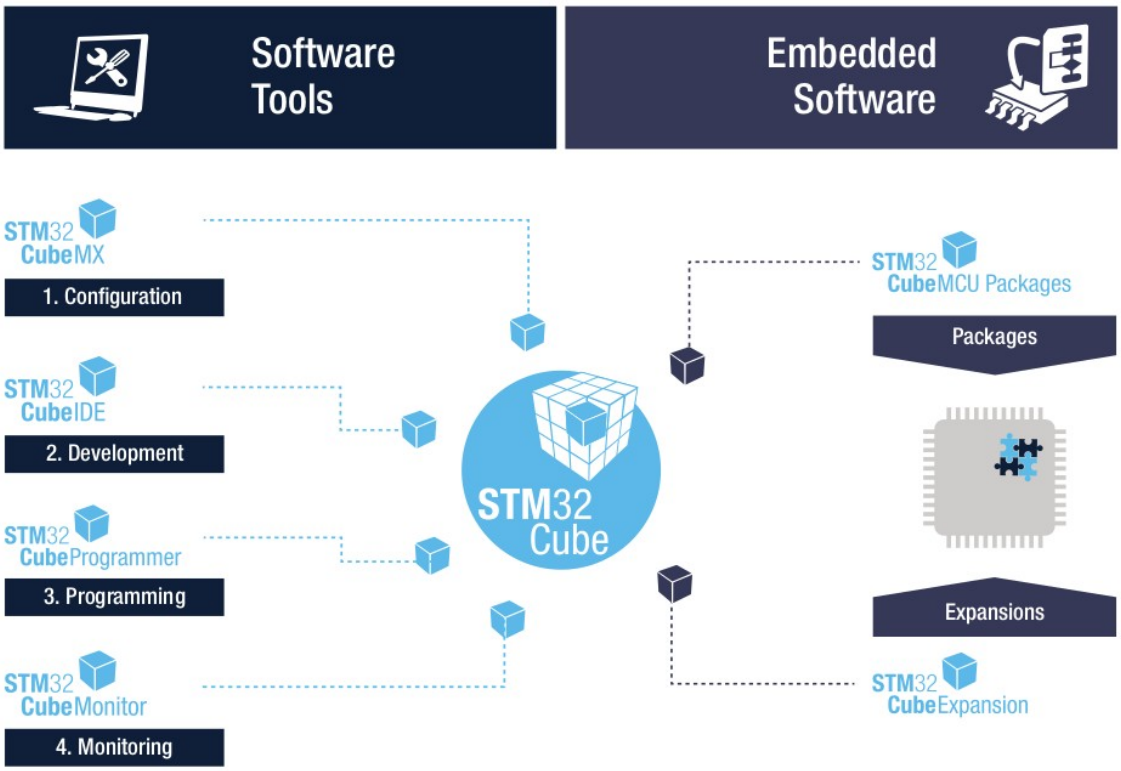
STM32CubeIDE

I. Presentation

STMicroelectonics offers a software suite to work on its own processors: **STM32Cube**[1]. 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-exisiting example projects provided by STMicroelectronics.*
3. Fill the project's name[2], choose its location and let the options by default (`C / Executable / STM32Cube`), and click on `Finish`.
4. « `Initalize 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 configurated in the previous step are now ready to use. To do so, see section "V HAL – Hardware Abstraction Layer" page 7.

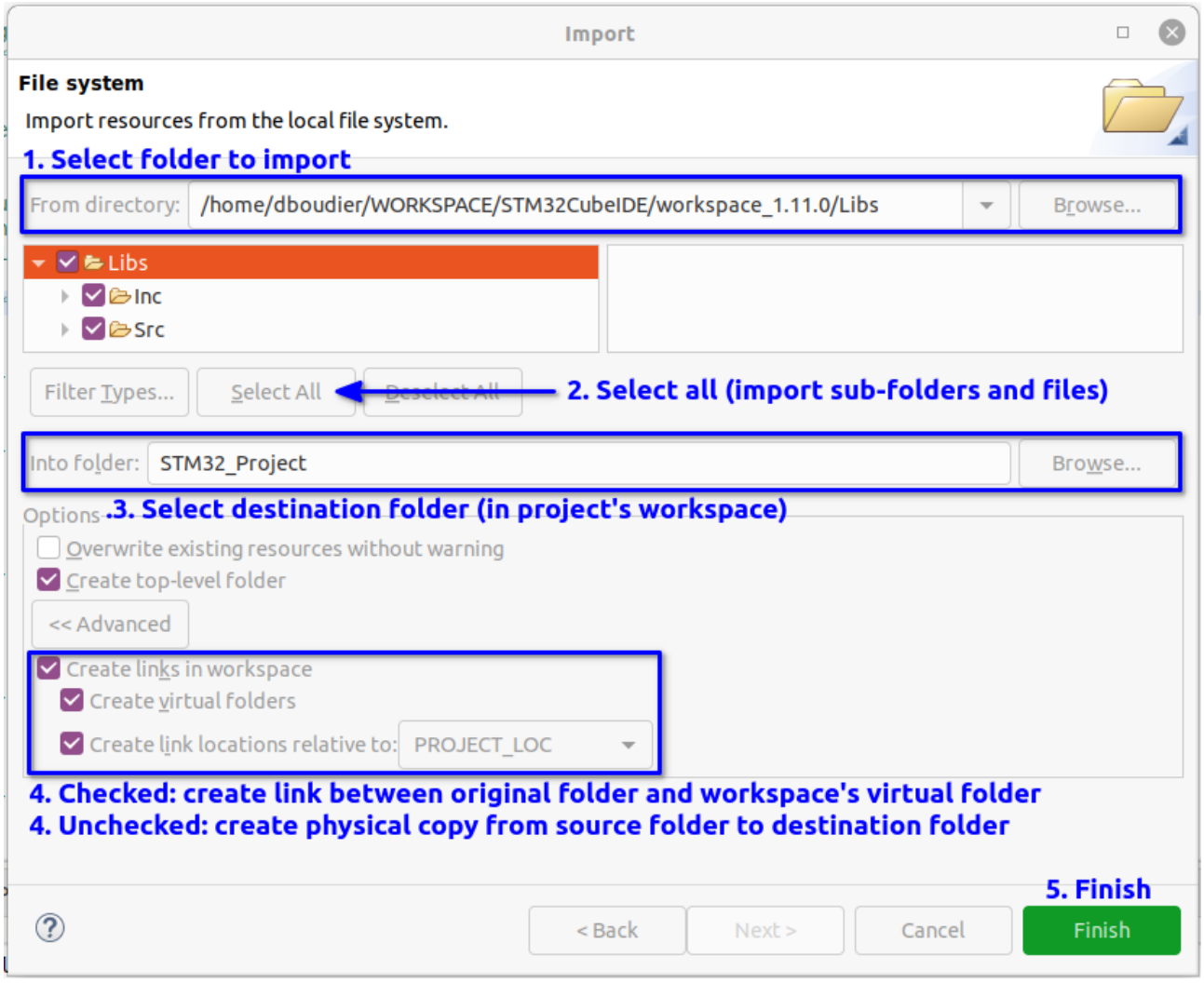
> **Warning:** After creating a project with STM32CubeMX (*.ioc configuration file), you must code between two matching comment tags..
> When modifying the MCU configuration, any code re-generation will erase any code line that is outside these tags.

2 Project name with no accent, no special character, no space, …

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 you 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 if 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.

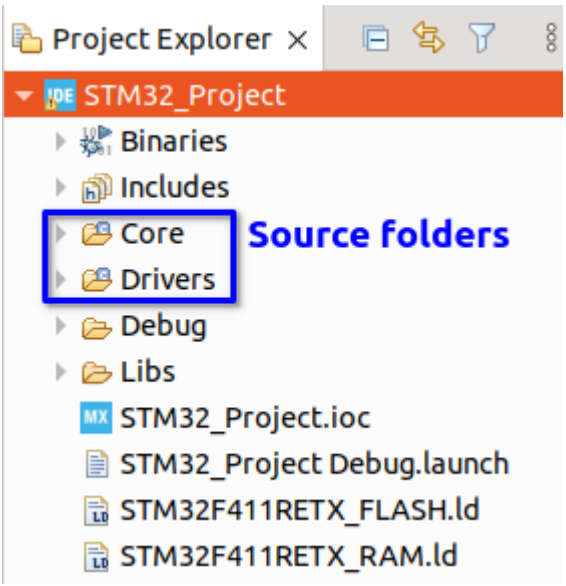
IV.2. Intégrer les fichiers C/C++ au processus de compilation

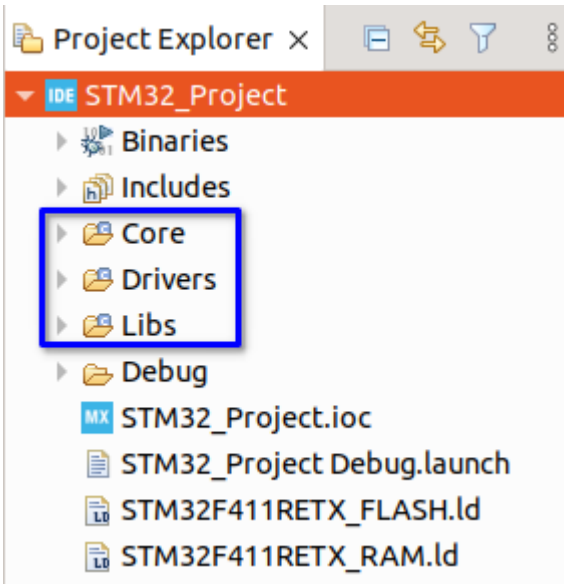
Pour être considéré comme un fichier source à compiler, le-dit fichier doit faire partie d'un « ***Source Folder*** », reconnaissable à son icône . Or par défaut un dossier importé avec la méthode précédente n'est qu'un dossier ordinaire. Mais en convertissant un dossier quelconque en ***Source Folder***, les fichiers C/C++ qui en font partie sont automatiquement considérés comme sources à compiler.

Pour convertir un dossier en ***Source Folder*** :

Clic droit sur le projet → `New` → `Source Folder` → Renseigner le champ `Folder Name` ou cliquer sur `Browse` pour le trouver → `Finish`.

On observe la transformation de l'icône du répertoire `Libs`, importé au projet dans l'étape-exemple précédente.

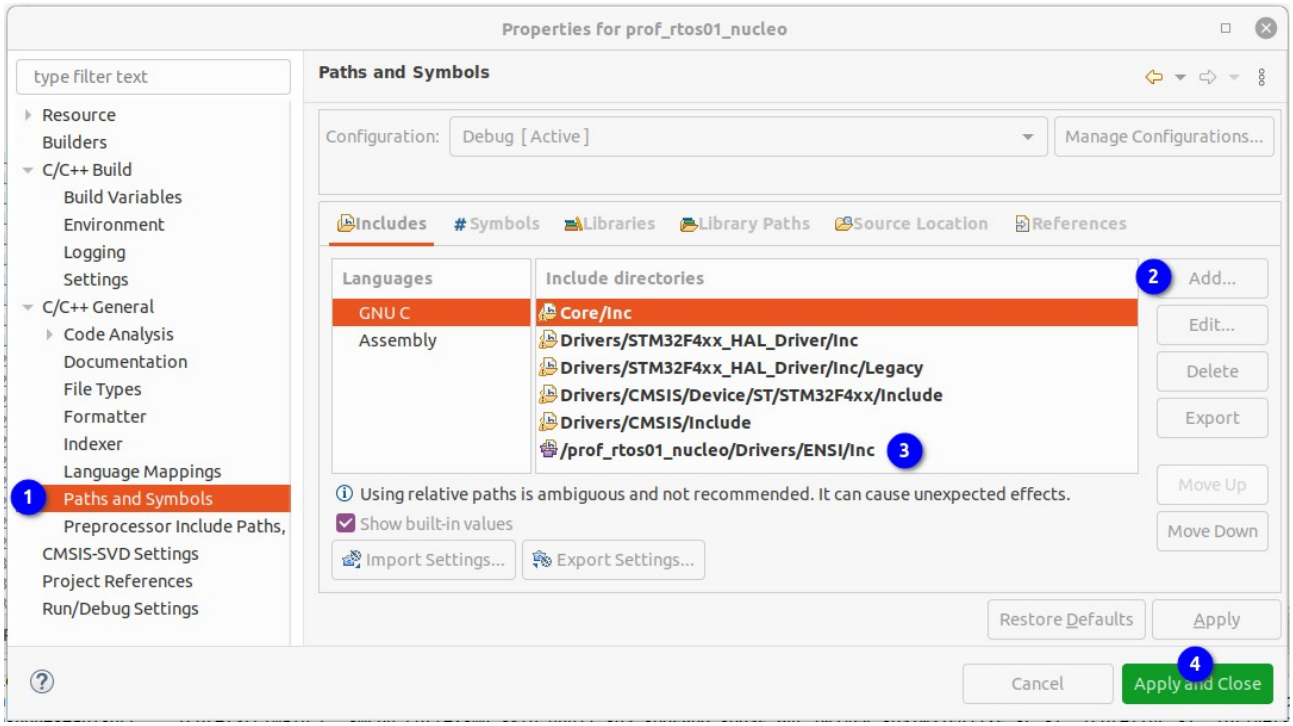




IV.3. Intégrer les fichiers d'en-tête (headers) au chemins d'inclusion

Comme pour les fichiers sources, les *headers* doivent être connus de la chaîne de compilation (*toolchain*). Si ces fichiers ne sont pas dans un répertoire déjà référencé, alors il faut préciser à la chaîne de compilation les nouveaux chemins d'inclusion (pour la *toolchain* GCC, il s'agit de l'option `-I`).

0. Clic droit sur projet → `Properties`
1. `C/C++ General → Paths and Symbols`
2. `Add`, puis renseigner le(s) répertoire(s) contenant les *headers*
   - *Note : si le répertoire a été ajouté par lien symbolique (et non par copie physique des fichiers) alors il faut renseigner le répertoire d'origine.*
3. Exemple de chemins d'inclusion fournis par STM32CubeIDE à la création du projet et d'un chemin d'inclusion ajouté à la main.
4. Appliquer et fermer



Note : les chemins d'inclusion doivent également être spécifiés pour les fichiers assembleur, si ceux-ci utilisent les headers nouvellement intégrés.

Conseil
`Ces étapes d'importation et d'inclusion sont souvent sources d'erreurs.`
`Ne pas hésiter à` importer et spécifier les chemins d'inclusion un à un, en compilant régulièrement le projet pour analyser les messages d'erreurs de la *toolchain*.

V. HAL – Hardware Abstraction Layer

V.1. Couches logicielles

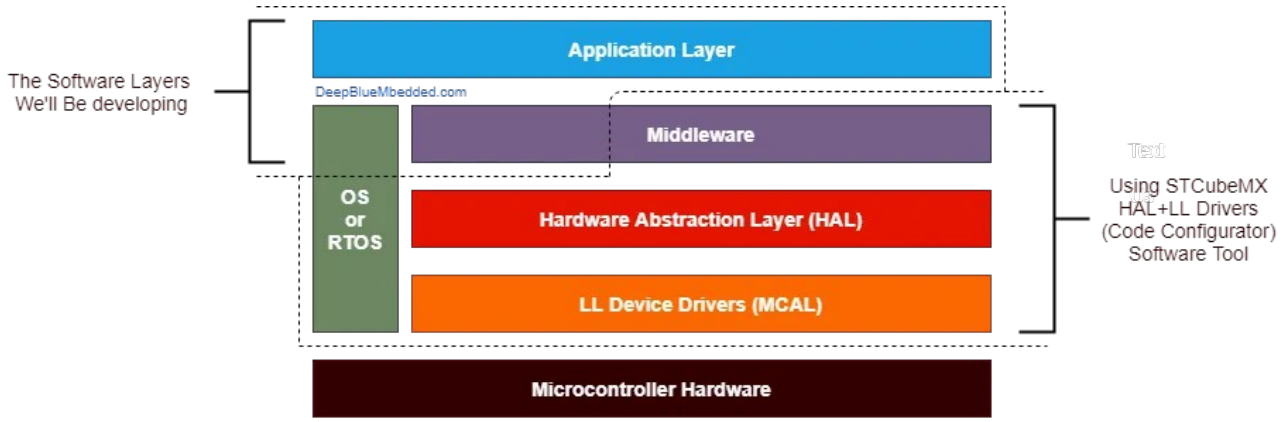
Les TP Systèmes Embarqués de première année portaient sur la programmation bas niveau (à l'étage registre) d'un micro-contrôleur (Microchip PIC18), avec pour premier objectif de produire un BSP (*Board Support Package*), puis en second objectif de développer une application (enceinte Bluetooth) en utilisant ce BSP.

Reproduire ce travail en TP RTOS serait encore plus long, étant donné la complexité d'un ARM Cortex-M en comparaison à un PIC18. Nous allons donc utiliser l'équivalent du BSP pour STM32 : la HAL et la LL. D'autant plus que ceci nous est gracieusement fourni par STM32CubeMX (cf. chapitre « III Project creation »).

La **HAL (*Hardware Abstraction Layer*)** est une **API (*Application Programming Interface*)**. À ce titre elle fourni un jeu de fonctions au développeur, fonctions ayant pour principal objectif d'être haut-niveau et portables (entre les MCU STM32). Ainsi une application utilisant la HAL pourra tourner sur n'importe quel STM32, du L0 au F4, sans ré-écriture du code (sous réserve d'avoir le même matériel). La HAL couvre tous les périphériques matériels des STM32.

La **LL (Low-Layer)** est également une API, mais orientée registres. Elle est donc plus légère et rapide que la HAL, mais n'est valable que pour certains périphériques et n'est pas forcément portable d'un STM32 à l'autre. Elle est donc utilisée à titre d'optimisation des performances.

Ces deux API sont proposées par STMicroelectonics afin de répondre à la norme MISRA-C, norme de programmation en C pour le monde de l'automobile. Notons que l'utilisation de la HAL ou de la LL permet de grandement réduire le temps de développement, mais un développeur aguerri peut modifier ou passer outre certaines fonctions afin d'optimiser le fonctionnement de certains périphériques.



thttps://deepbluembedded.com/adding-ecual-drivers-to-your-stm32-project-configurations-options/

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

Sur la figure précédente, on peut noter que le Middleware peut en partie être généré par STM32CubeMX, tout comme le RTOS. L'application reste quant à elle entièrement entre les mains du développeur.

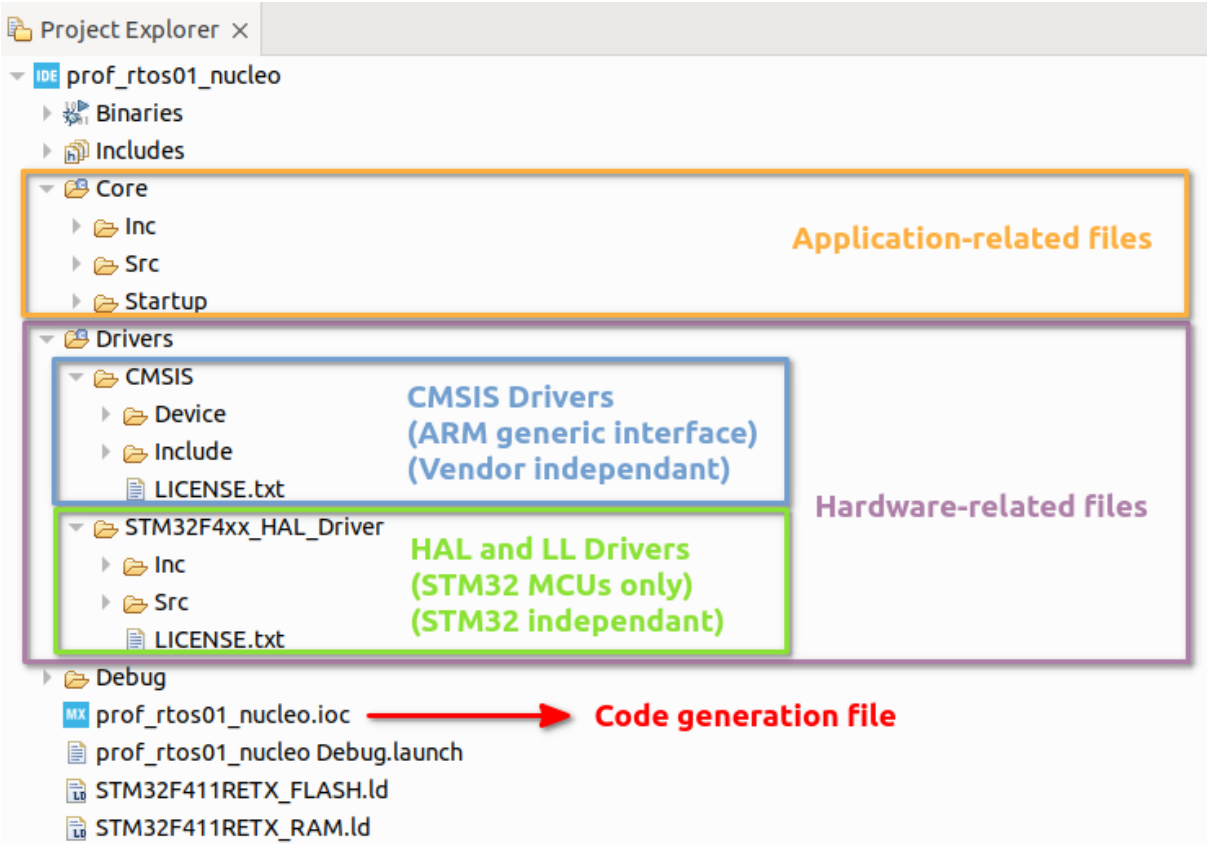
V.2. Utiliser la HAL

Prenons en exemple un extrait du « User Manual UM1749 – Description of STM32L0 HAL and Low Layer drivers »[3] :

> **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()*

À partir du moment où au moins une GPIO est configurée avec STM32CubeMX, alors toutes ces fonctions sont définies et fournies dans les fichiers du projet. Ainsi, des fonctions similaires sont définies pour l'UART, timer, … et tout autre périphérique configuré.

Dans l'arborescence du projet, toutes les fonctions de la HAL se trouvent déclarées et définies dans le répertoire `<Project>/Drivers/STM32xxxx_HAL_Driver`. En parcourant les fichiers, il reste à regarder les fonctions disponibles et comment s'en servir pour les appeler depuis les fichiers sources de l'application.

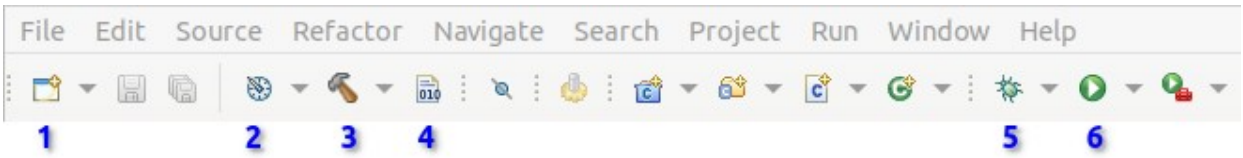


3 https://www.st.com/en/embedded-software/stm32cubel0.html#documentation

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 opos 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.