

# STM32CubeIDE

- I. Présentation.....2
- II. Téléchargement et Installation.....3
- III. Création d'un projet.....3
- IV. Intégrer des sources existantes à un projet.....4
  - IV.1. Importer les sources dans le workspace.....4
  - IV.2. Intégrer les fichiers C/C++ au processus de compilation.....5
  - IV.3. Intégrer les fichiers d'en-tête (headers) au chemins d'inclusion.....6
- V. HAL – Hardware Abstraction Layer.....7
  - V.1. Couches logicielles.....7
  - V.2. Utiliser la HAL.....8
- VI. Programmer et debugger avec STM32Cube.....9

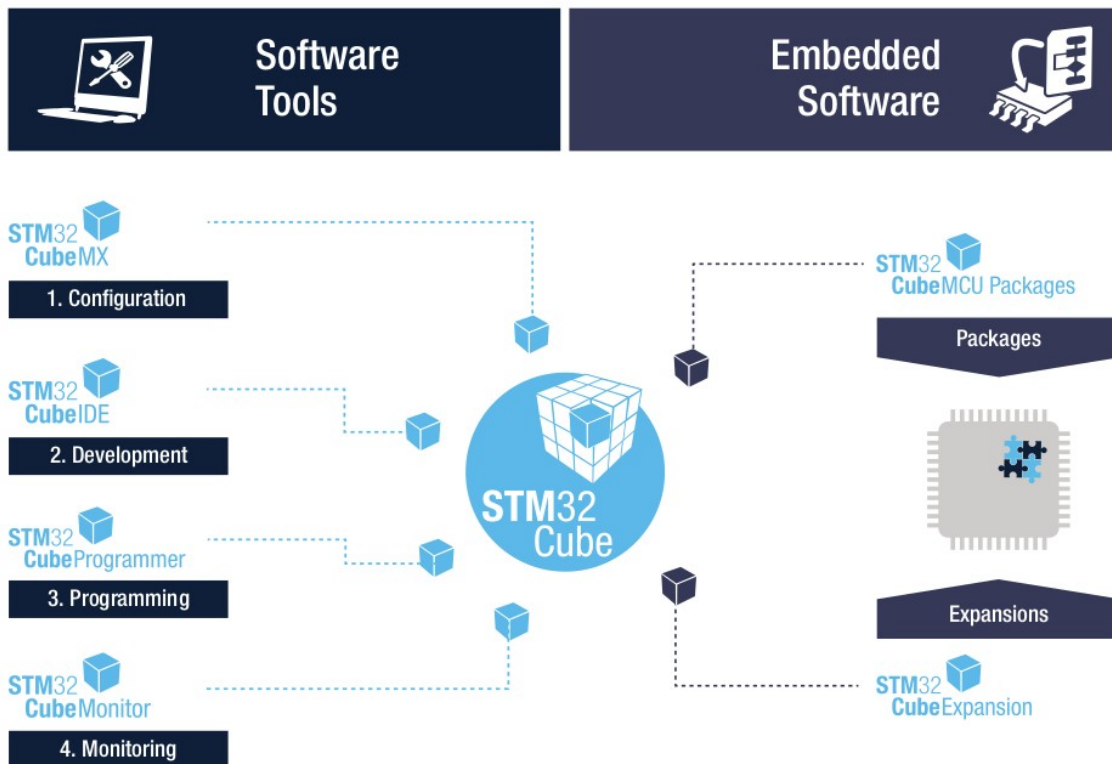


Except where otherwise noted, this work is licensed under <https://creativecommons.org/licenses/by-nc-sa/4.0/>

### I. PRESENTATION

STMicroelectronics offers a software suite to work on its own processors: **STM32Cube**<sup>1</sup>. It is an entire ecosystem that contains several software tools, even though we will focus on two of them:

- **STM32CubeIDE**, an Integrated Development Environment;
- **STM32CubeMX**, a graphical code generator tool.



The **Integrated Development Environment** (that will be called **IDE** from now on) is called **STM32CubeIDE**. It uses the Eclipse framework, which is the largest cross-platform open-free IDE. Eclipse works with perspectives, i.e. sets of windows that are configured for specific phases of the development (e.g. edit, debug).

Built into the IDE there is **STM32CubeMX**, which is a graphical configuration interface for the MCU and its internal peripherals. It gives configurations and control functions for the MCU peripherals, in few clicks and few minutes. In a professional context this tool is used to accelerate the prototyping phase and thus reduce the Time-to-market of software embedded solutions.

1 <https://www.st.com/en/ecosystems/stm32cube.html>

## II. DOWNLOAD AND SETUP


**STM32CubeIDE** can be downloaded right from the STMicroelectronics's website. It is cross-platform (Mac/Windows/Linux). This documents uses the **1.11.0 version**.

<https://www.st.com/en/development-tools/stm32cubeide.html>

Once it has been installed (no difficult step), you can open the IDE a make a a discovery tour with a 5-minute video, from the STM32CubeIDE:

Help → Tutorial Video → Discover your STM32 with SMT32CubeIDE  
→ How to use STM32CubeIDE

## III. PROJECT CREATION

1. In **STM32CubeIDE** (not STM32CubeMX): **File → New → STM32 Project**.
2. In the tab "**Board Selector**", find your board with the search bar (each board has a sticker on it), select the board and click the "**Next**" button.
  - *It is also to select pre-existing example projects provided by STMicroelectronics.*
3. Fill the project's name<sup>2</sup>, choose its location<sup>3</sup> and let the options by default (**C / Executable / STM32Cube**), and click on **Finish**.
4. "**Initialize all peripherals with their default Mode ?**"  
→ "**Yes**" or "**No**", depending on the lab's requirements!
5. The graphical configuration window opens. It is an **\*.ioc** file provided by STM32CubeMX. In the "**Pinout & Configuration**" tab, configure the peripherals **according to the application requirements**.
6. After the configuration: **Project → Generate Code** (or the  icon) to generate the files and functions corresponding to the peripheral configurations.

The project has been created and the configuration instructions have been written (startup, clock configurations, interrupts, ...). The peripherals that ave been configured in the previous step are now ready to use. To do so, see section "V HAL – Hardware Abstraction Layer" page 7.

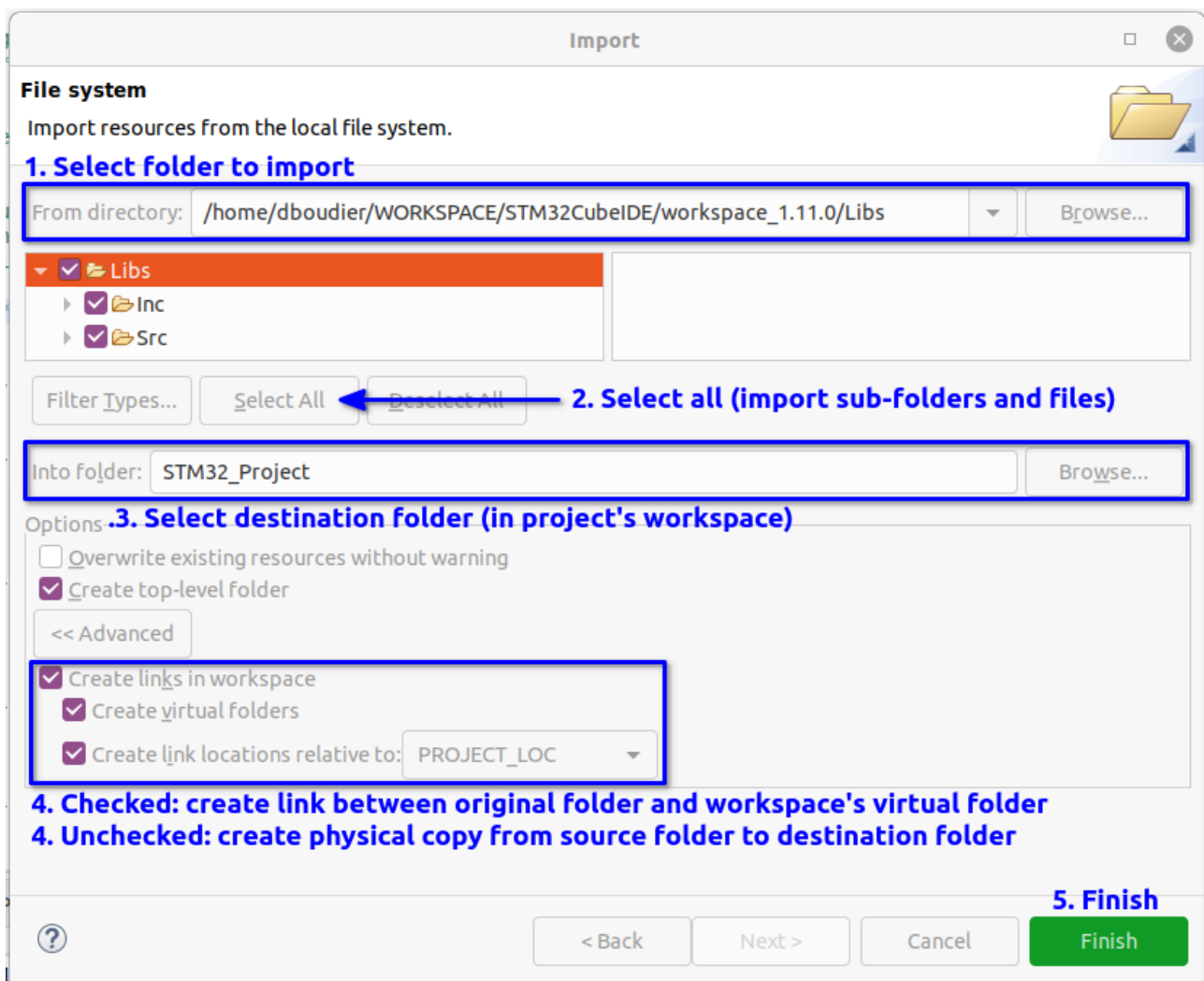
**Warning:** After creating a project with STM32CubeMX (\*.ioc configuration file), you must code between two matching comment tags.  
When modifying the MCU configuration, any code re-generation will erase any code line that is outside these tags.

<sup>2</sup> Project name with no accent, no special character, no space, ...  
<sup>3</sup> Project location must be a short path, with no special character.

### IV. ADD EXISTING SOURCES TO THE PROJECT

#### IV.1. Import sources into the workspace

- To add existing sources (files or folders) to a project, you can simply copy-paste them into an already existing project folder, using your OS's file browser.
- You can also add files from within the IDE, from the *Project Explorer* (left tab of the IDE):
  - Right-click on the project → **Import...** → **General** → **File System** → **Next**
  - Follow the configuration as shown below:




The imported folder (or imported file) should now appear in the project's tree-view, in the *Project Explorer* tab.

### Note

Importing a folder to a project has for sole effect to make its content (source and header files) reachable from the *Project Explorer* tab, to make it easier to read or edit those files.

This means that the imported files are not directly ready to be used by the toolchain. Depending on the file type (C/C++ sources or header files), two different operations must be performed. They are discussed in the next few pages.

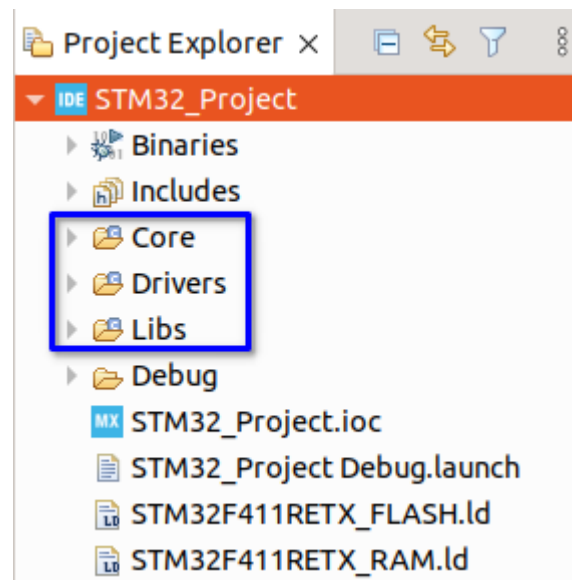
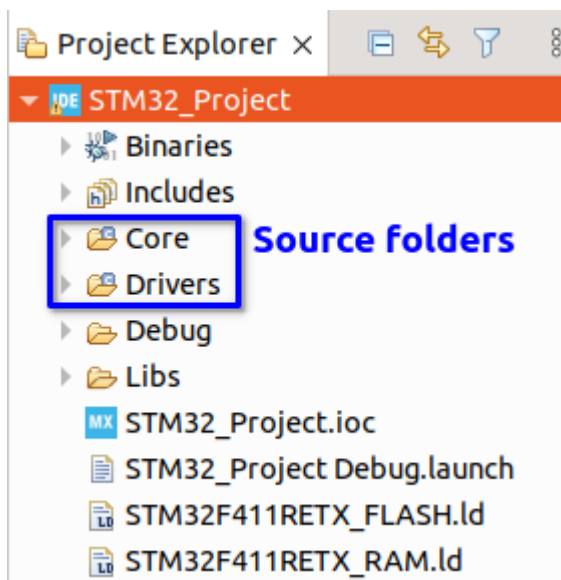
## IV.2. Add C/C++ sources files to the compilation process

To be considered among sources files to be compiled, imported files must be in a “**Source Folder**” recognizable with its  icon. When a folder has been imported with the method described before, it is a ordinary folder. Converting it as a **Source Folder** will add its files to the sources to be compiled.

Convert a folder into a **Source Folder**.

Right-click on the project → **New** → **Source Folder** → Fill in the **Folder Name** field or click on **Browse** to find it → **Finish**.

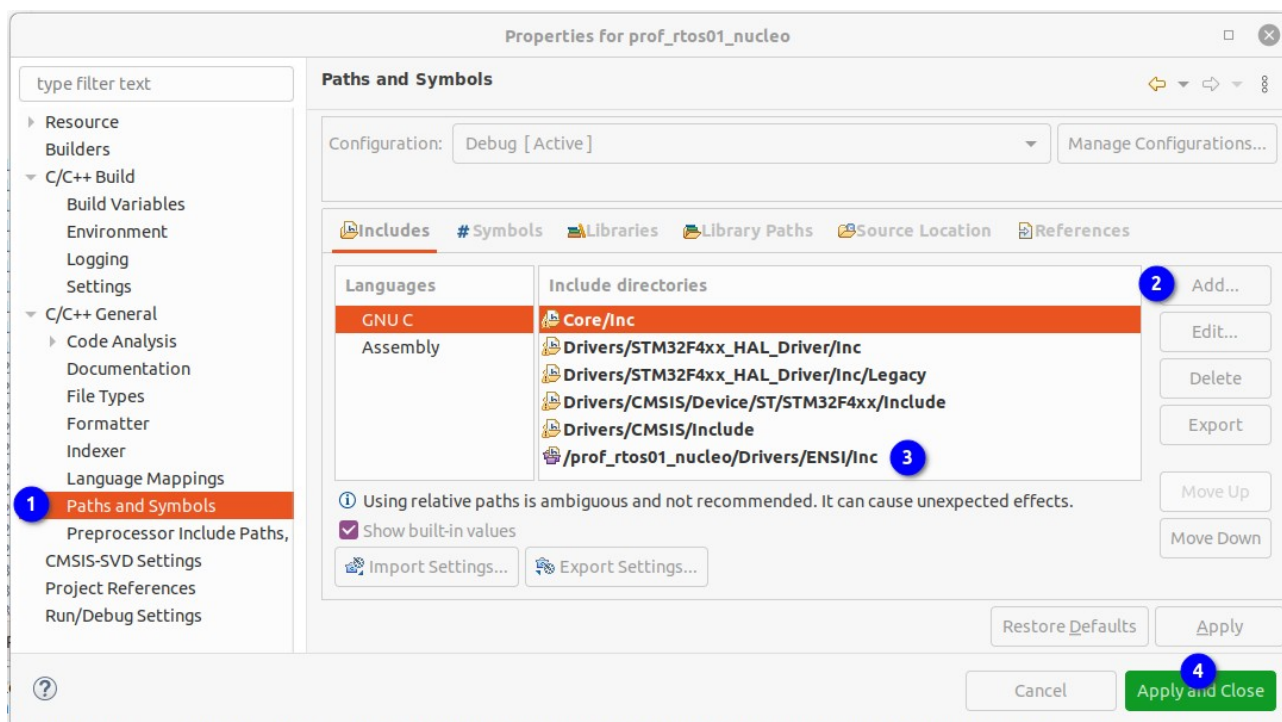
In the example figure below, we can see that the **Libs** changed from an ordinary folder to a Source folder.



### IV.3. Add header files to include paths

Such as sources files, headers files must be reachable for the toolchain. If these files are not in an already referenced directory, then the new **include paths** must be provided to the toolchain (for the GCC toolchain, this is the **-I** option).

0. Right-click on the project → **Properties**
1. **C/C++ General** → **Paths and Symbols**
2. **Add**, then fill in the headers directories
  - *Note : if the directory has been added using a symbolic link (and not by a physical copy-paste), then you must fill in the original directories (not the links).*
3. Number three on the figure below shows a manually added include directory, just below all directories set by STM32CubeIDE.
4. Apply and close



*Note: the include paths must also be set for the assembler stage of the toolchain if any assembly file uses one of the imported headers.*

#### Advice

Those import and includes steps are a common source of errors. Do not hesitate to import and fill in include paths one at a time, compiling after each one to analyze the toolchain error messages.

### V. HAL – HARDWARE ABSTRACTION LAYER

#### V.1. Software layers

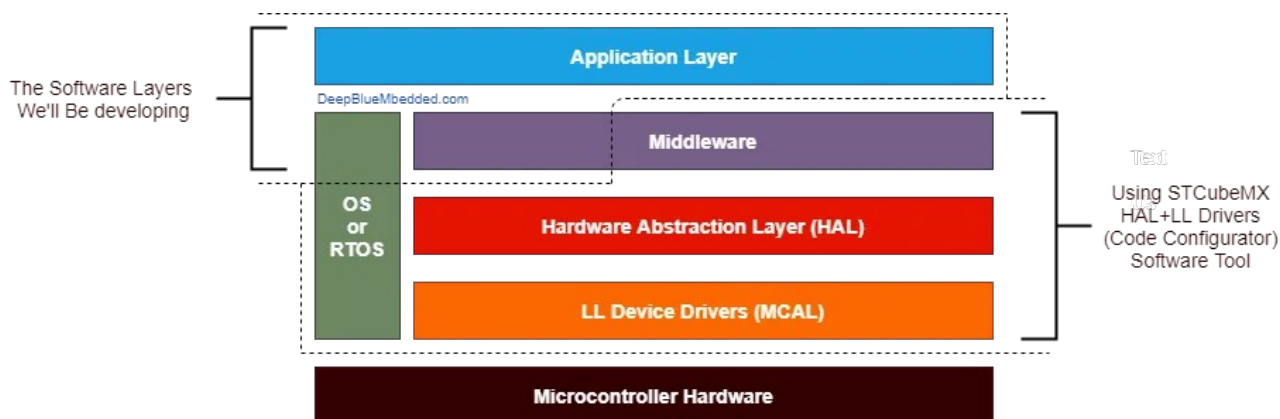
The “Embedded Systems” labs in first year discussed the low-level programming (at register level) of a Microchip PIC18 MCU. The first aim was to develop a BSP (*Board Support Package*), and the second one was to develop an application (Bluetooth speaker) using this BSP.

Reproducing this work on a STM32 microcontroller unit would be much longer due to the ARM Cortex-M complexity, as compared to a PIC18. We will use a BSP equivalent for the STM32: the HAL and the LL. The main reason is that all this hardware support is freely provided by STM32CubeMX (see chapter « III Project creation »).

The **HAL (Hardware Abstraction Layer)** is an **API (Application Programming Interface)**. To this end it supplies a functions set to the developer, with an aim of being high-level and portable (i.e. to all STM32 MCUs). Consequently any application written with HAL functions will be adaptable to any STM32, from the L0 series to the F4 series, with no need of re-working the code (as long as the hardware capabilities are the same). The HAL covers all of the STM32 peripherals.

The **LL (Low-Layer)** is also an API, which takes place at the register-level. It is thus lighter and faster than the HAL, but it is available only for few peripherals and is not necessarily portable from an STM32 to another. It mainly used for performance optimization.

Both APIs are offered by STMicroelectronics in order to comply with the MISRA-C standard, which leads C programming in the automobile engineering. Note that using the HAL or the LL allows to reduce the development time by a lot, but a skilled developer can modify or even use a peripheral with no HAL/LL in order to optimize some peripherals’ operation.



<https://deepbluembedded.com/adding-ecual-drivers-to-your-stm32-project-configurations-options/>

<https://deepbluembedded.com/stm32-hal-library-tutorial-examples/>

On the figure above, we can see that the Middleware can partly be generated by STM32CubeMX, just like the RTOS (Real-Time Operating System). However the application still remains the developer’s work as it should fit the project requirements.

### V.2. Use the HAL

Let us have a look at this extract of the “User Manual UM1749 – Description of STM32L0 HAL and Low Layer drivers”<sup>4</sup>:

#### 22.2.3 Initialization and de-initialization functions

This section contains the following APIs:

- [HAL\\_GPIO\\_Init\(\)](#)
- [HAL\\_GPIO\\_DeInit\(\)](#)

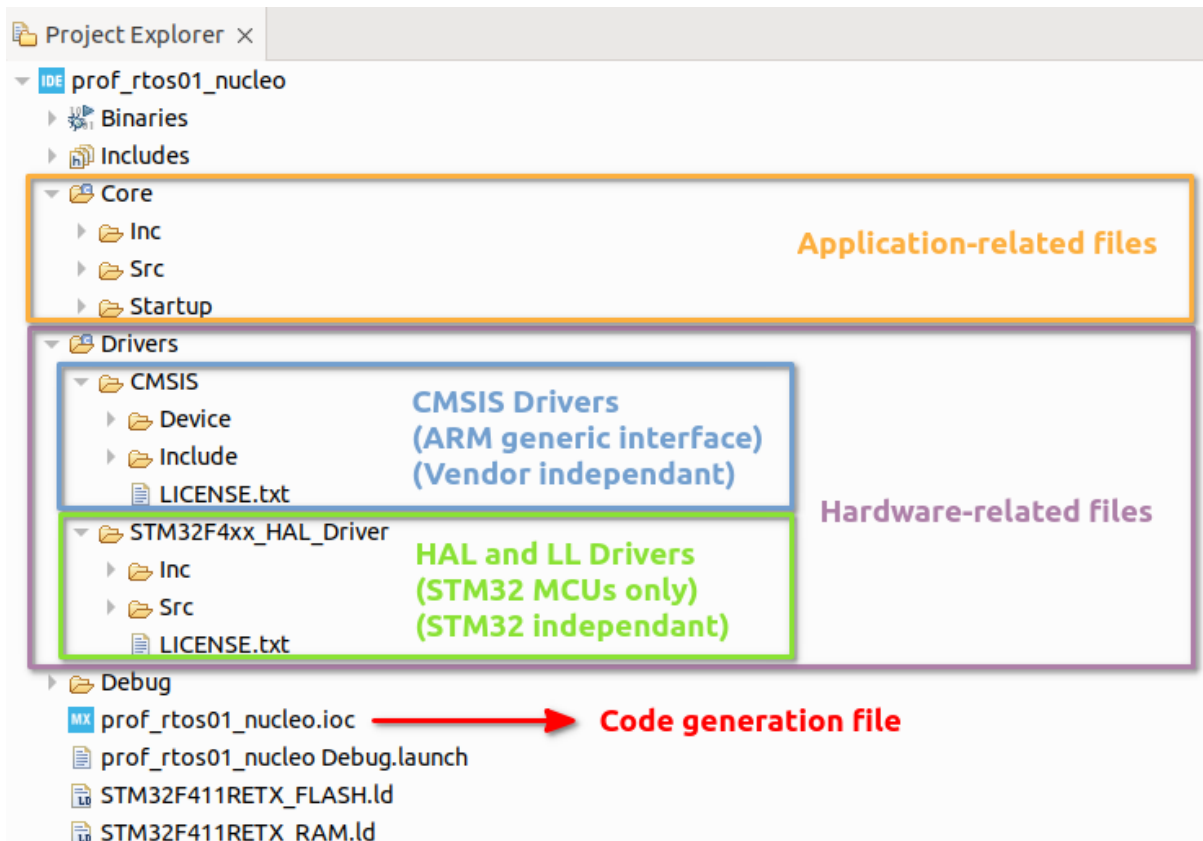
#### 22.2.4 IO operation functions

This section contains the following APIs:

- [HAL\\_GPIO\\_ReadPin\(\)](#)
- [HAL\\_GPIO\\_WritePin\(\)](#)
- [HAL\\_GPIO\\_TogglePin\(\)](#)
- [HAL\\_GPIO\\_LockPin\(\)](#)
- [HAL\\_GPIO\\_EXTI\\_IRQHandler\(\)](#)
- [HAL\\_GPIO\\_EXTI\\_Callback\(\)](#)

As long as at least one GPIO is initialized with STM32CubeMX, all of those functions will be defined and provided in the project files. Similar functions are defined for the UART, the timer, ... and any other configured peripheral.

In the project tree view, all the HAL functions are declared and defined in the directory `<Project>/Drivers/STM32xxxx_HAL_Driver`. Browsing those files, you can discover all the available functions and how to use them. You will then be able to call them from your application file.



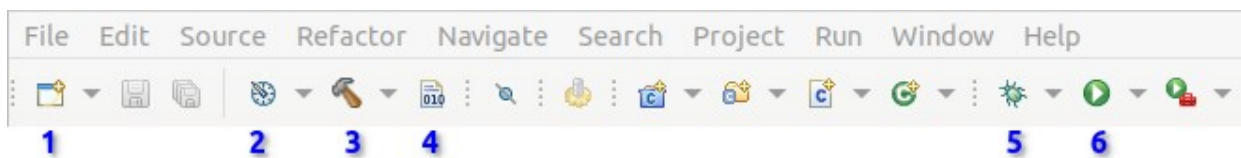
<sup>4</sup> <https://www.st.com/en/embedded-software/stm32cubel0.html#documentation>



## VI. PROGRAM AND DEBUG WITH STM32CUBE

### *In short*

Let's keep it simple: the buttons below are the most commonly used in STM32CubeIDE. Among them, the top two consist of the Build all button (compilation) and debug mode button.



1. New project (File → New → STM32 Project) [Alt+Shift+N]
2. Configuration (Debug mode / Release mode)
3. Build current configuration of current project
4. **Build all (Project → Build All) [Ctrl+B]**
5. **Debug current project (Run → Debug) [F11]**
6. Run current project (Run → Run) (upload firmware onto the target and debug).

### *Debugger*

The first time the debugger is launched, the « Edit Configuration » window opens up. Click on the OK button. The IDE will change its perspective (the windows and menus change). It is possible to come back to the edit perspective by stopping the debug mode.



1. Reset the debug session of the target MCU.
2. Make the debugger stop/skip the breakpoints
3. Stop and restart
4. Start / resume [F8]
5. Pause
6. Stop (stop the debug session, go back to the Edit mode)
7. Step into: go to the next instruction
8. Step over: go to the next line
9. Step return: resume until the current function returns.