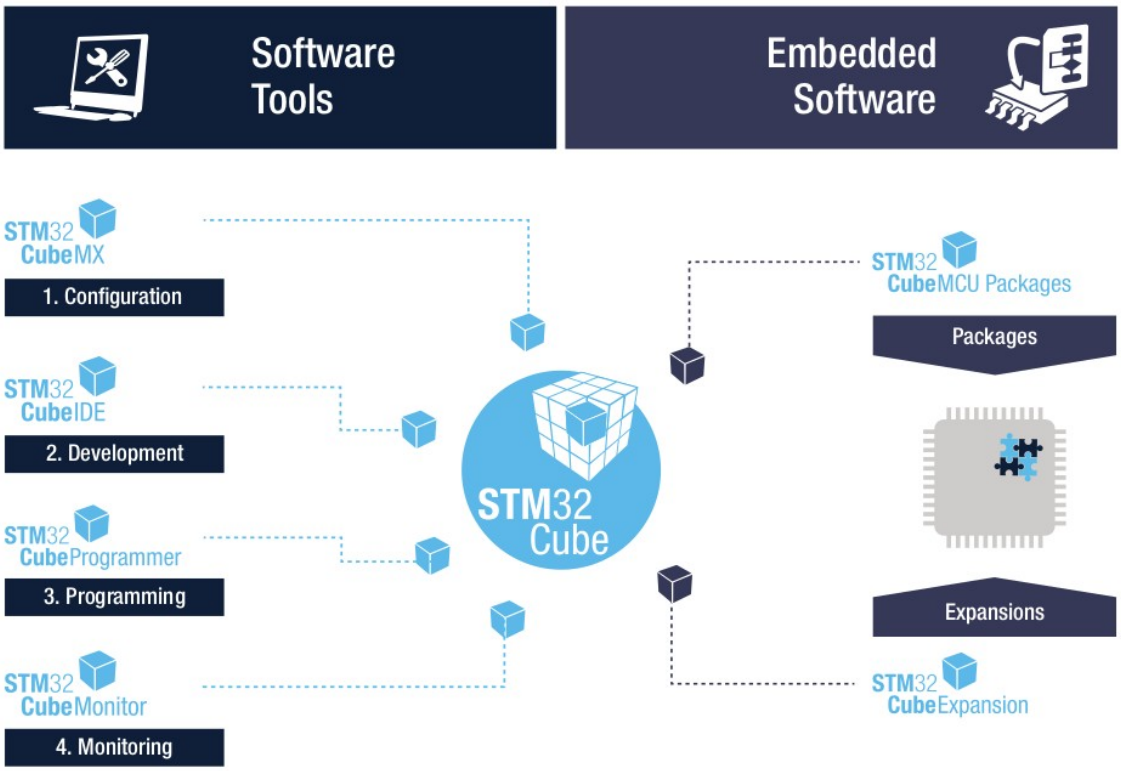
STM32CubeIDE

I. Présentation

STMicroelectonics propose une suite logicielle pour développer sur ses processeurs : **STM32Cube**[1]. Il s'agit d'un écosystème complet qui propose plusieurs logiciels, même si nous nous concentrerons sur les deux principaux :

- **STM32CubeIDE**, un IDE ;
- **STM32CubeMX**, un outil de configuration et de génération de code.



L'environnement de développement intégré (qu'on appellera désormais **IDE** pour *Integrated Development Environment*) s'appelle donc **STM32CubeIDE**. Il est construit à partir de l'IDE Eclipse, *aka* le plus gros projet d'IDE libre et multi-plateforme. Son fonctionnement s'articule autour de *perspectives* (des vues) correspondant à des usages spécifiques (par ex : *edit*, *debug*).

Intégré à l'IDE se trouve **STM32CubeMX**, qui permet de configurer graphiquement le MCU et ses périphériques pour une utilisation en quelques minutes. Celui-ci fourni en quelques clics diverses fonctions de configuration et d'utilisation des périphériques du MCU. Dans un contexte d'entreprise, l'objectif de ce genre d'outils est de réduire le *Time-to-market* (temps de développement) des solutions logicielles embarquées.

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

II. Téléchargement et Installation

STM32CubeIDE est téléchargeable depuis le site STMicroelectronics et est *cross-platform* (Mac/Windows/Linux). Ce document présente la **version 1.11.0**.

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

Une fois installé (pas de difficulté particulière), vous pouvez ouvrir l'IDE et en faire le tour avec une vidéo de 5 minutes, depuis STM32CubeIDE :

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

III. Création d'un projet

1. Dans **STM32CubeIDE** (pas STM32CubeMX) : `File → New → STM32 Project`.
2. Dans l'onglet « `Board Selector` », retrouver avec la barre de recherche la carte de TP (l'étiquette est dessus), la sélectionner et cliquer sur « `Next` ».
   - *Notez qu'il est possible de sélectionner des projets exemples fournis par STMicroelectronics (avec ou sans OS).*
3. Renseigner le nom[2] du projet, vérifier son emplacement, laisser les options telles quelles (`C / Executable / STM32Cube`) et cliquer sur `Finish`.
4. « `Initalize all peripherals with their default Mode ?` »

   → « **Yes** » ou « **No** », selon la consigne du sujet !
5. La fenêtre de configuration graphique s'ouvre. Il s'agit d'un fichier `*.ioc`, géré par le logiciel STM32CubeMX. Dans l'onglet « `Pinout & Configuration` », configurer les composants **en fonction du cahier des charges**.
6. Une fois la configuration effectuée : `Project → Generate Code` pour générer le code correspondant aux configurations des périphériques (ou l'icône).

Le projet est créé et les instructions de configuration du micro-contrôleur sont générées (démarrage, configuration des horloges, interruptions, …). Les périphériques sélectionnés dans l'étape précédente sont également prêts à l'emploi. Pour les utiliser, voir section « V HAL – Hardware Abstraction Layer » page 7.

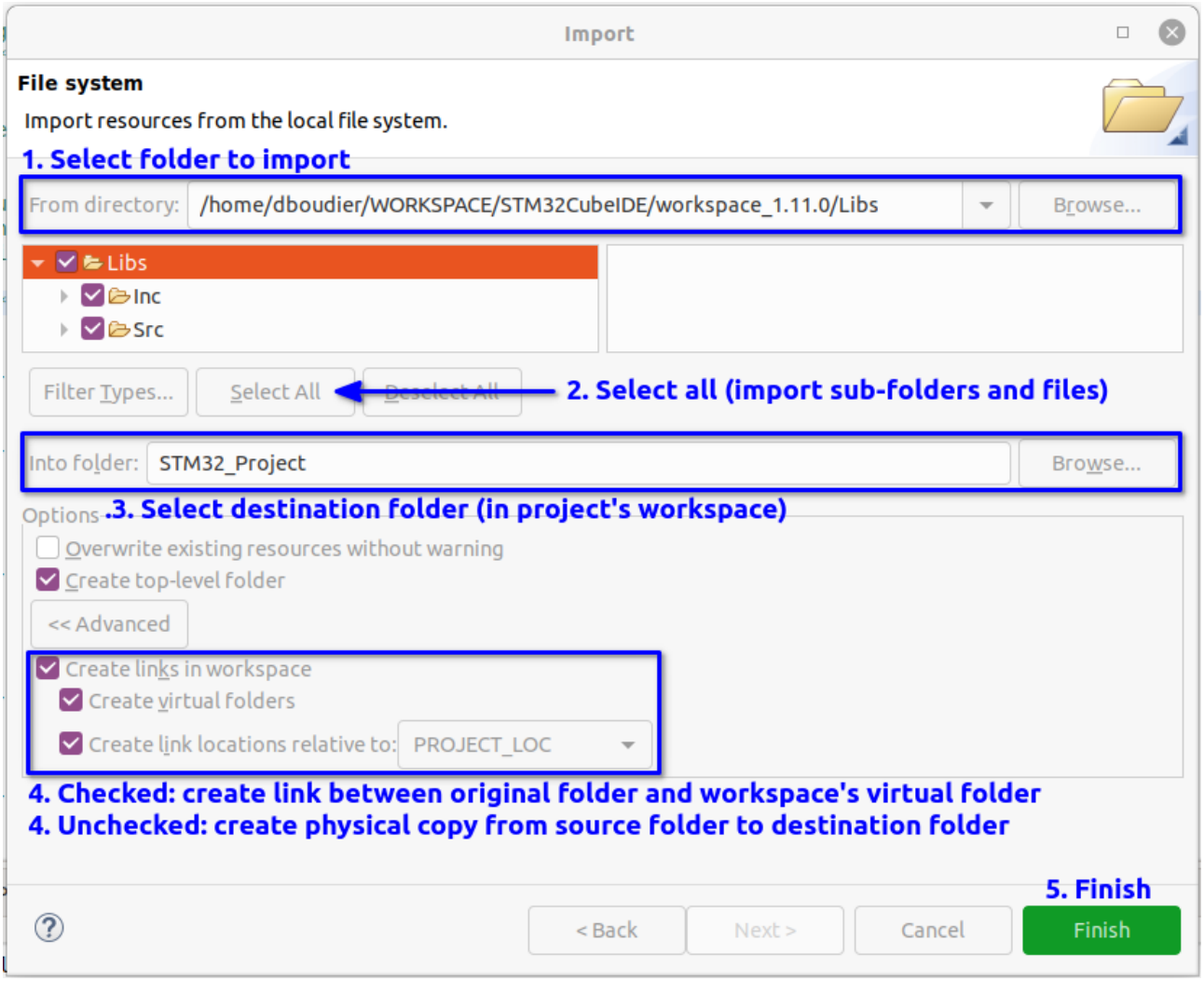
`Attention :` Après avoir créé un projet en passant par STM32CubeMX (fichier de configuration *.ioc), faîtes attention à bien coder entre les balises de commentaires fournis par l'IDE.
En cas de reconfiguration d'un périphérique et de re-génération du code, STM32CubeMX supprimera tout ce qui se trouve en dehors de ces balises !

2 Attention : choisir un nom sans accent, sans caractère spécial, sans espace.

IV. Intégrer des sources existantes à un projet

IV.1. Importer les sources dans le workspace

- Pour importer des sources (répertoire ou fichier) à un projet, on peut simplement copier-coller les sources dans un des répertoires déjà existants, depuis le système d'exploitation.
- Il est également possible de réaliser cette étape à partir de l'IDE, depuis l'explorateur de projet (onglet de gauche de l'IDE, en perspective *Edit*).
  - Clic droit sur projet → `Import... → General → File System → Next`
  - Suivre la configuration suivante.



Le répertoire (ou le fichier) importé doit maintenant apparaître dans l'arborescence du projet, dans l'onglet de l'explorateur de projet. De plus si une copie physique a été réalisée, les fichiers se trouvent physiquement sur le disque, dans le répertoire du projet.

Remarque
Importer un répertoire à un projet n'a pour seul effet que de rendre son contenu (fichiers sources et headers) accessible depuis l'explorateur de projet, afin de faciliter la consultation ou l'édition de ses fichiers.

Ceci implique que les fichiers importés ne sont pas directement prêts à être utilisés par la chaîne de compilation. Selon qu'il s'agit de sources C/C++ ou de headers, deux manipulations distinctes doivent être réalisées. Celles-ci sont expliquées sur les pages suivantes.

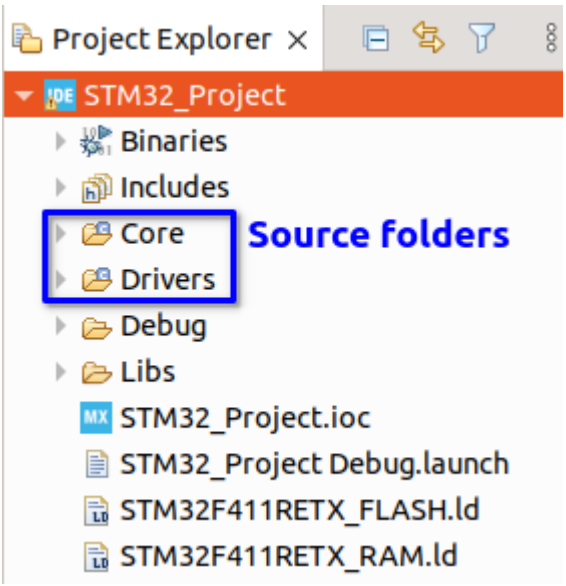
IV.2. Intégrer les fichiers C/C++ au processus de compilation

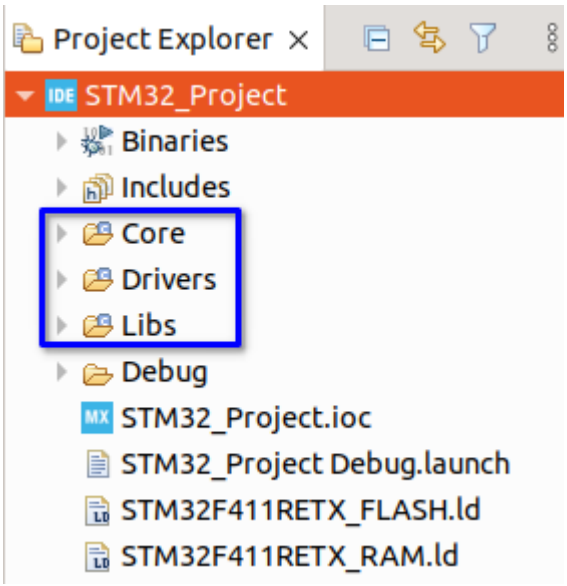
Pour être considéré comme un fichier source à compiler, le-dit fichier doit faire partie d'un « ***Source Folder*** », reconnaissable à son icône . Or par défaut un dossier importé avec la méthode précédente n'est qu'un dossier ordinaire. Mais en convertissant un dossier quelconque en ***Source Folder***,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

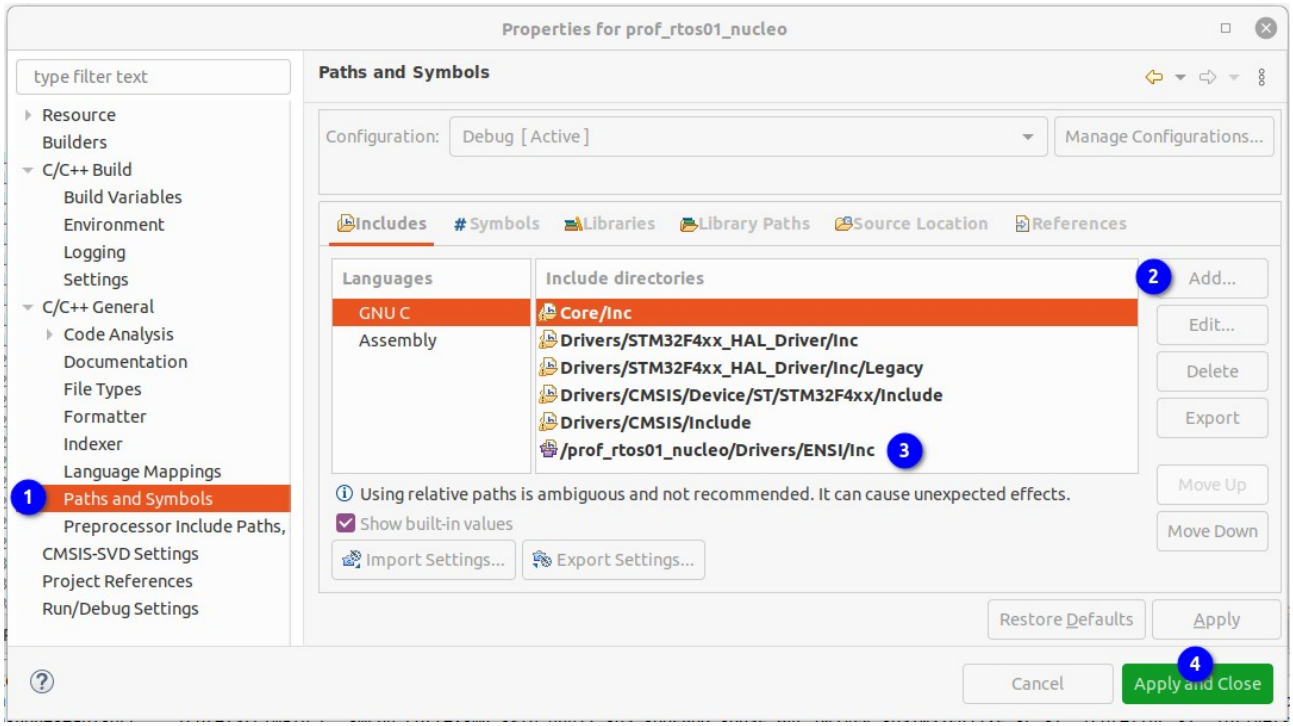




IV.3. Intégrer les fichiers d'en-tête (headers) au chemins d'inclusion

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

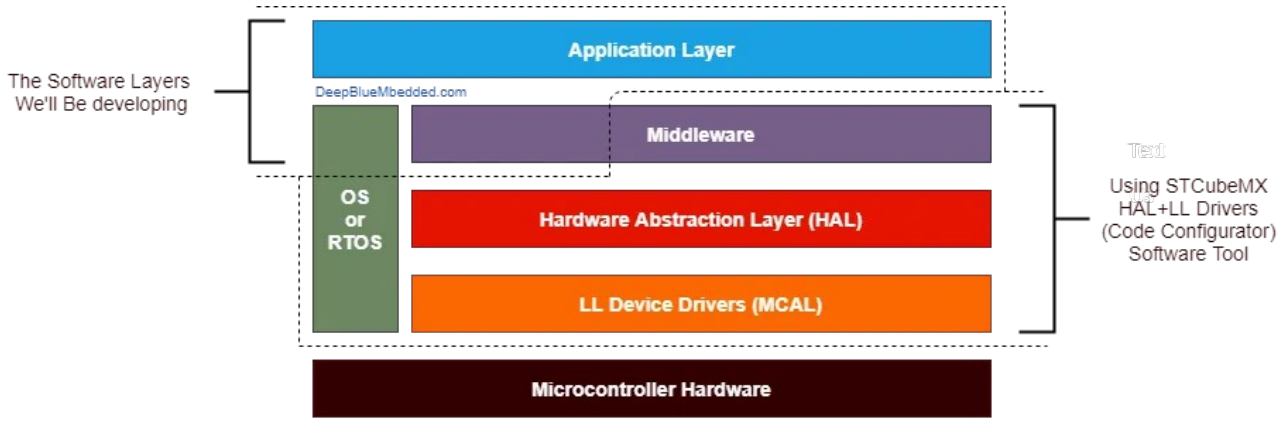
Les TP Systèmes Embarqués de première année portaient sur la programmation bas niveau (à l'étage registre) d'un micro-contrôleur (Microchip PIC18), avec pour premier objectif de produire un BSP (*Board Support Package*), puis en second objectif de développer une application (enceinte Bluetooth) en utilisant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

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.

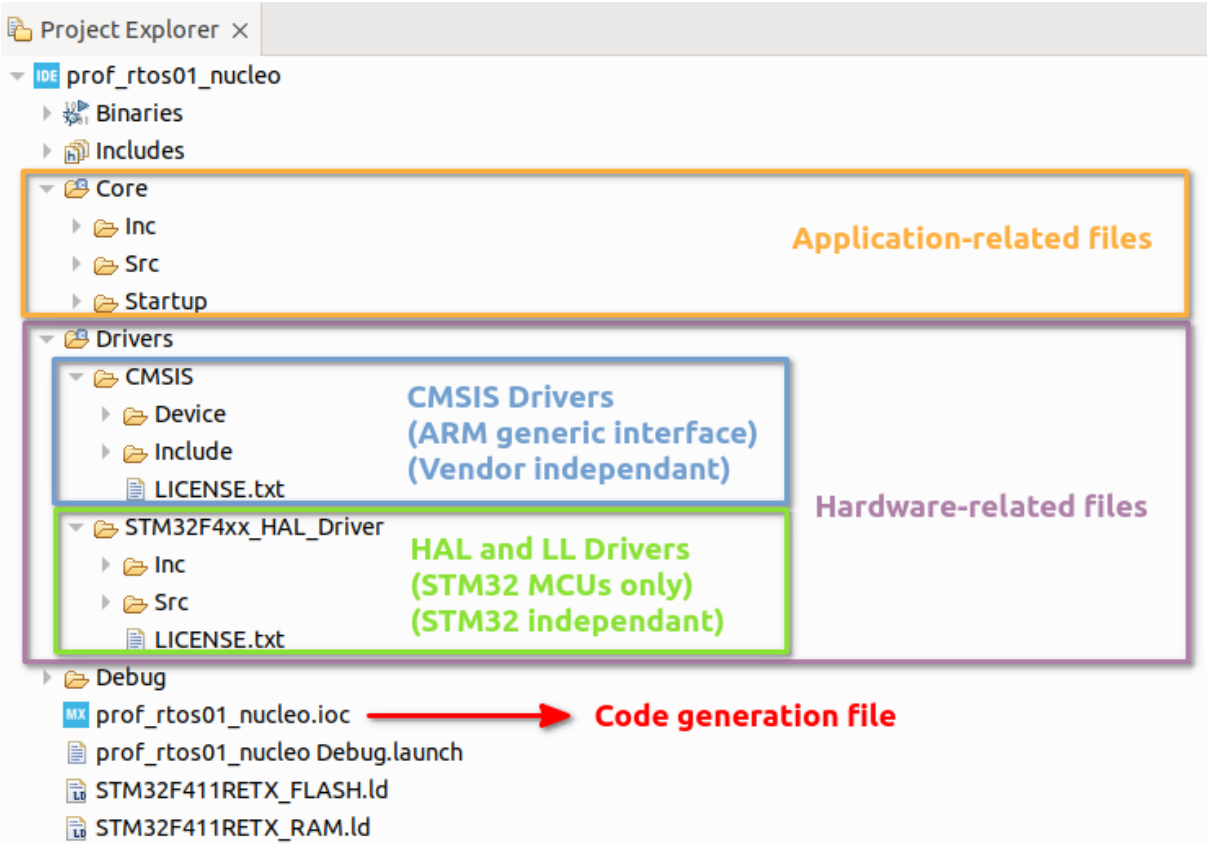
V.2. Utiliser la HAL

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.

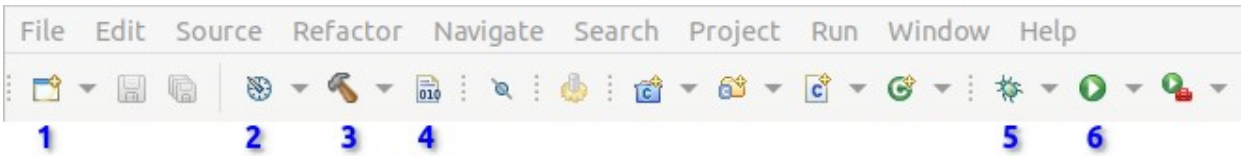


3 https://www.st.com/en/embedded-software/stm32cubel0.html#documentation

VI. Programmer et debugger avec STM32Cube

En bref

Faisons simple : ces menus et boutons sont les plus utilisés de STM32CubeIDE. Les deux que vous emploierez le plus sont la compilation (Build All) et le mode debug.



1. Nouveau projet (File → New → STM32 Project) [Alt+Shift+N]
2. Configuration (mode Debug / mode Release)
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) (programmation et exécution de la cible)

Debugger

La première fois que vous lancez le debugger, la fenêtre « Edit Configuration » s'affiche. Cliquez sur OK. En mode debug, l'IDE change de perspective (la configuration des fenêtres et menus change). Il est évidemment possible de revenir à la perspective d'édition en arrêtant le mode debug.

1. Reset du composant et de la session de debug
2. Le debugger s'arrête / ne s'arrête pas aux breakpoints
3. Arrêt et redémarrage
4. Démarre / continue [F8]
5. Pause
6. Stop (arrête la session de debug, revient au mode édition)
7. Step into : passe à l'instruction suivante
8. Step over : passe à la ligne suivante
9. Step return : continue jusqu'à quitter la fonction en cours