Rapid Embedded Course

Getting Started Guide

Contents

[1 Introduction 1](#_Toc35943539)

[1.1 Course Labs Overview 1](#_Toc35943540)

[1.2 Scope of this Guide 1](#_Toc35943541)

[2 Requirements 2](#_Toc35943542)

[2.1 Software and Hardware 2](#_Toc35943543)

[3 Mbed Studio Setup 3](#_Toc35943544)

[3.1 Mbed Studio 3](#_Toc35943545)

[3.1.1 Preparing a workspace 3](#_Toc35943546)

[3.1.2 Creating or importing programs 3](#_Toc35943547)

[3.1.3 Build and running a program 5](#_Toc35943548)

[4 Mbed CLI 6](#_Toc35943549)

[4.1 Installation and Setup 6](#_Toc35943550)

[4.2 Creating or importing programs 6](#_Toc35943551)

[4.2.1 Creating a program 6](#_Toc35943552)

[4.2.2 Importing a program 7](#_Toc35943553)

[4.2.3 Managing libraries 7](#_Toc35943554)

[4.3 Compile and run a program 7](#_Toc35943555)

[4.3.1 Setting a default target and toolchain 7](#_Toc35943556)

[4.3.2 Compile and Run 8](#_Toc35943557)

[5 Mbed OS Simulator 9](#_Toc35943558)

[5.1 Installation and Setup 9](#_Toc35943559)

[5.1.1 Software requirements 9](#_Toc35943560)

[5.1.2 Install Mbed Simulator 9](#_Toc35943561)

[5.2 Create and Run a program 10](#_Toc35943562)

[5.2.1 Create a project 10](#_Toc35943563)

[5.2.2 Run a demo 10](#_Toc35943564)

[5.3 Modify the Mbed Simulator 11](#_Toc35943565)

[5.3.1 Overview of the Mbed Simulator’s Architecture 11](#_Toc35943566)

[5.3.2 Add a component – Push Button 11](#_Toc35943567)

[5.3.3 Modify mbed-simulator-hal 12](#_Toc35943568)

[After changing anything in the simulator HAL, you need to recompile the libmbed library, run: 12](#_Toc35943569)

[or in windows 12](#_Toc35943570)

[5.4 Limitations 13](#_Toc35943571)

# Introduction

## Course Labs Overview

Throughout the labs in this course, we will utilise several tools that can be used to easily prototype and develop for embedded devices. Complementary to the content shown in the lectures, the labs will provide hands on experience making use of the capabilities of MCU development boards.

The labs will be centred around the Mbed platform, which aims to provide a universal API and development environment for MCU boards. This means that applications developed in these labs should work with a large range of different hardware.  
  
This first lab will focus on getting started with Mbed studio, the Mbed Command-line Interface (CLI), and the Mbed-Simulator.

## Scope of this Guide

This getting started guide should give you enough information to set up and start working with Mbed Studio, Mbed CLI, and Mbed Simulator.

Note that some labs in the course may require more specific set-up than is covered here - anything additional will be introduced and explained as necessary for each lab.

# Requirements

## Software and Hardware

In the following labs we will be using the following software and hardware:

* **Mbed Studio**, an IDE designed to streamline development and prototyping using Mbed enabled microcontrollers and development boards. Can be found here: <https://os.mbed.com/studio/>

**OR**

**Mbed online compiler**, which can be found here: <https://ide.mbed.com/compiler>

* **NUCLEO-F401RE**, or another suitable Mbed OS 5 compatible development board. A full list of compatible devices can be found here: <https://os.mbed.com/platforms>, note that outputs may have to be reconfigured for devices following a different standard.
* A breadboard, 4 x 330 Ω resistors, 4 x buttons, 1 red LED, 1 blue LED, 1 green LED

# Mbed Studio Setup

## Mbed Studio

Mbed Studio is a free IDE that allows developers to create Mbed OS 5 applications and libraries. This will provide the functionality we require for completing the further labs.

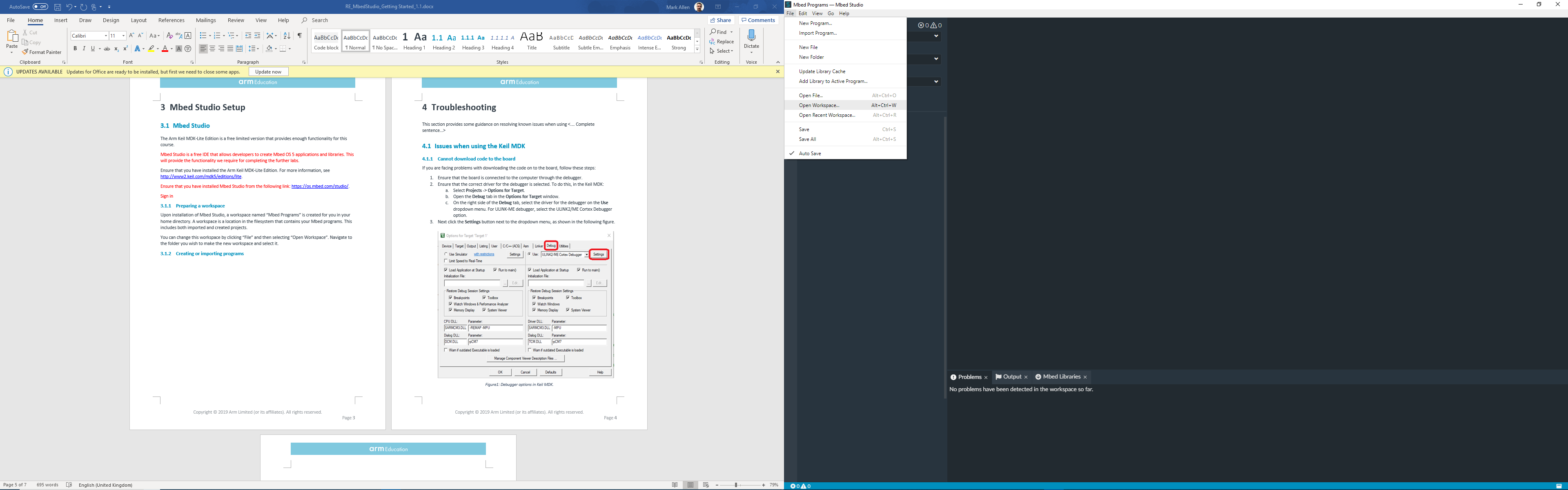
Ensure that you have installed Mbed Studio from the following link: <https://os.mbed.com/studio/>.

You will also need to register an account and use it to sign in when launching the IDE.

### Preparing a workspace

Upon installation of Mbed Studio, a workspace named “Mbed Programs” is created for you in your home directory. A workspace is a location in the filesystem that contains your Mbed programs. This includes both imported and created projects.

You can change this workspace by clicking “File” and then selecting “Open Workspace”. Navigate to the folder you wish to make the new workspace and select it.



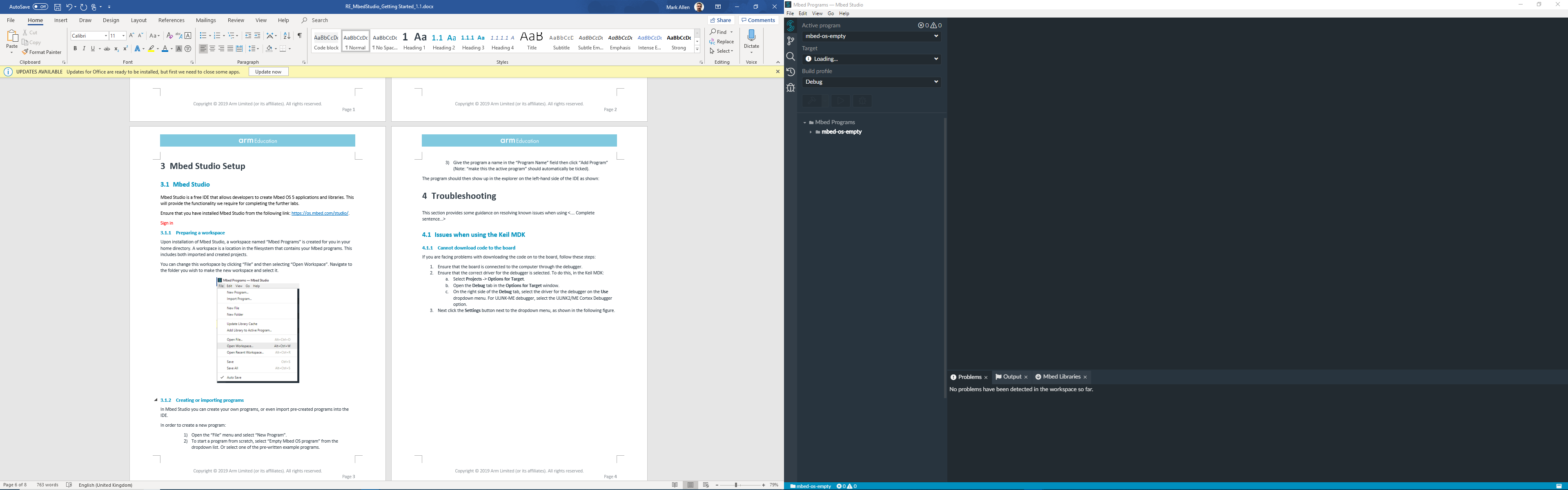
### Creating or importing programs

In Mbed Studio you can create your own programs or import pre-created programs into the IDE.

In order to create a new program:

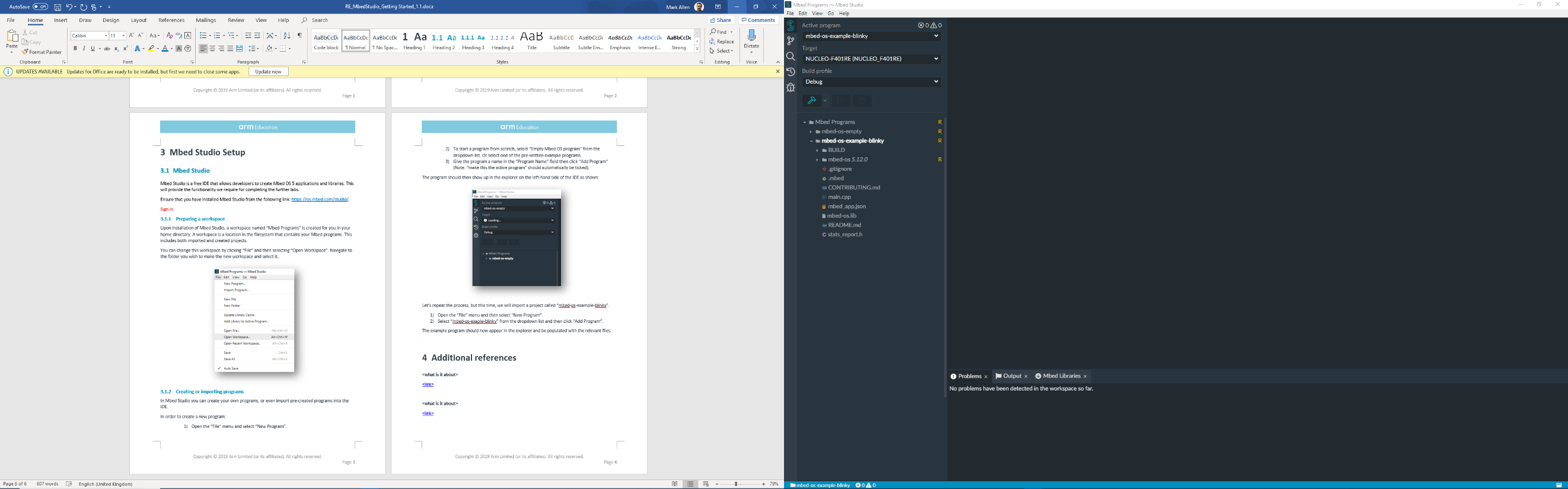
1. Open the “File” menu and select “New Program”.
2. To start a program from scratch, select “Empty Mbed OS program” from the dropdown list. Or select one of the pre-written example programs.
3. Give the program a name in the “Program Name” field then click “Add Program” (Note: “make this the active program” should automatically be ticked).

The program should then show up in the explorer on the left-hand side of the IDE as shown:



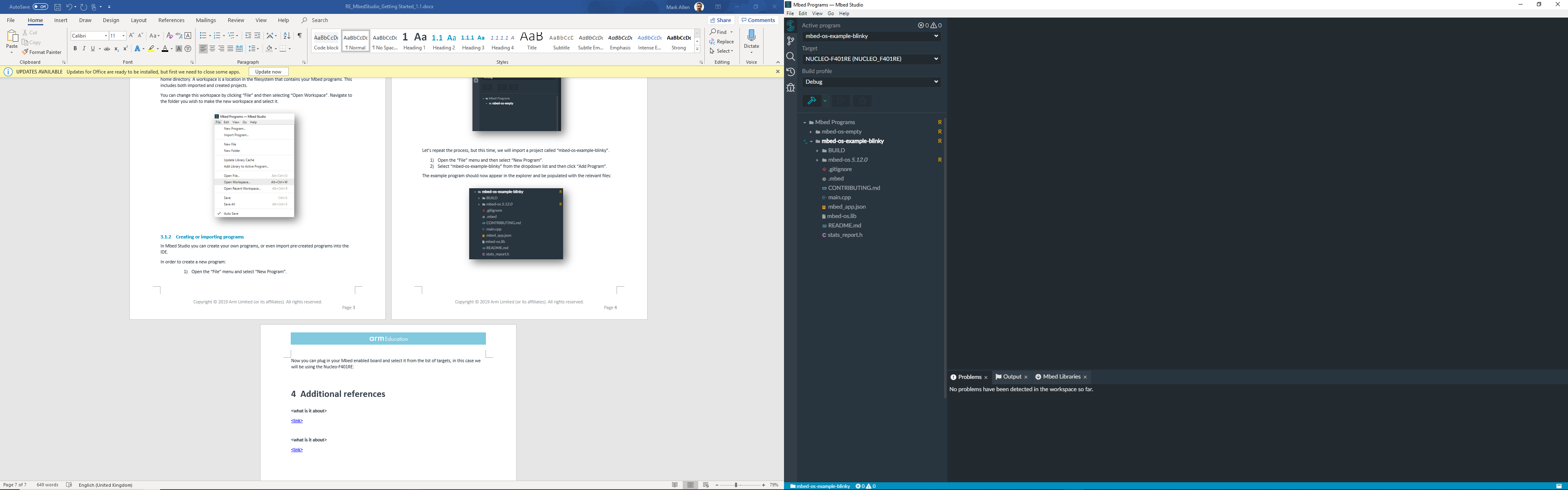
Let’s repeat the process, but this time, we will import a project called “mbed-os-example-blinky”.

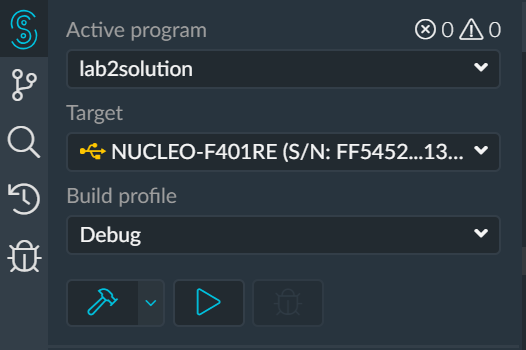
1. Open the “File” menu and then select “New Program”.
2. Select “mbed-os-example-blinky” from the dropdown list and then click “Add Program”.

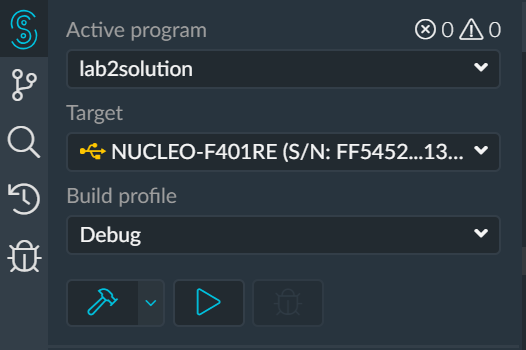
The example program should now appear in the explorer and be populated with the relevant files (this may require reopening the workspace to work):

### Build and running a program

Now you can plug in your Mbed enabled board and select it from the list of targets, in this case we will be using the Nucleo-F401RE:



Once the board is selected, ensure that “mbed-os-example-blinky” is the current active program and then select the “Build program” button:

As a result of the build process, Mbed will create a .bin file, which will contain result of compiling your program into machine code for whichever device you are using. Copying this file to your boards internal storage will have the effect of loading the program onto your board, at which point the program (for instance a blinking LED) should run!

Alternatively, pressing “Run program” should automatically build and load the program to your board. You may need to press the reset switch on the board in order for the new program to run.

# Mbed CLI

Mbed Studio is a free python-based command-line tool that enables developers to work with Mbed offline. Through the CLI you can use all the standard features of Mbed studio, including making and importing programs as well as compilation. It also supports version control through both git and mercurial.

## Installation and Setup

Installers are available for Windows and macOS and can be found here:

<https://os.mbed.com/docs/mbed-os/v5.13/tools/installation-and-setup.html>

There is no installer for Linux, you can follow the manual installation guide:

<https://os.mbed.com/docs/mbed-os/v5.13/tools/manual-installation.html>

Or follow the steps presented in this video: <https://www.youtube.com/watch?v=cM0dFoTuU14>

## Creating or importing programs

Using Mbed-CLI, you can create your own programs or import pre-created programs and add or remove libraries.

### Creating a program

In order to create a new program (called mbed-os-program), run:

$ mbed new mbed-os-program

This will create a new folder mbed-os-program, initializes a new repository and imports the latest revision of the mbed-os dependency to your program tree.

Note that Mbed CLI uses the current directory as a working context, in a similar way to Git, Mercurial and many other command-line tools. This means that before calling any Mbed CLI command, you must first change to the directory containing the code you want to act on.

For example, if you want to list all the imported libraries in your project:

$ cd mbed-os-program

$ mbed ls -a

To list all Mbed CLI commands, use mbed --help. A detailed command-specific help is available by using mbed <command> --help.

Now, in the project directory, create main.cpp file and write your down your code.

### Importing a program

In order to import a program, run $ mbed import <project link>:

$ mbed import https://github.com/ARMmbed/mbed-os-example-blinky

With Mbed CLI you can import hosted repositories from GitHub, GitLab and mbed.org.

### Managing libraries

Use the $ mbed add <project link> command to add a library to a program.

Use $ mbed remove <library name> command to remove a library from your project.

Use $ mbed update command to update a library in your project. As with any Mbed CLI command, mbed update uses the current directory as a working context. For example, if you're updating mbed-os, use $ cd mbed-os before you begin updating.

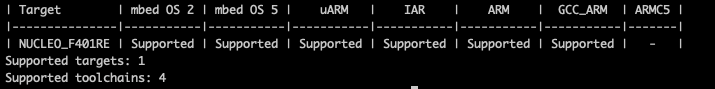
## Compile and run a program

### Setting a default target and toolchain

In order to detect the supported targets and toolchains on your device, run:

$ mbed detect

You should have an output similar to:



To set a default target, use the mbed target command with the name of your target (in this example, the NUCLEO\_F401RE), run:

$ mbed target NUCLEO\_F401RE

To set a default toolchain (in this example GCC\_ARM), run:

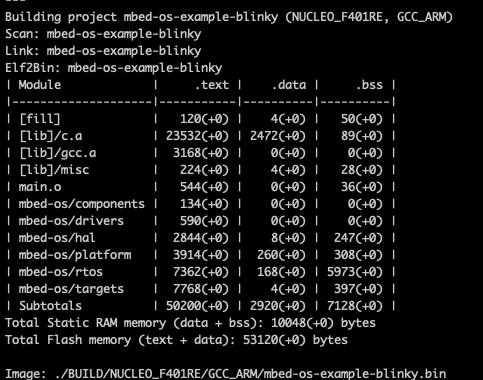
$ mbed toolchain GCC\_ARM

### Compile and Run

In order to compile your project, run in your project directory:

$ mbed compile

This is an example for the NUCLEO\_F401RE and ARM Compiler:



To run, the project, copy the .bin file (in this example mbed-os-example-blinky.bin) from ./BUILD/<target name> to your device.

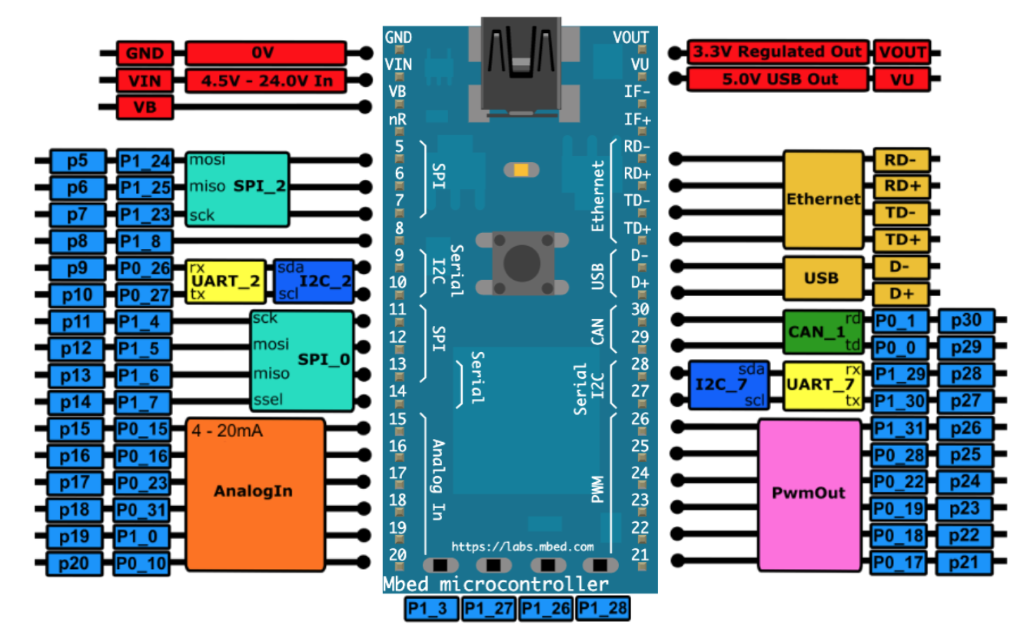
More Mbed commands can be found:

<https://os.mbed.com/docs/mbed-os/v5.13/tools/compiling.html>

# Mbed OS Simulator

The Mbed OS Simulator is an online tool for testing programs on a generic Mbed device.

It is currently in development and can be buggy for larger programs but can be used to run simple examples such as blinky without the need for a physical board.



*Figure 1: The Mbed Simulator board pin descriptions*

## Installation and Setup

Using the offline version will allow the developers to create their own Mbed projects and add the required libraries.

### Software requirements

Before installing the Mbed Simulator, make sure that the following utilities are installed

* Mbed CLI.
* Python 2.7 - not Python 3!.
* Git.
* Mercurial.
* Node.js v8 or higher.

### Install Mbed Simulator

Installation instructions for the Mbed Simulator can be found:

<https://github.com/janjongboom/mbed-simulator#celi>

First, Install Emscripten using the information provided.

Then install the simulator from the source (through git)

## Create and Run a program

### Create a project

You can run the Mbed Simulator on any Mbed OS 5 project, therefore creating or importing a project for the Mbed Simulator can be done with the Mbed CLI as described in 4.2.1 and 4.2.2.

You can delete the mbed-os library since it is already present in the simulator folder and, therefore, it is useless.

Note that the pin definitions are different for the Mbed simulator (check 5.)

### Run a demo

In order to run the binky demo, Run $ mbed-simulator . in the demos/blinky:

$ cd demos

$ cd blinky

$ mbed-simulator .

Note that this will download all dependencies (including Mbed OS) and will build the common libmbed library so this'll take some time.

The Mbed Simulator should now launch in your default browser.

You can add components manually once the simulator has been launched, by choosing the component and the pin(s) that it should be connected to.

Let’s have a look at the files in the ‘pwmout’ demo. First go into the directory of the lcd, then use the ls command to check all the files and folders.

$ cd ..

$ cd pwmout

$ ls

You should have as an output:

main.cpp simconfig.json

You are already familiar with the main.cpp file. Do not worry if you don’t understand the code in the main.cpp, it will be covered throughout the course.

The simconfig.json file is used to declare the specifications of the simulator. This will allow you to add components to the project and to setup the connections with the pins of the board, instead of adding them manually every time you run the project. More details can be found in the docs/simconfig.md file.

## Modify the Mbed Simulator

The Mbed Simulator is a project in development. Therefore, when you test your projects you might need to modify it, to add a new hardware component or adapt the already existing library to satisfy your project requirements.

### Overview of the Mbed Simulator’s Architecture

Large parts of the Mbed OS API are generic between targets. Target-specific code, such as which registers to write to when toggling a GPIO pin, are implemented using the Mbed C HAL. The simulator uses the same approach, implementing a new target (`TARGET\_SIMULATOR`) that implements the Mbed C HAL which passes events through to a JavaScript HAL. Then the UI subscribes to these events and updates the simulator accordingly.

In the scope of this course we will be interested in 3 folders of the mbed-simulator:

* demos
* mbed-simulator-hal
* viewer

The demos folder groups different examples which demonstrate a variety of applications using the mbed-simulator.

In the mbed-simulator-hal folder, we can find the mbed-os folder which is a fork Mbed OS 5.10.2 but with a new target for the simulator, TARGET\_SIMULATOR. Also, the JavaScript HAL and UI files of the peripherals (lcd, touchscreen and temperature sensor) can be found in the peripherals folder.

The JS HAL lives in viewer/js-hal, and dispatches events around between JS UI components and C++ HAL. It implements an event bus to let the UI subscribe to events from C++. For instance, see js-hal/gpio.js for GPIO and IRQ handling.

UI lives in viewer/js-ui, and handles UI events, and only communicates with JS HAL. The components Javascript files, such as the LED and thermistor, can be found in viewer/js-ui/component.

Further details about the architecture can be found in doc/architecture.

### Add a component – Push Button

We are going to create a Push Button and add it the Mbed Simulator.

1. Find a picture of a push button and save it to to viewer/img as push\_button*.png*.
2. Add the following code in viewer/js-ui/viewer.js:

{component: 'PushButton', name: 'Push button', pins: [ 'Button' ]},

inside var components = [ … ]

1. Add the push\_button.js provided file to viewer/js-ui/components.
2. Add the following code in viewer/viewer.html:

<script ="text/javascript" src="/js-ui/components/push\_button.js">. </script>

under   <!-- UI handling -->

### Modify mbed-simulator-hal

We are going to increase the resolution PWM pin's period from ms to us. This upgrade will be useful in several future labs. We need to modify ***viewer/js-hal/gpio.js*** and ***mbed-simulator-hal/mbed-os/target/TARGET\_SIMULATOR/pwmout\_api.c*** files.

In ***viewer/js-hal/gpio.js***,

* 1. Change the variable name period\_ms to period\_us, in the function init\_pwmout()
  2. Change the name of the function: ‘***function period\_ms(pin, pw)’***  to ‘***function period\_us(pin, pw)’***  and the line ‘***declaredPins[pin].period\_us = pw;’*** to ‘***declaredPins[pin].period\_us = pw;***’
  3. Change line 255 ‘***obj.period\_ms = period\_ms;’*** to ‘***obj.period\_us = period\_us;’***

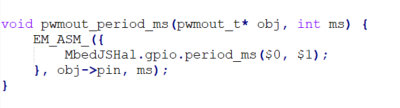
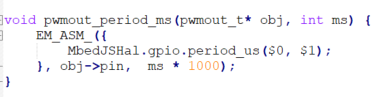
In ***mbed-simulator-hal/mbed-os/target/TARGET\_SIMULATOR/pwmout\_api.c***, in the functions,

***‘void pwmout\_period(pwmout\_t\* obj, float seconds)’***

***‘void pwmout\_period\_ms(pwmout\_t\* obj, int ms)’***

***‘void pwmout\_period\_us(pwmout\_t\* obj, int us)’***

Change ‘***MbedJSHal.gpio.period\_ms($0, $1);’*** to ‘***MbedJSHal.gpio.period\_us($0, $1);’*** and convert the input in the next line to ‘us’. If the input is already in ‘us’ you don’t need to do this (i.e. for the function ‘***pwmout\_period\_ms(pwmout\_t\* obj, int ms)***’ the input it is in ms so to convert this into us you will need to multiply the output by 1000).



## After changing anything in the simulator HAL, you need to recompile the libmbed library, run:

$ rm mbed-simulator-hal/libmbed.bc

## or in windows

$ del mbed-simulator-hal\libmbed.bc

Rebuild your application. libmbed will automatically be generated.

## Limitations

Currently, the simulator has the following limitations:

* Busy-looping (while (1) {}) without calling wait anywhere will make the browser hang. Make sure you use wait () when you want to pause.
* No RTOS. This is a single-threaded environment. You can use mbed-events to make it a bit easier to deal with complex environments, but you can't spin up multiple threads.
* UART and SPI interfaces are not implemented on a low level, if you would like to use the LCD screen or temperature sensor, you can only use the and modify the corresponding high-level library.

Further information about the simulator can be found:

<https://os.mbed.com/blog/entry/introducing-mbed-simulator/>