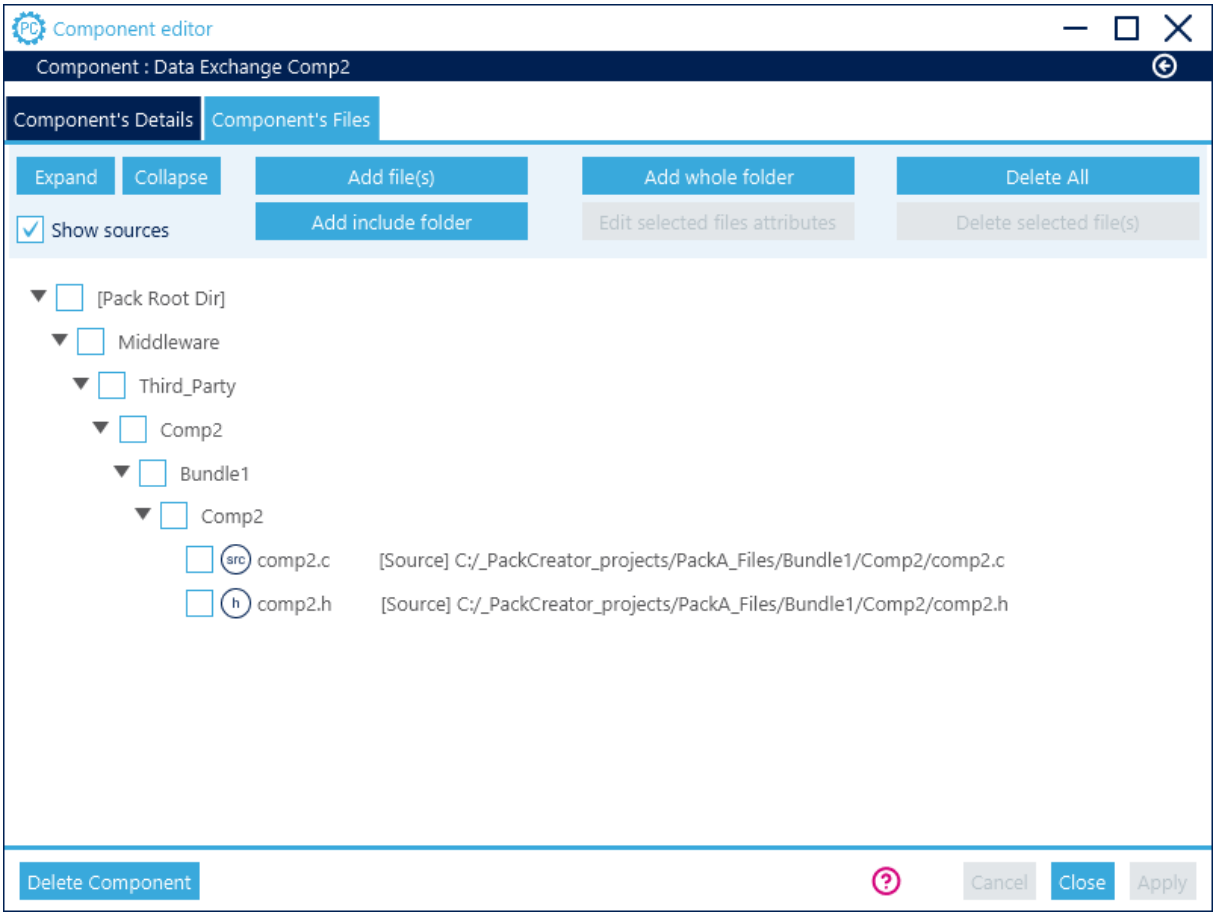
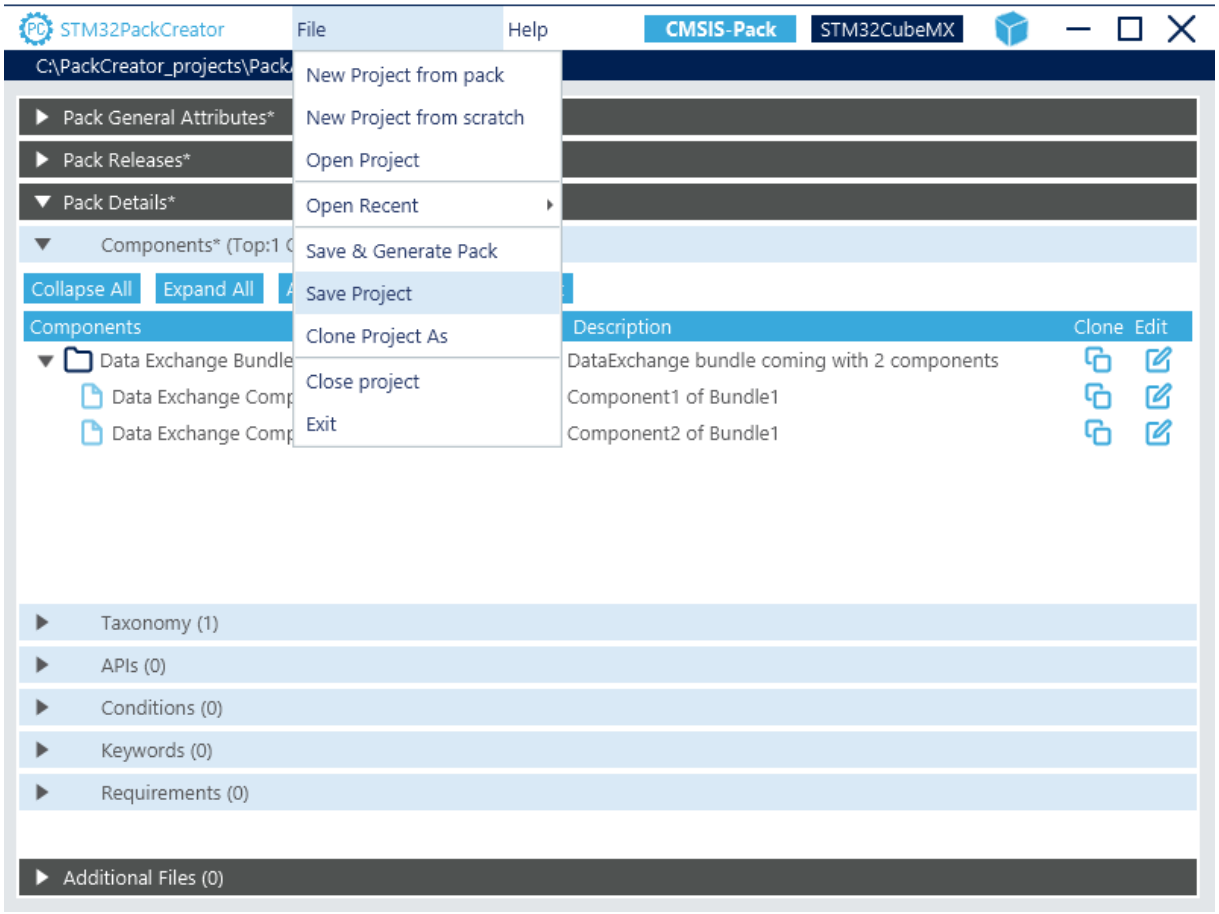


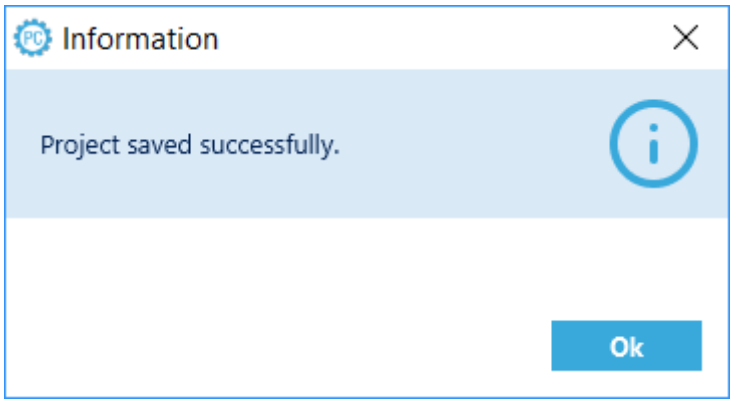
Figure 30. **Component2 with component files specified**



9. Save the project

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

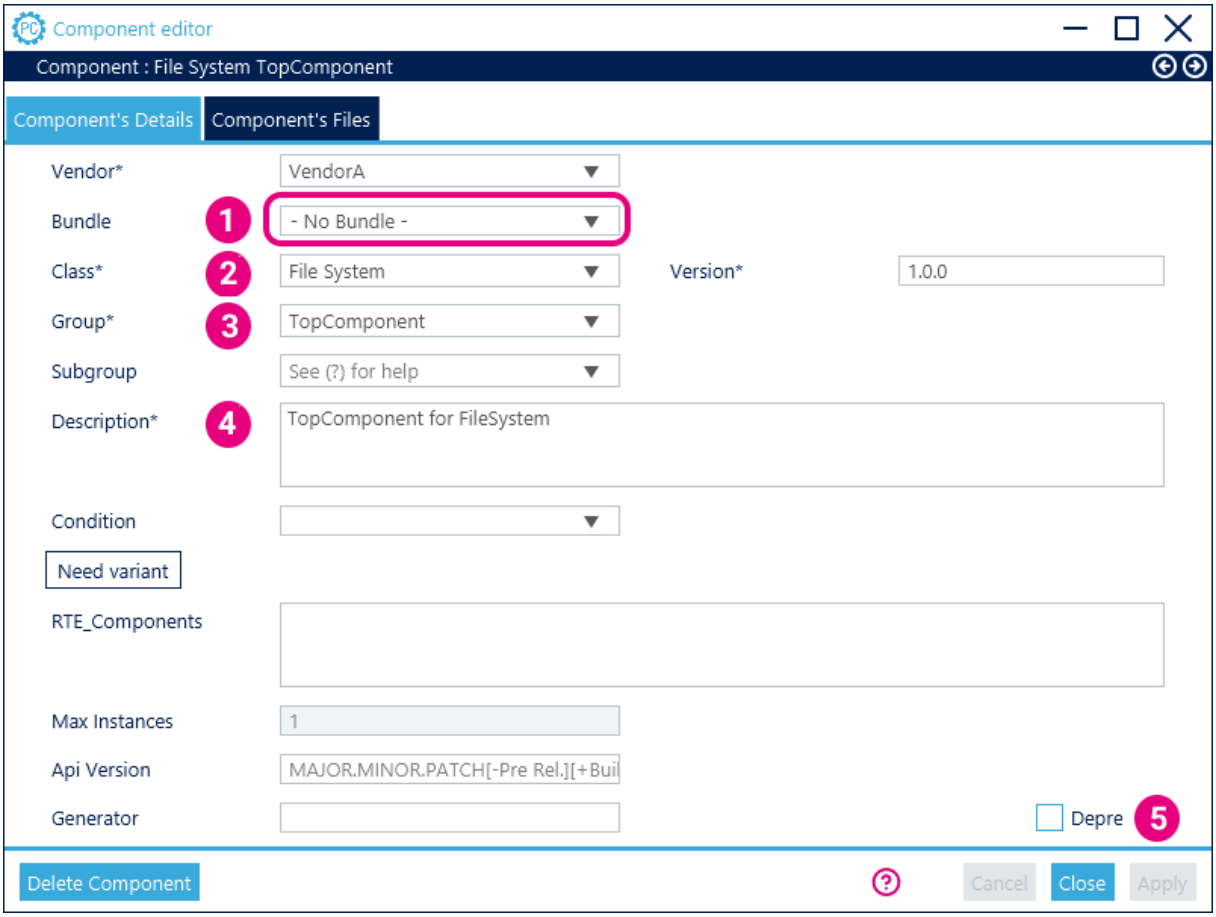




5.3.5 Create the top component

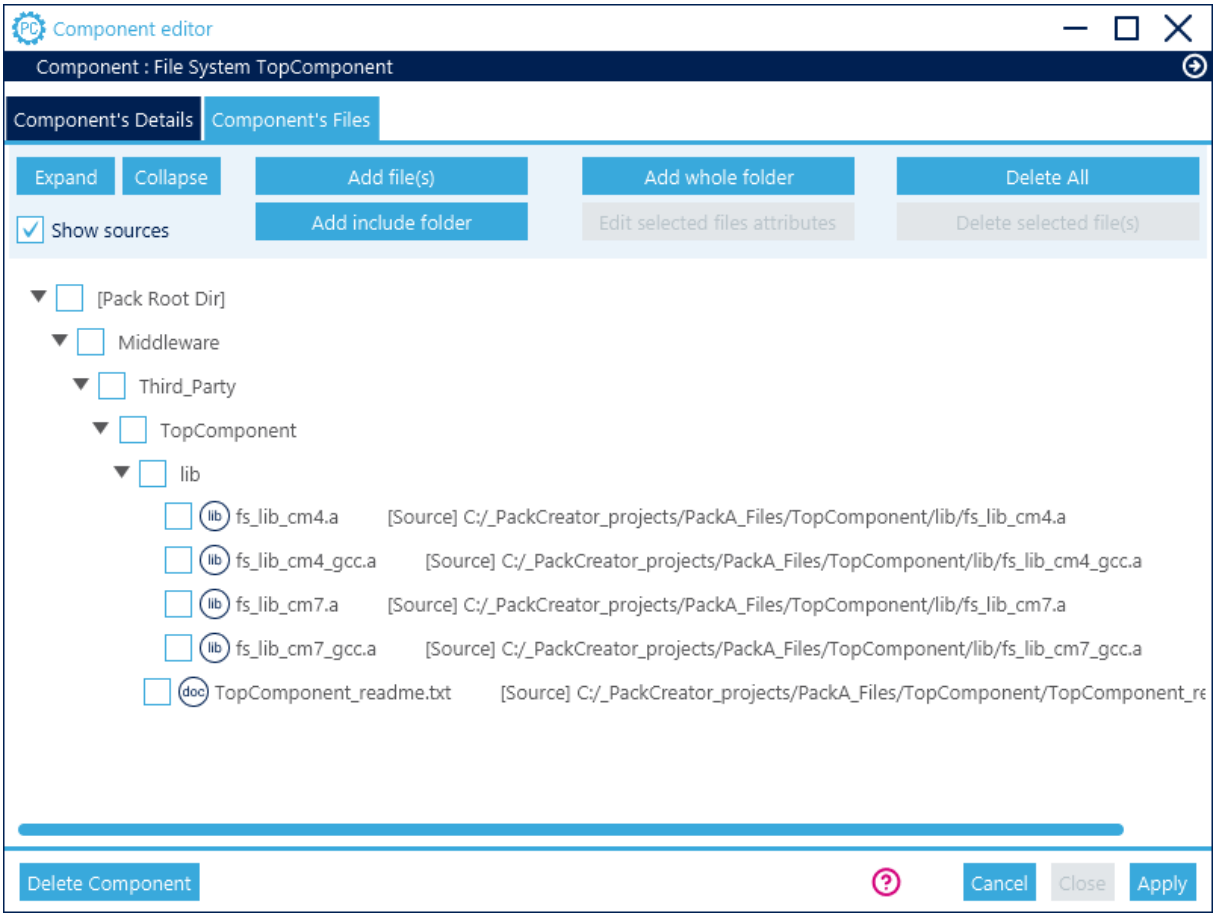
1. Back to the Pack Details section, click `Add Component` and fill in the component details, then click `Apply`. Since the component is a top component, the bundle is set to `No Bundle`.

Figure 31. Creating a top component



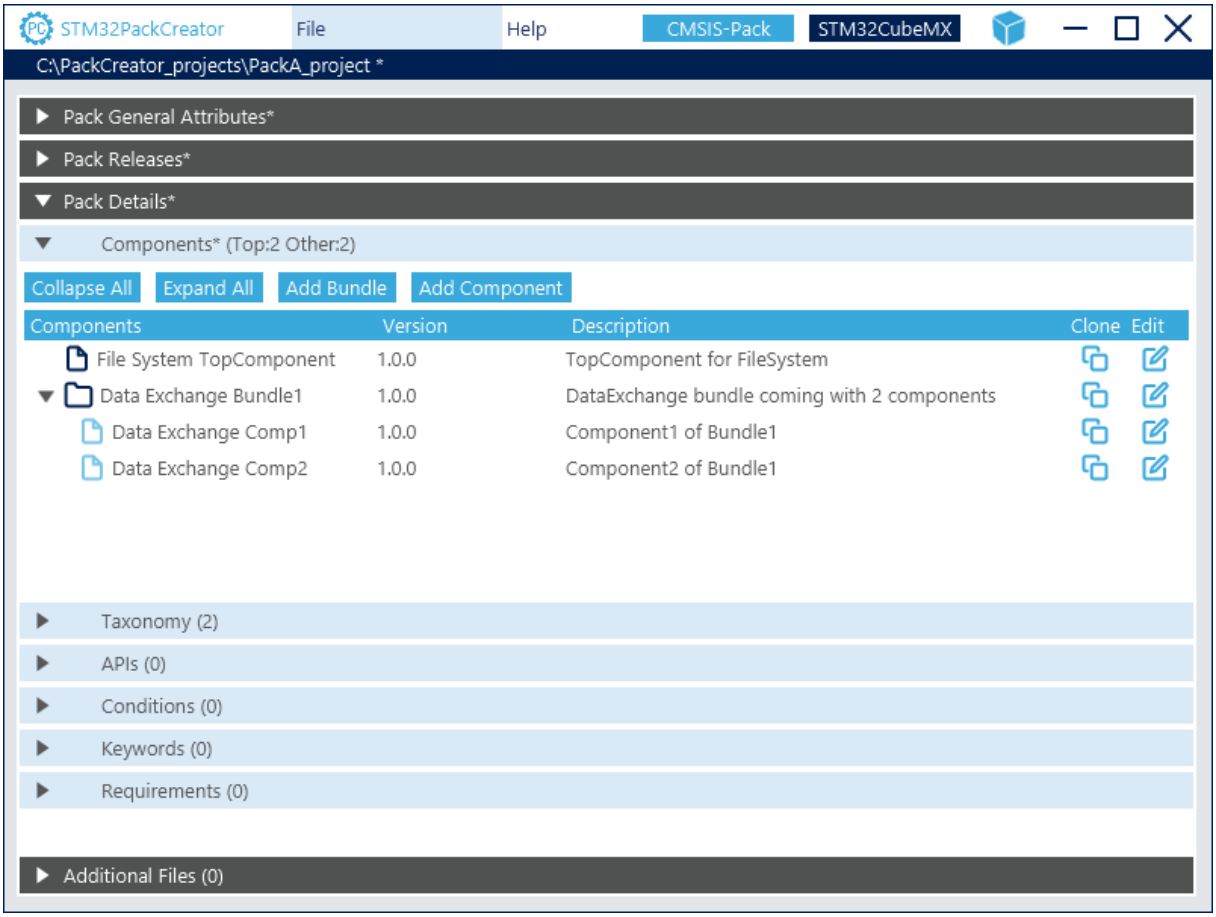
2. Select the Component's files tab then click `Add whole folder:` browse to select the folder location. Set the path in the pack where the selected files may be found.

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

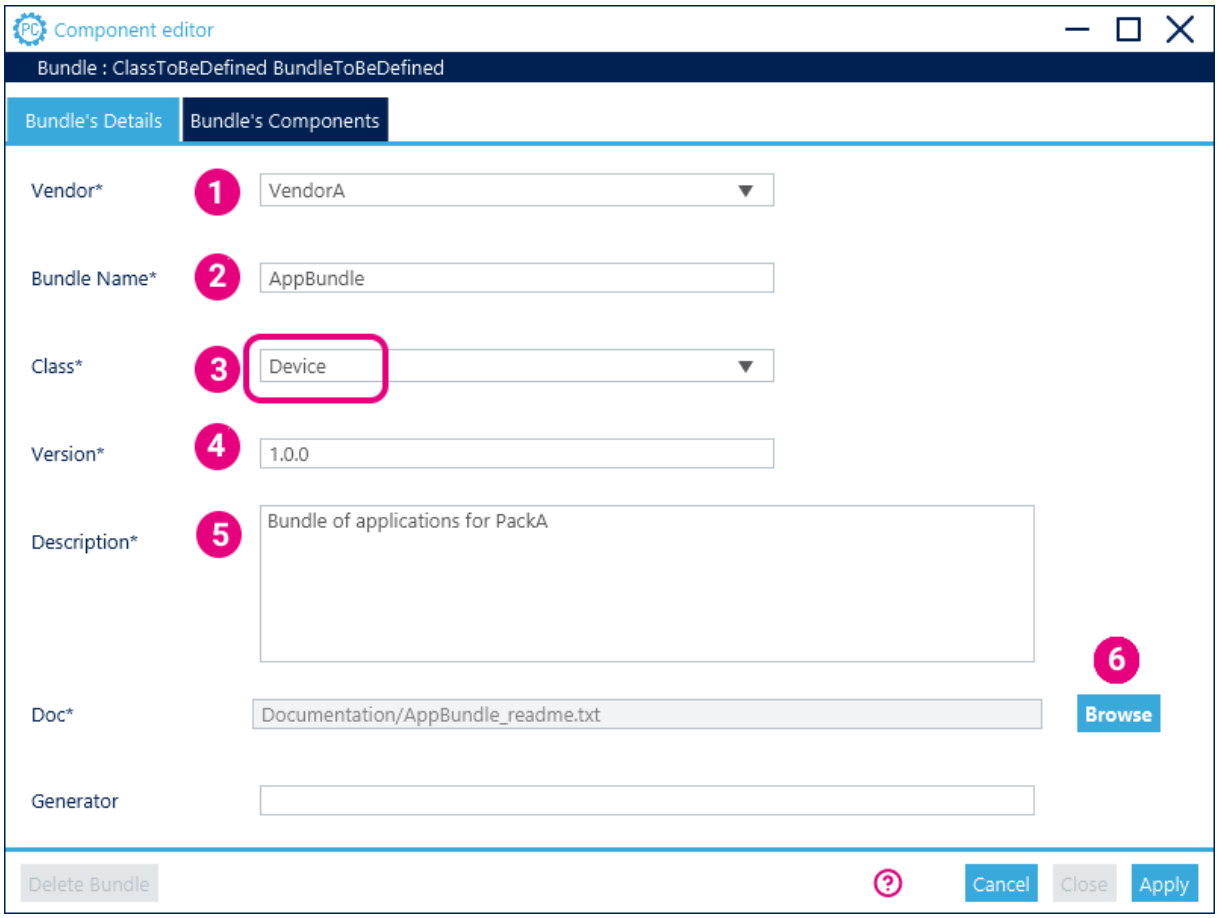


Figure 35. **First application component**

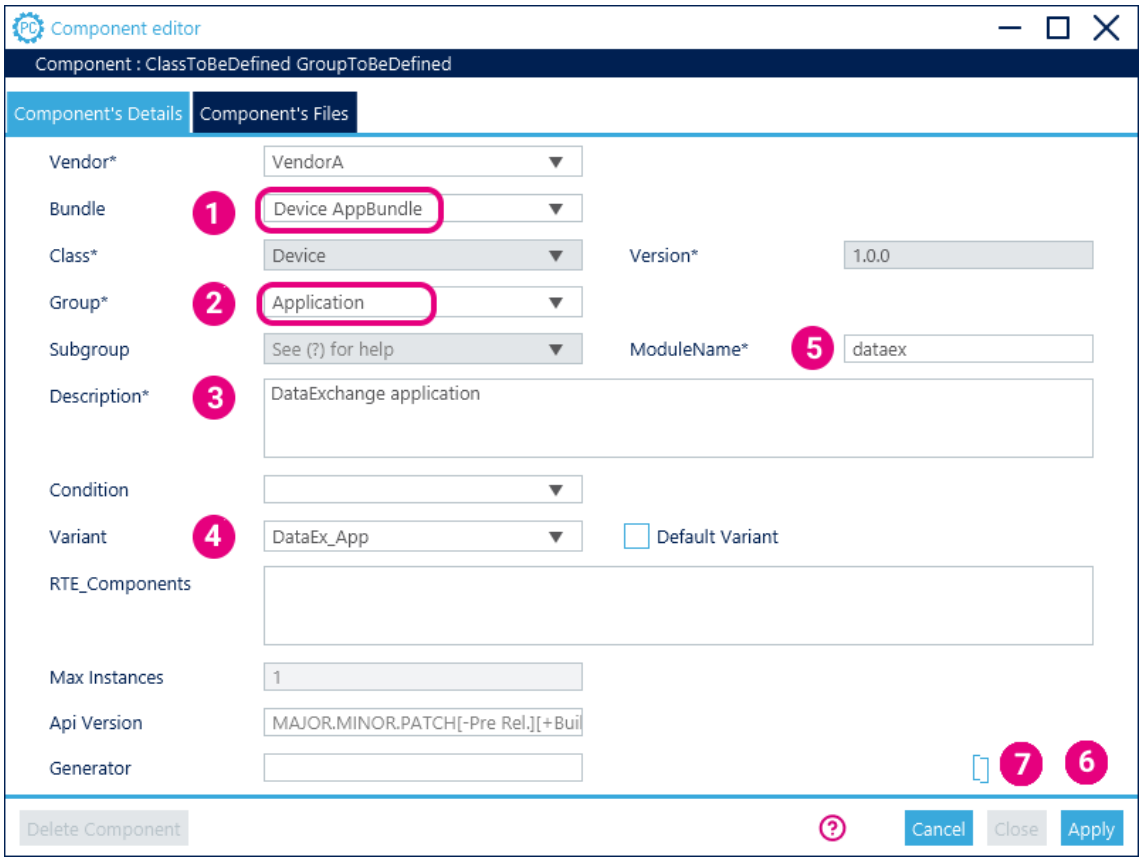
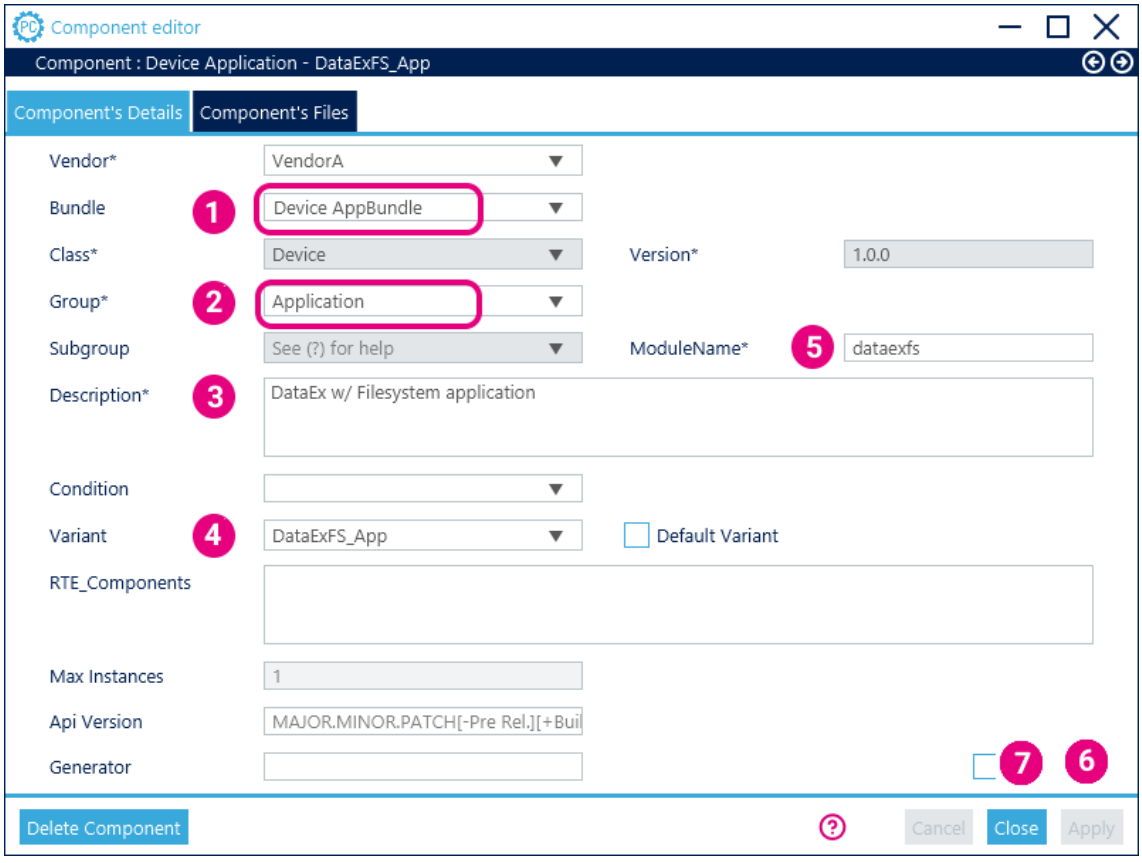


Figure 36. **Second application component**



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	*Not managed*	-
-	Dmpu	*Not managed*	-
-	Dtz	*Not managed*	-
-	Dsecure	*Not managed*	-
-	Ddsp	*Not managed*	-
-	Dendian	*Not managed*	-
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 overrules 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

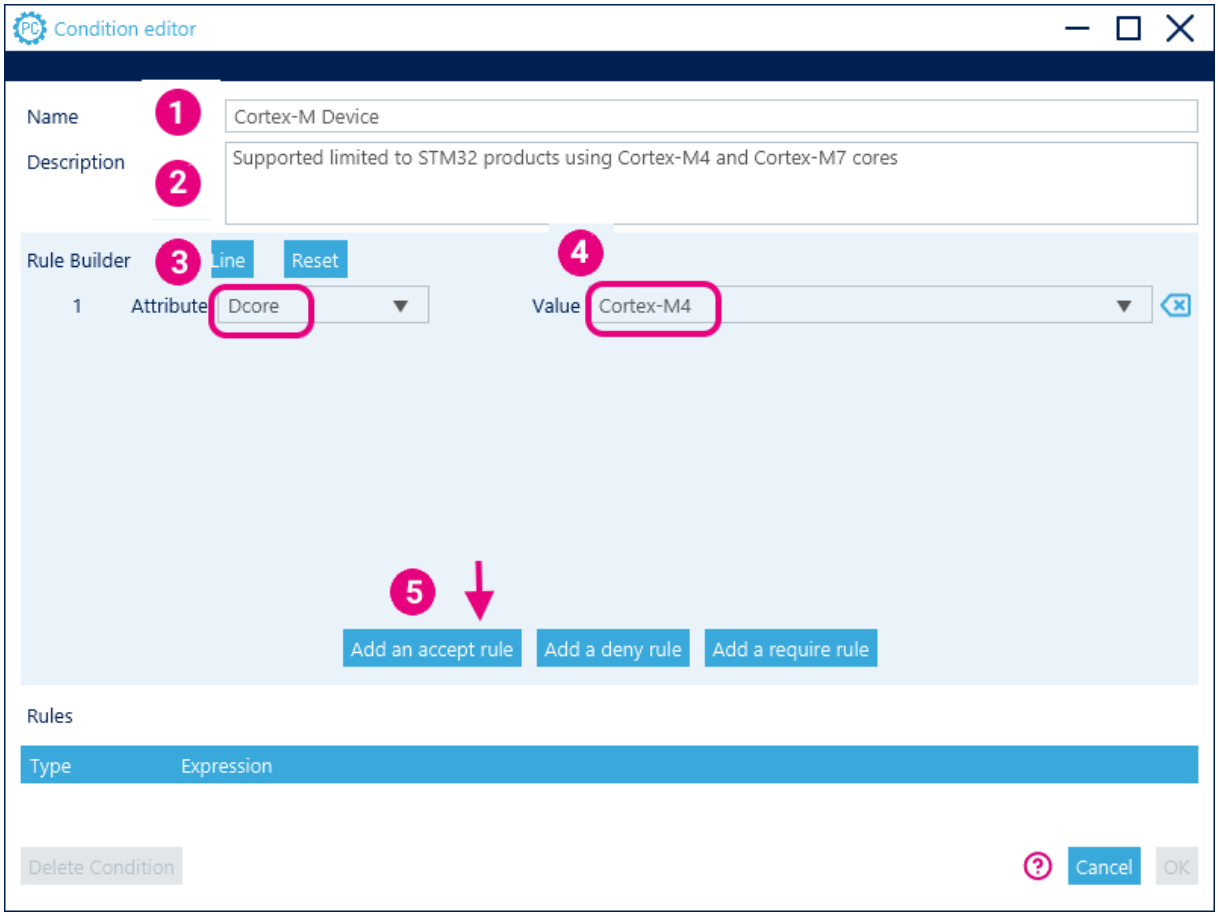
From the `Pack Details` section, expand the `Conditions` sub-section.

Click `Add Condition` to open the `Condition editor`.

Enter the condition `Name` and a `Description`.

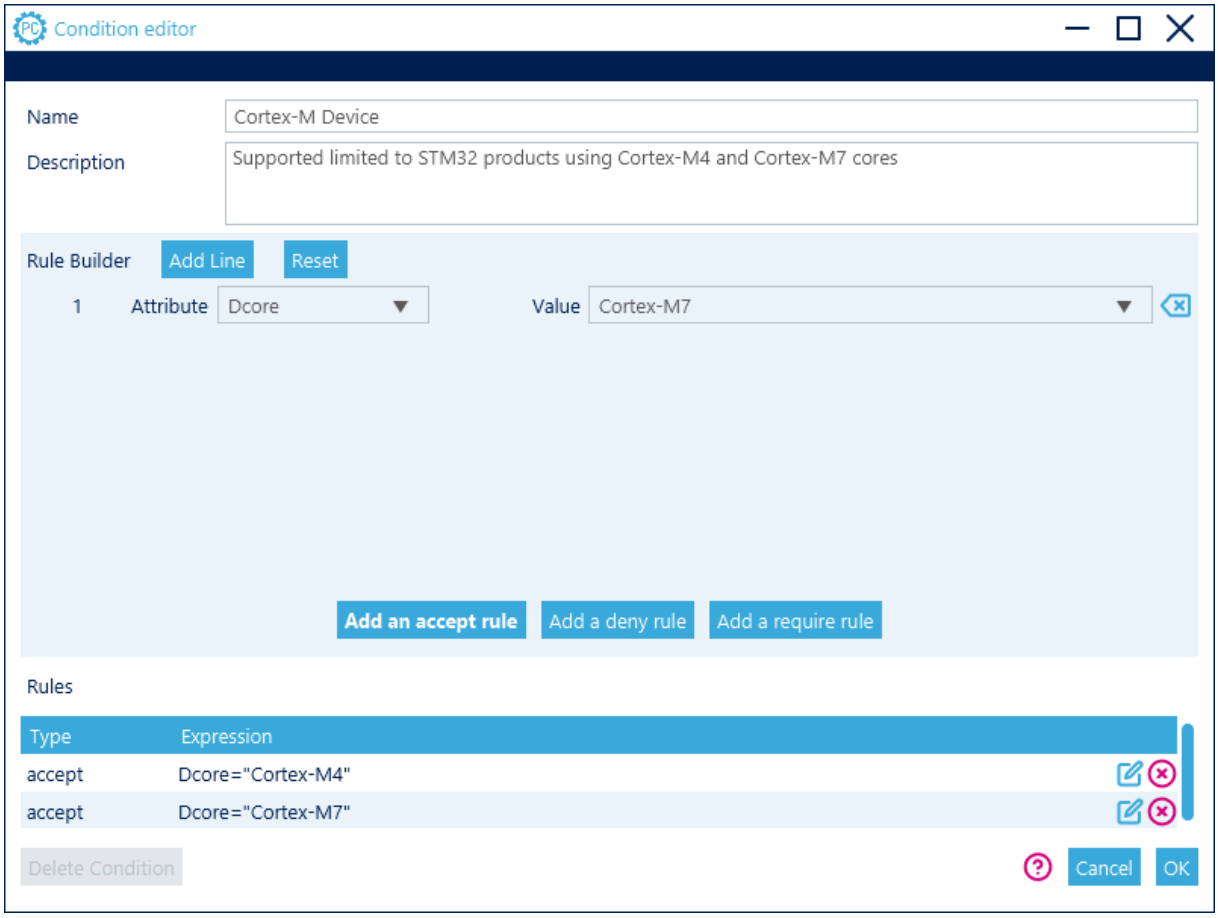
`Add an accept rule` on Cortex-M4.

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



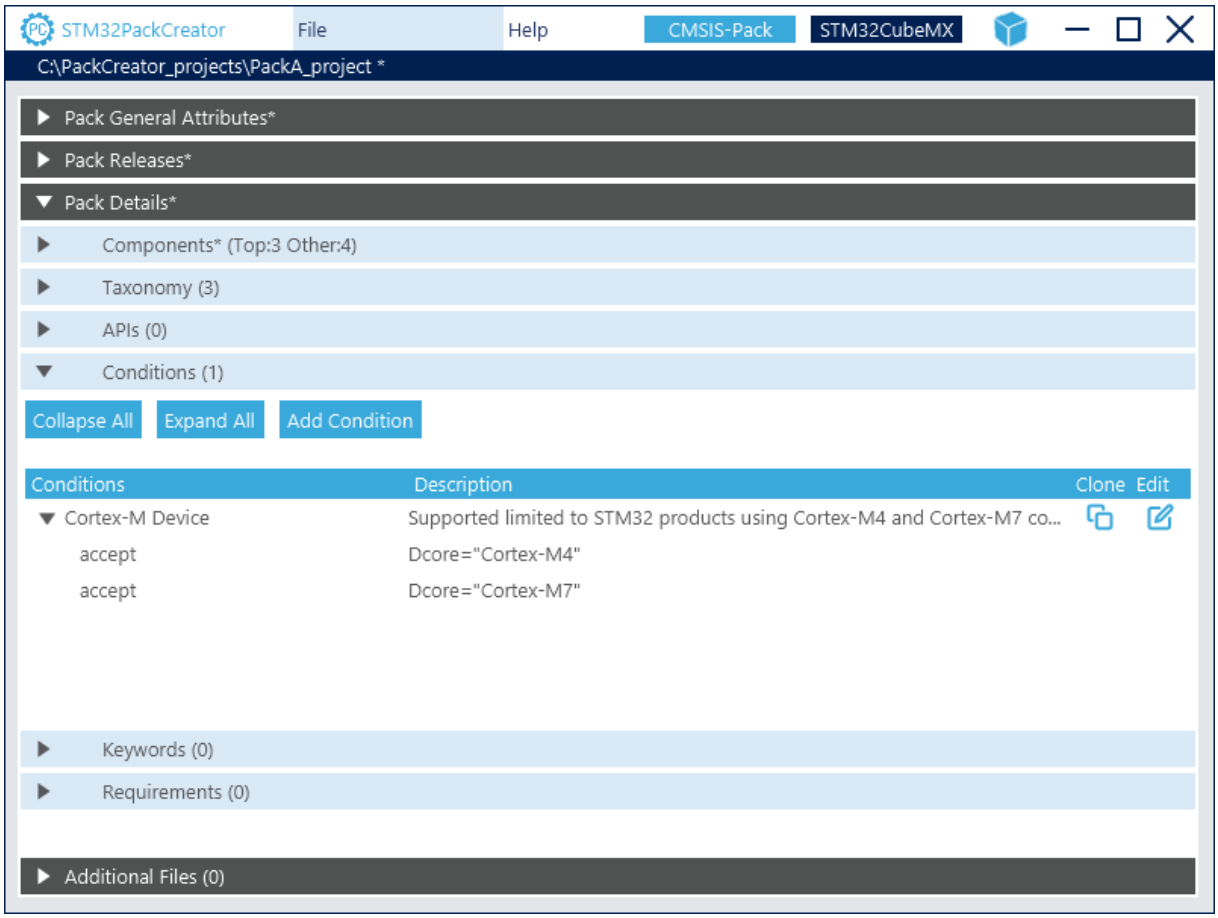
Add an accept rule on Cortex®-M7.

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



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

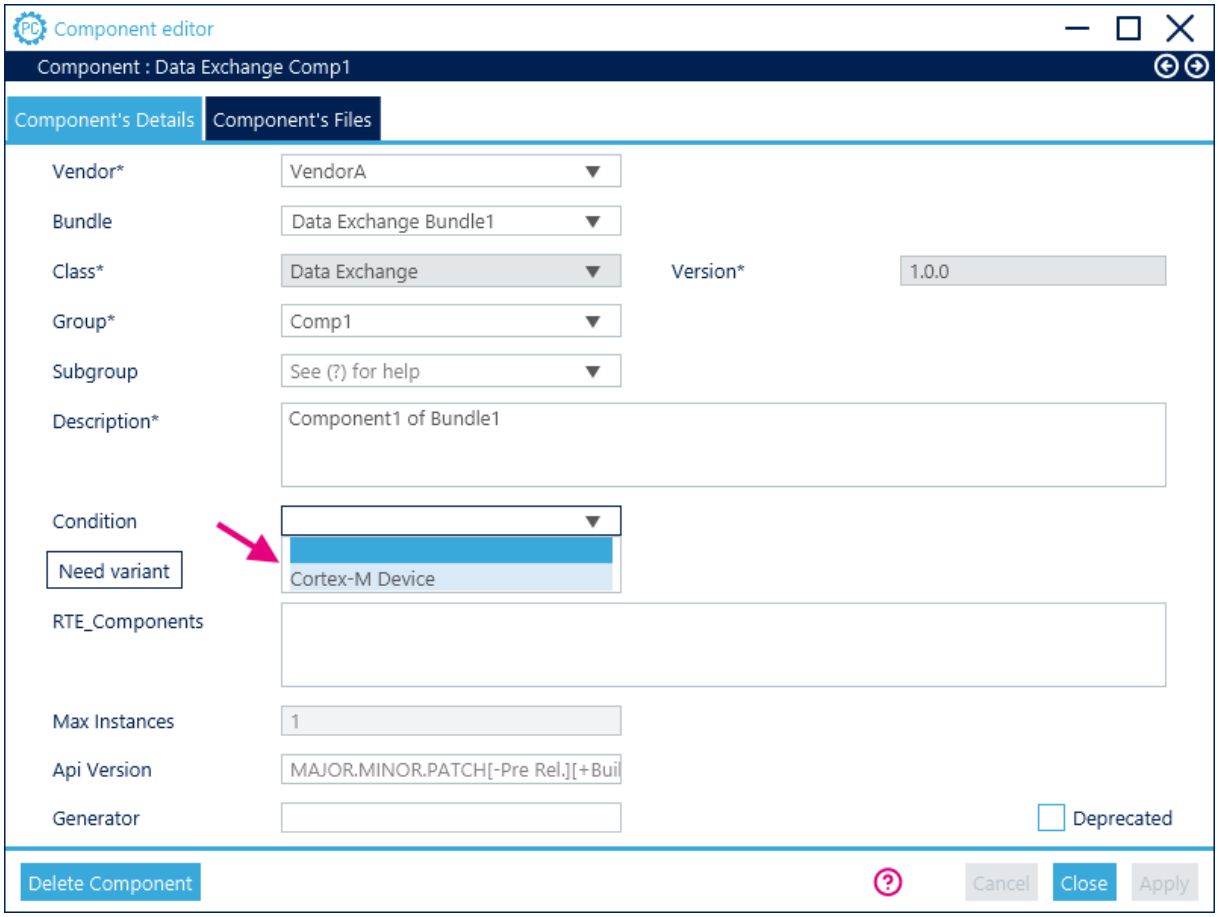
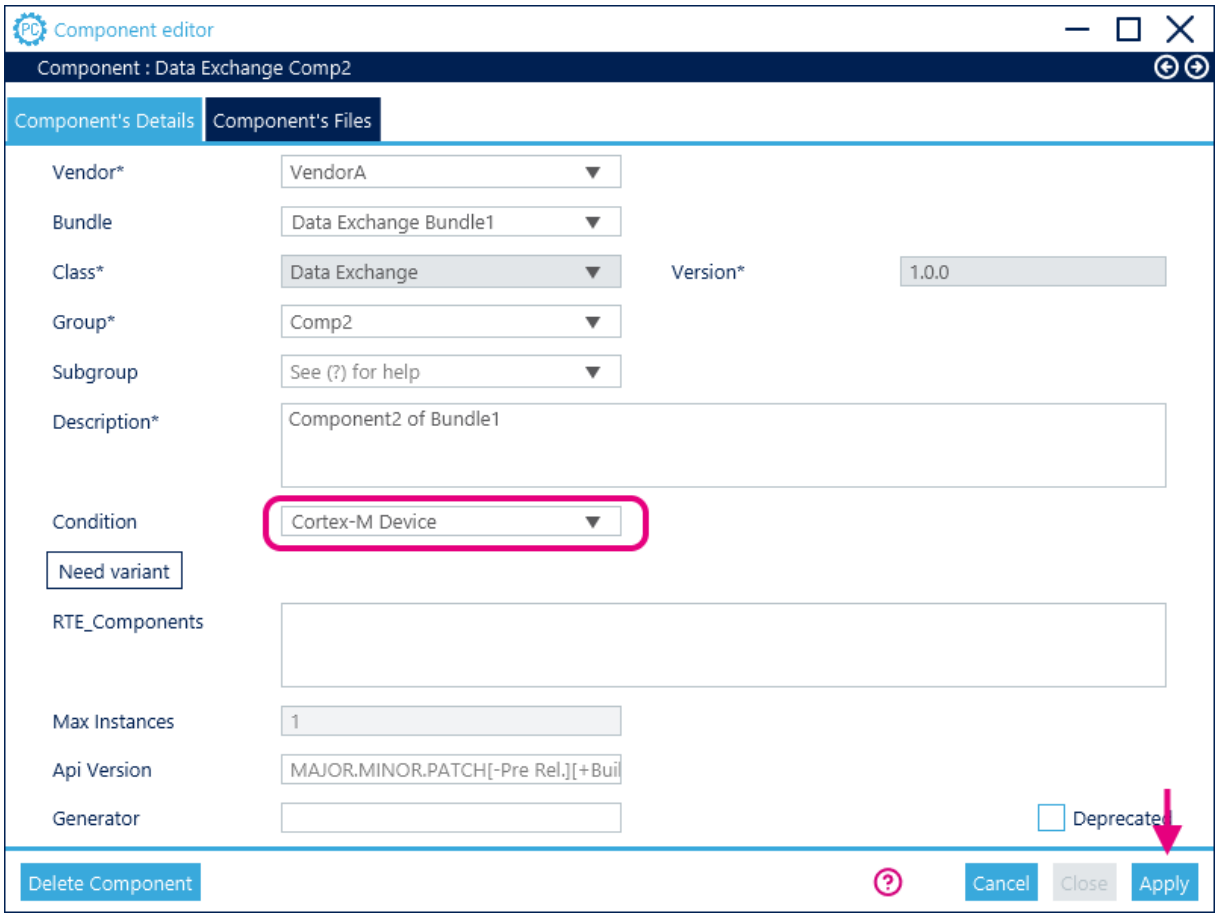


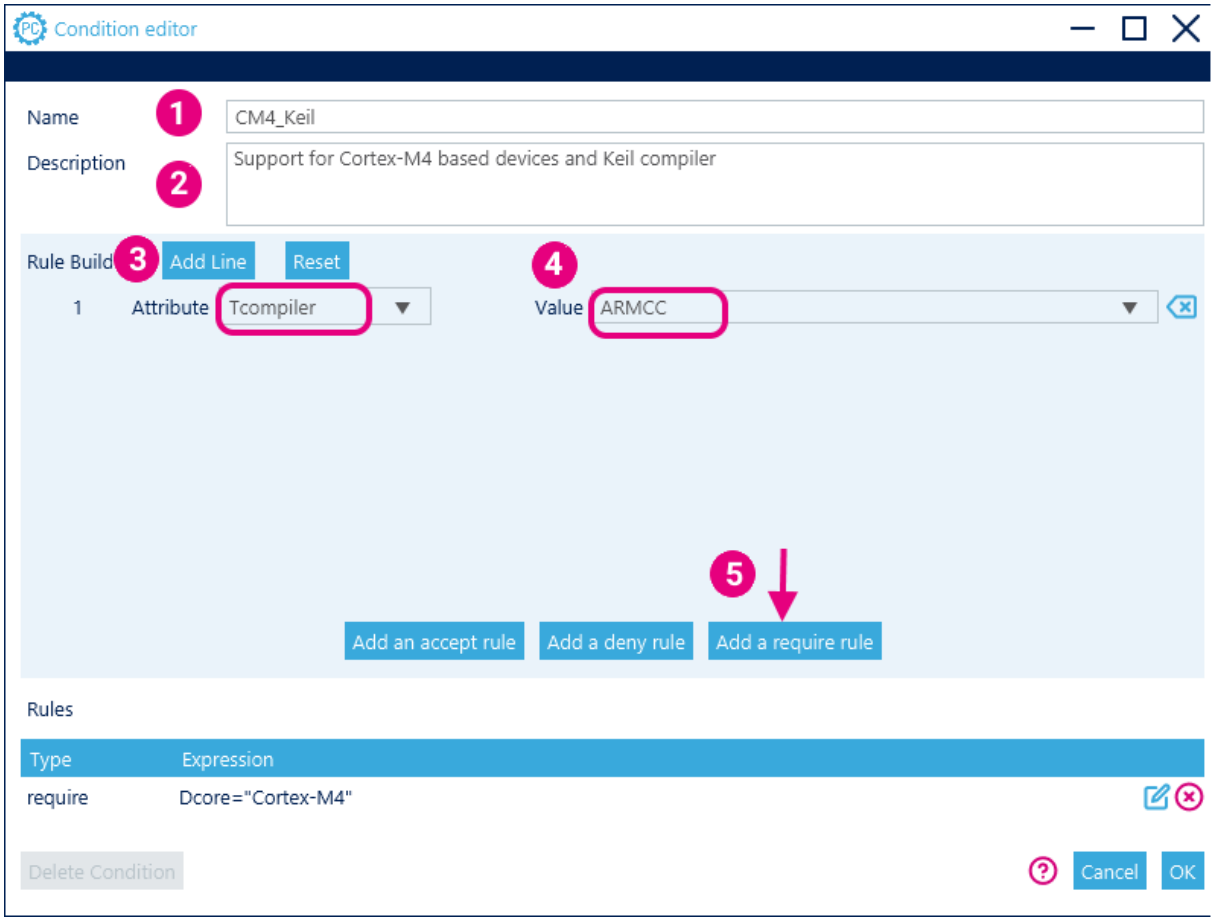
Figure 41. **Condition assigned to component 2**



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



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**

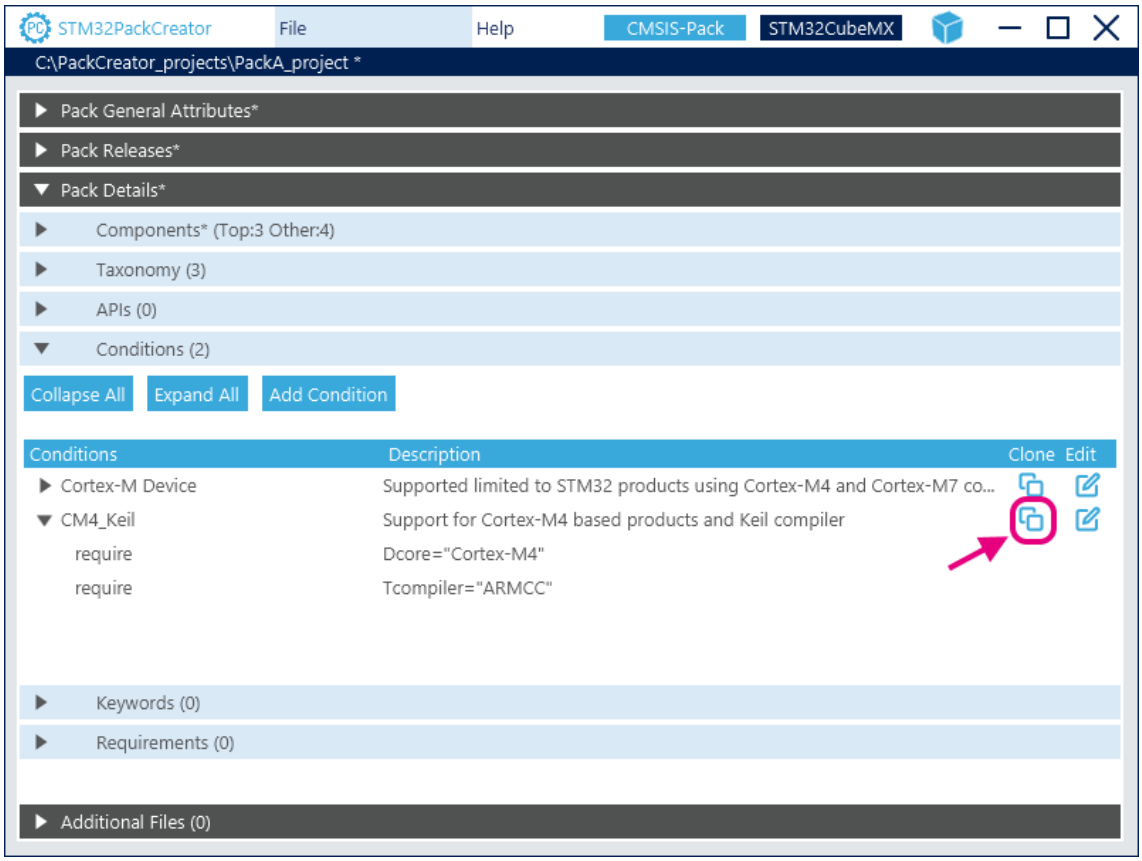
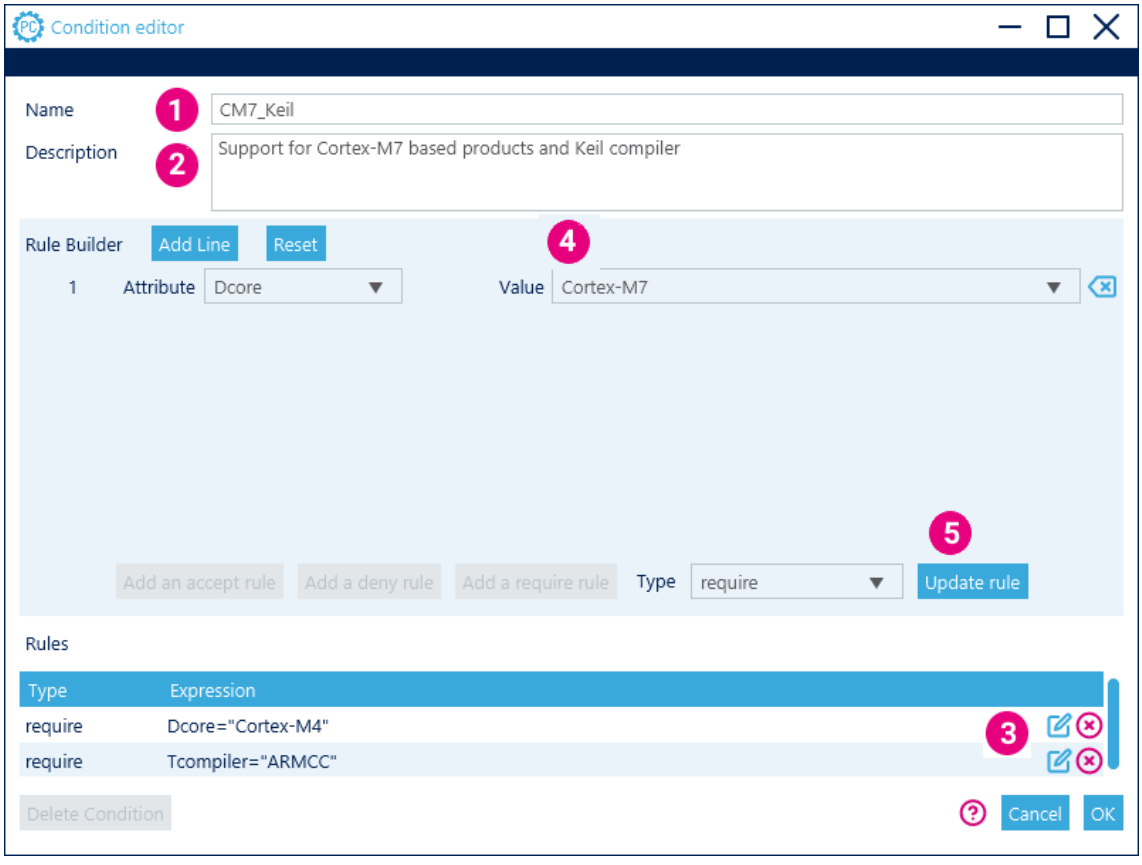
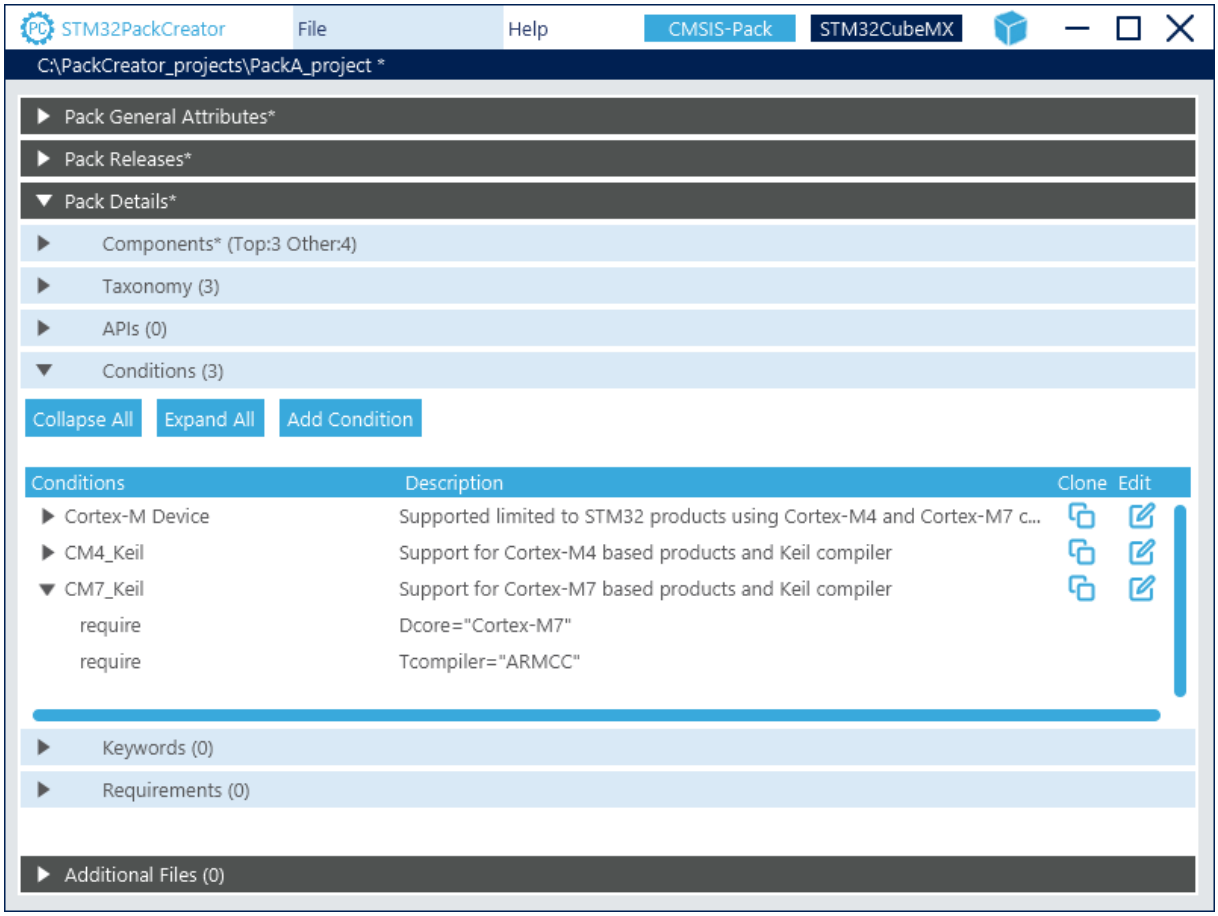


Figure 44. **CM7_Keil condition**

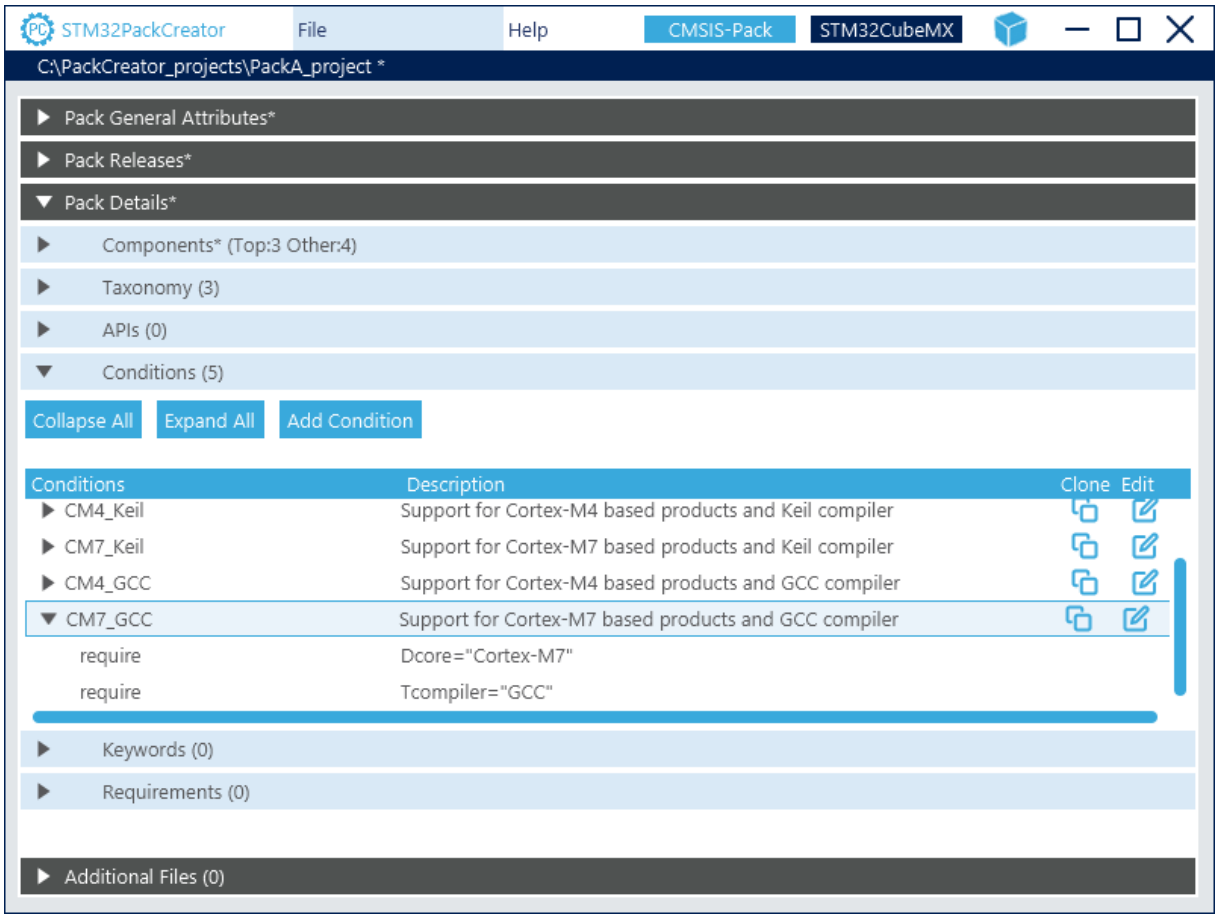


Click OK to close.



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, click edit on the TopComponent line to open the component editor.

Go to the Component's files tab.

Assign 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**

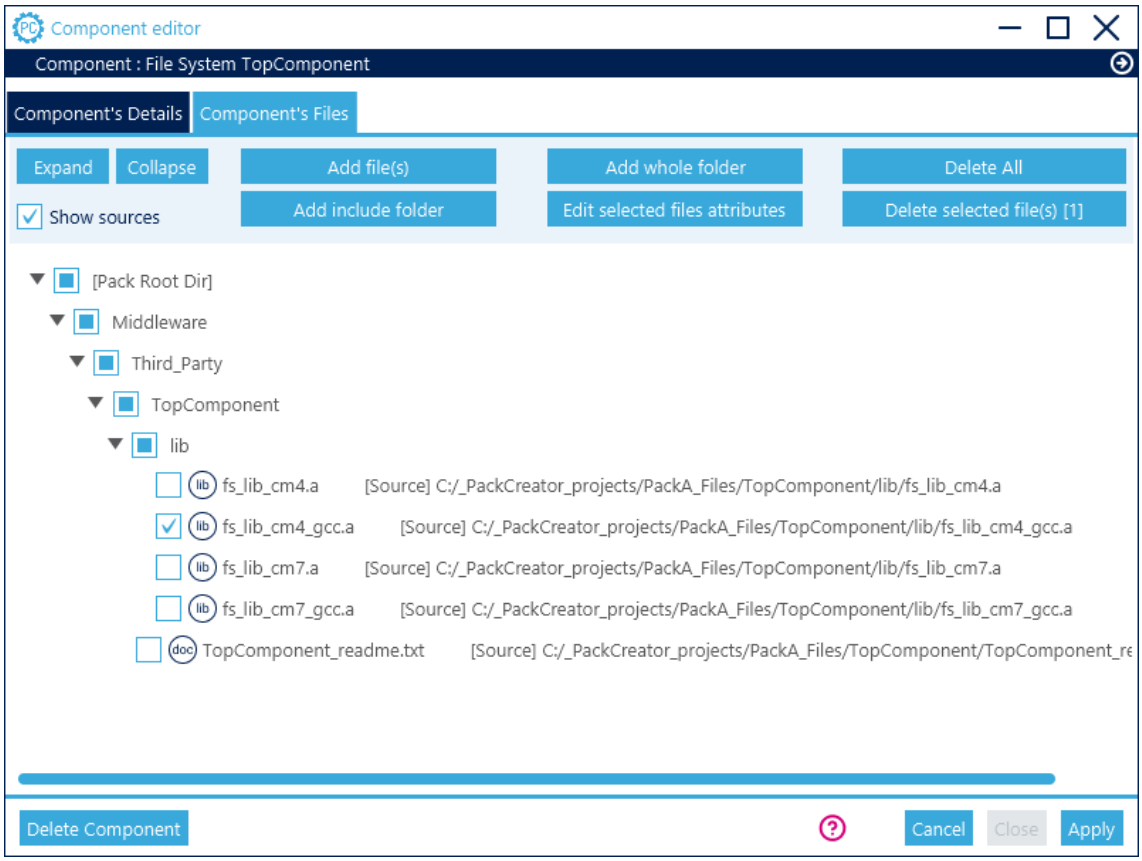


Figure 47. **Assigning CM4_GCC condition on file**

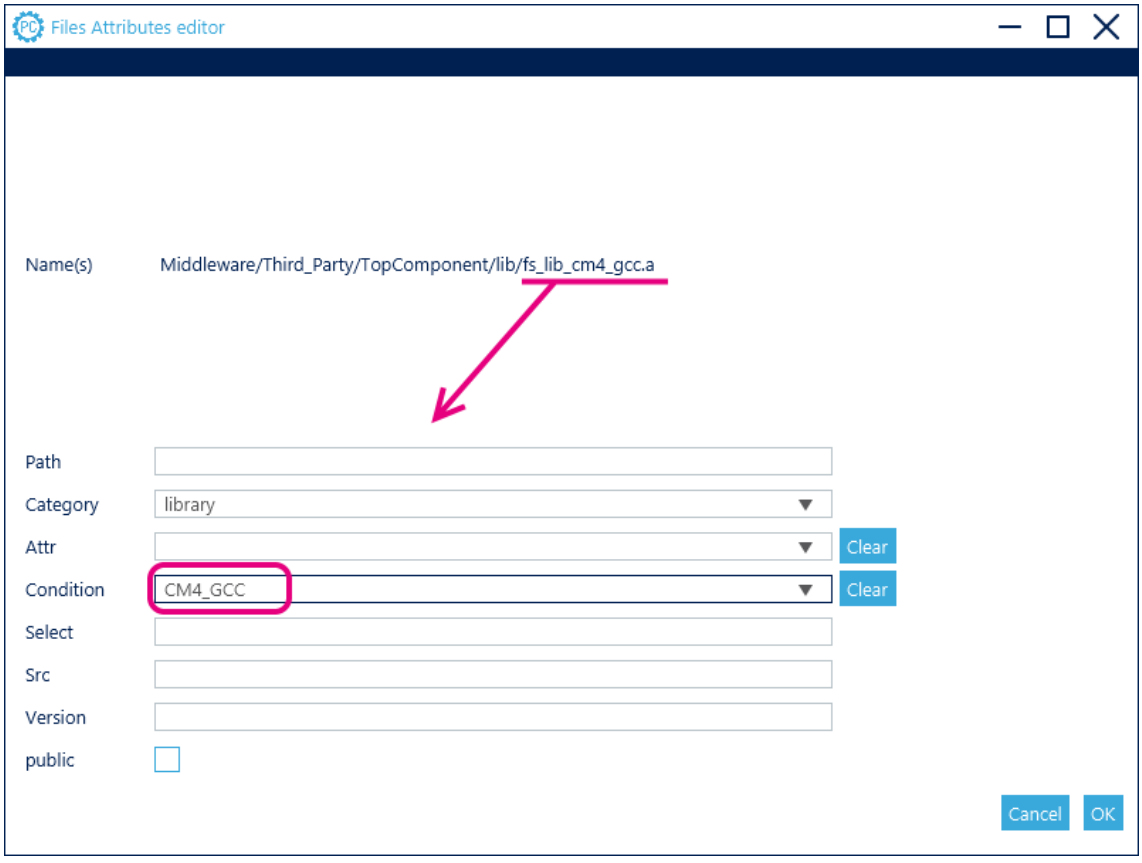
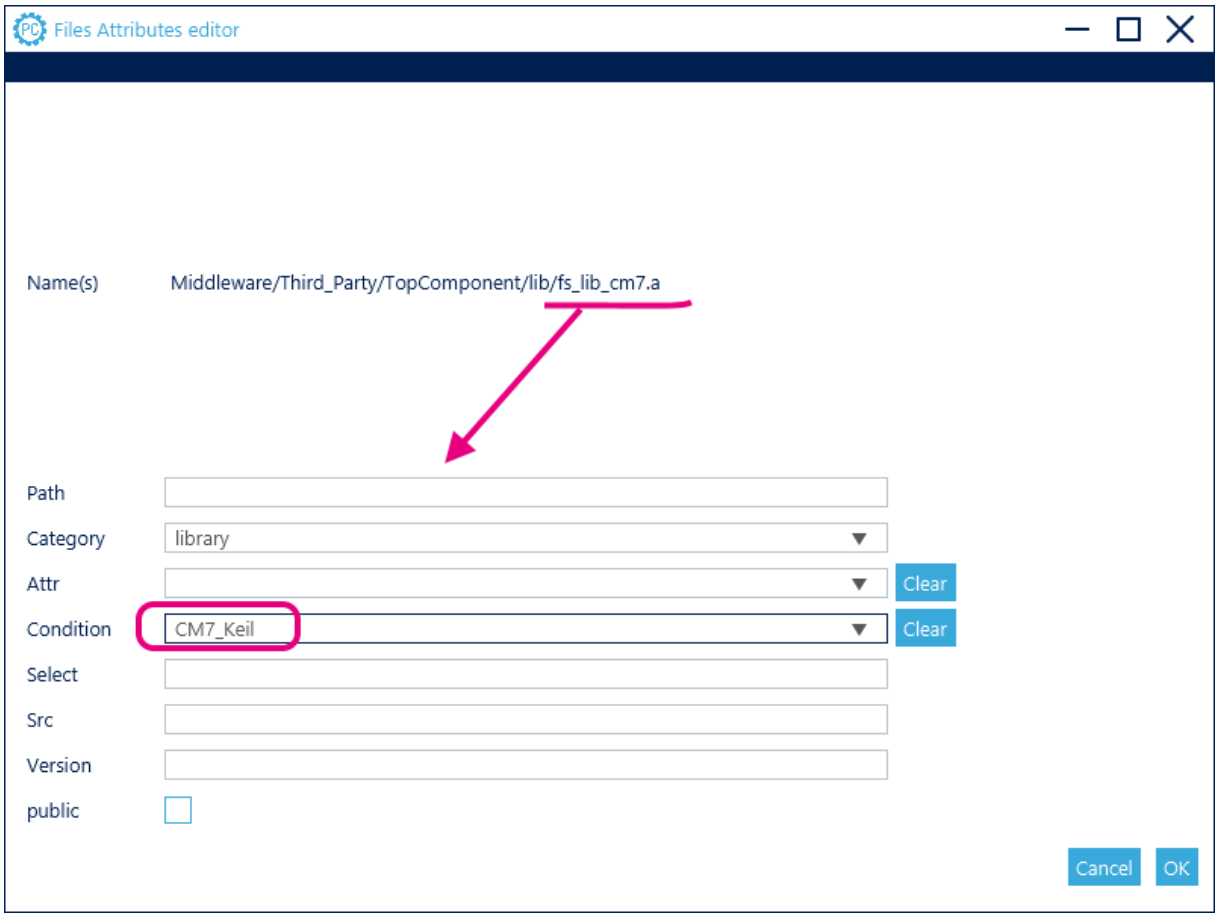


Figure 48. **Assigning CM7_Keil condition on file**



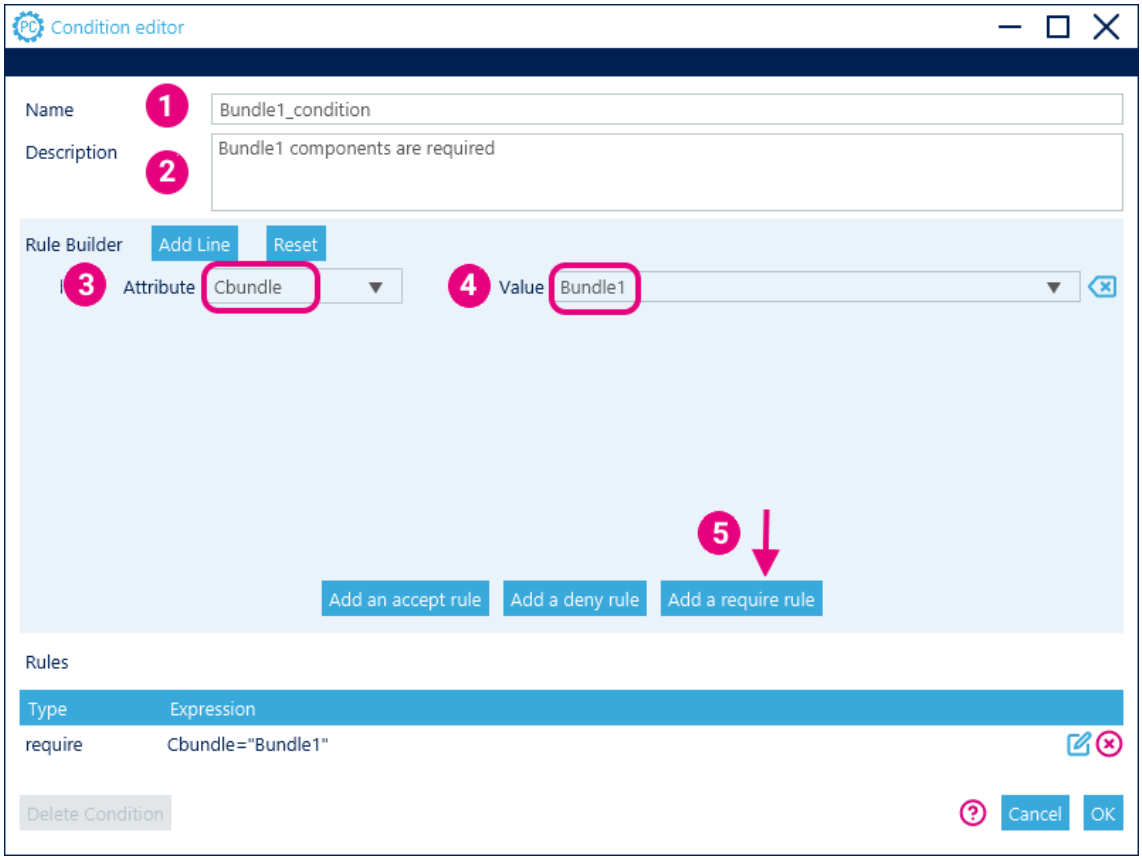
5.3.13 Create conditions on components

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.

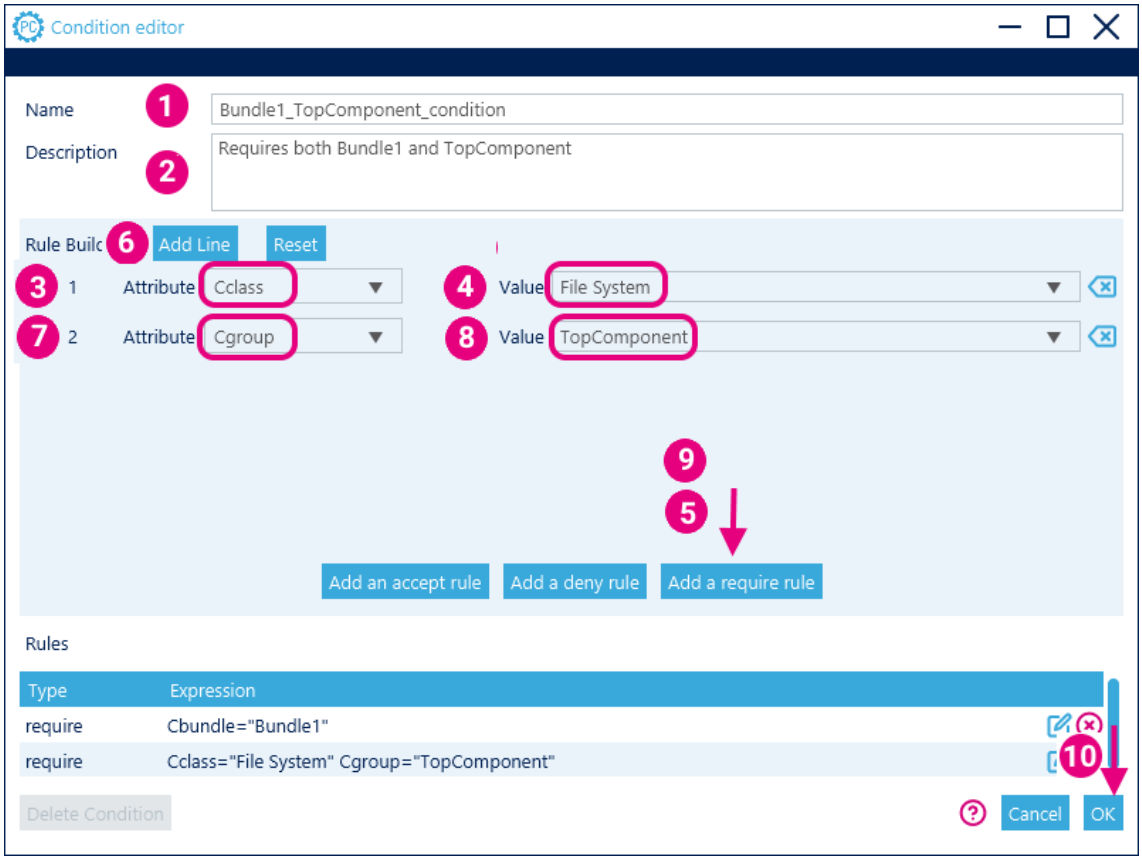
From the Conditions sub-section, Click `Add Condition` to open the condition editor. Proceed as shown on Figure 49.

Figure 49. Creating a condition for Bundle 1



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

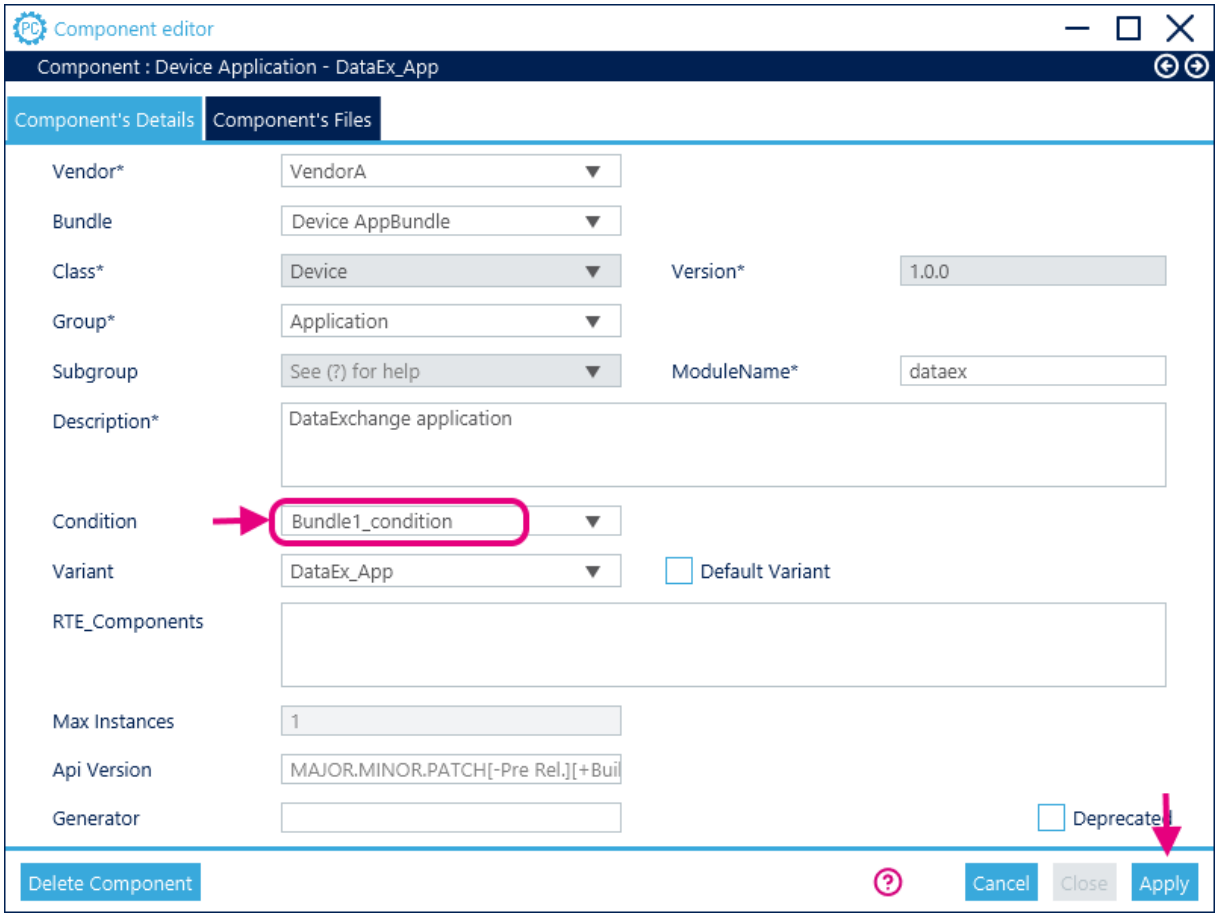
Figure 50. Creating condition for Bundle 1 and TopComponent



5.3.14 Assign conditions on application components

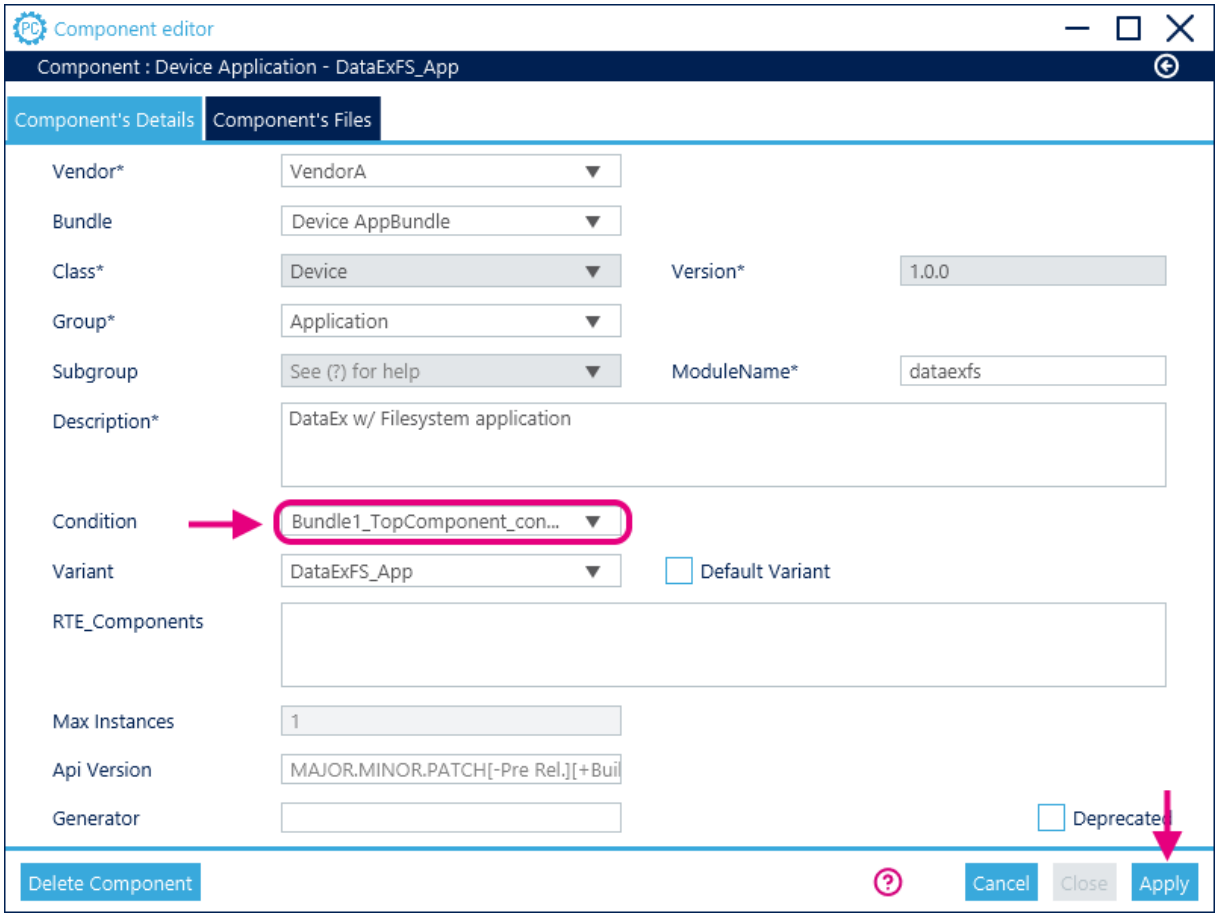
Go back to the Components section and click the edit icon to edit the `DataEx_App` component. Proceed as shown in Figure 51.

Figure 51. Assigning conditions to DataEx_App component



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 DataExFS_App component



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

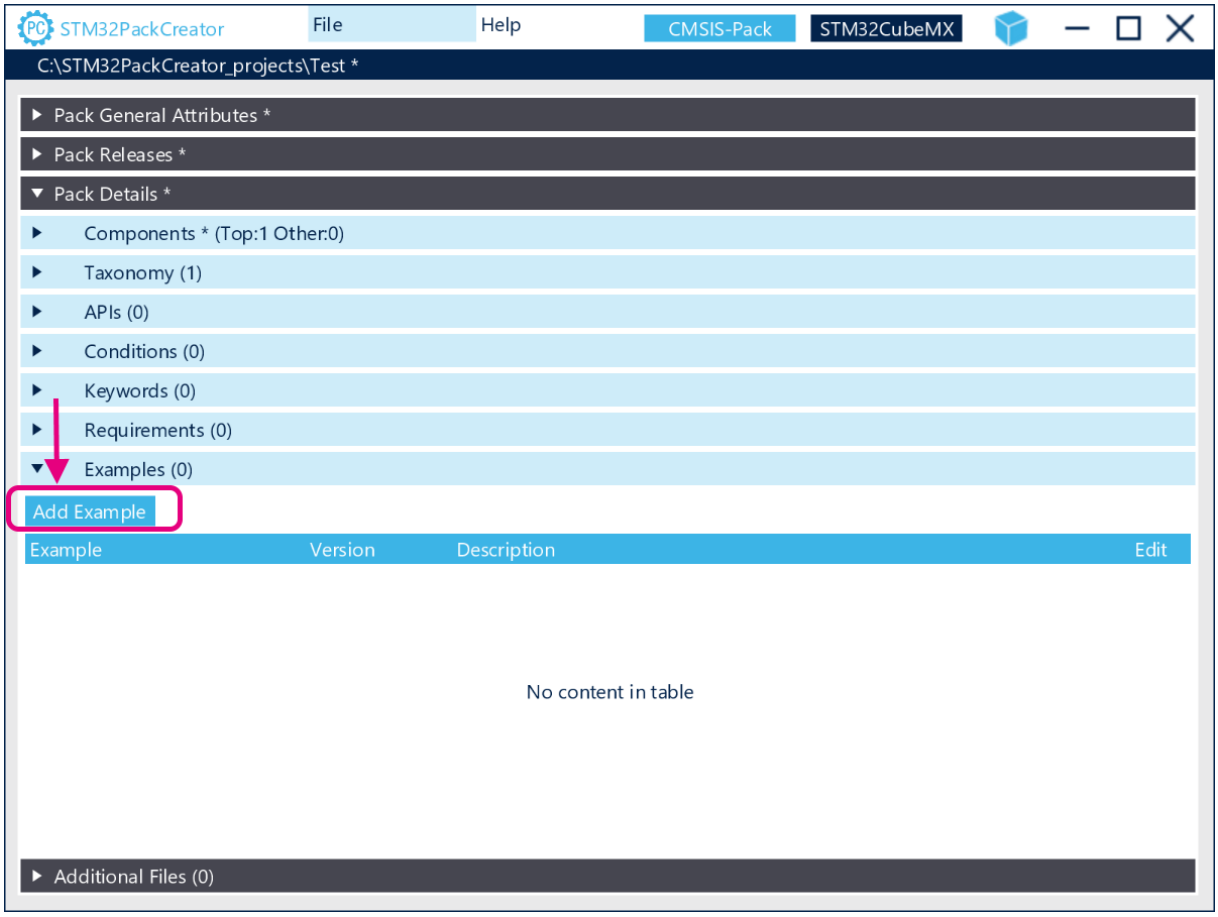


Figure 54. **Specifying example details – part 1**

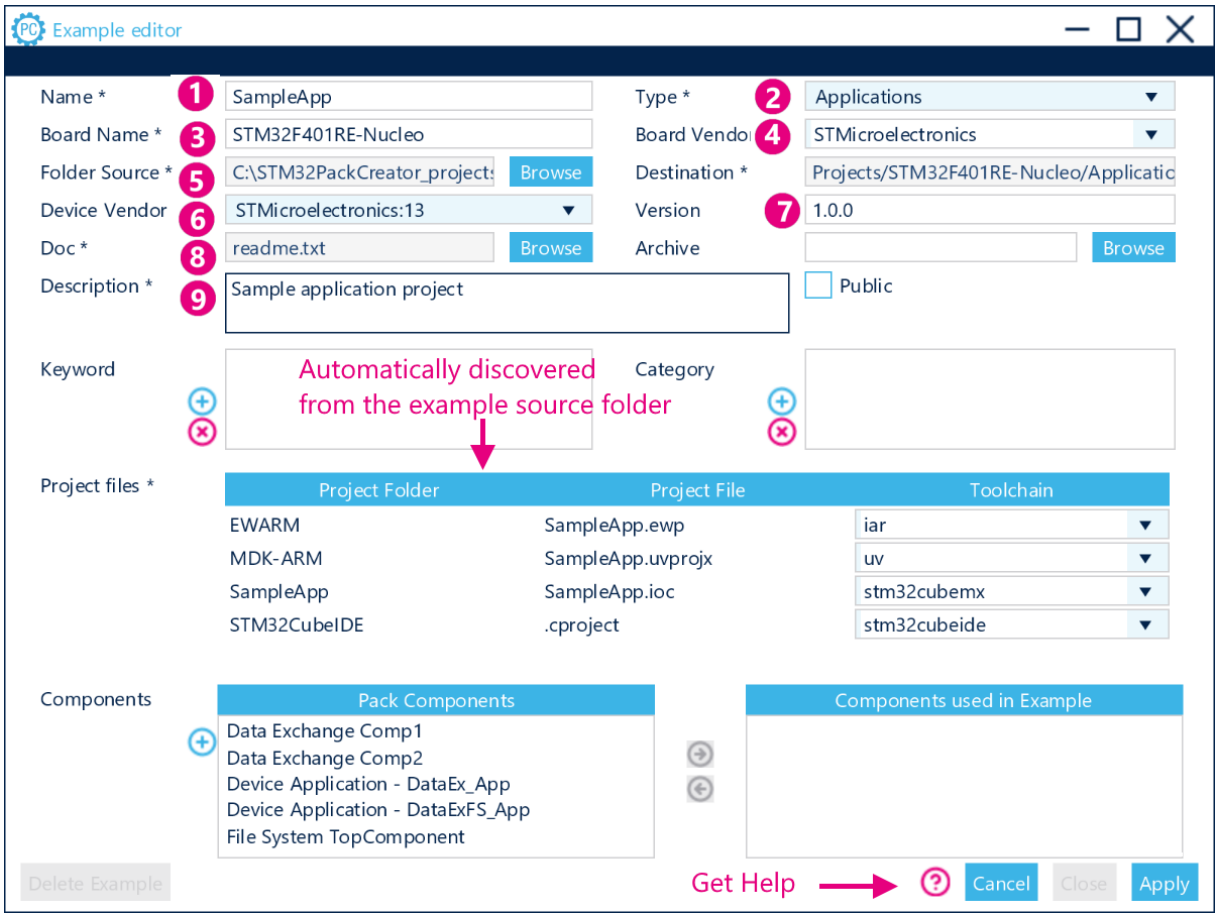


Figure 55. **Specifying example details – part 2**

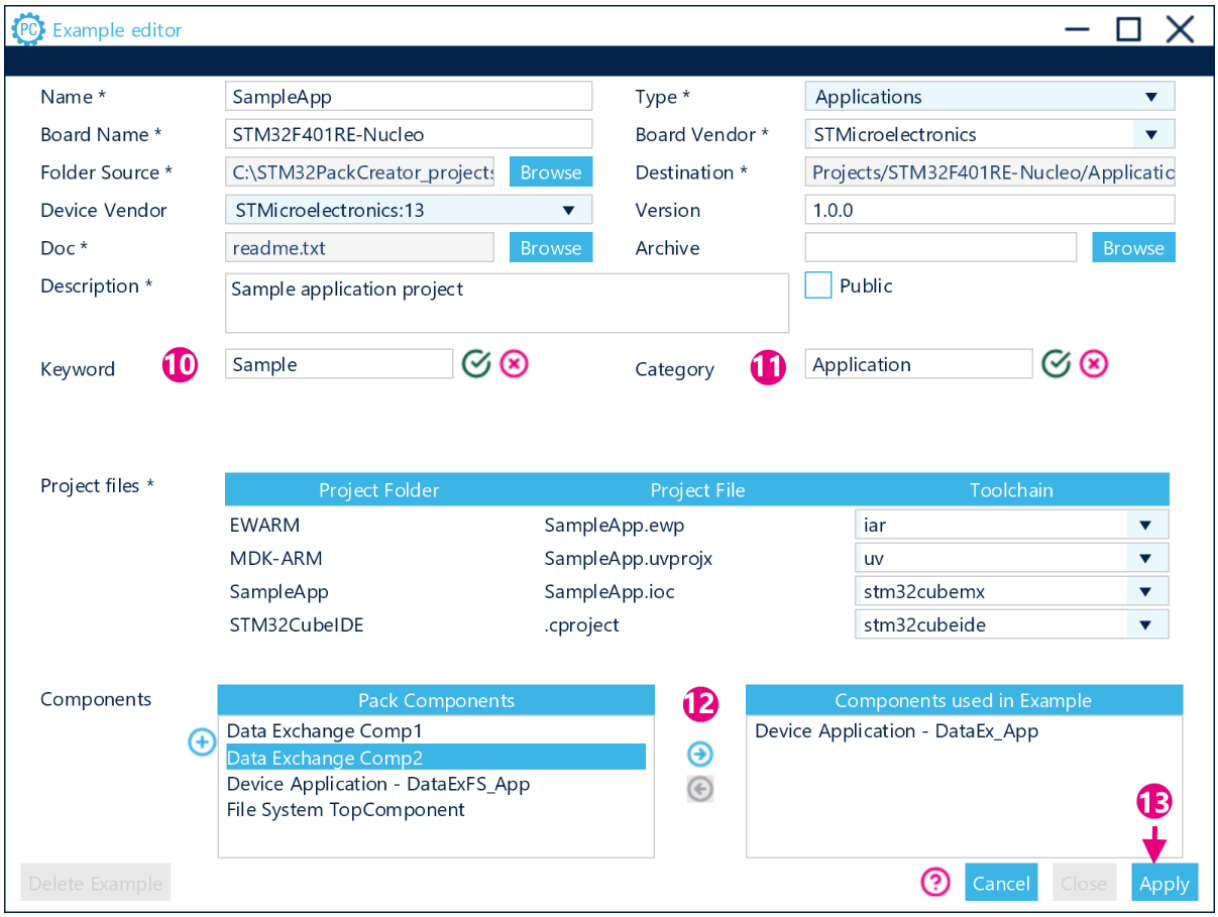
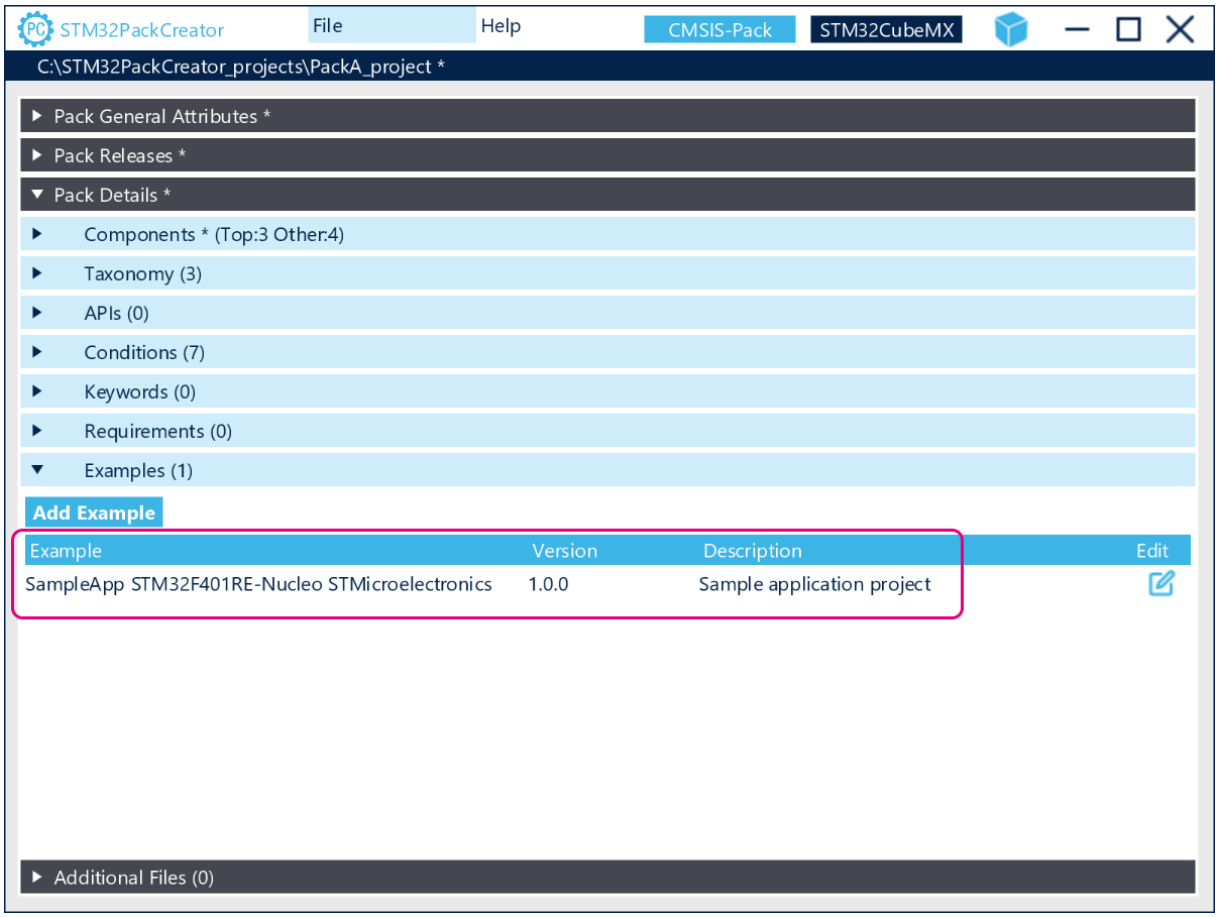


Figure 56. **New example added to the pack**



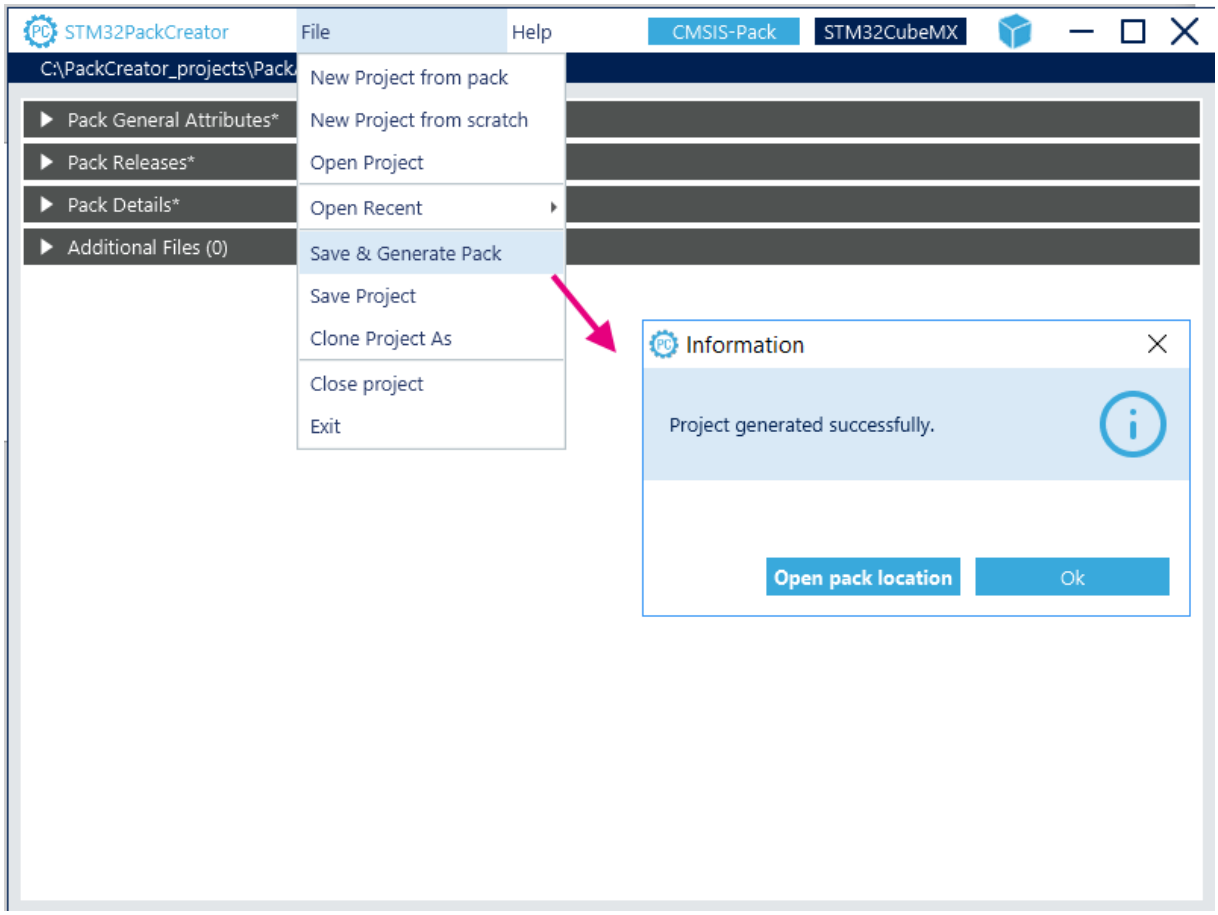
5.4 Save the software pack and use it in STM32CubeMX

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.

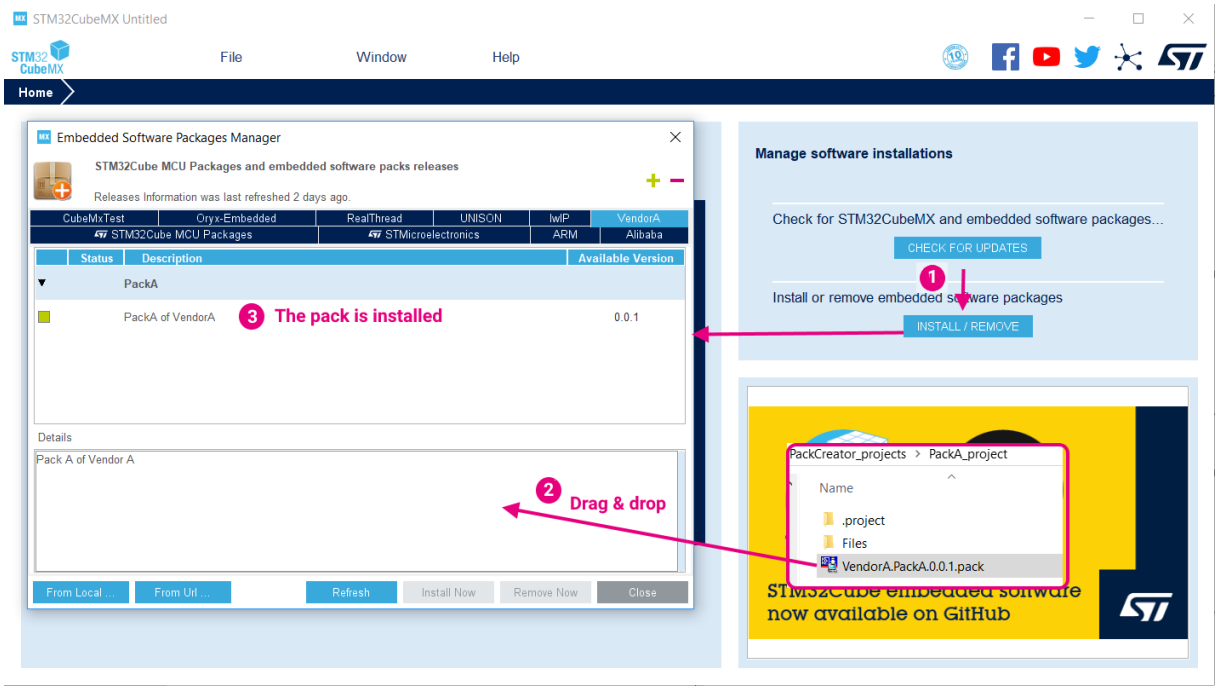
Figure 57. Generating a CMSIS-Pack pack with STM32PackCreator



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

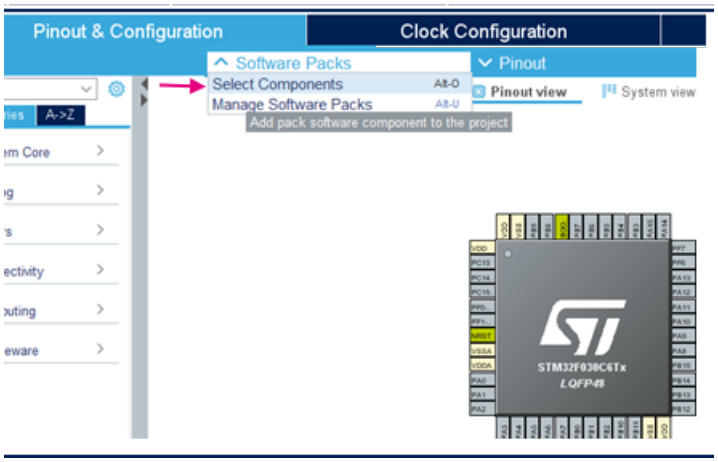


5.4.2 Create a project and check if the pack is available

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

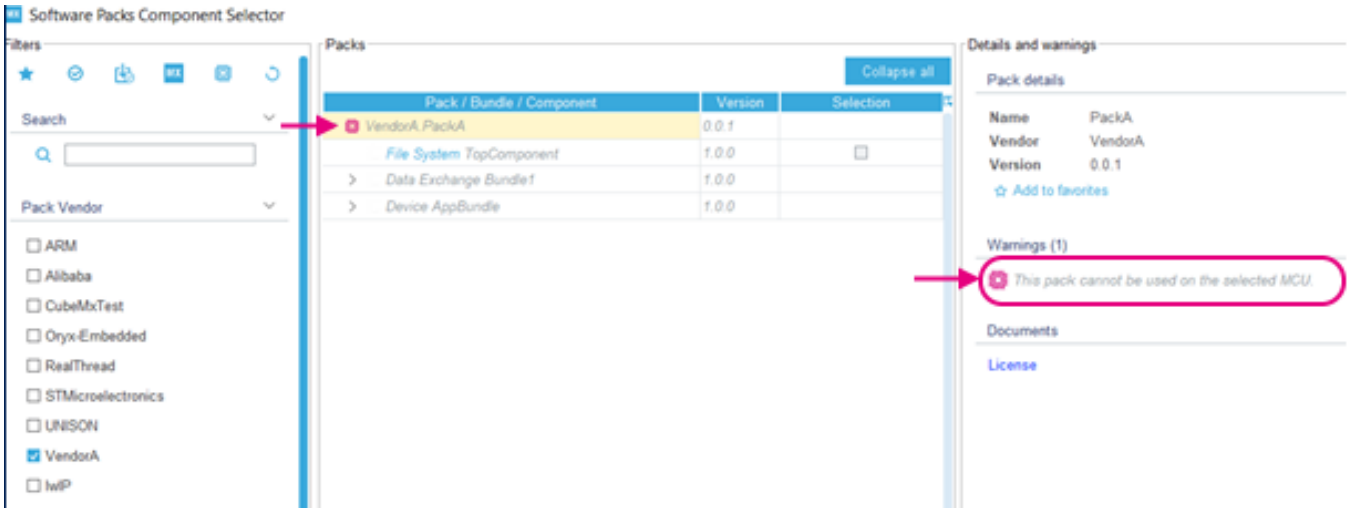
From the `Pinout & Configuration` view, select `Software Packs> Select Components`. Refer to Figure 59.

Figure 59. Selecting pack components



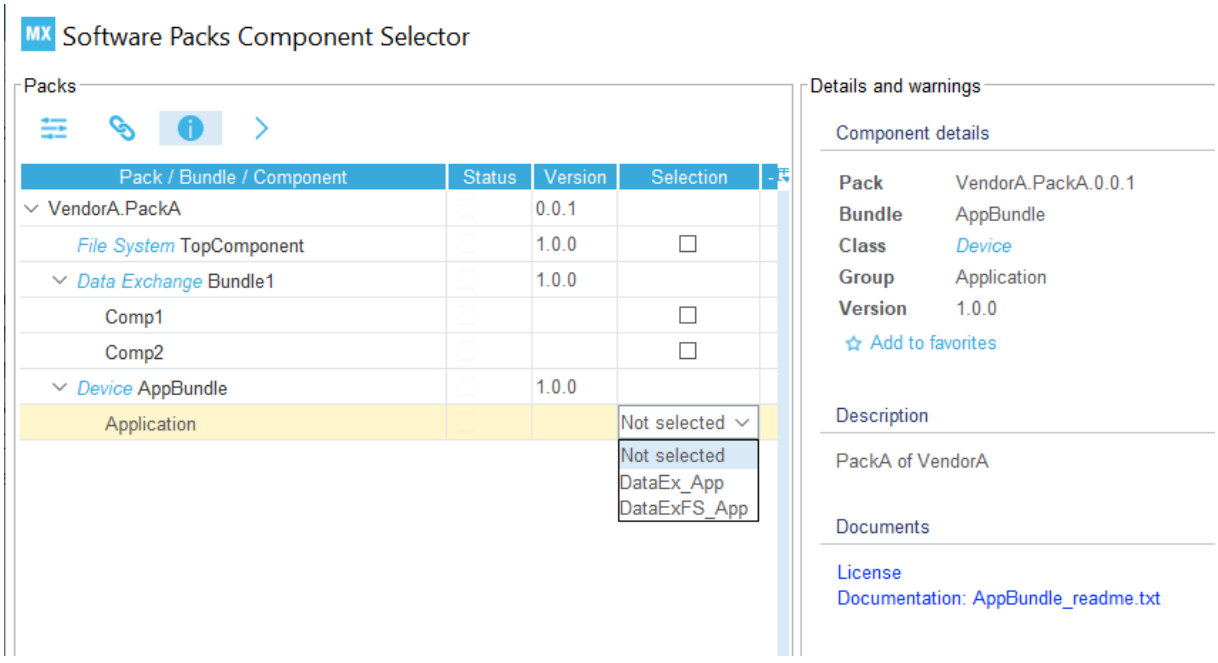
In the component selector, check that the pack components cannot be selected since the Dcore condition is not met: the pack requires an Arm®Cortex®-M4 or Arm®Cortex®-M7 while STM32F0 MCU embeds an Arm®Cortex®-M0 core. Refer to Figure 60.

Figure 60. Pack unavailability when 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

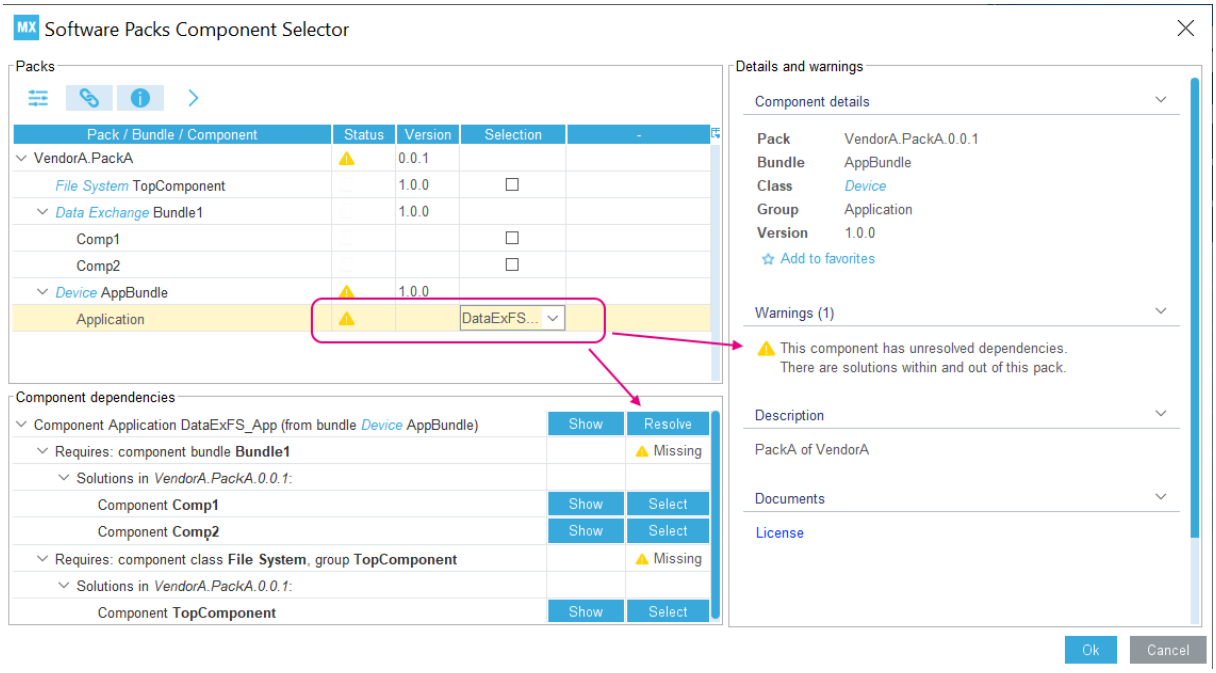


5.4.3 Select pack components, check and solve dependencies

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

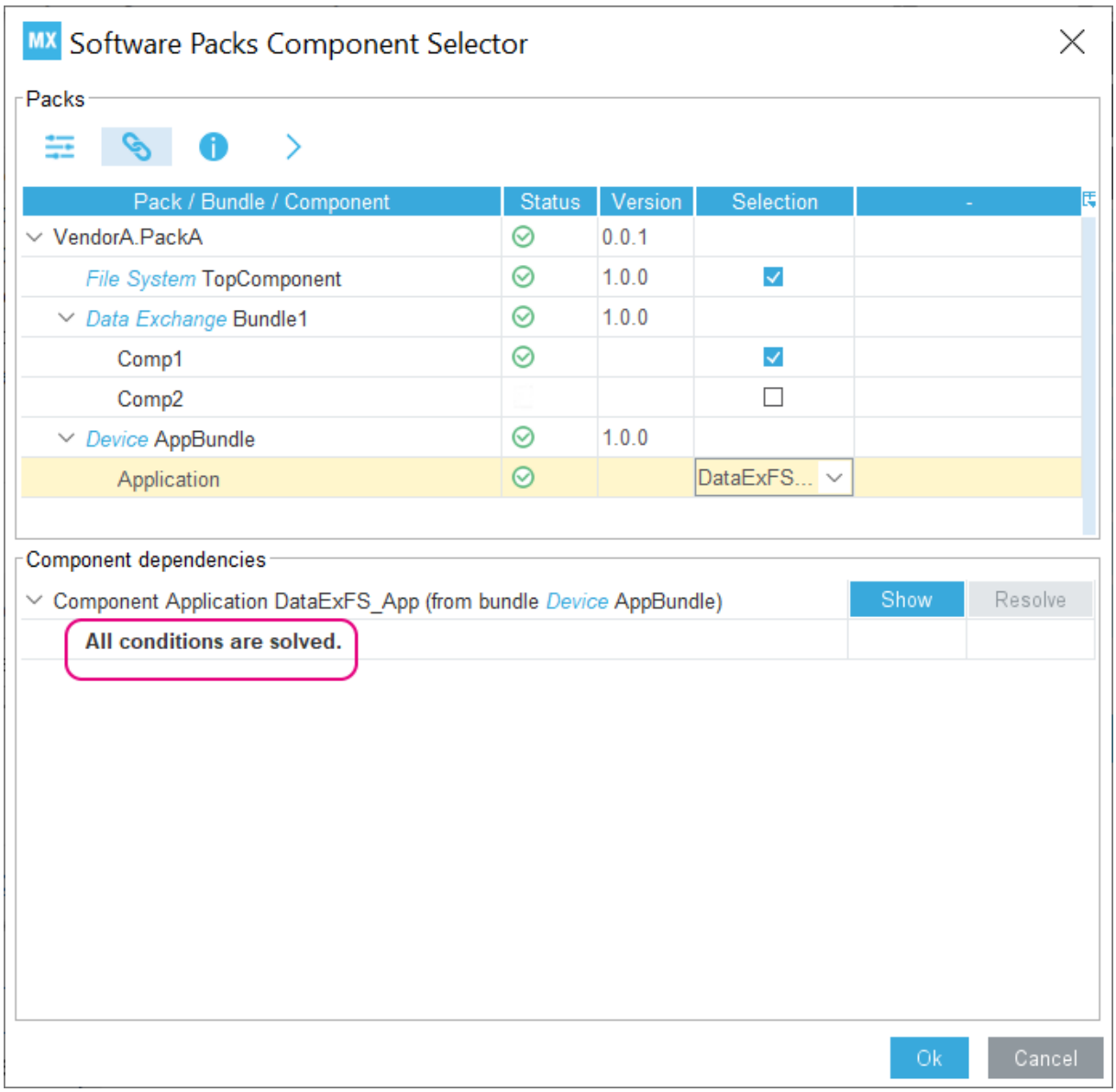
1. Enable the DataExFS_App.
2. Click the DataExFS_App component line to highlight dependencies: Top component and Bundle1 are highlighted as required components. Refer to Figure 62.

Figure 62. Highlighting dependencies on components



3. Click the Resolve button to automatically resolve dependencies: this is possible when only one choice is available. Click select 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

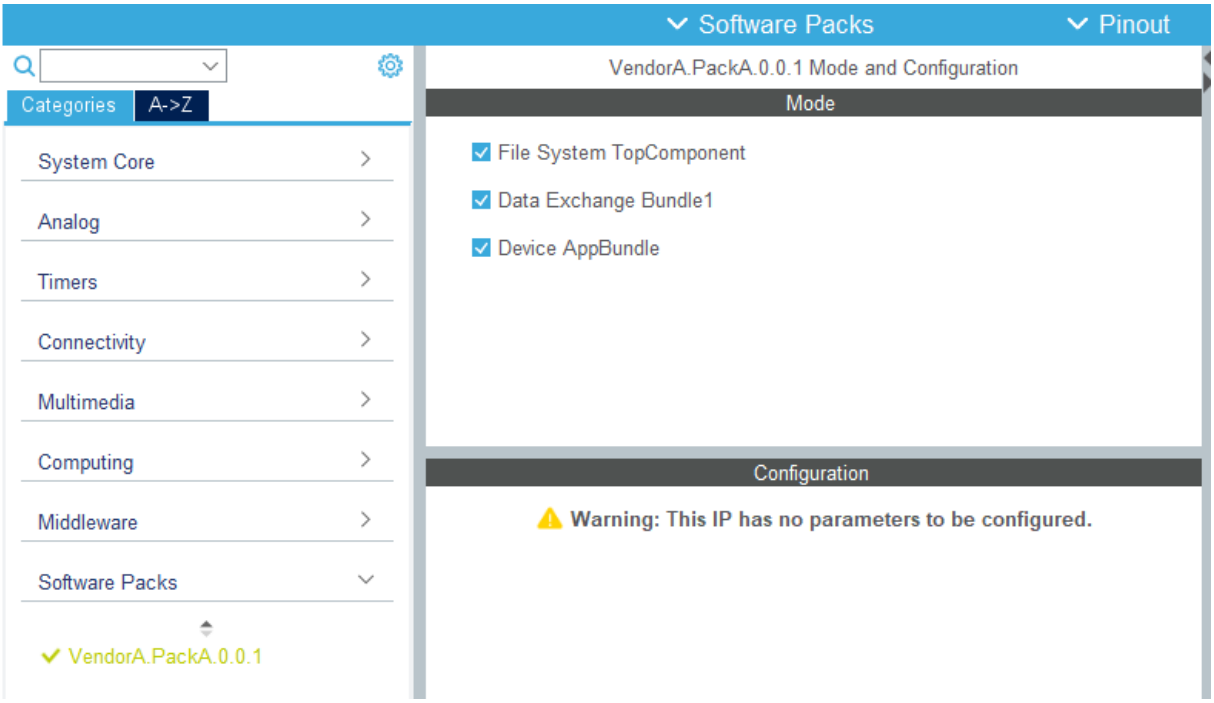


5.4.4 Check the pack in the STM32CubeMX pinout configuration view

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.

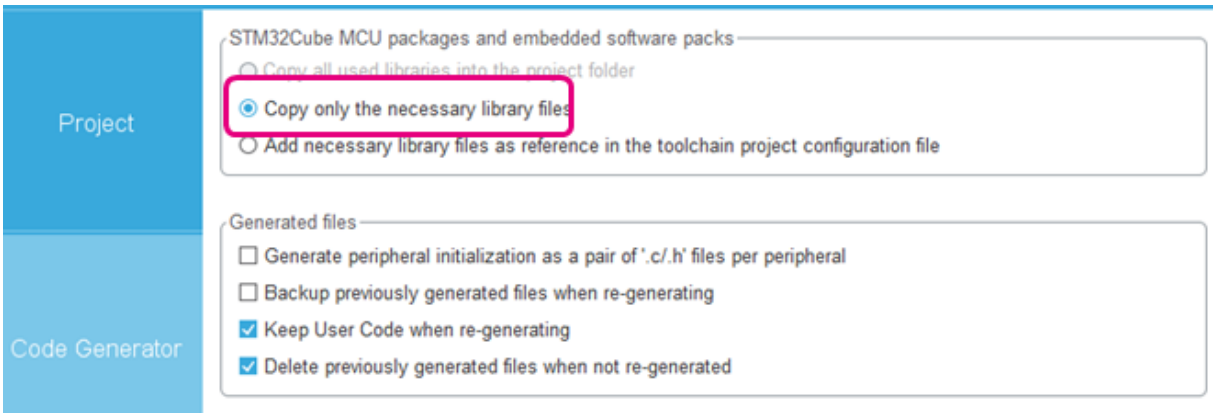
Figure 64. Unconfigurable software pack in STM32CubeMX user interface



5.4.5 Generate the project

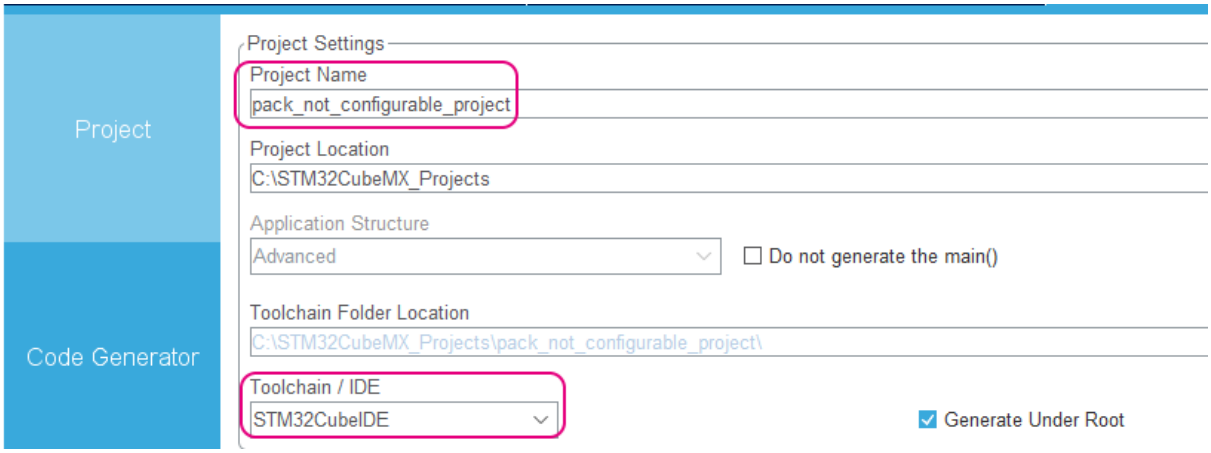
1. Go to the `Project Manager`'s view.
2. On the `Code generator` tab, select the option to copy only the necessary library files.

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



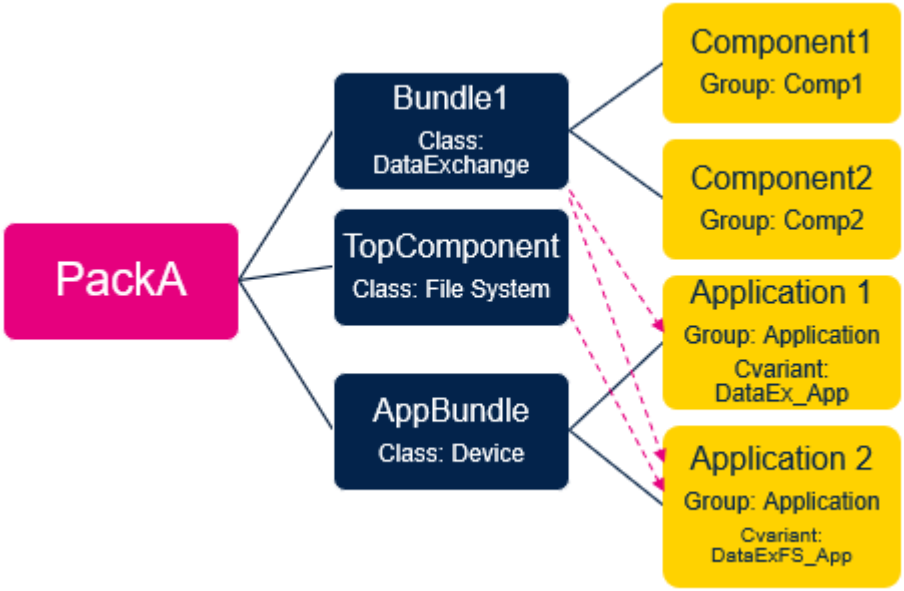
4. Click GENERATE CODE to generate the project. If requested, the user must accept to download the STM32Cube MCU Package relevant to the project.
5. Check that the pack files are copied accordingly under the folder `Middlewares\Third_Party`. Specifically, check that among the four library files of TopComponent, 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

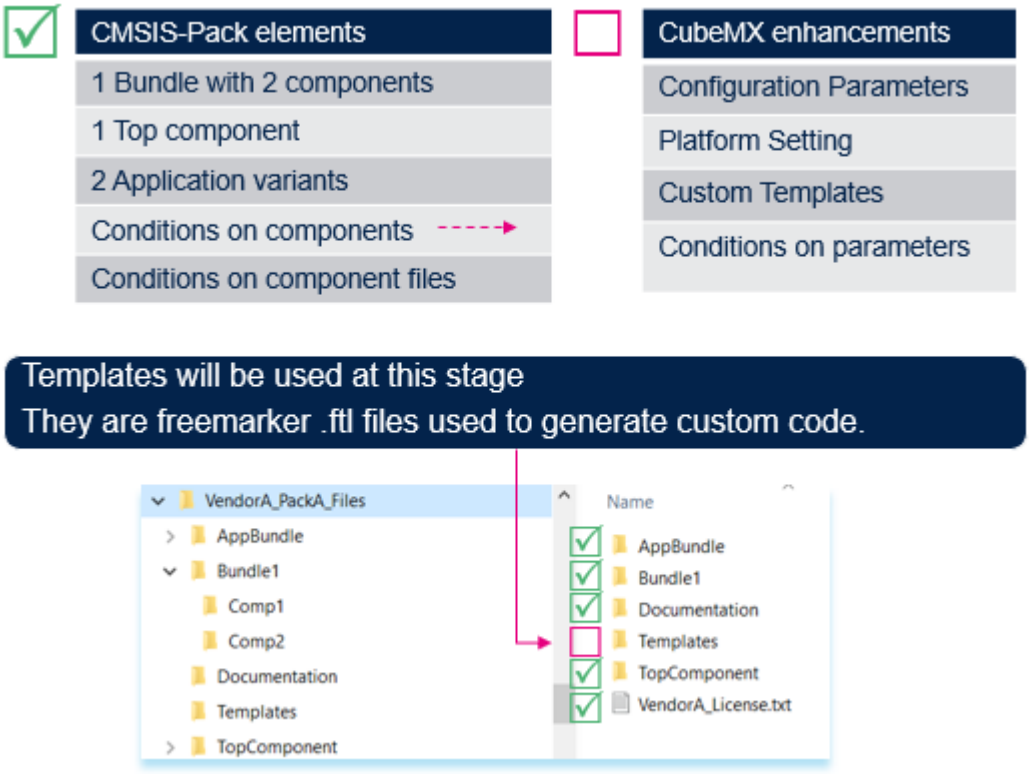
At this stage, the pack is available and can already be used in STM32CubeMX.

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

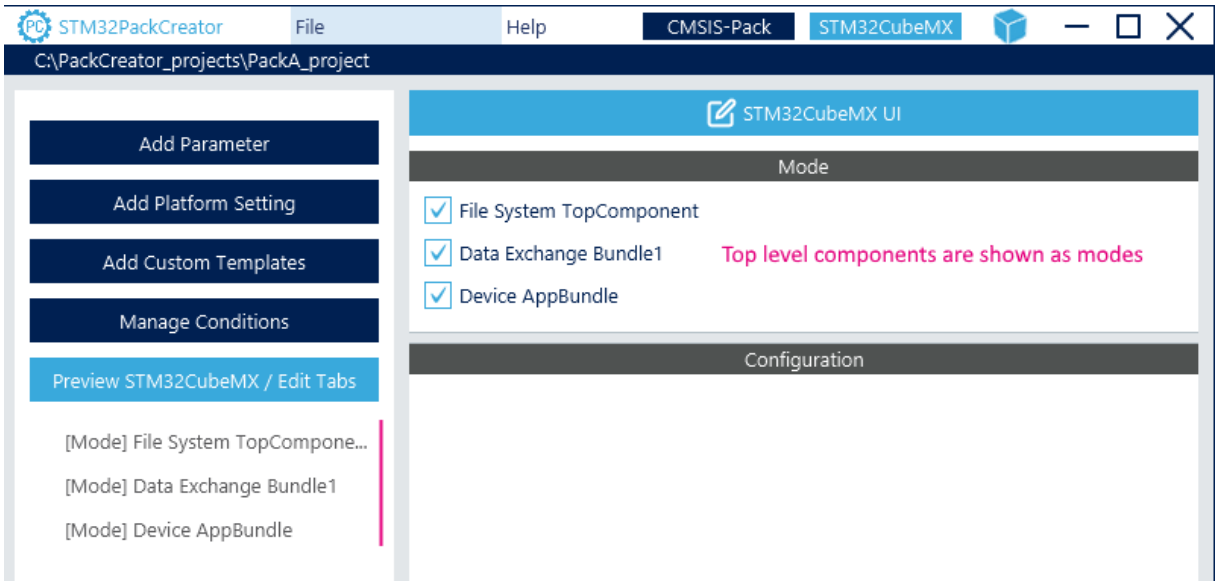


5.5.2 Switch to STM32CubeMx view

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

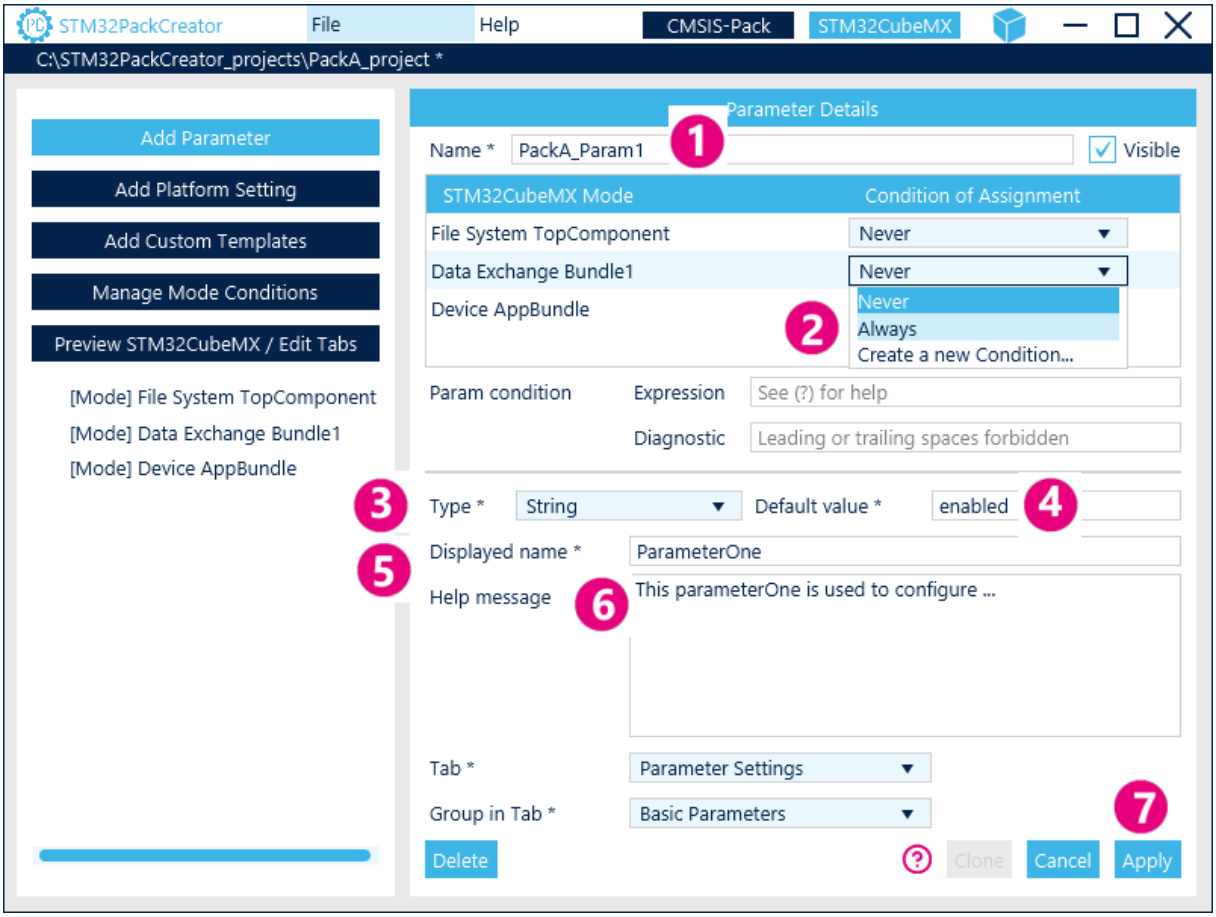


5.5.3 Add the first parameter for Bundle1

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

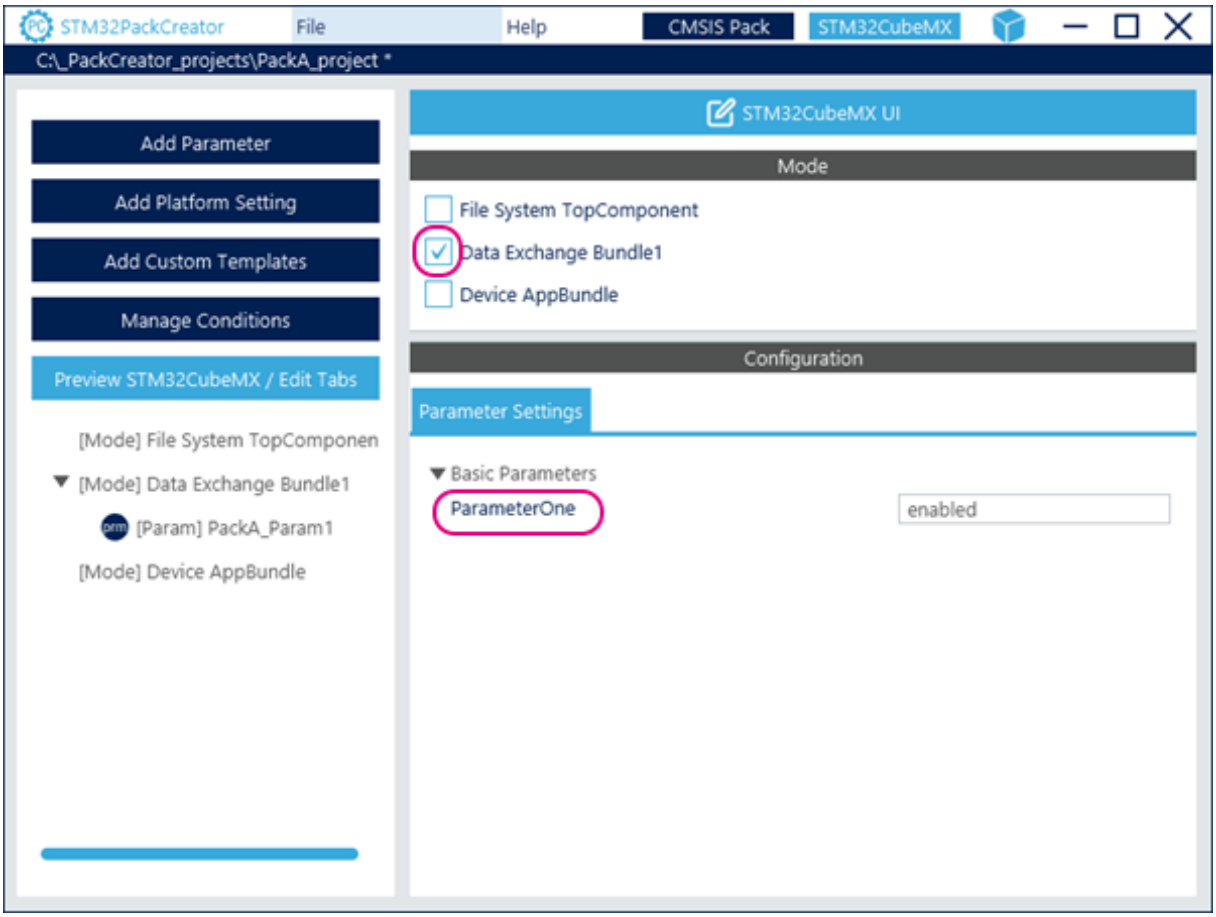
1. Enter the parameter name.
   – This name must not contain spaces and be unique per project, so it is advised to include the Pack name into the Parameter name.
2. Choose for which top-level components the parameter may be available.
   – Here: available only if Bundle1 is enabled
   – It can also be 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



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

Figure 71. Preview first parameter in STM32CubeMX UI

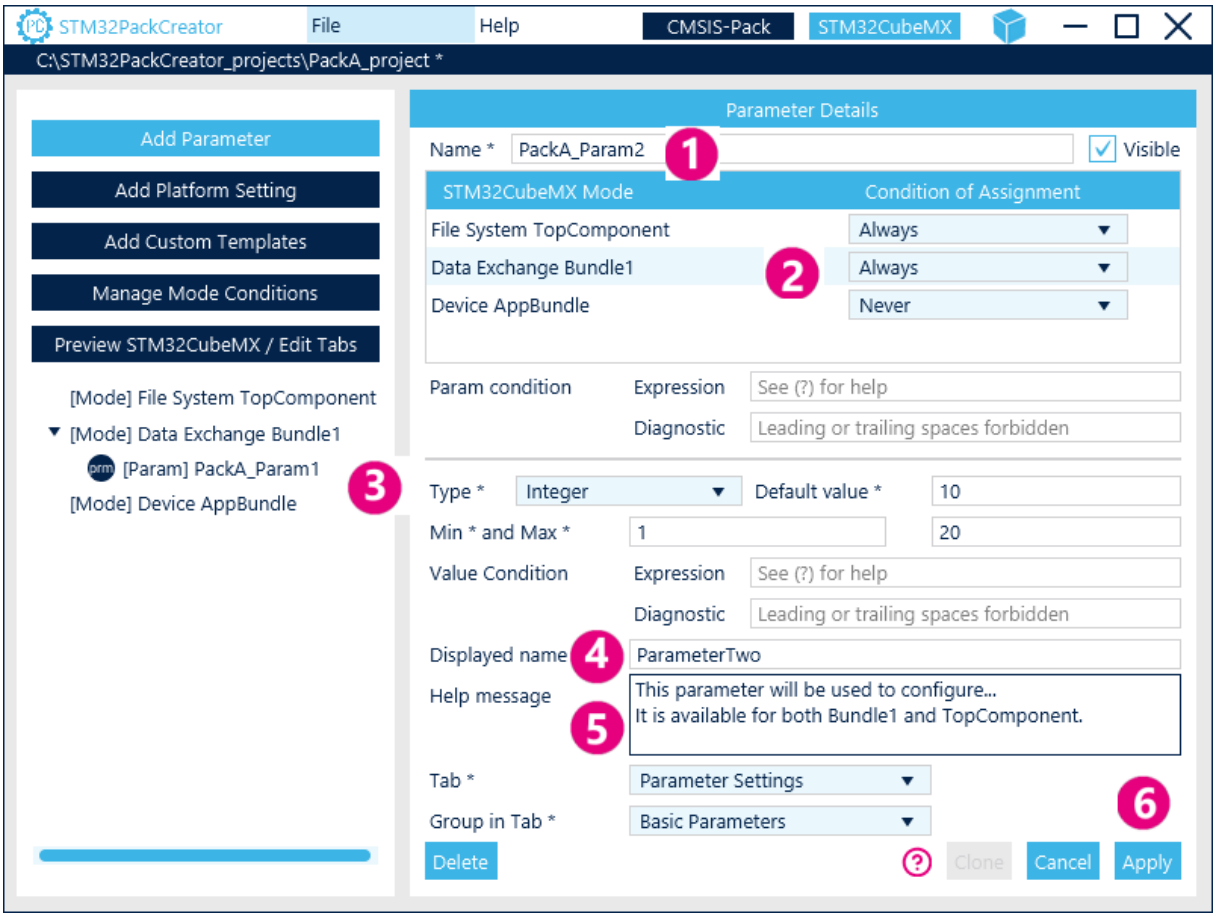


5.5.4 Add a second parameter for Bundle1 and TopComponent

Create a second parameter as shown in Figure 72.

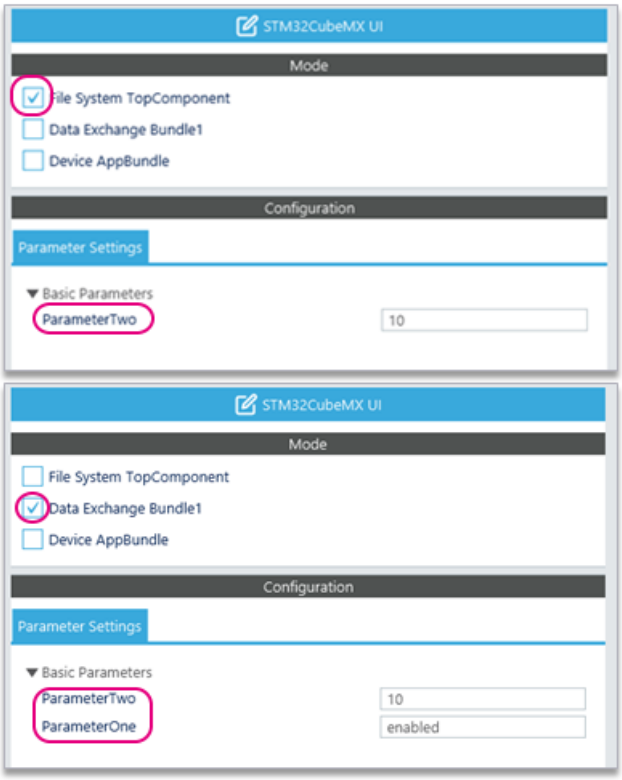
1. Enter the parameter name
2. Choose for which top-level components the parameter may be available.
   – Select Always for TopComponent and Bundle1.
3. Choose the parameter type.
   – Select Integer.
   – Enter Default, Min, and Max values.
4. Specify the label.
5. Enter a description that is displayed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

Figure 72. Adding the second parameter to Bundle1 and TopComponent



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

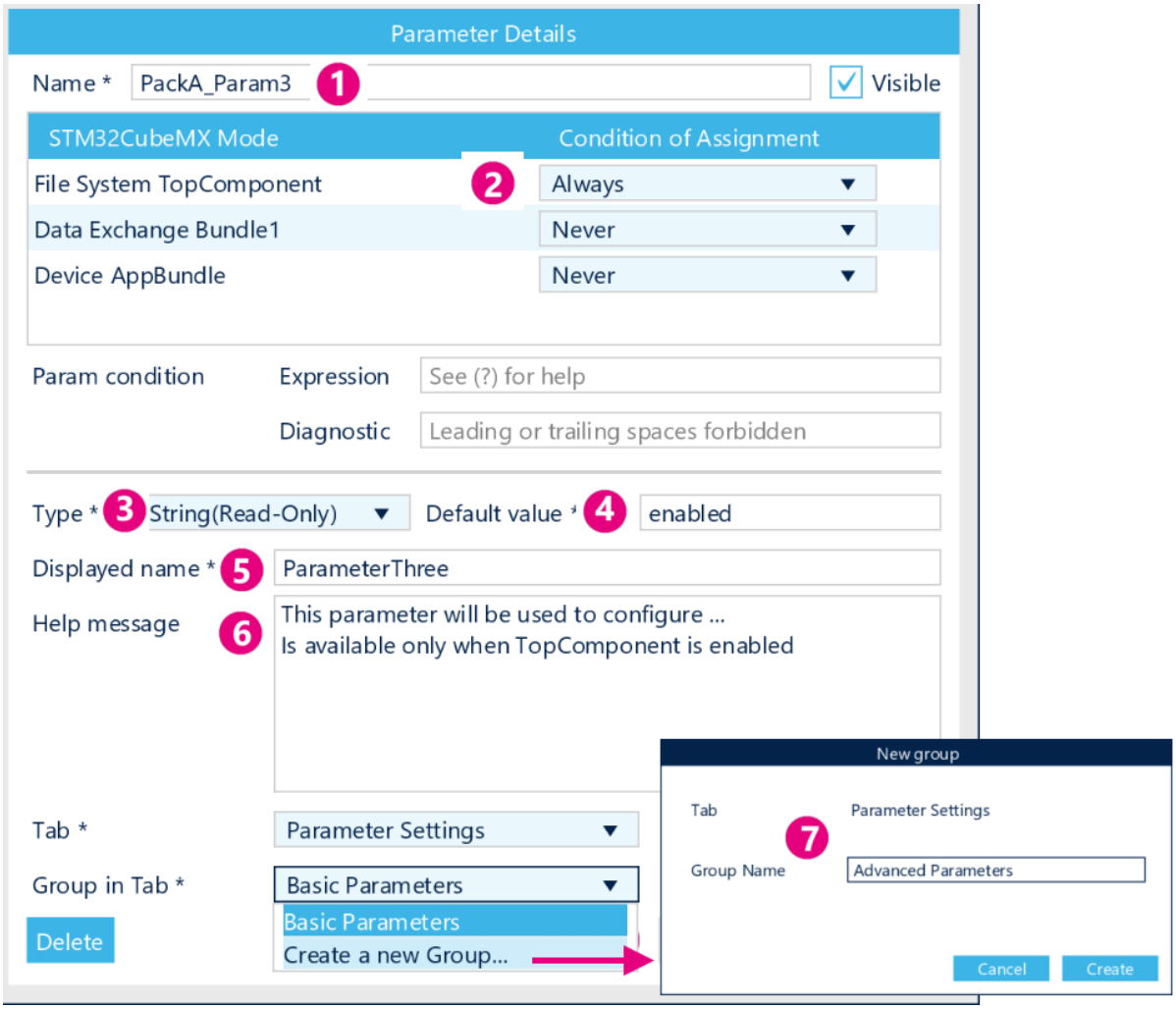


5.5.5 Add the third parameter for TopComponent in the user-defined tab and group

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

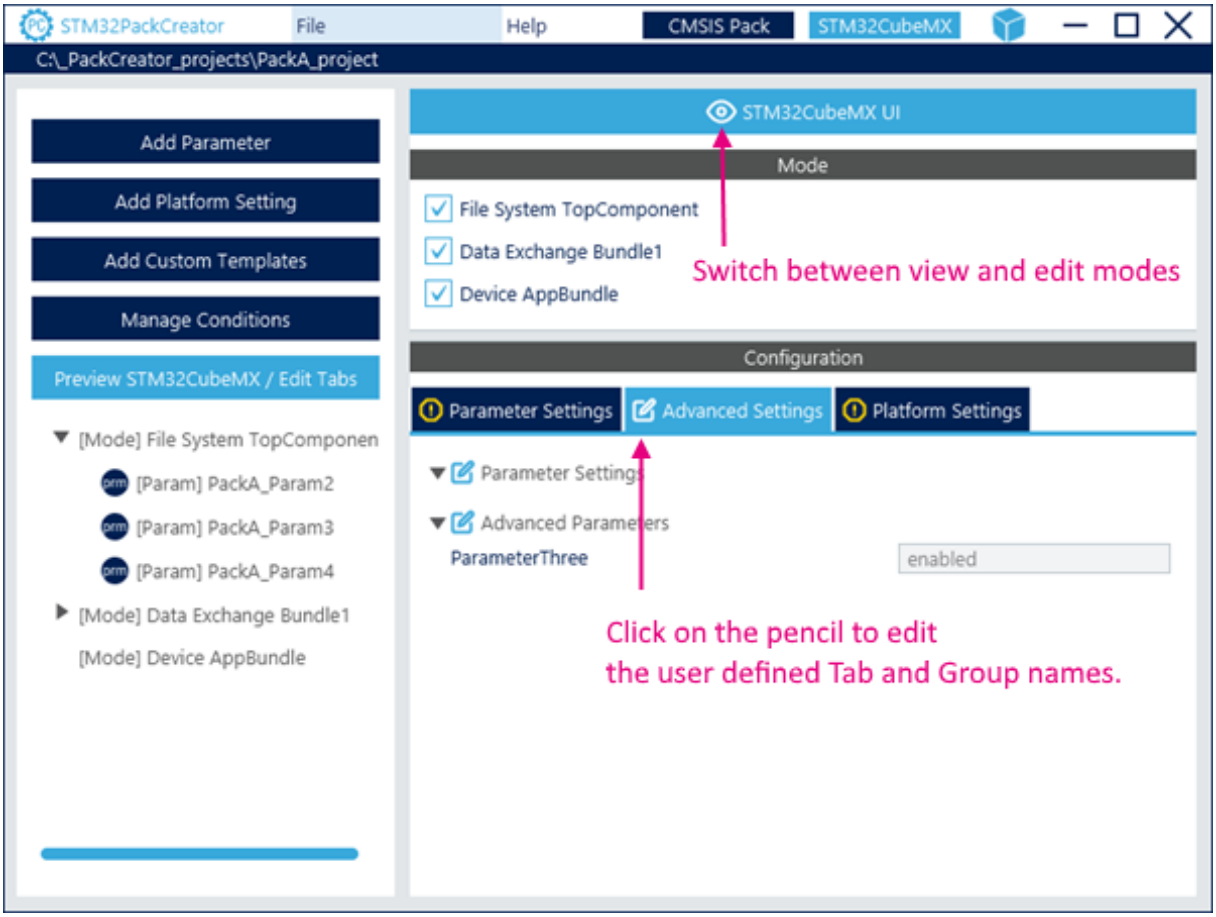
1. Enter the parameter name.
2. Select `Always` for TopComponent.
3. Choose the parameter type as a read-only string. Set its default to `enabled`.
4. Specify its label.
5. Enter its description.
6. For the Tab, select `Create a new Tab` and name it `Advanced Settings`.
7. For the Group in Tab, select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

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



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

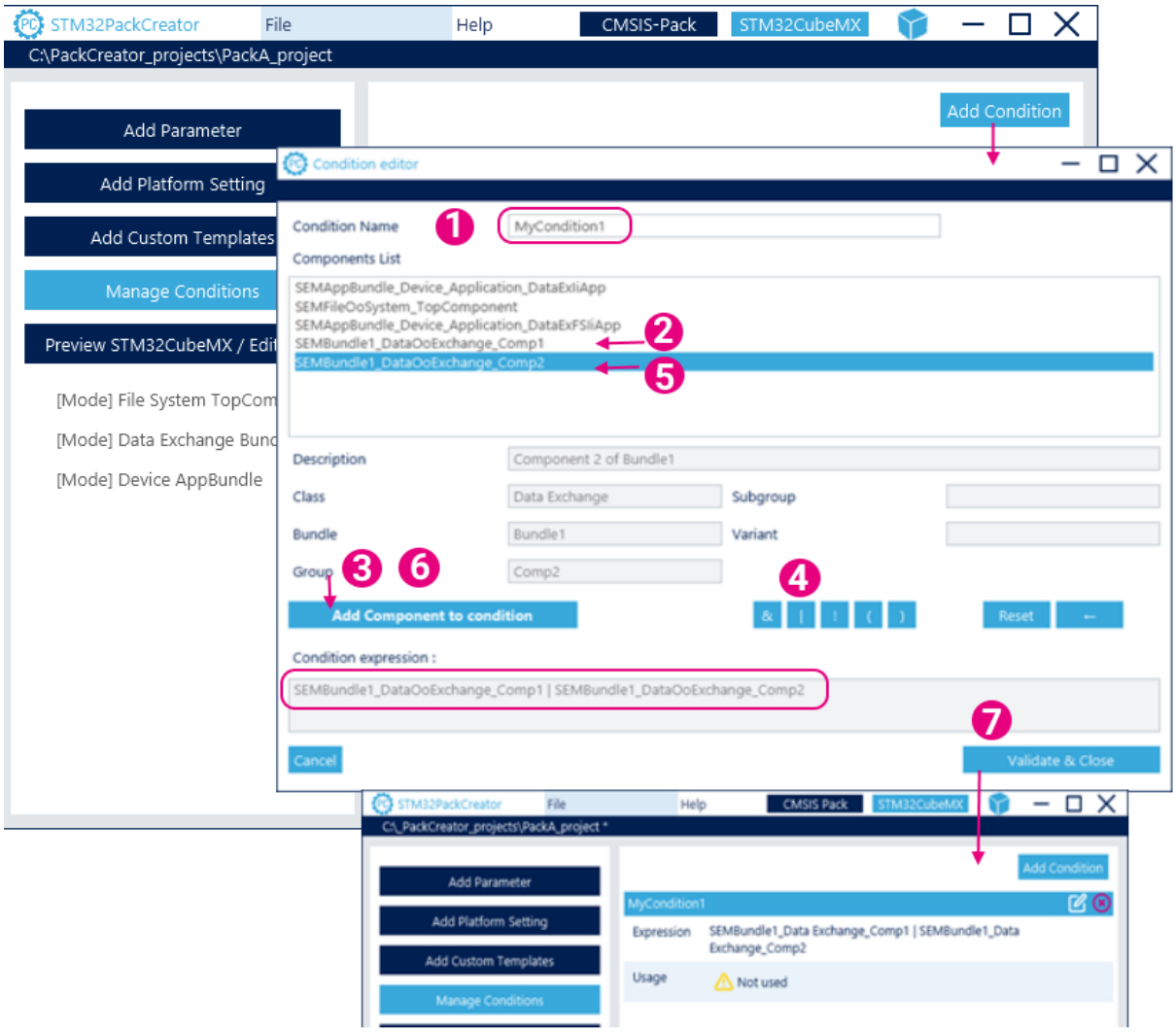
Figure 75. Preview third parameter in STM32CubeMX UI



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.

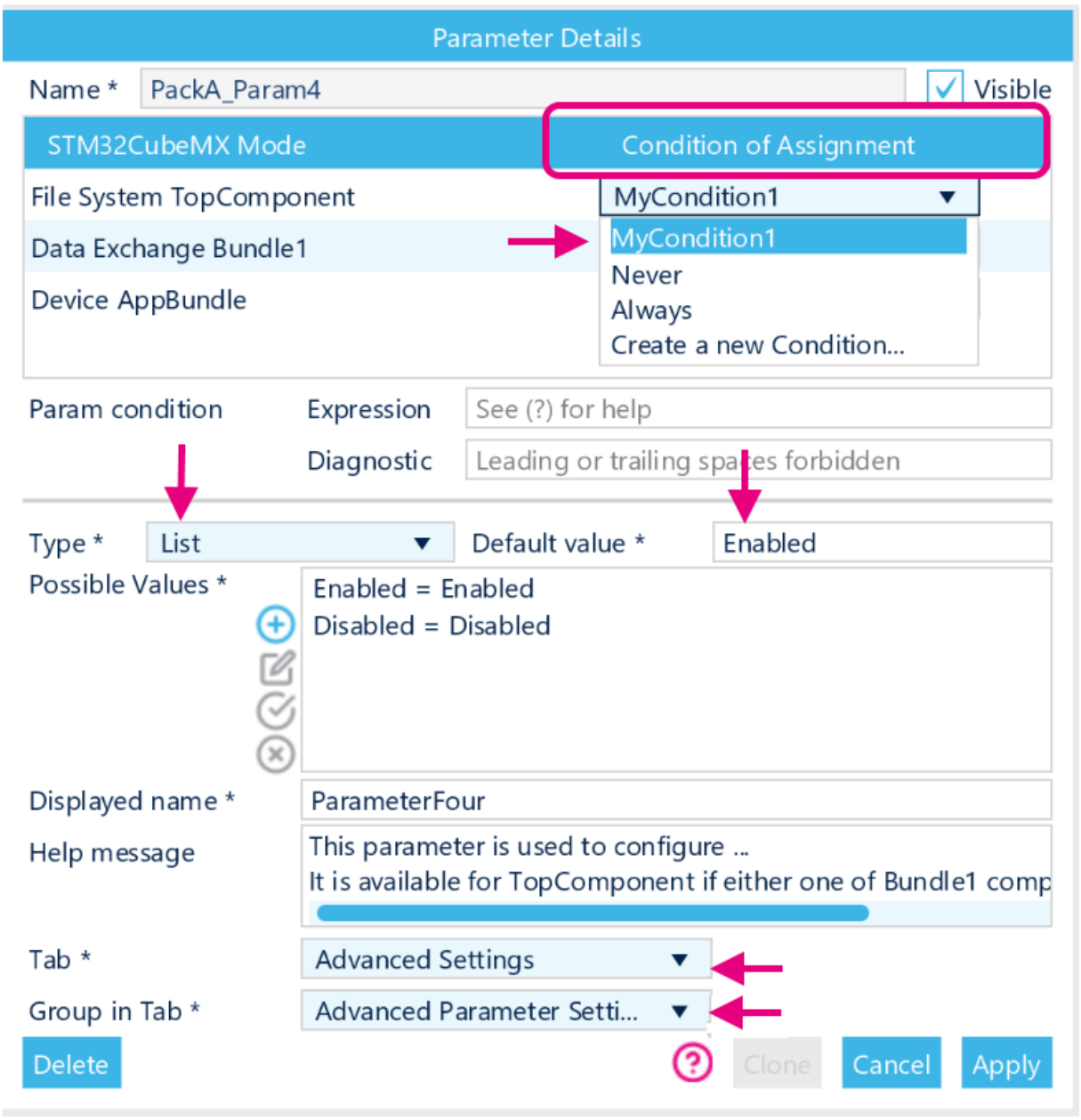


5.5.7 Create a fourth conditional parameter

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

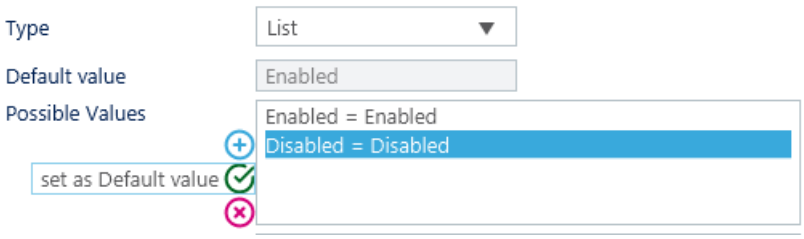
1. Enter the parameter details.
2. Assign it to TopComponent on the condition MyCondition1 is met. Refer to Figure 76.

Figure 76. Creating parameters with conditions



3. Create a list parameter: click the + sign to add values, cross icon to delete, checkmark 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.

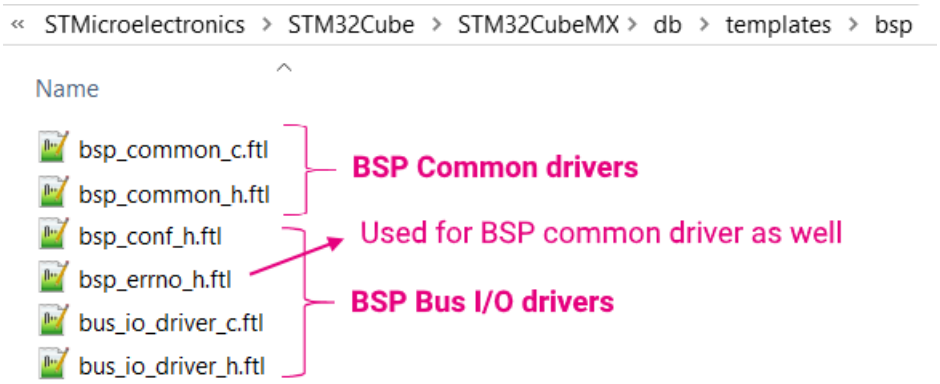
STM32Cube specifications define a set of Board Support Package APIs among which the BSP Bus and BSP Common driver APIs.

- The BSP bus driver API, which exports the transport functions used by the components IO operations. 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

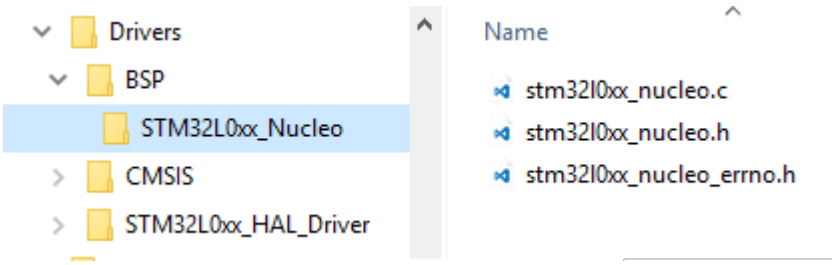
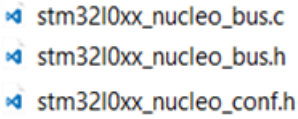


Figure 80. STM32CubeMX generated Bus I/O driver files



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 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

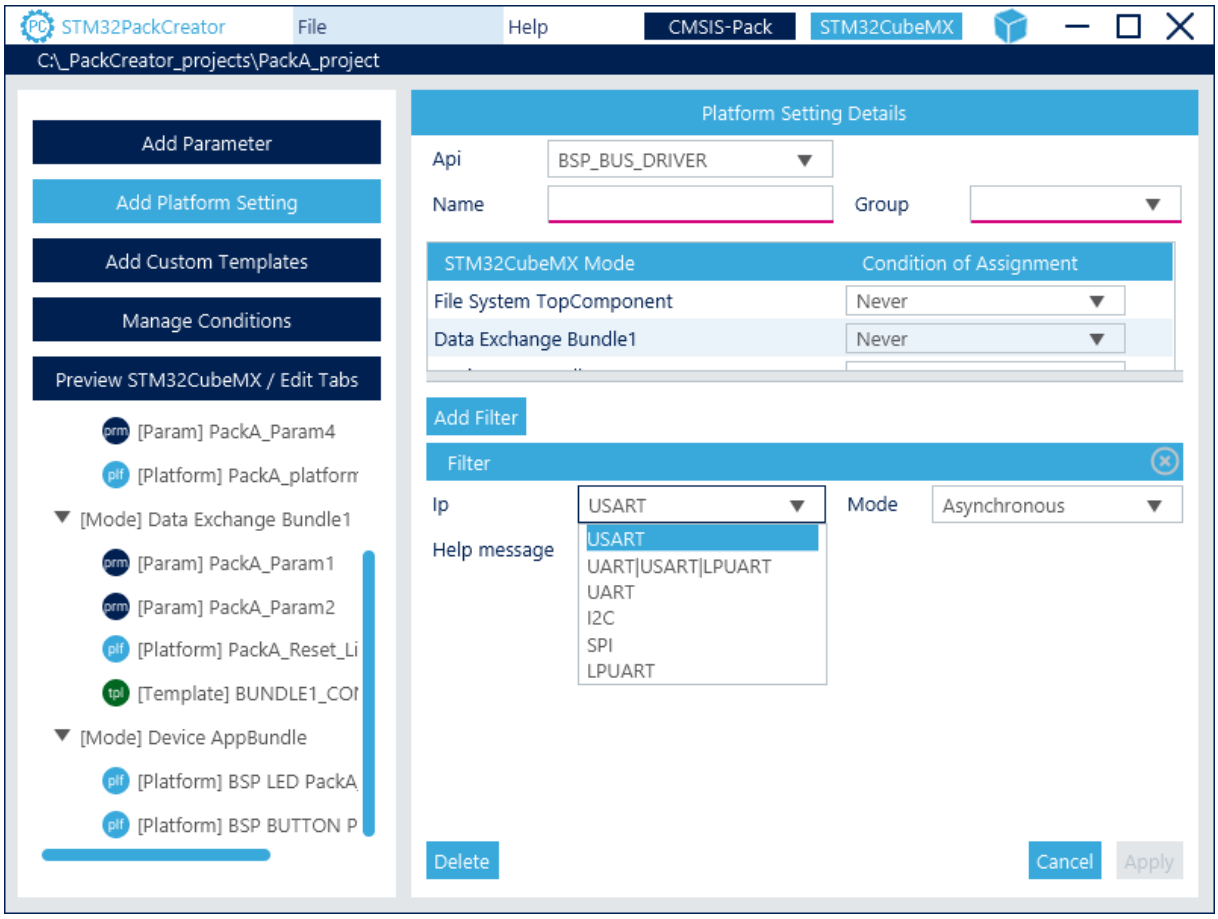


Figure 82. **Platform settings for BSP common API**

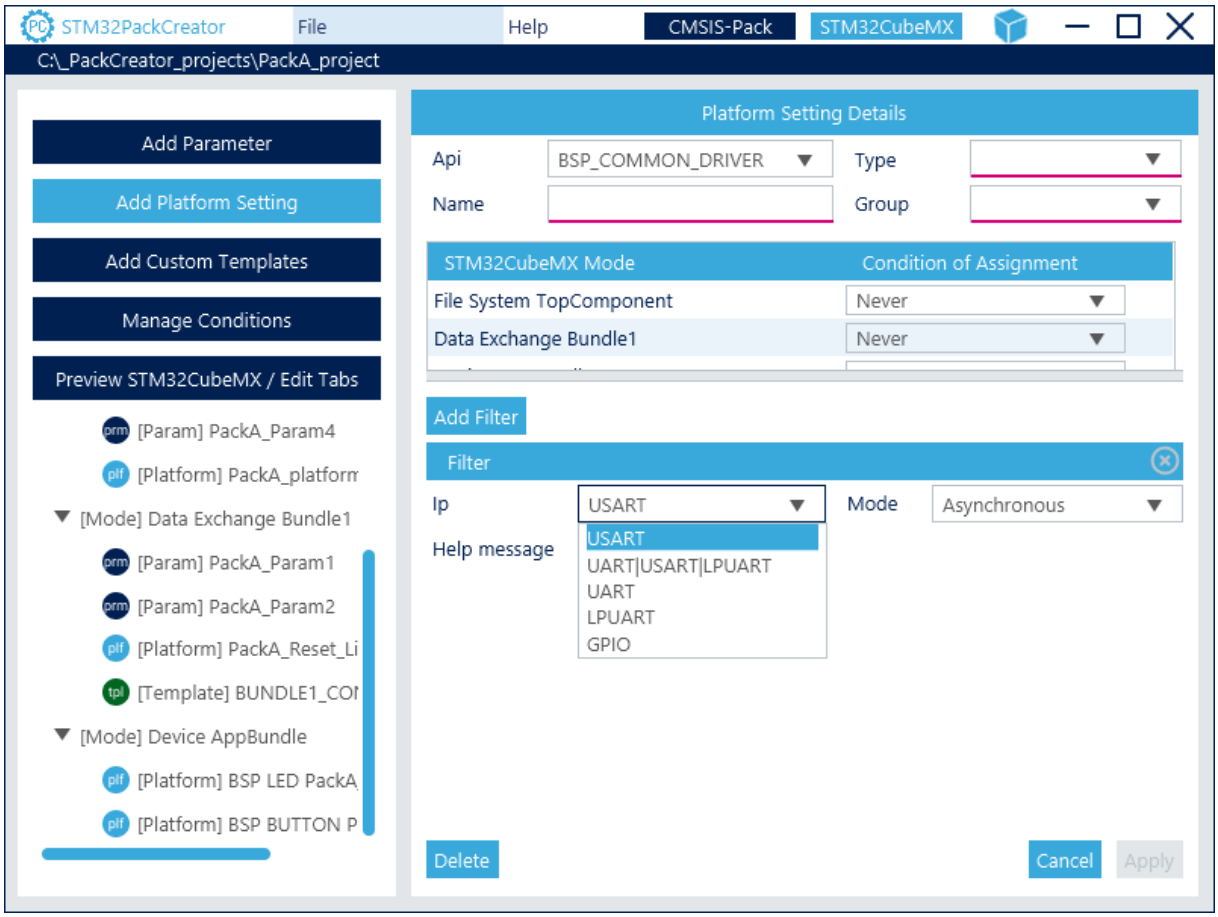
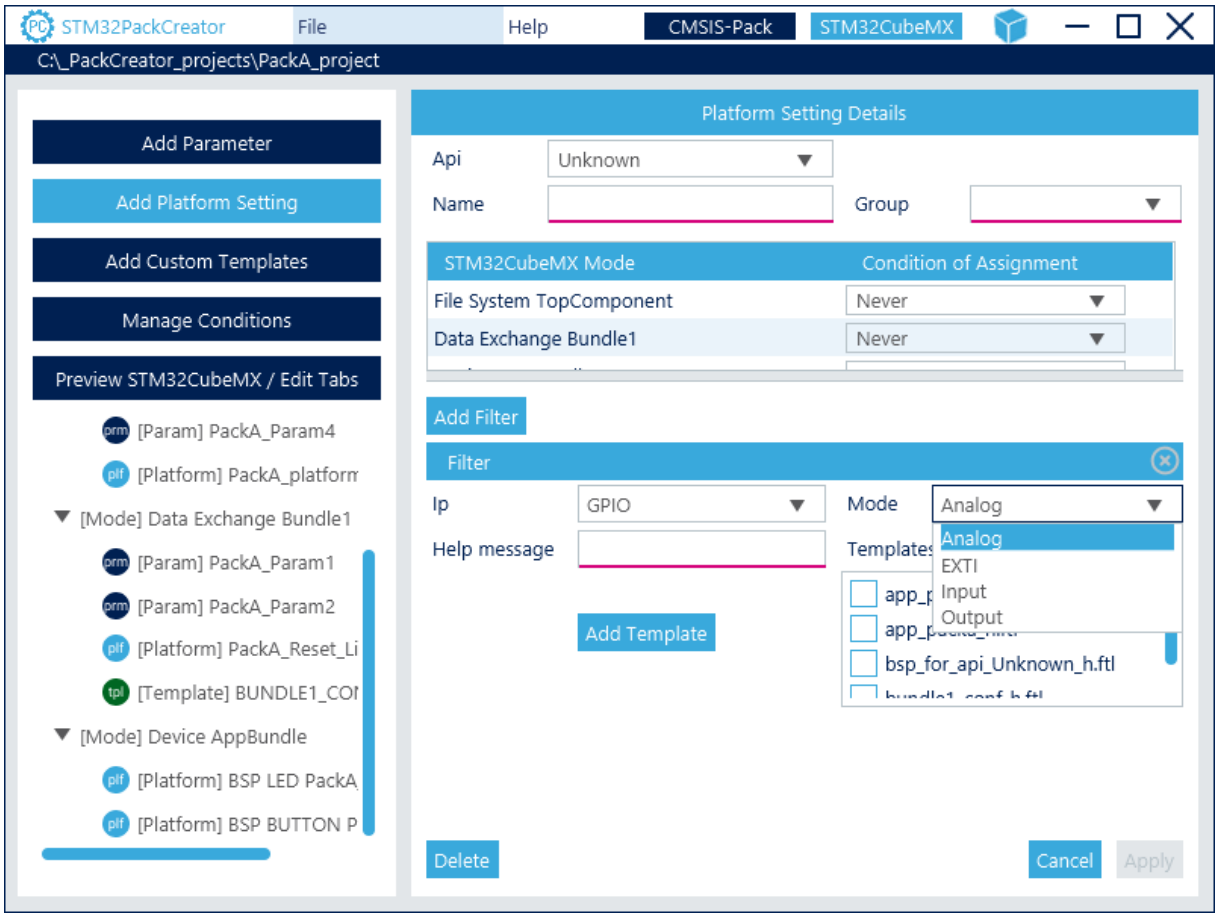


Figure 83. **Platform settings with custom BSP API**



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.

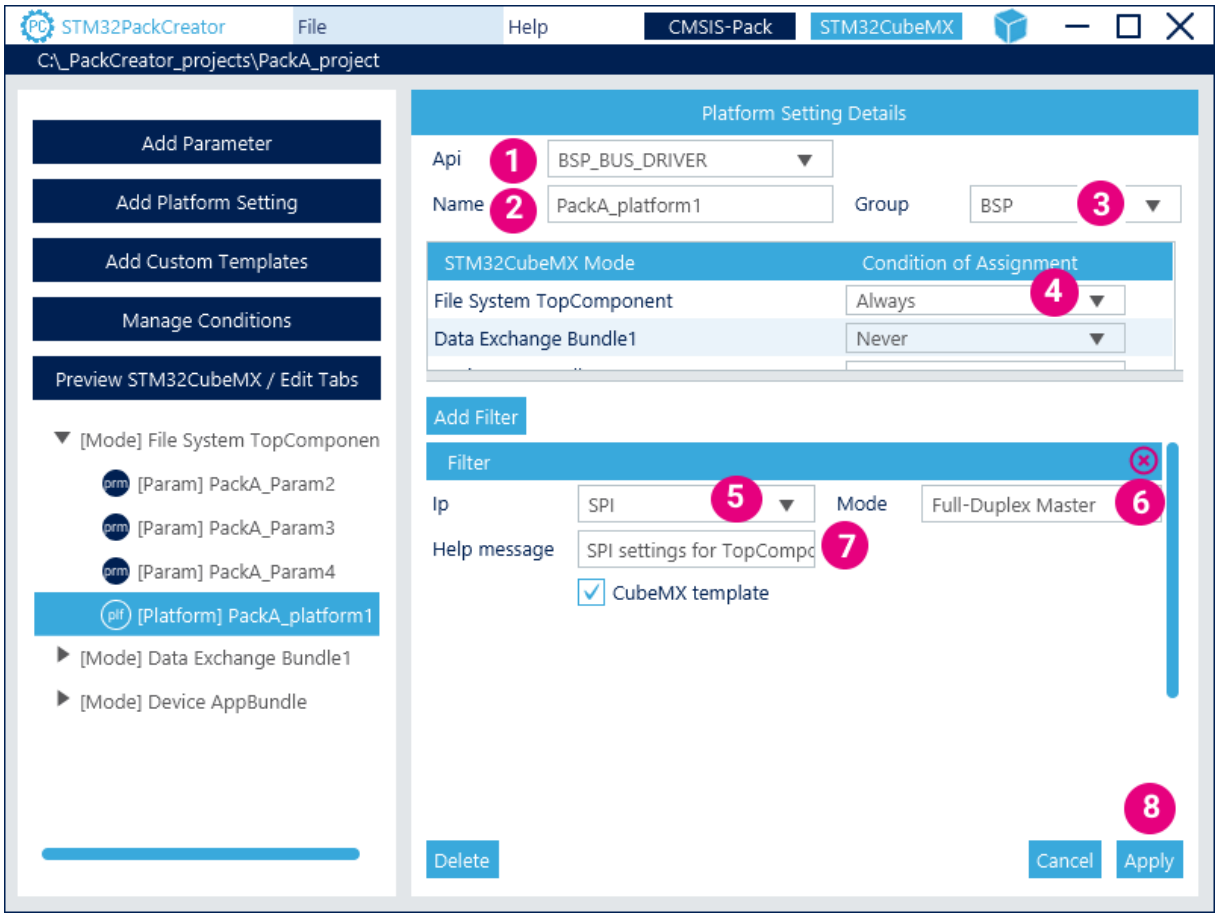
- Example for Timer 1 up to Timer 8 peripherals: IP = `TIM1_8`, Mode = `Internal Clock`
- Example: IP= `RTC`, Mode = `Activate Clock Source`

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 I^2C 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



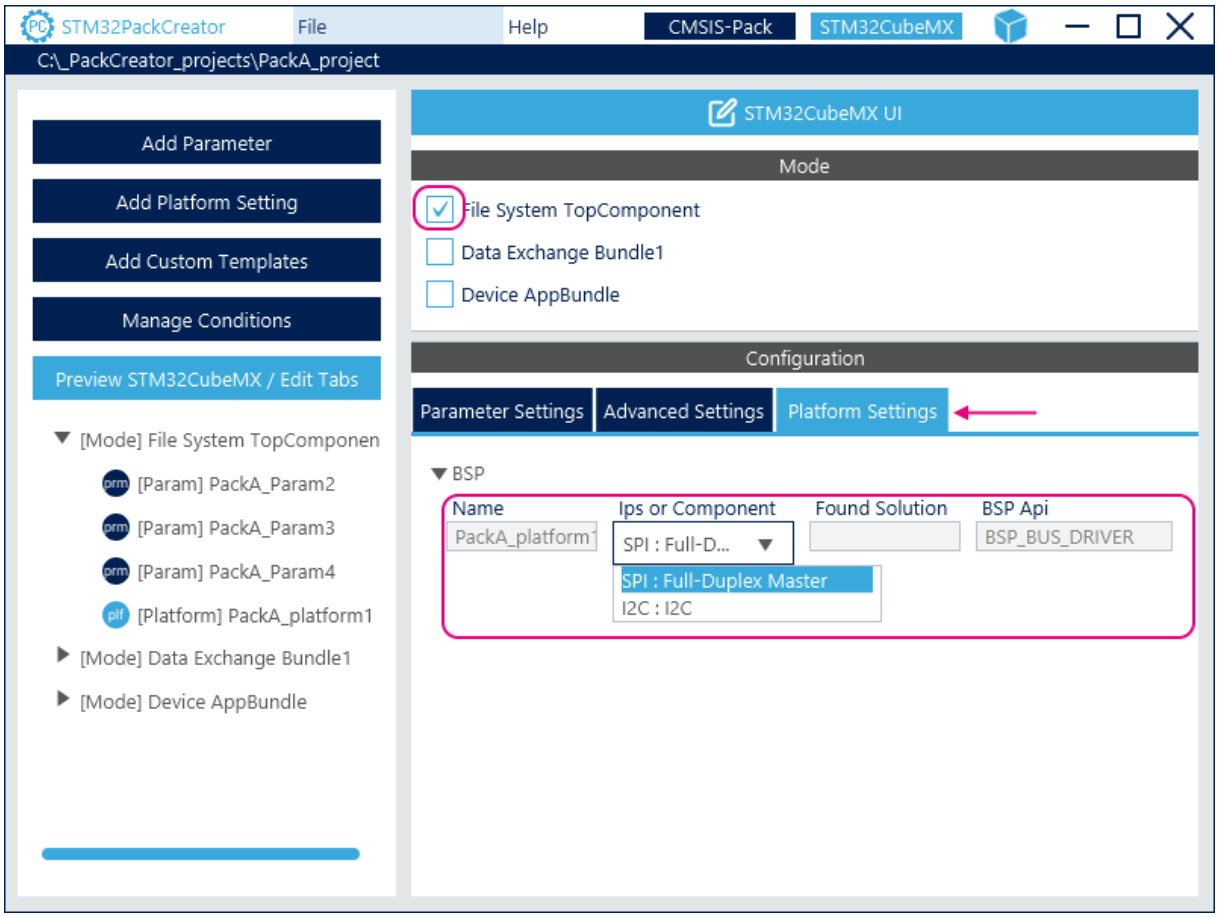
1. Enter the name (unique, no space) `PackA_platform1`.
2. Select the API `BSP_BUS_DRIVER`.
3. Select the `BSP` group.
4. Assign it to a Mode `Always on TopComponent`.
5. Under Filter, select the peripheral to use `SPI`.
6. Select the peripheral mode to use the `Full-duplex master`.
7. Fill in the `help`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

Then add the option to use I^2C 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

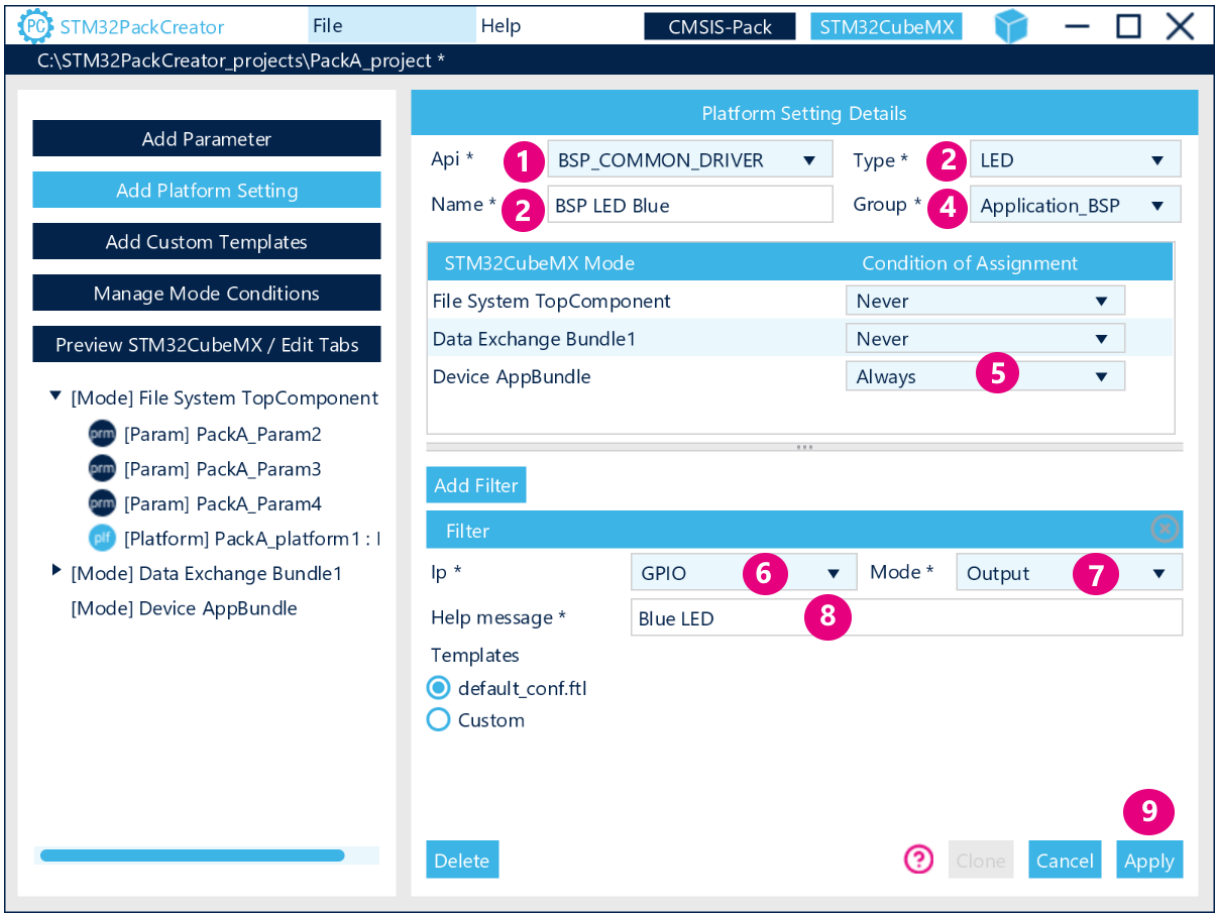


5.6.3 Create platform settings for bus common drivers (LED and button)

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.

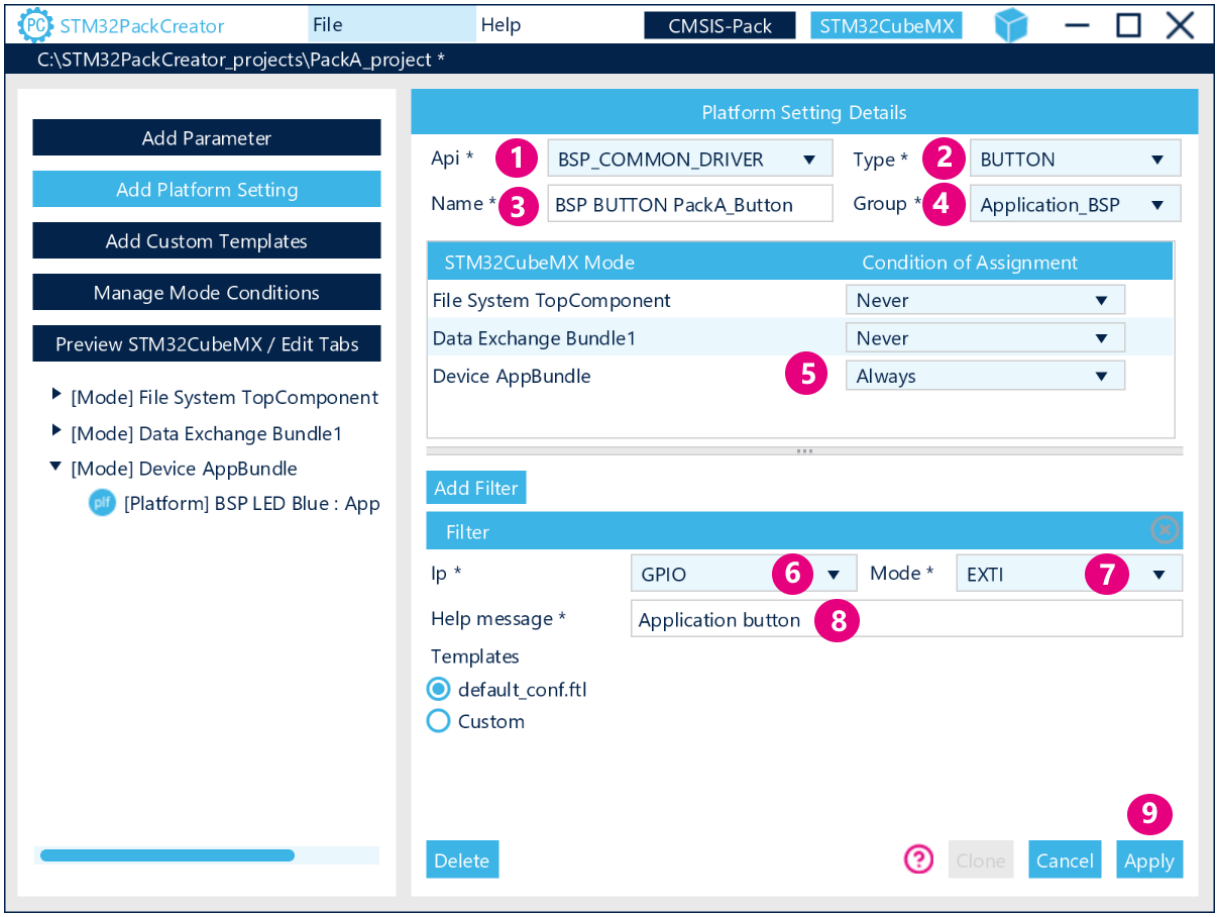
Figure 86. Creating platform settings for a LED



1. Select the API `BSP_COMMON_DRIVER`
2. Select the type `LED`
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

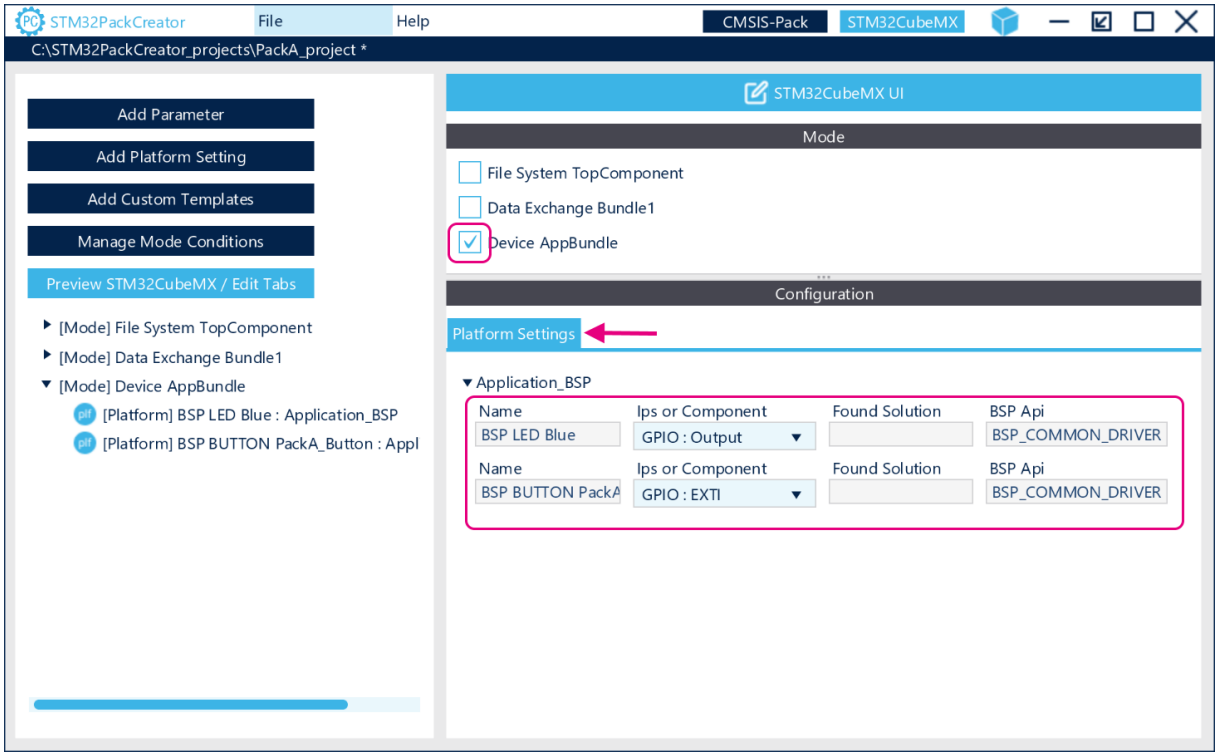
Click Add Platform settings and proceed similarly to add the Button (Type=Button, IP=GPIO, Mode=EXTI) as shown in Figure 87.

Figure 87. Creating platform settings for a button



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

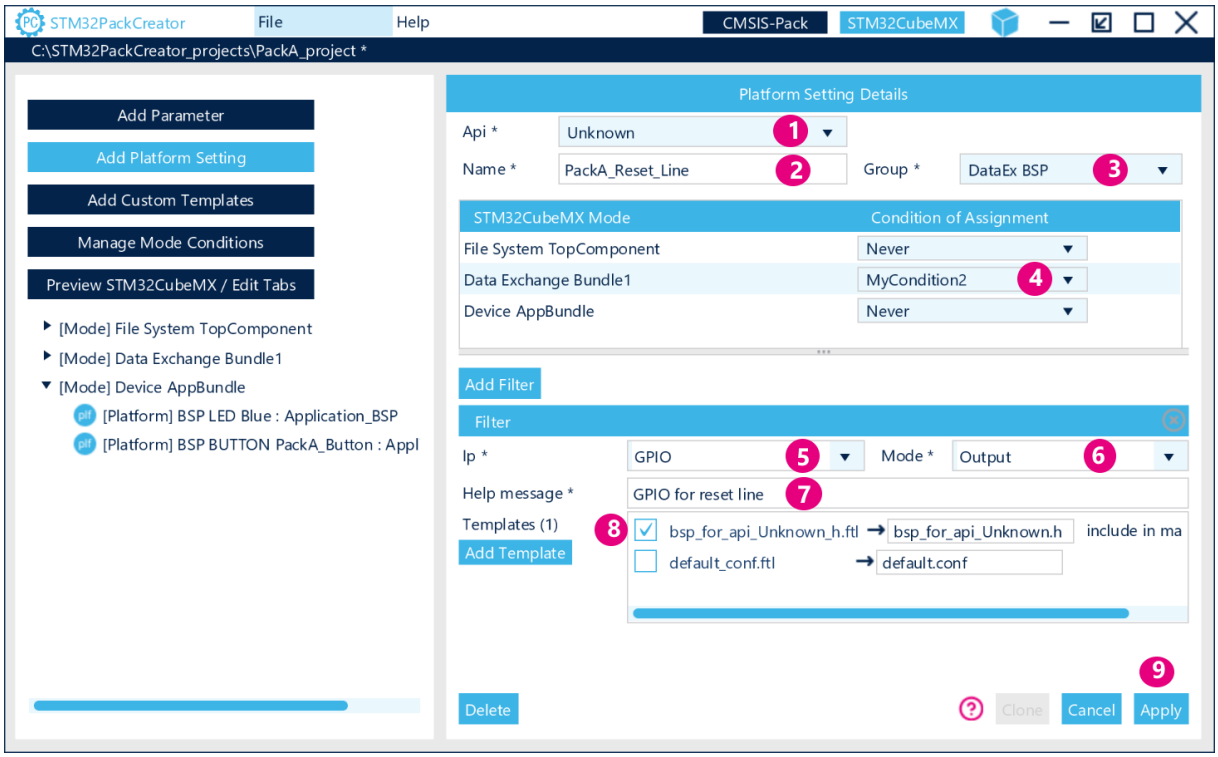


5.6.4 Create platform settings for unknown API

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



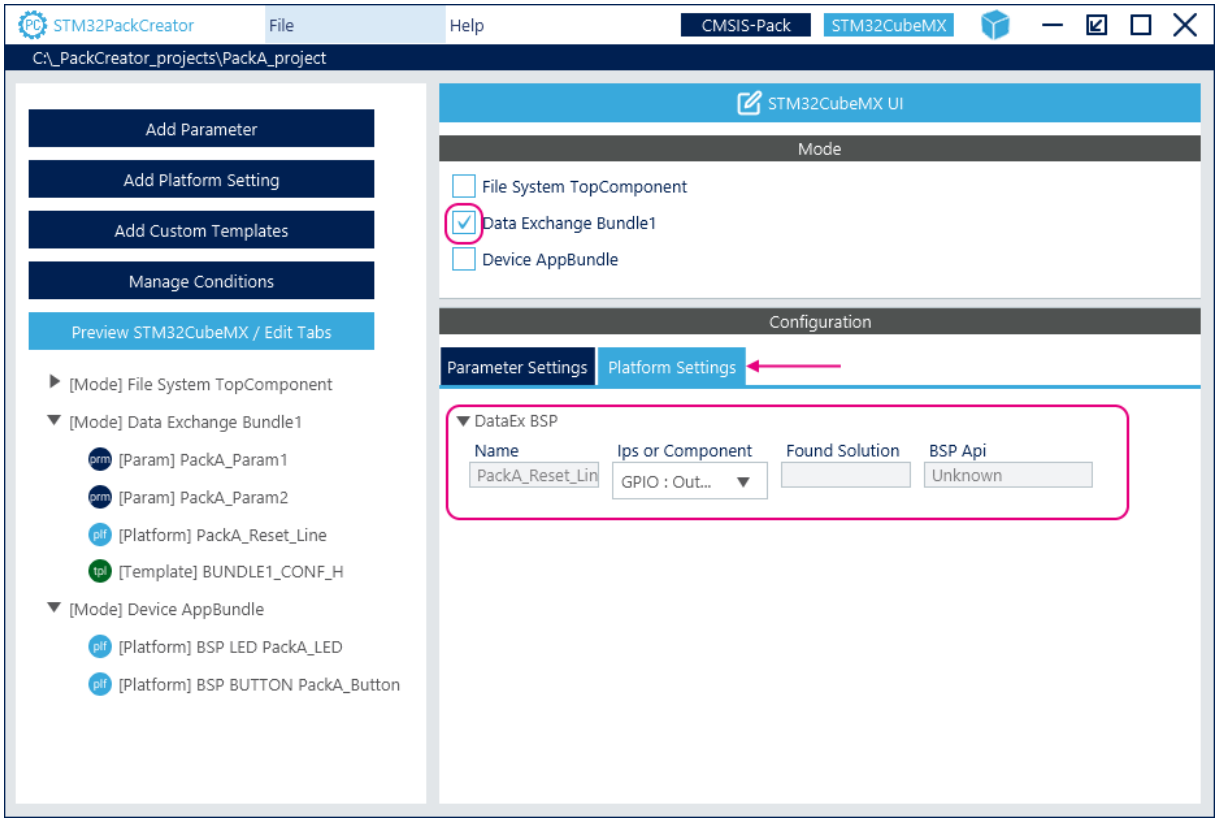
1. Select the API `Unknown`.
2. Enter the name (unique, no space) `PackA_Reset_Line`.
3. Create a new group `DataEx_BSP`.
4. Assign to the Mode Bundle1.
   – Create a condition: `MyCondition2` (TopComponent is present)
5. Under Filter select `GPIO` and `Output` mode.
6. Fill in the Help message
7. Select `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, 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 parameter 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]
```

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)***

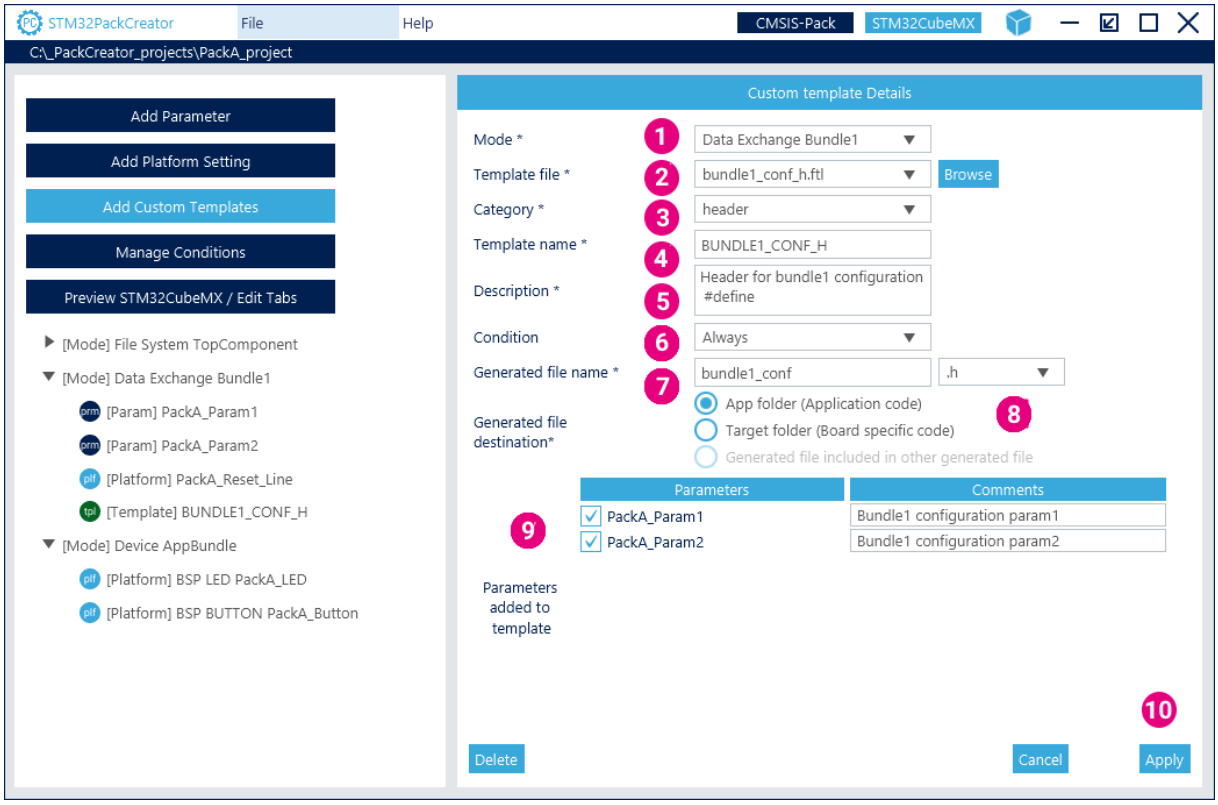
```
#ifndef __API_UNKNOWN_H
#define __API_UNKNOWN_H

#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 customs 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



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 `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.

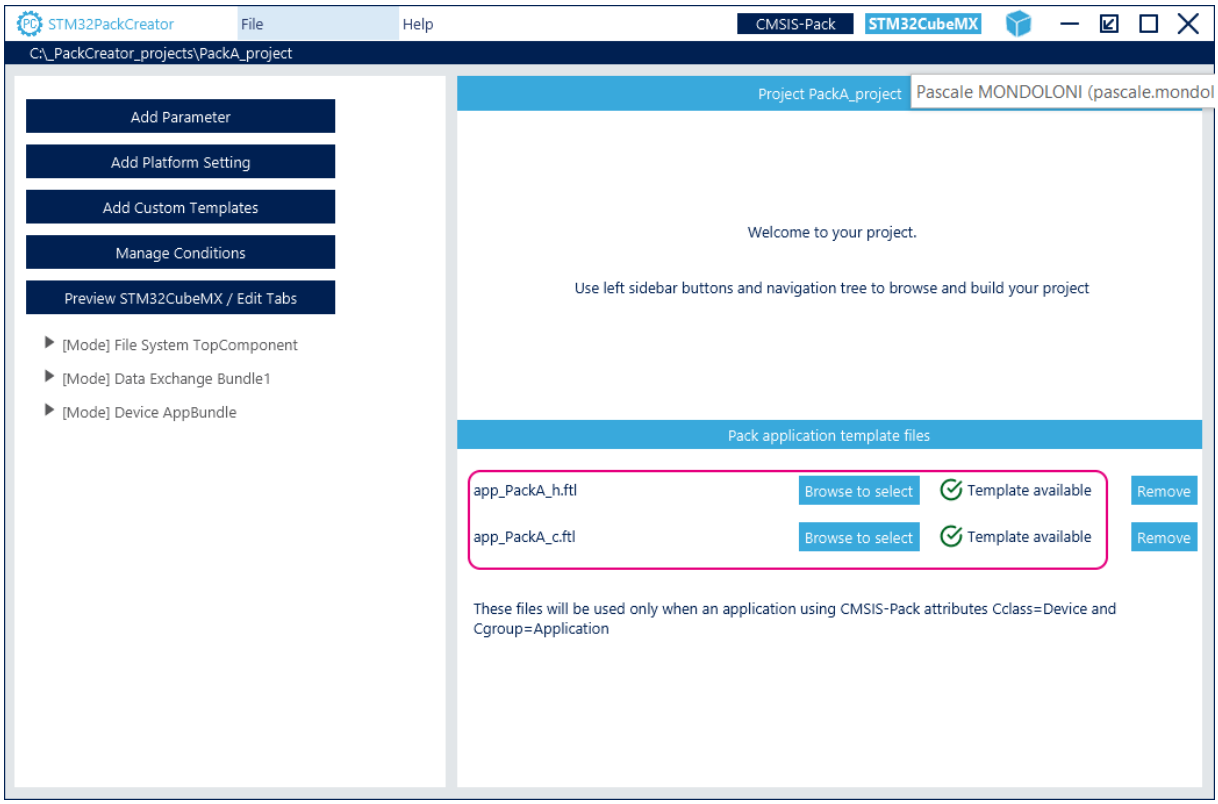
The main purpose of application components is for STM32CubeMX to generate application-ready code

- `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 application custom templates can be designed so that code is generated differently according to the RTE component defines or module names.

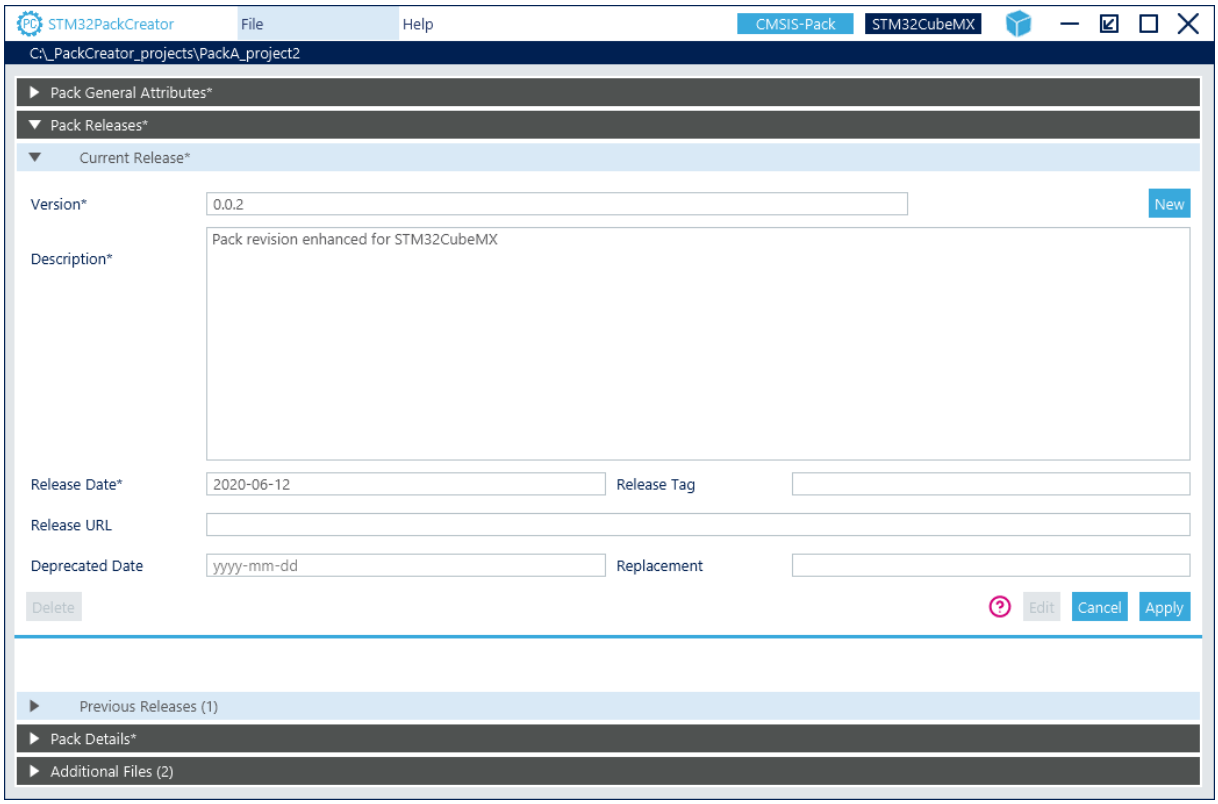
- Use RTE Components defines:
  - RTE Components are specified in the components section of the CMSIS-Pack view.
  - Most useful when all applications share the same module name, which is recommended to support multi-pack configurations. Use the following Freemarker statement: `#if define?contains("<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



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

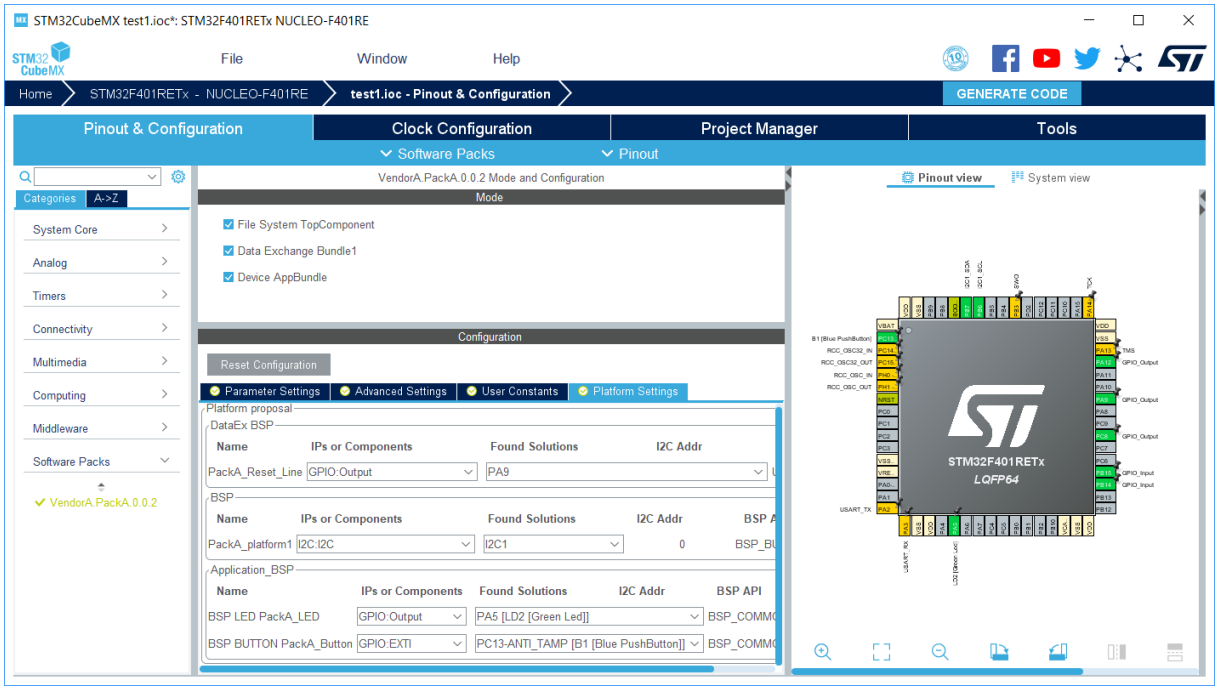
Select the pack components using `Software Packs > Select Components` menu and click `OK` to close the component selector window.

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 I^2C 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 `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`.

Note: 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.

Revision history

Table 7. **Document revision history**

Date	Revision	Changes
12-Jul-2020	1	Initial release.
2-Nov-2020	2	Deleted compatibility limitation with CMSIS-pack standard in *Section 4.3 Rules and limitations* Updated: • JRE™ requirements in *Section 4.4 System requirements* • Program launch in *Section 4.5.1 From STM32CubeMX user interface* and *Section 5.2 Create a new project from scratch* • *Figure 5*, *Figure 7*, *Figure 8*, *Figure 9*, *Figure 15*, *Figure 19*, and *Table 3* handling example in pack details Added *Section 5.3.15 Specify the example projects to be delivered with the pack*
11-Feb-2021	3	Updated: • JRE™ requirements in Section 4.4 System requirements • Command lines in Section 4.5.2 Standalone option • Table 4. STM32CubeMX view - Main features • Section 5.4.3 Select pack components, check and solve dependencies • Section 5.4.5 Generate the project • 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 Removed: • Former *Figure 63*

Contents

List of tables

List of figures