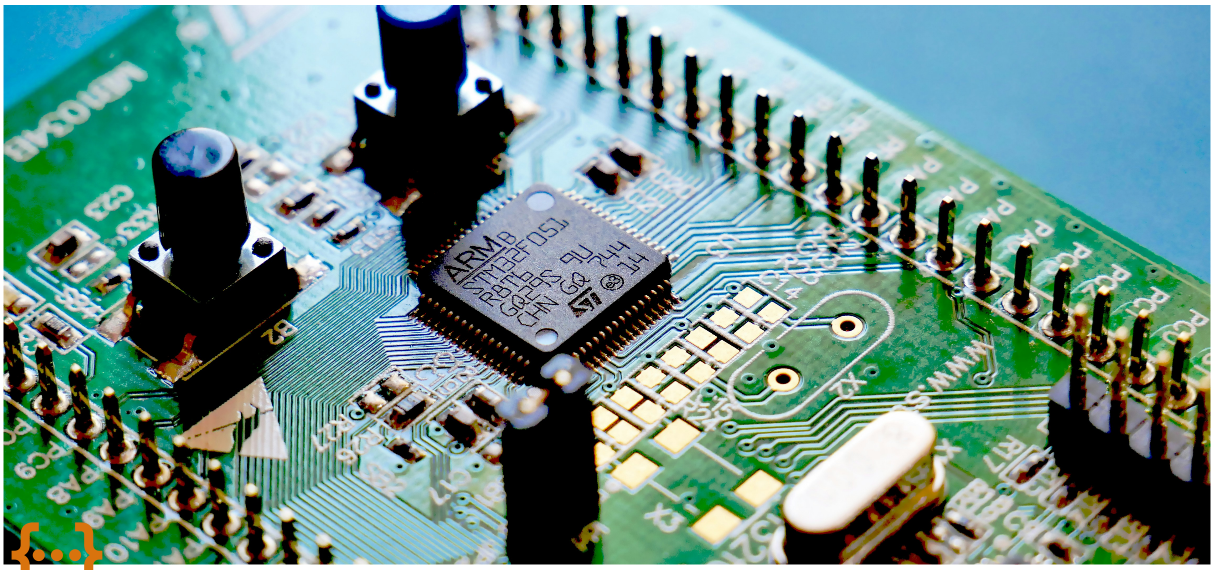


DESN2000 (Computer Engineering) Getting Started With STM32cubeIDE

Author: Riley Haydon

Created: T2 2024

STM32CubeIDE is STMicroelectronics's integrated development environment (IDE) for their microcontrollers. The development you have got by now has a NUCLEO-F303RE board equipped with a 32-bit STM microcontroller. The STM32CubeIDE includes the necessary drivers, compilers, and a debugger, all in a single package.

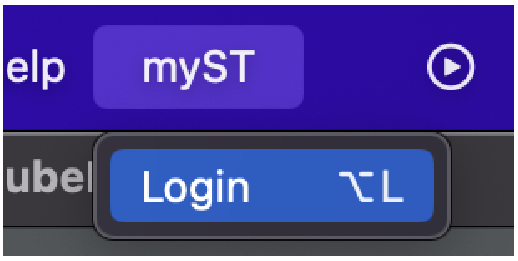
Install STM32CubeIDE

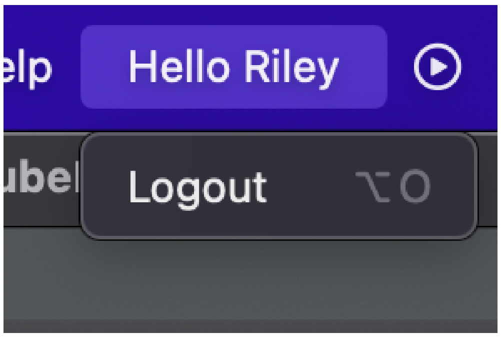
You can download and install CubeIDE from the link below:

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

It's free, although you do have to sign up for an account / give them your email address. The good thing is it is supported on Windows, Linux and MacOS. Download the package for your operating system and install the package as you would install any other software on your computer.

When launching the app for the first time, you may be prompted to create/login to an account. If not you can access the login by going ***mySt*** → ***login*** in the menu bar. I know creating an account to use an IDE is a pain and unnecessary, but I found that some features simply didnt work when I wasnt logged in.

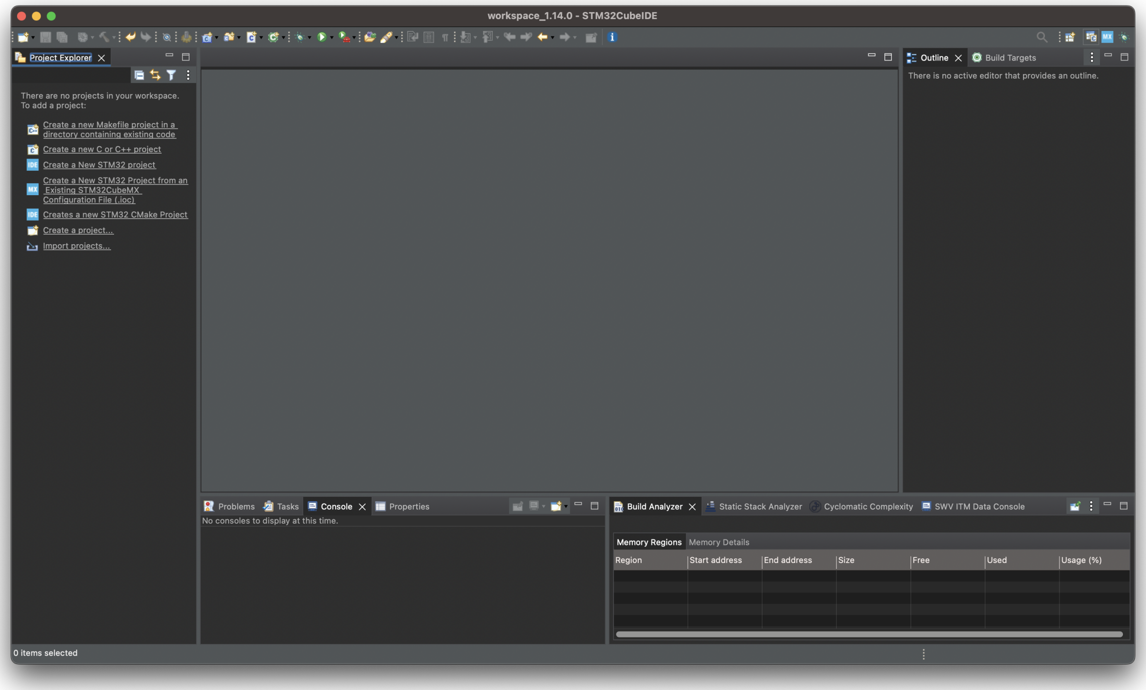




Creating a Project

Start the CubeIDE application, it should look something like this → i.e an empty workspace with no projects.

(Mine is in dark mode)


workspace_1.14.0 - STM32CubeIDE

There are no projects in your workspace.
To add a project:

- Create a new Makefile project in a directory containing existing code
- Create a new C or C++ project
- Create a New STM32 project
- Create a New STM32 Project from an Existing STM32CubeMX Configuration File (.ioc)
- Creates a new STM32 CMake Project
- Create a project...
- Import projects...

There is no active editor that provides an outline.

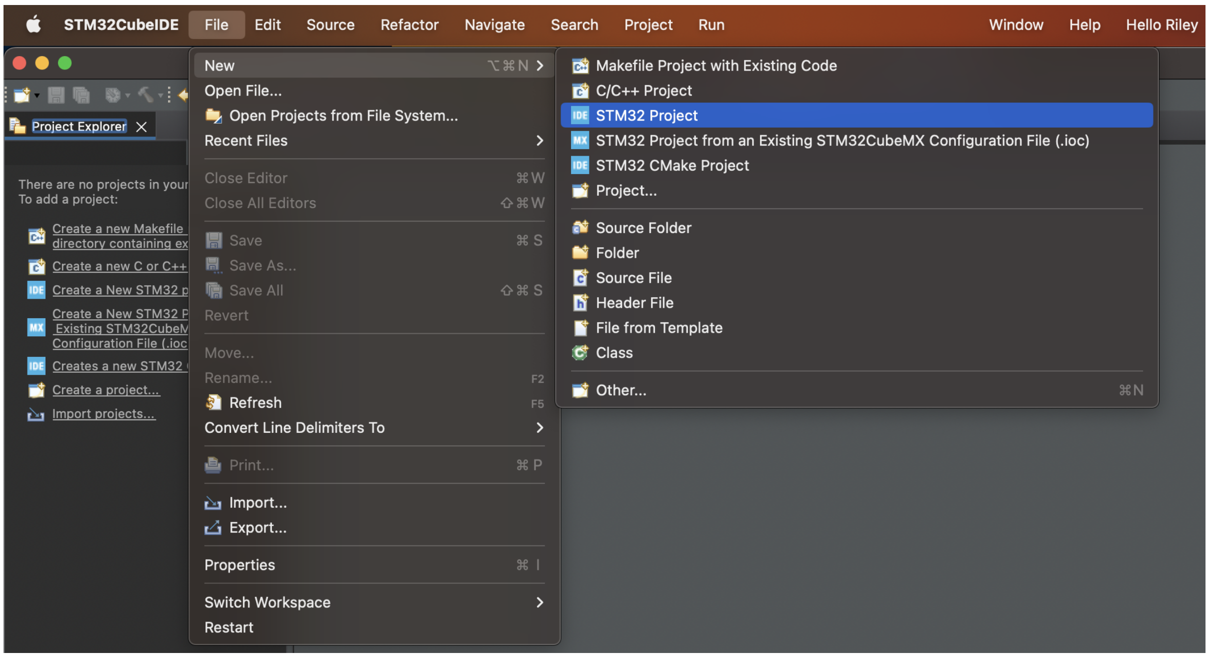
No consoles to display at this time.

Region	Start address	End address	Size	Free	Used	Usage (%)

0 items selected

Under the menu bar at the top: ***File*** > ***New*** > ***STM32 Project***

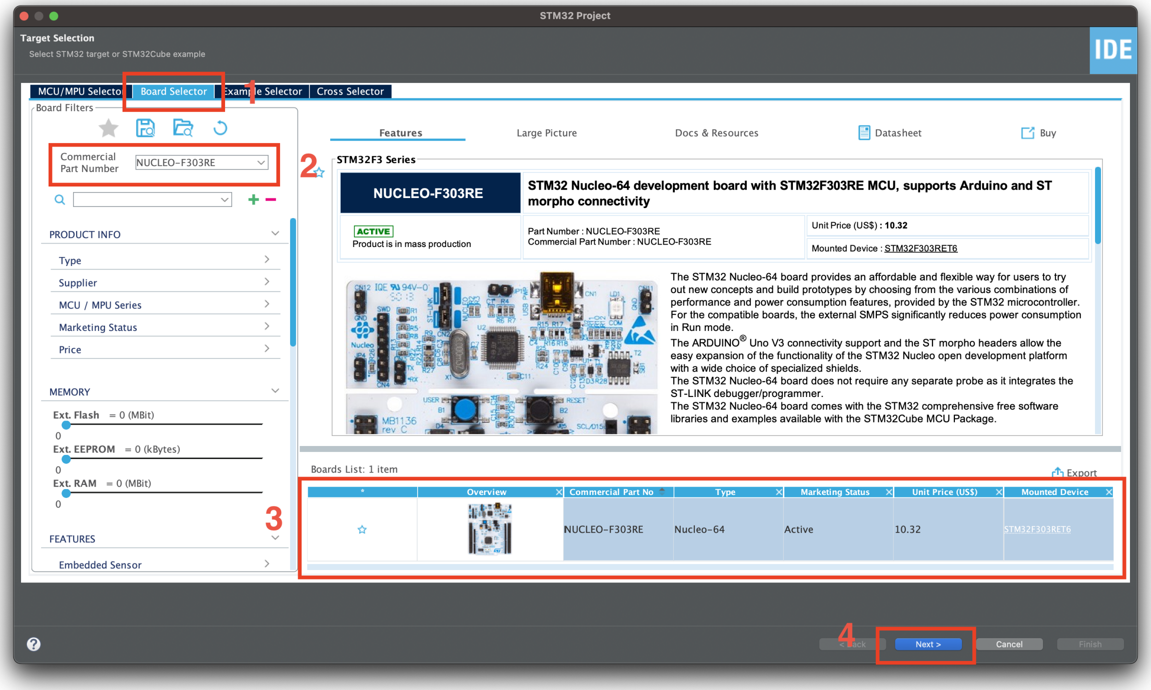
⚠ Dont be alarmed - It will download some stuff after you create the new project and initialise the GUI for the board selector it can be a bit laggy, ***so be patient***



Page 1

For this project, we are using the **Nucleo-F303RE**.

1. Select 'Board Selector' tab in the top left.
2. Search **Nucleo-F303RE** inside the 'Part Number Search' dropdown/textbox
3. In the table of boards in the lower RHS, select the board we searched for
4. Click 'next' down the bottom right.



Target Selection

Select STM32 target or STM32Cube example

Commercial Part Number: NUCLEO-F303RE

Ext. Flash = 0 (MBit) 0
Ext. EEPROM = 0 (kBytes) 0
Ext. RAM = 0 (MBit) 0

STM32F3 Series

NUCLEO-F303RE

STM32 Nucleo-64 development board with STM32F303RE MCU, supports Arduino and ST morpho connectivity

ACTIVE
Product is in mass production

Part Number : NUCLEO-F303RE
Commercial Part Number : NUCLEO-F303RE

Unit Price (US$) : **10.32**

Mounted Device : STM32F303RET6

The STM32 Nucleo-64 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the external SMPS significantly reduces power consumption in Run mode.

The ARDUINO® Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

The STM32 Nucleo-64 board does not require any separate probe as it integrates the ST-LINK debugger/programmer.

The STM32 Nucleo-64 board comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package.

Boards List: 1 item

*	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		NUCLEO-F303RE	Nucleo-64	Active	10.32	STM32F303RET6

Page 2

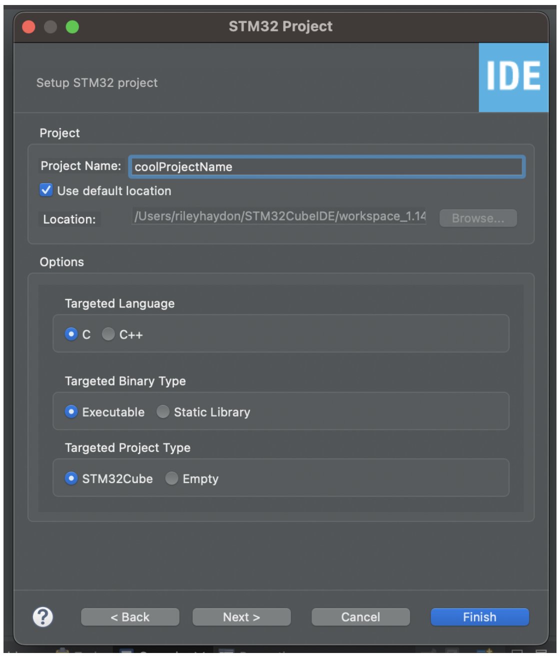
You can now pick a project name. You may pick any location for the project you like, I will be leaving mine in the default location.

Verify the other fields under the options category are as follows.

Targeted Language: C

Targeted Binary Type: Executable

Targeted Project Type: STM32Cube



Setup STM32 project

Project

Project Name: coolProjectName

[✓] Use default location

Location: /Users/rileyhaydon/STM32CubeIDE/workspace_1.14

Options

Targeted Language
[✓ C] C++

Targeted Binary Type
[✓ Executable] Static Library

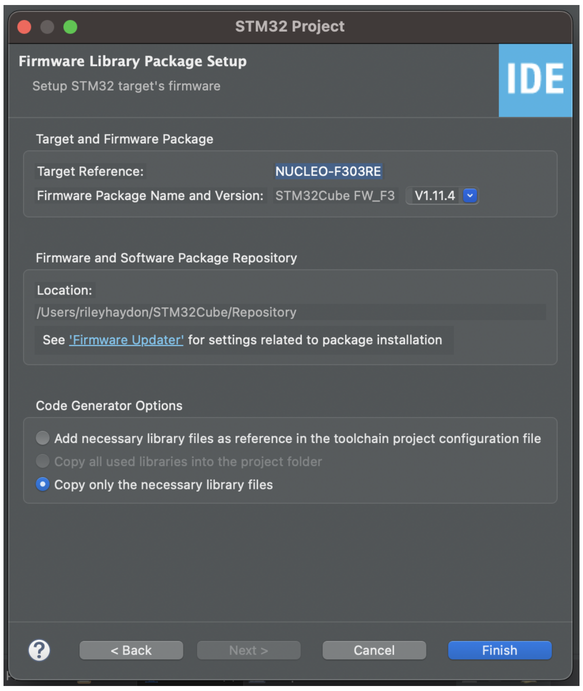
Targeted Project Type
[✓ STM32Cube] Empty

Page 3

Verify the target reference is NUCLEO-F303RE. And press **Finish**

(If you missed this step because you pressed finish before next, dont worry its just a checking page - everything should be okay)

💡 Your version number for 'Firmware Package Name and Version' may be different to mine, as long as its V1.11.4 or higher it should be fine


STM32 Project

Firmware Library Package Setup
Setup STM32 target's firmware

Target and Firmware Package
Target Reference: NUCLEO-F303RE
Firmware Package Name and Version: STM32Cube FW_F3 V1.11.4

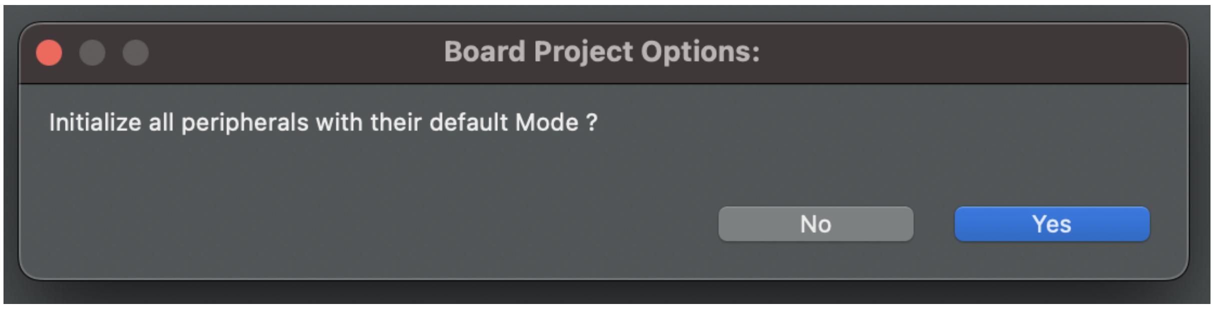
Firmware and Software Package Repository
Location:
/Users/rileyhaydon/STM32Cube/Repository
See 'Firmware Updater' for settings related to package installation

Code Generator Options
Add necessary library files as reference in the toolchain project configuration file
Copy all used libraries into the project folder
(●) Copy only the necessary library files

You will then get a popup asking if you want to 'Initialise peripherals to their default mode.

Press **Yes**

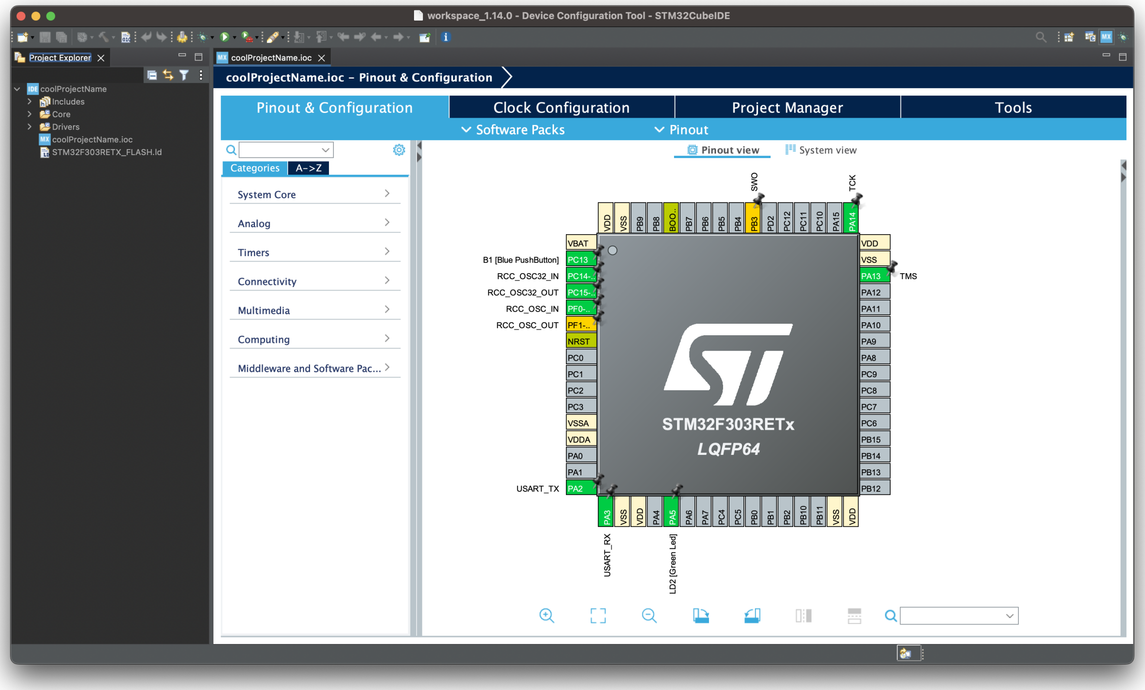
It may take some time after this to load everything for the first time.


Board Project Options:

Initialize all peripherals with their default Mode ?

Finally

After the project has been created, you should now see a nice GUI of the F303re showing the pinout diagram.



Here, we can set up our peripherals. By choosing “yes” a moment ago, it has pre-populated some settings for us.

- **PA5** is LD2, the NUCLEO board’s Green LED
- **PC13** is B1, the NUCLEO board’s Blue Push Button

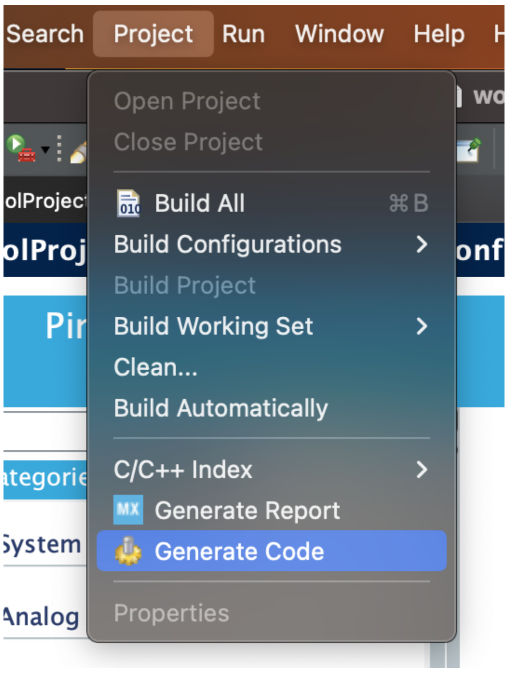
Keep the defaults for now, we will learn all these shortly, so don’t worry.

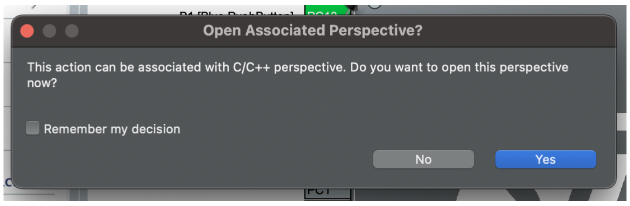
💾 Writing to the Board

Generating the Code

In the top menu bar go ***Project*** > ***Generate Code***

If you get a popup asking 'This action can be associated...' → Press **Yes**





Open Associated Perspective?

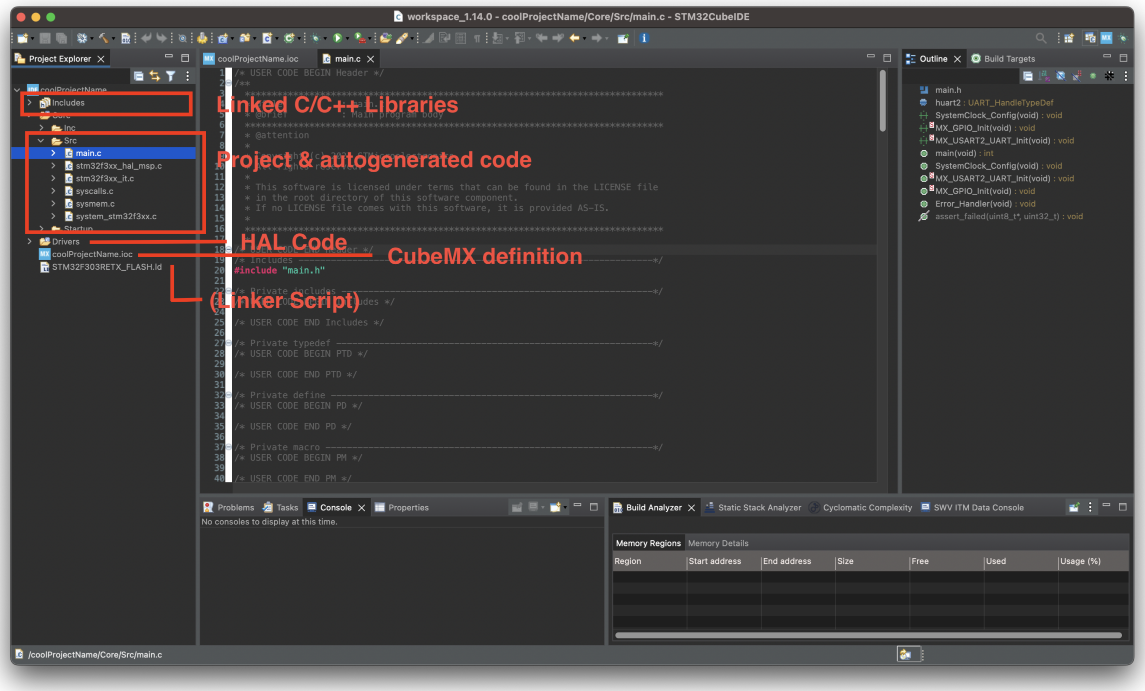
This action can be associated with C/C++ perspective. Do you want to open this perspective now?

Remember my decision

No Yes

This genrates C files to work with under a `Src` directory and puts a `HAL` (Hardware Abstraction Layer) into an Includes directory.

Expand the folders right under the project explorer and see what it has generated:



```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */
```

👀 Looking at `main.c`

In this tutorial, we'll focus on the project and autogenerated code section, starting with `main.c`.

You'll see that main.c is already quite large, containing a fair amount of autogenerated code.

⚠ **A key piece of information** to remember here is that `main.c` can be edited by the code generator, so it's important to only write code in the ***USER sections***

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------
------------------------------
--------------*/

  /* Reset of all peripherals, I
nitializes the Flash interface a
nd the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock
*/
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured p
eripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
```

Observe in this block of code that there are several `USER CODE BEGIN` and `USER CODE END` sections marked out by comments

Any code written **inside** these blocks is **safe** and will not be deleted by the code autogenerator.

Any code written **outside** those blocks is **unsafe** and will be *deleted* by the code autogenerator any time you edit the device configuration settings that we looked at earlier.

Files that you yourself add to the project (e.g. `MySuperCFile.c`) are also totally safe. It's just this set of autogenerated `.c` (and `.h`) files that you must be careful with when adding code.

Now to look at the function names used in the autogenerated code.

- Any function call beginning with `HAL_` is from the STM32 HAL, and is provided in the library files. There's HAL functions to do all sorts of things, including using the UART, writing a Pin, etc.
- Any function call beginning with `MX_` is autogenerated by CubeIDE. These functions tend to be used to initialise functions.
- There are exceptions to these rules, including, annoyingly, `SystemClock_Config()`, which is also an autogenerated function.

```
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

👨‍💻 Writing Our Own Code!

Let’s add a smidge of C code of our own now! After the `Infinite Loop` area, we’re going to add code to toggle the LED under section 3. To get the autosuggest to show up you press Ctrl+Space:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    //Here's my new code that I've added to toggle the Green LED (LD2)

    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

    HAL_Delay(1000);
}
/* USER CODE END 3 */
```

Note the following:

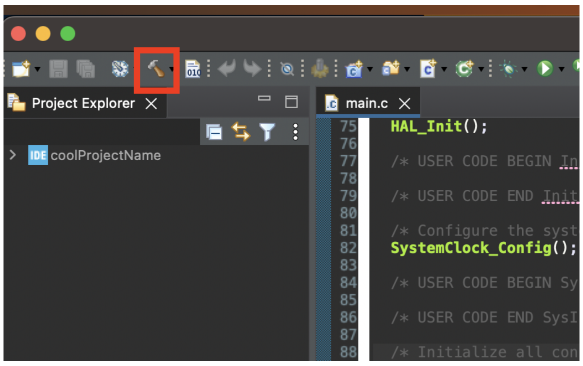
- We have put my code only inside section 3
- Under the device configuration tool, the Led GPIO pin was named LD2 - observe how a name has automatically been generated for both the Pin and the Port
- We have used two HAL functions, one to toggle a GPIO pin, and one to cause a delay of 1000 milliseconds.

This tiny project is all we needed to blink that on-board LED at around once per second.

Compiling and Flashing

STM32CubeIDE actually makes it pretty easy to compile our work and get it onto the STM32 chip. The first step is to produce the first version of the compiled `.elf` (a binary version of our code). We need this `.elf` so that we can point the download tool to it.

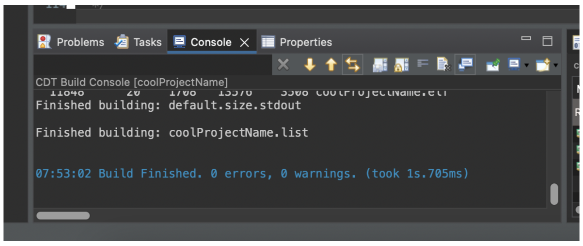
To generate the `.elf`, we need to do a build. Press the *build button on the toolbar*:



```
HAL_Init();

/* USER CODE BEGIN In

/* USER CODE END Init

/* Configure the syst
SystemClock_Config();

/* USER CODE BEGIN Sy

/* USER CODE END SysI

/* Initialize all con
```

Pressing the build button

Now, build information is presented in the console at the bottom of the screen:

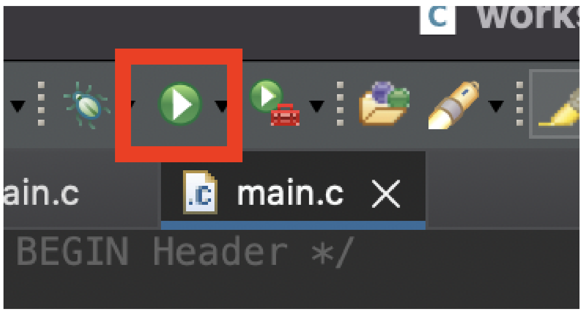


```
  11848      20    1708   13576    3508 coolProjectName.elf
Finished building: default.size.stdout

Finished building: coolProjectName.list


07:53:02 Build Finished. 0 errors, 0 warnings. (took 1s.705ms)
```

Result of the build

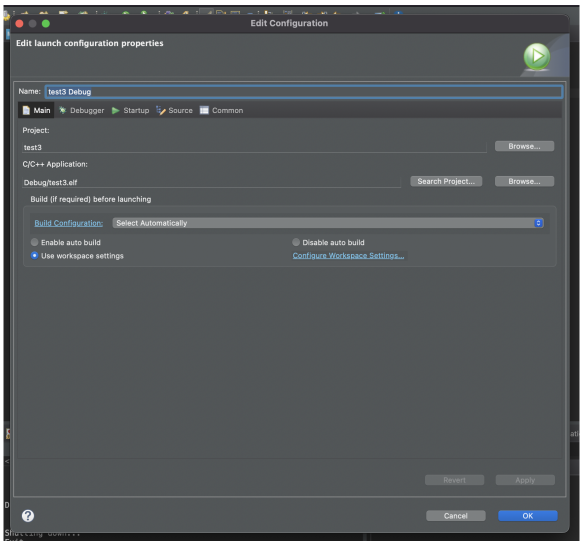
Now we want to send this compiled binary onto the STM32 microcontroller. Let's plug in the development board. The Red power LED (below the blue switch) is lit, as is the larger communication LED (by the USB cable).

Inside STM32CubeIDE, select the **run** button.



This will open the Run dialog (as it's the first time we've run it). The settings we choose now will be saved as a run configuration which we can re-use or edit later. Press **OK** and the download will proceed.

The LED next to the usb connector should flash red and green several times indicating a download is in progress.



Name: test3 Debug

Project: test3

C/C++ Application: Debug/test3.elf

Build (if required) before launching

Build Configuration: Select Automatically

Enable auto build [✓ Use workspace settings] Disable auto build

Interestingly, we do not get the option to choose a board or USB port or anything during this process, it all just happened automagically. The NUCLEO board's communication LED should light up during this time, and after that, it seems that the board should be running the programme.

It is as easy as that - you've got everything set up now. For any new code from here, you just need to hit the run button - it will compile it for you automatically.

This tutorial was adapted from the blog post by Dr Hammond Pearce. If you prefer a detailed version, you may follow that blog post.

Tutorial: Getting started with an ST development board using STM32CubeIDE

Project initialisation, intro to debugging, and the virtual COM port

https://01001000.xyz/2020-05-11-Tutorial-STM32CubeIDE-Getting-started/