AN5952

Application note

How to use CMake in STM32CubeIDE

Introduction

This application note describes how to control the software compilation process using the CMake utilities for C/C++ projects in the STMicroelectronics STM32CubeIDE integrated development environment.

1 General information

STM32CubeIDE supports STM32 32-bit products based on the Arm® Cortex® processor.

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

1.1 Purpose

STM32CubeIDE offers the user-requested CMake feature, which developers can leverage for their developments in the STM32 MPU and MCU ecosystems.

1.2 The use cases in this document

In the STM32CubeIDE context, a user can compile C/C++ projects using either the makefile or the CMake solutions. This document details the use of CMake for two use cases:

- The user wants to work with an existing CMake project structure
- The user wants to start a CMake development from scratch

1.3 Compatible toolchain

The STM32CubeIDE CMake support presented in this application note works with the following minimum version of the toolchain:

- STMicroelectronics STM32CubeIDE v1.13.0

1.4 Prerequisites

CMake must be installed on the user's computer. The compatibility with STM32CubeIDE is from CMake v3.13 onwards.

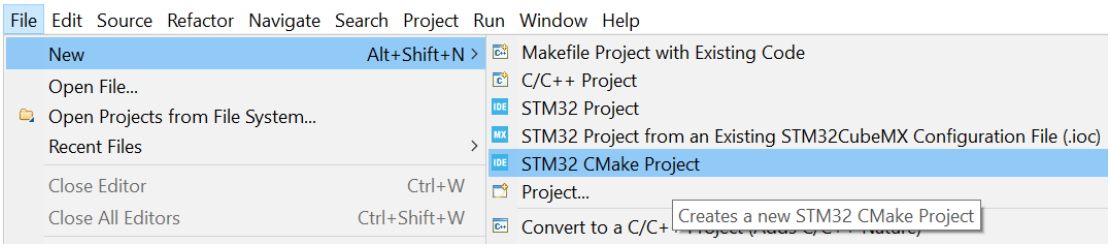
2 Create projects

2.1 Creation with an existing CMake project structure

Creating a new project with an existing CMake structure offers an easy way to use an already developed or a downloaded project or library for use in STM32CubeIDE. To do so, start by following the steps below:

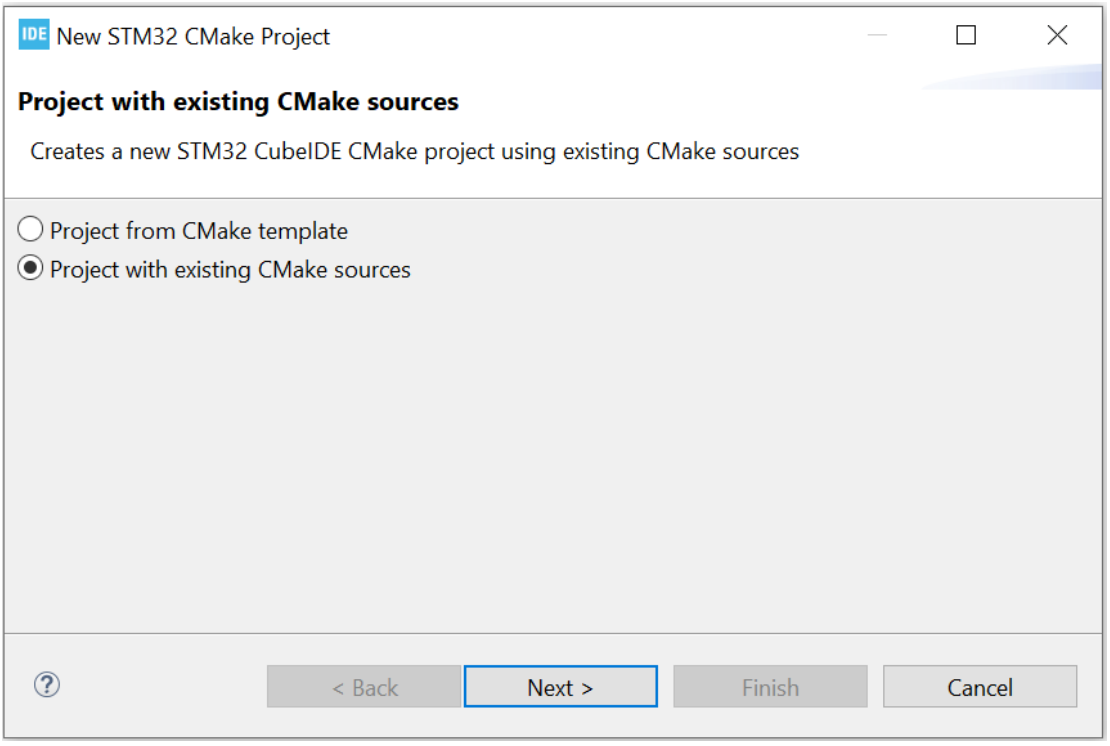
- Select [**File**]>[**New**]>[**STM32 CMake Project**]

Figure 1. CMake project creation



- Select [**Project with existing CMake sources**]

Figure 2. CMake project types



- Select [**Next >**]

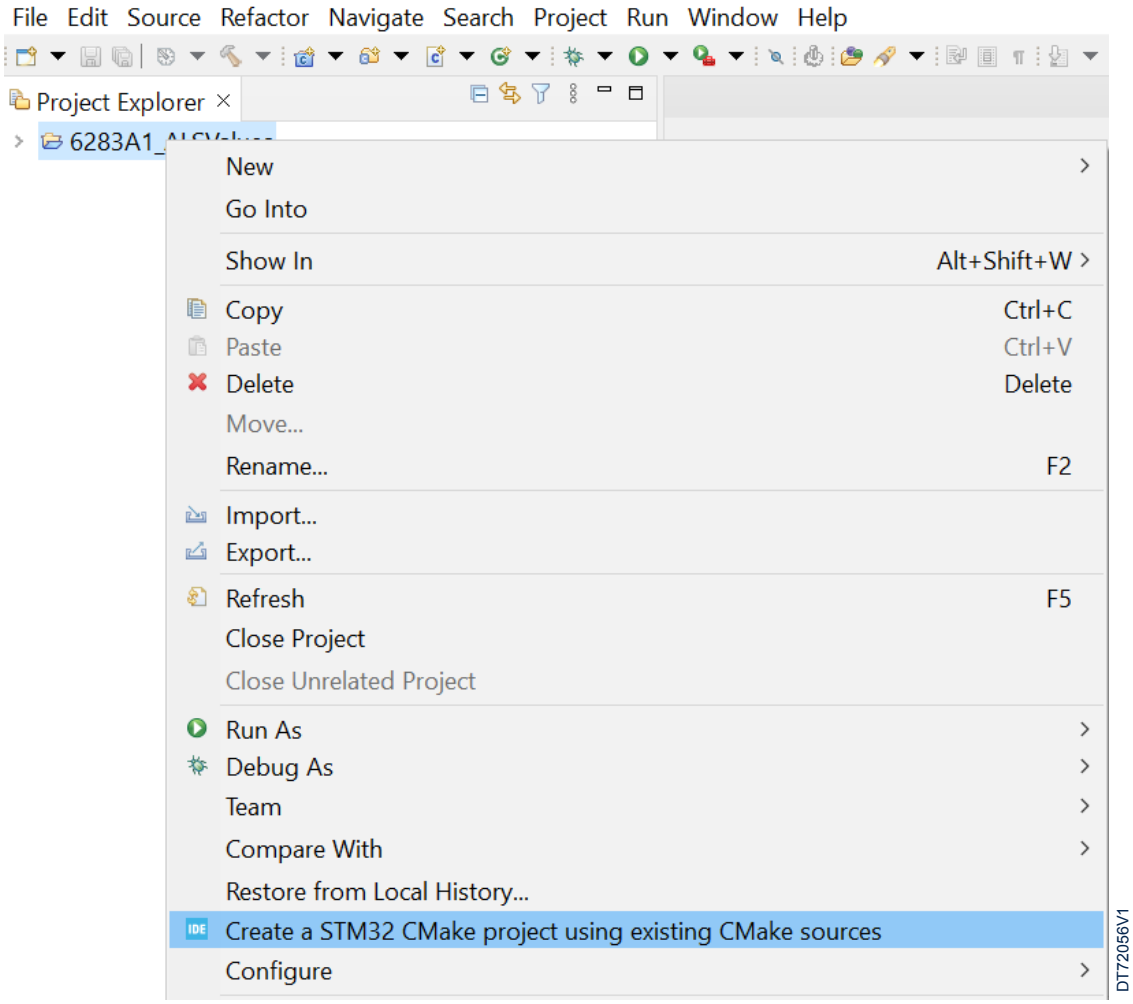
The next wizard page allows the setup of a CMake project in two different ways:

- Project creation inside an existing CMake project structure (refer to Section 2.1.2)
- Project creation external to an existing CMake project structure (refer to Section 2.1.3)

2.1.1 CMake project as a subproject

Alternatively, to create a CMake project as a subproject in a more complex project structure, use the following context menu option.

Figure 3. CMake project as a subproject



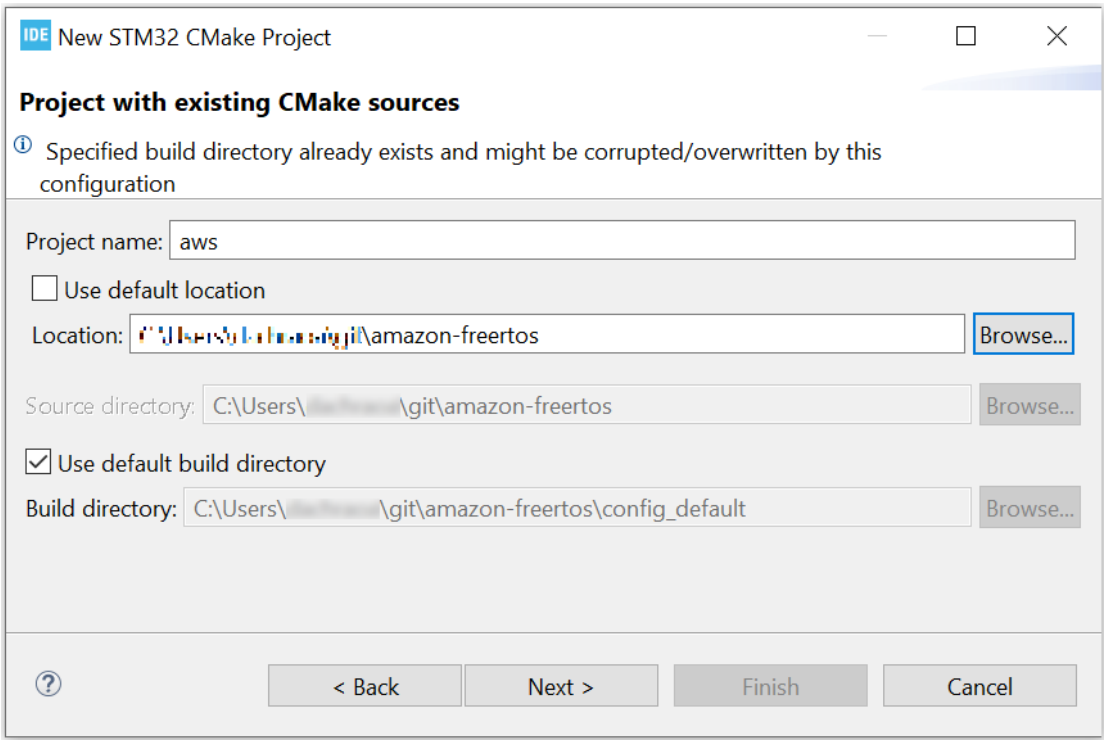
Then, for a project creation inside or external to an existing CMake project structure, refer to Section 2.1.2 or Section 2.1.3.

2.1.2 Creation inside an existing CMake project structure

To create a new project inside an existing package that was already downloaded or created, do the following:

- Specify a name for the project.
- Uncheck [**Use default location**].
- Use the [**Browse…**] button and select the root directory of a CMake-based project. This disables the [**Source directory**] field since the project location and the CMake source directory are the same.
- Specify the path to the build directory to be used for the CMake configuration. By default, the build directory is `config_default` relative to the CMake source directory.

Figure 4. Project creation inside an existing CMake project structure



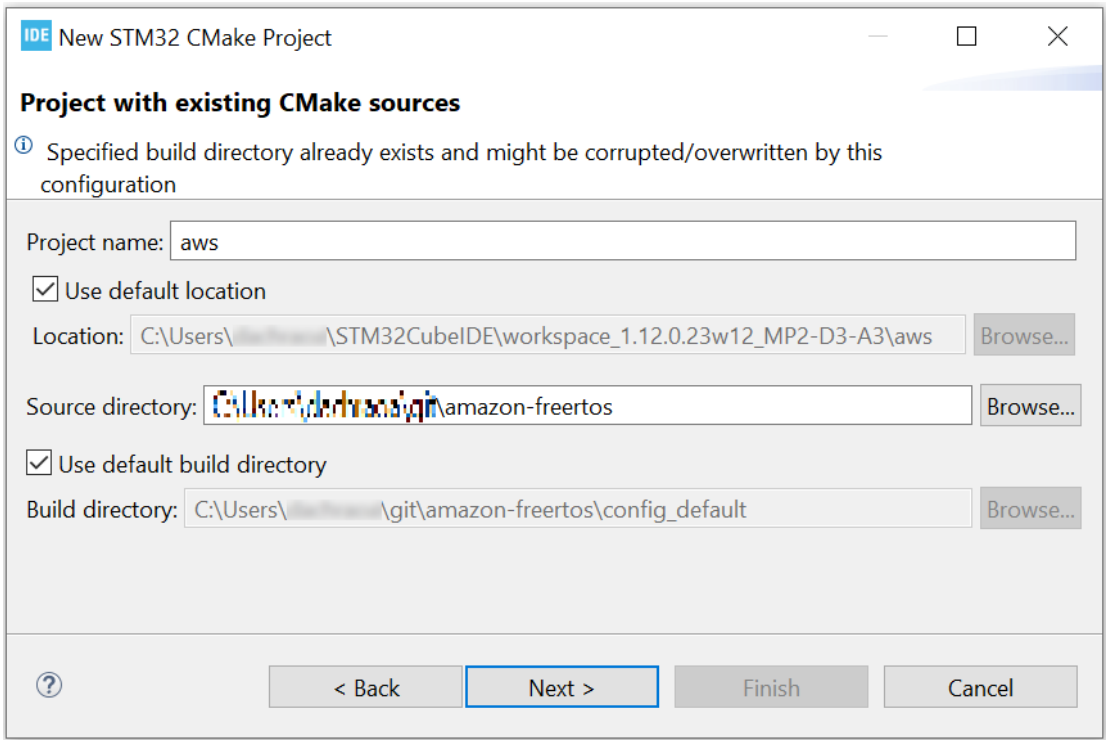
- Click on [**Next >**].

2.1.3 Creation external to an existing CMake project structure

In this other way to set up the project, it is created in the user's workspace. This is useful in the case of a complex project structure to link the existing package from the source directory to the project.

- Specify a name for the project.
- Specify an empty directory for the project. Keep [**Use default location**] checked so that the project is created inside the current workspace directory.
- Use the [**Browse…**] button and select the root directory of a CMake-based project. The selected directory is linked into the project.
- Specify the path to the build directory to be used for the CMake configuration. By default, the build directory is `config_default` relative to the CMake source directory.

Figure 5. Project creation external to an existing CMake project structure

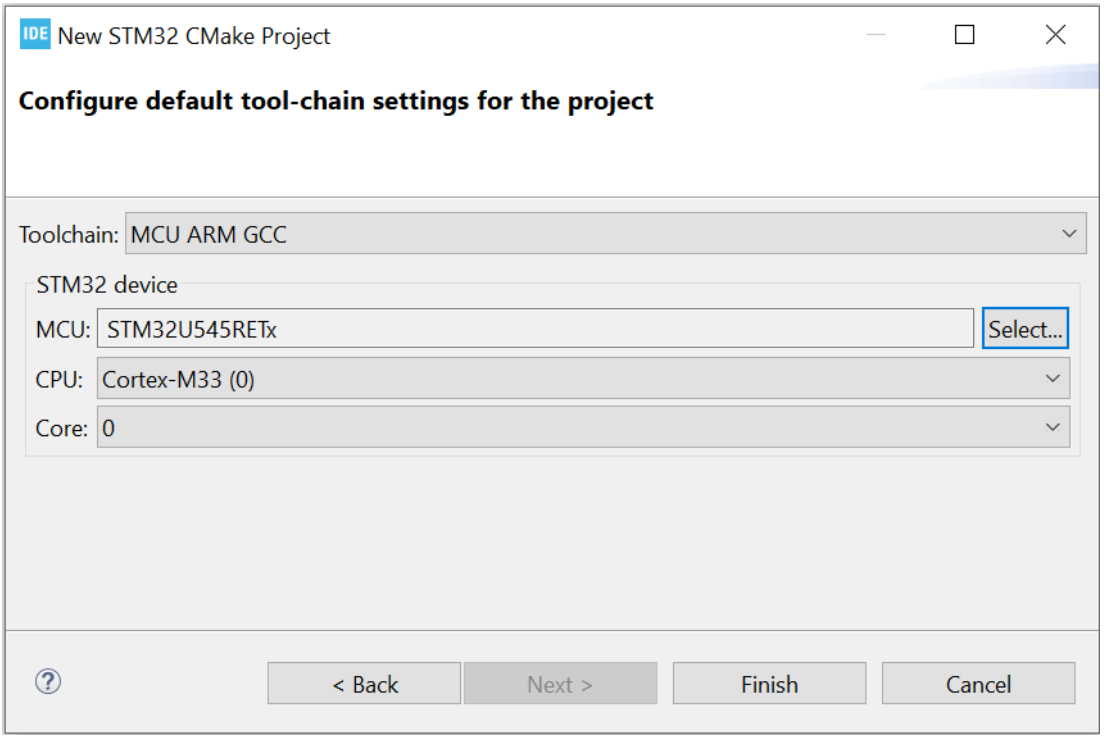


- Click on [**Next >**].

The next wizard page allows the configuration of a default toolchain and its relevant options for the created project. This is not directly relevant for building the CMake project structure but the IDE might require this information for certain of its features to function properly.

- Fill in:
  - The information about the MCU and core for the debugger
  - The proper indexing, for instance for code completion
  - The selected toolchain, which is added to the PATH variable when building the project

Figure 6. CMake default toolchain



- Select [**Finish**]

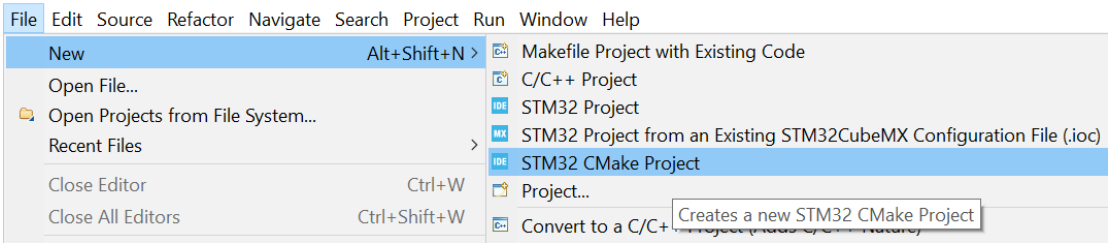
At this stage, the project is created and visible in the *Project Explorer* view.

2.2 Starting CMake project development from scratch

STM32CubeIDE also offers the possibility to create an own user's project using CMake. To do so, proceed as per the steps below:

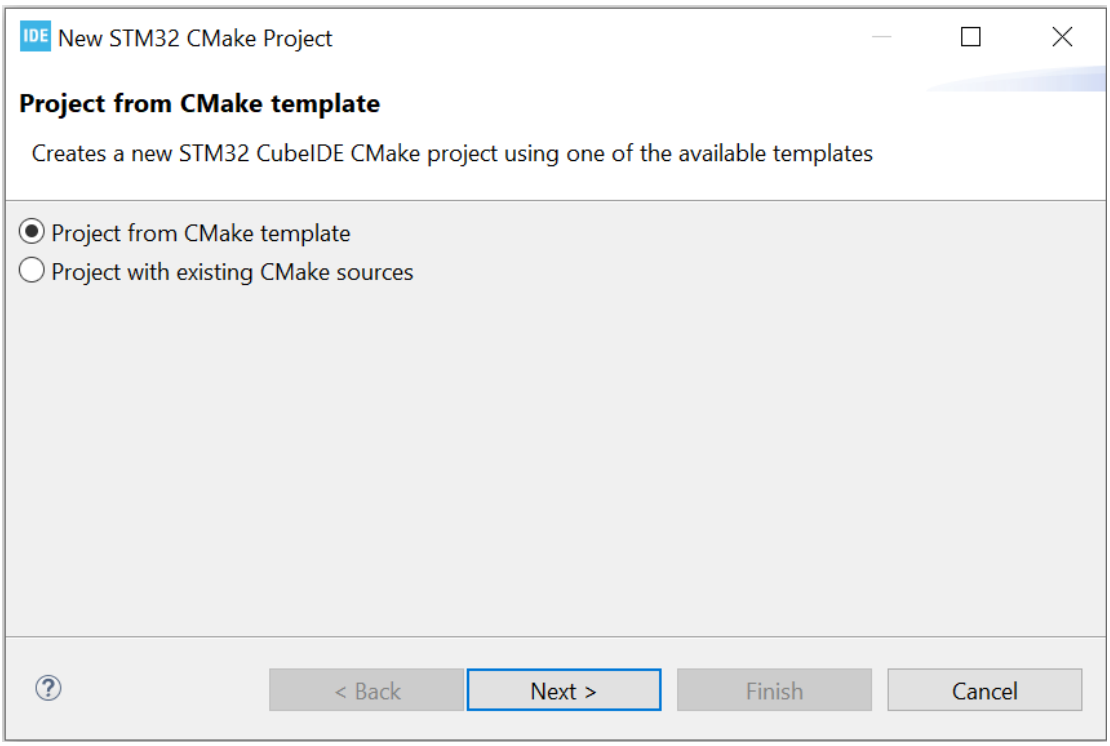
- Select [**File**]>[**New**]>[**STM32 CMake Project**]

Figure 7. CMake project creation (alt.)



- Select [**Project from CMake template**]

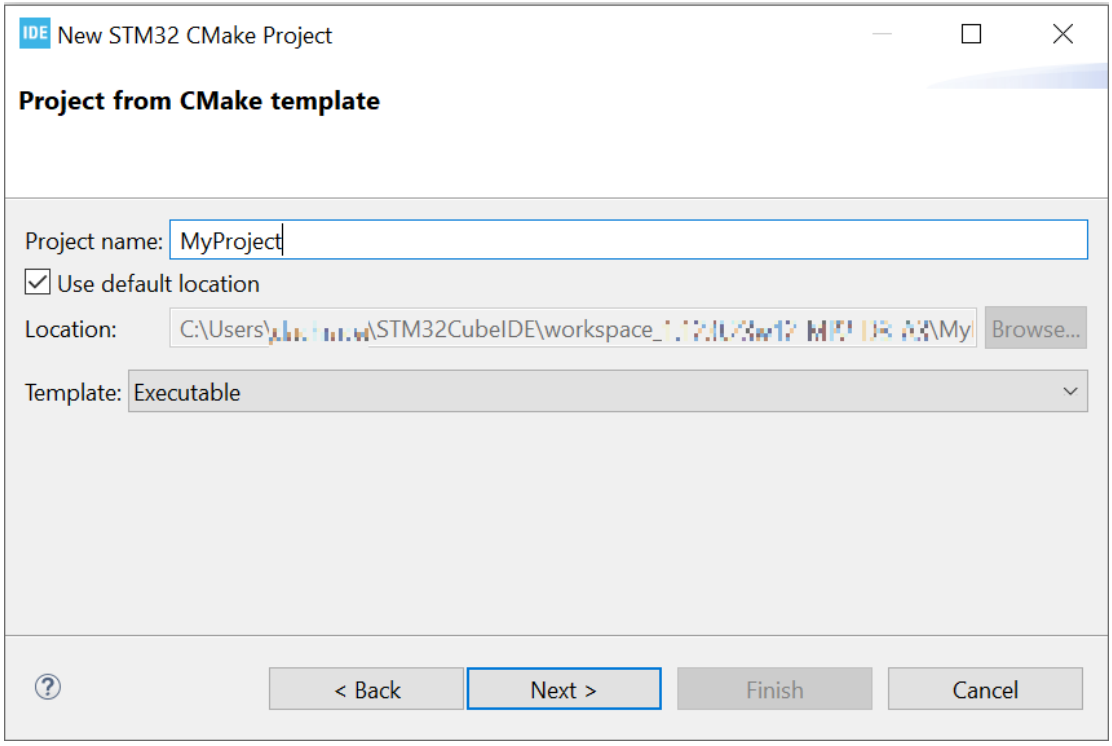
Figure 8. CMake project types (alt.)



- Select [**Next >**]

- Provide the information requested for the project creation:
  - Specify a name for the project.
  - Specify an empty directory for the project. Keep [**Use default location**] checked so that the project is created inside the current workspace directory.
  - Select the template to use for the project. Currently, the templates for *"Executable"* and *"Static Library"* targeting the *"MCU ARM GCC toolchain"* are available.

Figure 9. CMake project from template

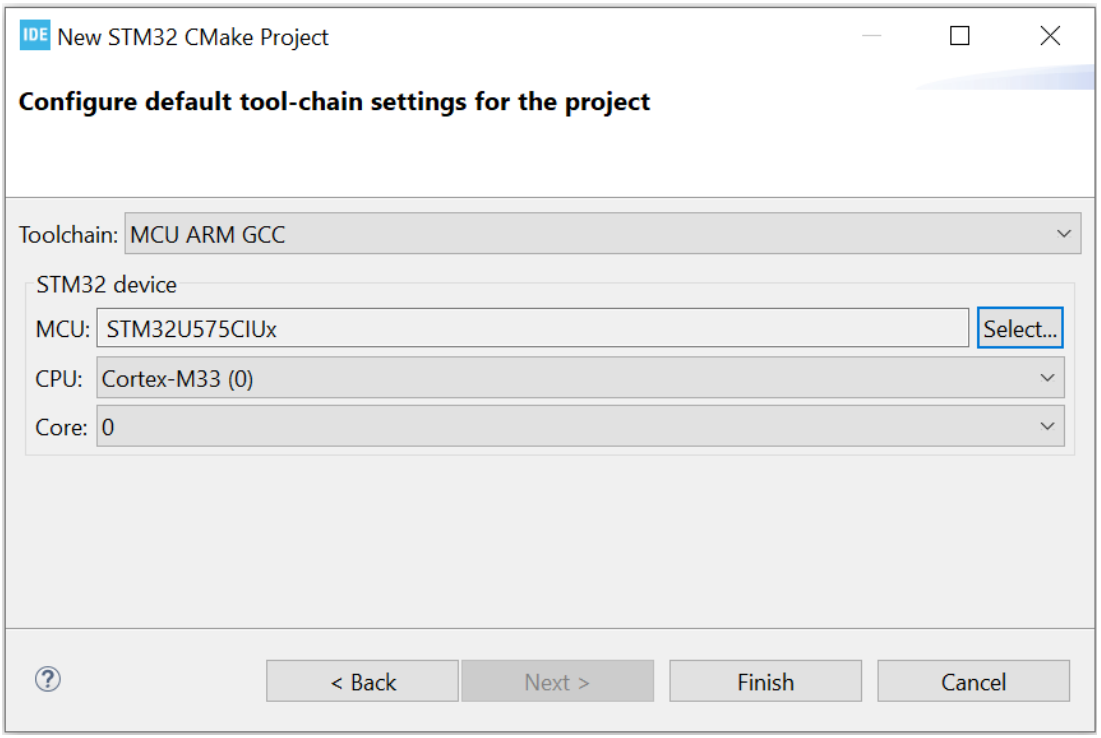


- Click on [**Next >**]

The next wizard page allows the configuration of a default toolchain and its relevant options for the created project.

- Select the options for the STM32 device to be used for the new project

Figure 10. CMake default toolchain (alt.)



- Click on [**Finish**]

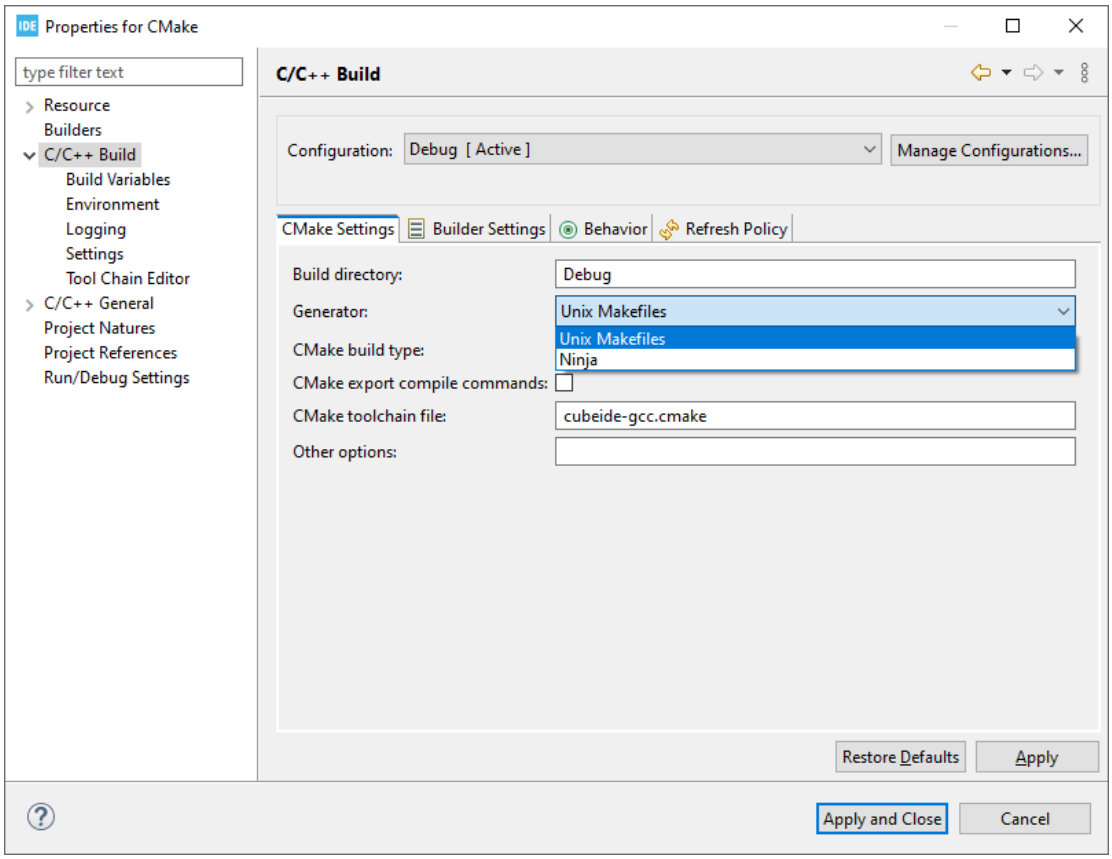
3 Configure and build

3.1 CMake build settings

The CMake-related build settings can be specified in the project properties:

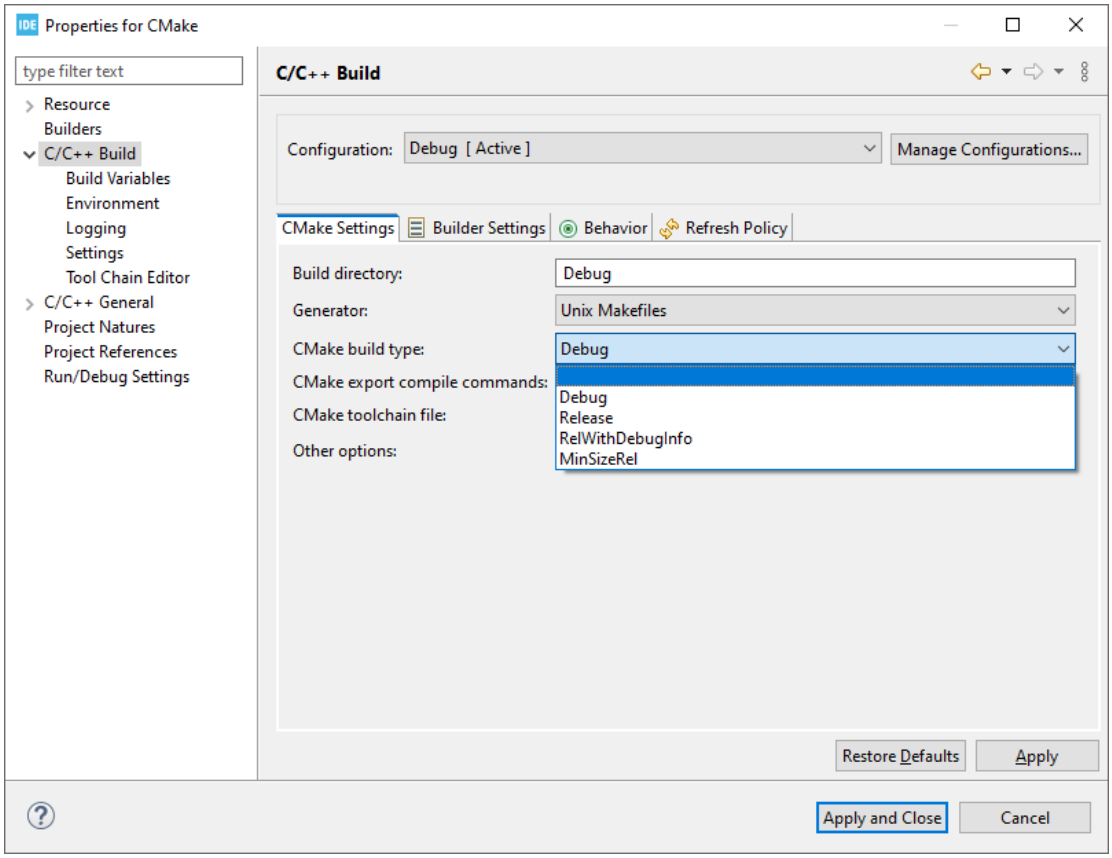
- Select *C/C++ Build* available in the project properties.
- Select the *CMake Settings* tab.
- Select the build system in [**Generator**]:
  - *"Unix Makefiles"*
  - *"Ninja"*
- Specify the initial values for the CMake configuration step in [**Other Options**]. The example presented in Figure 11 is for configuring the `amazon-freertos` CMake package for the B-L475E-IOT01A STM32 Discovery kit.

Figure 11. CMake build options (Generator)



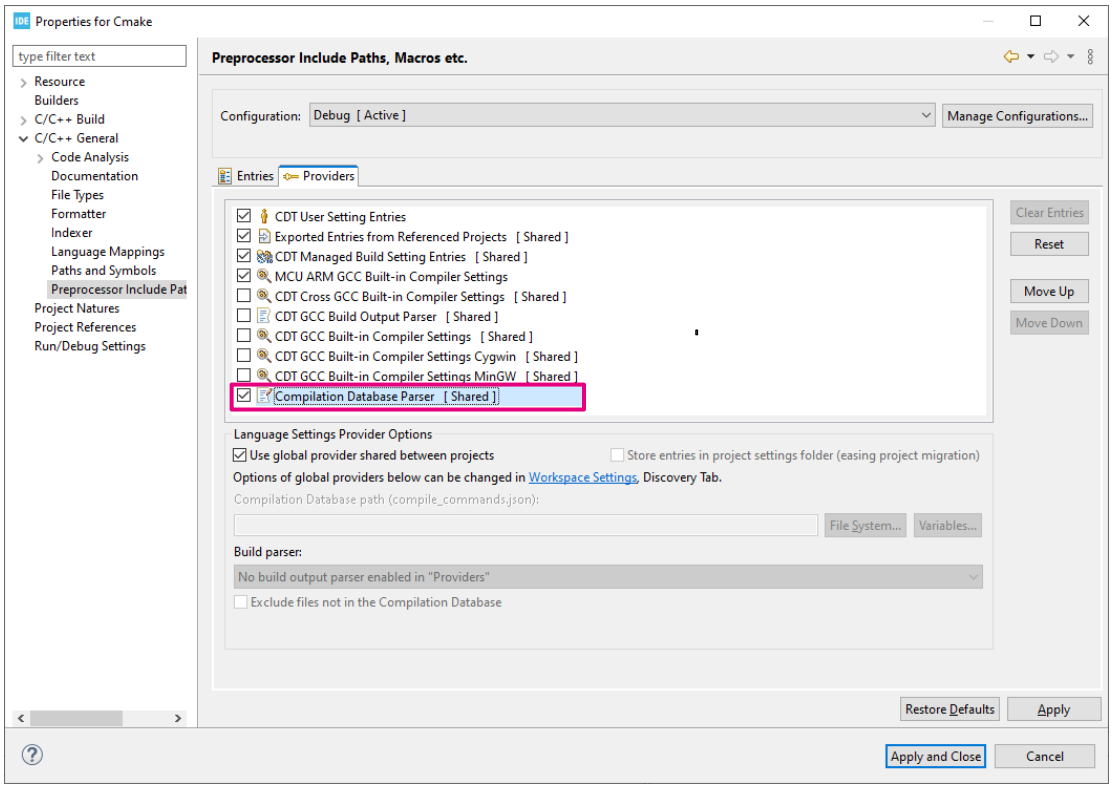
- Select the [**CMake build type**]:
  - Empty (default value)
  - *"Debug"*
  - *"Release"*
  - *"RelWithDebugInfo"* (release with debug information)
  - *"MinSizeRel"* (minimum size release)

Figure 12. CMake build options (CMake build type)



- Configure the parameters as shown in Figure 13, when working with CMake projects in STM32CubeIDE, to mitigate indexer issues resulting from CDT™ limitations.

Figure 13. Compilation database parser



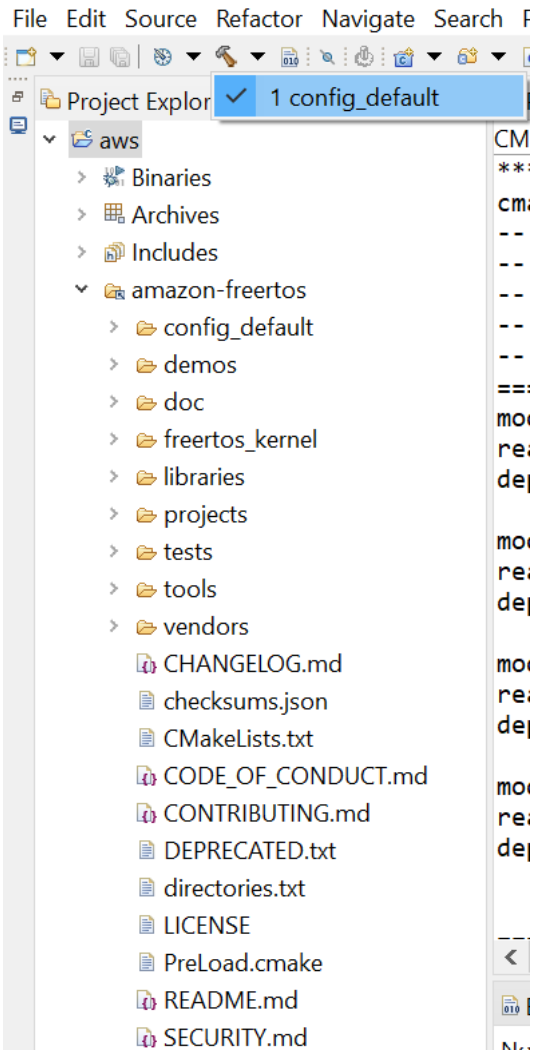
3.2 Building a CMake-based project

For CMake-based projects, the IDE build configuration management and user interfaces are similar to those for non-CMake-based projects, including:

- Menus
- Toolbars
- Buttons

Multiple project build configurations can be created and associated with different sets of CMake settings, such as *“Debug”* and *“Release”*.

Figure 14. CMake project manual build



DT72070V1

During the first project build, the IDE automatically performs:

- The CMake configure step
- The setup of the CMake cache
- The build files generation into the specified build directory

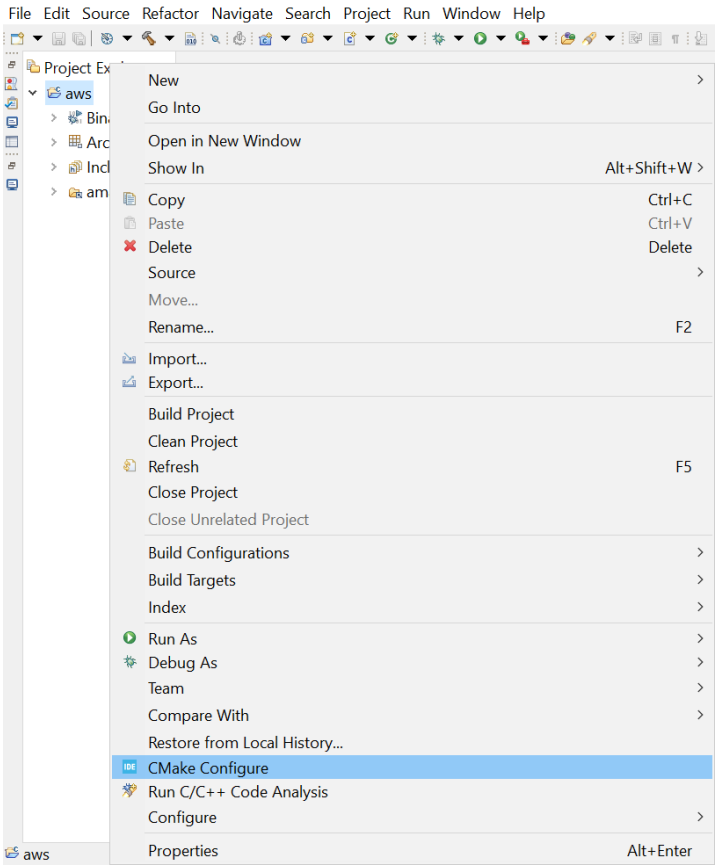
Any subsequent build is performed in the build directory using the existing CMake cache.

3.3 CMake manual configure step

The CMake configure step and build files generation can also be performed manually from:

- Either the project context menu

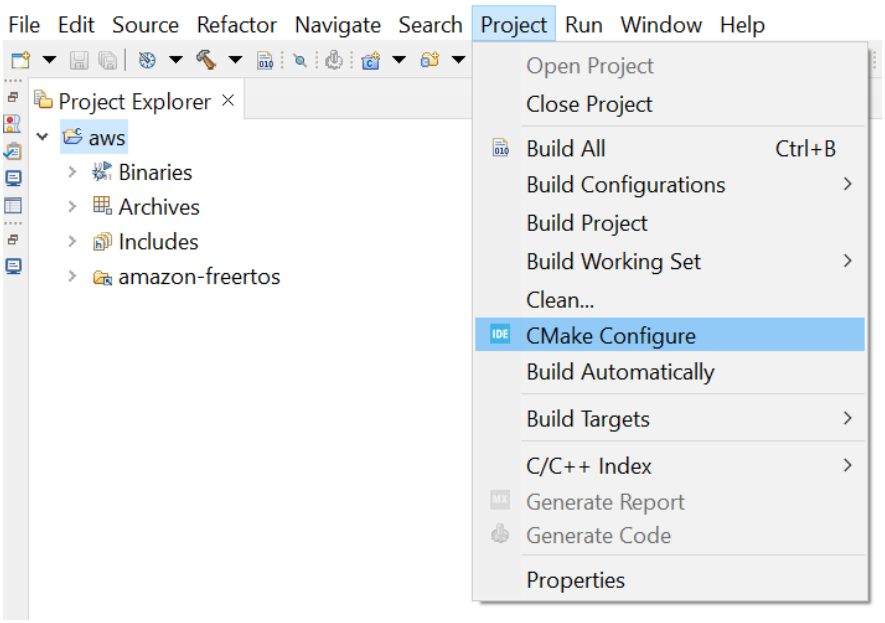
Figure 15. CMake project context menu



DT72068V1

- Or the *Project* menu

Figure 16. CMake project menu



Revision history

Table 1. **Document revision history**

Date	Revision	Changes
07-Jul-2023	1	Initial release.
19-Feb-2025	2	Updated Section 3.1: CMake build settings: • Updated Figure 11. CMake build options (Generator) • Added Figure 12. CMake build options (CMake build type) and Figure 13. Compilation database parser

Contents

List of figures