Create a New Project: Step by Step

Here is a record of steps needed in to create a new System Workbench project in mbed 5.4. This process uses mbed-cli to make a new project. These instructions assumes the DISCO L476VG development board is used.

Make New Empty Project

Run CMD prompt in administrator mode. Use mbed cli to create a new mbed project. Here "**projectname**" is used to represent the actual project name.

```
        mbed new projectname
```

Update Library

To update the mbed libraries, open the command line in administrator mode. Navigate to the project folder.

```
        cd projectname
```

The update command is:

```
        mbed update master
```

It should be noted that mbed-cli program has its own project builder and potentially offers a second way of building the project. But it does require GCC or a valid Keil or IAR compiler license.

Set Compiler (optional)

You only have to set the compiler if you want to use mbed-cli build tools. Skip this step if you are only exporting. You have to tell mbed-cli where GCC compiler lives. Search for:

```
        arm-none-eabi-g++
```

For GCC, the command is GCC_ARM_PATH.

```
mbed config --global GCC_ARM_PATH
"C:\Ac6\SystemWorkbench\plugins\fr.ac6.mcu.externaltools.arm-
none.win32_1.7.0.201602121829\tools\compiler\bin"
```

Build in mbed-cli (optional)

We won't be building in mbed-cli, but the command looks like is:

```
        mbed compile -t GCC_ARM -m DISCO_L476VG
```

Export to System Workbench

Exporting and setting up a new project is a bit tricky. The export tool does not necessarily give you a clean, new working project, though it has gotten better with each release of mbed cli. Exports to other toolchains like Keil and Embitz work better. To start run this command in mbed-cli for your target.

```
        mbed export -i sw4stm32 -m DISCO_L476RG
```

Open System Workbench and import the project.

```
File -> Import -> General -> Existing Projects Into Workspace
```

Click Next. Select root directory, navigate to the folder made above. Usually leave the project where it is, so uncheck Copy Project into Workspace.

Tip: Watch Configurations

When changing all settings, watch the configuration (Release/Debug) at the top. It is possible to change Release and Debug configs separately. If making a change, normally you want to apply the Changes to **All Configurations** and All **Languages**.

Enable parallel Build

This drastically speeds up build time but is disabled by default in a new project. Usually set to Optimal to make use of all cores.

```
Properites->C/C++ Build->Behavior->Enable Parallel Build->Optimal
```

Set FPU

This may or may not be set correctly after making a new project. Obvious Cortex M4 is the only micro that might have an FPU. For M0 or M3 parts choose No unit and Floating point ABI = soft. Right click on project, properties:

```
C/C++ Build -> Settings -> MCU Settings
```

For M0 or M3: choose No Unit and Soft
For M4: Choose the FPU in the list, choose softfp or hard

Enable GCC Newlib

Make sure Newlib and not the nano version is selected by the linker if using a multithread RTOS. Nano specs must **not** be selected.

```
C/C++ Build -> Settings -> MCU G++ Linker -> Miscellaneous
```

delete: `--specs=nano.specs`

No Sys Specs

Add the no system specs flag to the linker:

```
C/C++ Build -> Settings -> MCU G++ Linker -> Miscellaneous
```

add: `-specs=nosys.specs`

Modify Linker File

If you are using a micro not directly supported by a devboard, the linker memory ranges may need to be adjusted to reflect what is on the chip.

Point to Linker File

The linker file for the target is included in mbed-os 5.4:

```
C/C++ Build -> Settings -> MCU G++ Linker -> General
```

```
mbed-os\targets\TARGET_STM\TARGET_STM32L4\TARGET_STM32L476xG\device\TOOLCHAIN_GCC_ARM
```

Post Build Steps

For the Release build, you may need to turn the .elf file into a .bin or an .srec file. And be sure to show the size of the build.

```
C/C++ Build -> Settings -> Build Steps -> Post Build Steps
```

```
arm-none-eabi-objcopy -O binary "${BuildArtifactFileBaseName}.elf"
"${BuildArtifactFileBaseName}.bin" && arm-none-eabi-size -B "${BuildArtifactFileName}" &&
arm-none-eabi-objcopy -O srec "${BuildArtifactFileBaseName}.elf"
"${BuildArtifactFileBaseName}.srec"
```

Optimizations

For Release it is probably best to start with –O0 (none). For Debug config, you have to use –Og, optimize for debug. You have to change setting for both C and C++ compilers.

```
C/C++ Build -> Settings -> MCU GCC Compiler
C/C++ Build -> Settings -> MCU G++ Compiler
```

System Workbench: Search Paths

For both Debug and Release configurations and for all toolchains, you have to add each search path in your project. These may or may not be set correctly by mbed-cli export. Every sub-folder inside mbed-os that your project needs must be added to the search path.

```
C/C++ General -> Paths and Symbols
```

The form is:

```
${ProjDirPath}/.\mbed-os
```

Exclude from Build

So mbed-os includes files for all targets which will confuse gcc if it tries to build them. In the project tree, go through every folder and sub-folder, and exclude folders that are clearly not for your target micro or your target compiler. Also exclude any test folders. This will be a lot of folders and files. Right click on the file or folder you do not need:

```
Resource Configuration -> Exclude from Build
```

Include mbed_config.h

This file is not included in the project in default export. You can include it inside mbed.h.

```
#ifndef MBED_H
#define MBED_H

#include "mbed_config.h"
```

However there are some files in mbed-os that cannot see mbed.h. A better solution is to include mbed_config.h globally as a compiler argument. Use -include to add mbed_config.h to gcc and gpp under project settings.

```
C/C++ Build -> Settings -> MCU GCC Compiler -> Includes -> Include Files
C/C++ Build -> Settings -> MCU G++ Compiler -> Includes -> Include Files
```

Change Default Baud Rate

Add to build symbols to change default baud rate from 9600 to faster 115200. This is important to speed up printf debug statements. In mbed os 5.4 this is defined in a header, mbed_config.h

```
MBED_CONF_CORE_STDIO_BAUD_RATE 115200
```

Prevent Eclipse from Modifying Your .gitignore

You can also configure Eclipse to automatically ignore derived resources, e.g. class files. This is important otherwise it keeps ignoring the Release build. And we want to save the final binaries with the project.

```
Window → Preferences → Team → Git → Projects → Automatically ignore derived resources ..
setting.
```

STLink Openocd Script .cfg

For a development board like Nucleo or Disco, this is all setup and you can skip.

For a production PCB, you have to modify the Openocd script for stlink v2 not v2-1. V2.1 is the chip on Nucleo dev boards. Locate the script by browsing to the debugging setup. Right Click the project folder and select Debug As. Choose Ac6 STM32 Debugging to make a new debug file. On the Debugger Tab go to Script and use Local script. The script is a file named something like:

similar to: `stm32l476e_stlink.cfg`

You may have to create a custom file for the specific micro. But there are lots of examples for STM32 parts. Big difference is `workareasize` Which refers to RAM on target that can be used to help debugging.

http://openocd.org/doc/html/CPU-Configuration.html

Find the file, make a copy and put in the main project. Edit the file to point to v2 debugger.

change this: `source [find interface/stlink-v2-1.cfg]`
to this: `source [find interface/stlink-v2.cfg]`

Adjust Target Board

If the micro is not directly supported on a devboard you will need to update Target information. Make a new board with the specific micro being used so that the flash and ram data is right. Linker file needs to be adjusted manually.

```
        C/C++ Build -> Settings -> Target
```

Build and Inspect Errors

From here the strategy is to build the project and inspect the build errors. Then track them down one by one and resolve until you can compile the project successfully.