Link 1: Updated: Using The stm32f4 discovery board with Mac OSX 10.9 Mavericks (Link 2 follows)

My first posting about the stm32f4 discovery board and Mac OSX 10.9 Mavericks became a little bit outdated. That is because the used software has been updated since then. The following sotware is used in this tutorial: But wait with the download. We will do the installation step by step together.

1. Mac OSX Mavericks (10.9.3)
2. Xcode Command Line Tools (5.1.0.0.1.1396320587)
3. homebrew, package manager for OSX (0.9.5)
4. Eclipse Kepler for C/C++ Developer (4.3.2)
5. ST-Link (Commit ID 4782ab0ca736202e31d1afcbc62ce469a5daedf7)
6. gcc-arm-none-eabi GCC arm crosscompiler Toolchain for Mac (4.8-2014-q1-update)

Xcode Command Line Tools

First of all, make sure you have installed Apples free Xcode command line tools installed or the full Xcode IDE. If this is not the case please check this website.

http://railsapps.github.io/xcode-command-line-tools.html

Install Brew and Dependencies

i recommend to install homebrew which is a package manager that will help us to install some dependencies. Homebrew is package manager for OS X similar to macports or fink. You could also use macports to download the dependencies by your own.
After you have installed your favorite package manager open your preferred Terminal application and run the command:

```
$ brew install libusb autogen automake wget pkg-config
```

Stlink Utility

Now we are ready to downlaod and install the stlink utility. It is used for programming and debugging different micro-controllers. First download/clone the source code from github. Therefor we create a new directory.

```
$ mkdir ~/dev/
$ cd ~/dev/
```

Clone the source from Github into the new created directory. This can take a while, depending on your internet connectivity.

```
 $ git clone https://github.com/texane/stlink.git
 $ cd ~/dev/stlink/
```

Now we can configure and compile the source code of the stlink utility.

```
$ ./autogen.sh
$ ./configure
$ make
```

a couple of binaries are generated where the most interesting for use is

- st-util

varify that everything worked fine. Connect the STM32 Board to your Mac and in the stlink directory fire the command

```
$ ./st-util
2014-06-01T18:20:05 INFO src/stlink-common.c: Loading device parameters....
2014-06-01T18:20:05 INFO src/stlink-common.c: Device connected is: F4 device, id
 0x10016413
2014-06-01T18:20:05 INFO src/stlink-common.c: SRAM size: 0x30000 bytes (192 KiB)
, Flash: 0x100000 bytes (1024 KiB) in pages of 16384 bytes
Chip ID is 00000413, Core ID is  2ba01477.
Target voltage is 2878 mV.
Listening at *:4242...
```

If you get a different output you should check if the board is recognized by your Mac.

Crosscompiler for Mac: gcc-arm-none-eabi

Download a pre compilted version GNU Tools for ARM Embedded Processors into the same dev directory. There are versions available for windows, mac and linux.

```
$ cd ~/dev/
$ wget https://launchpad.net/gcc-arm-embedded/4.8/4.8-2014-q1-update/+download/g
cc-arm-none-eabi-4_8-2014q1-20140314-mac.tar.bz2
$ tar xvf gcc-arm-none-eabi-4_8-2014q1-20140314-mac.tar.bz2
```

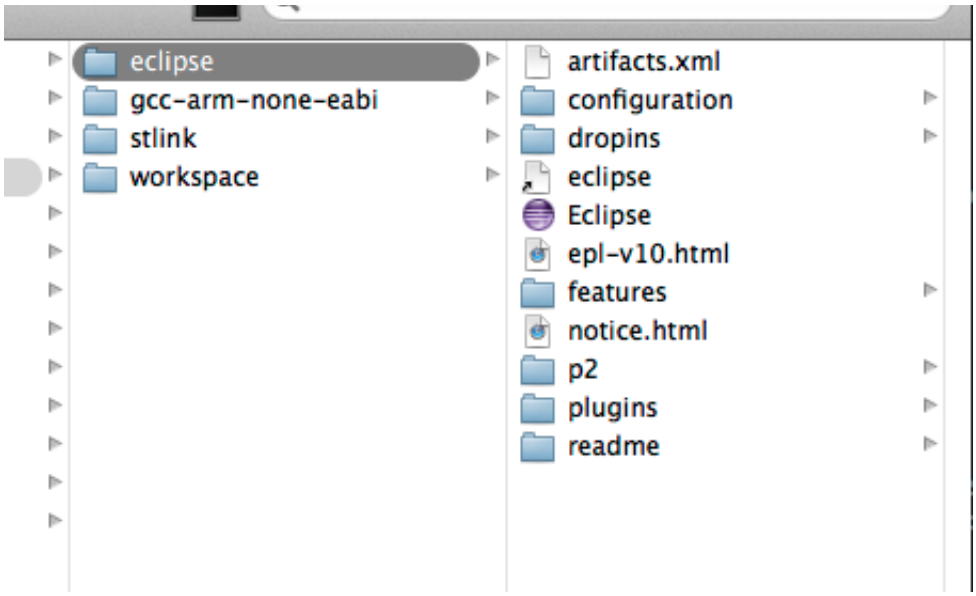
dont worry, we will realy need all this tools.

Install and setting up the Eclipse IDE

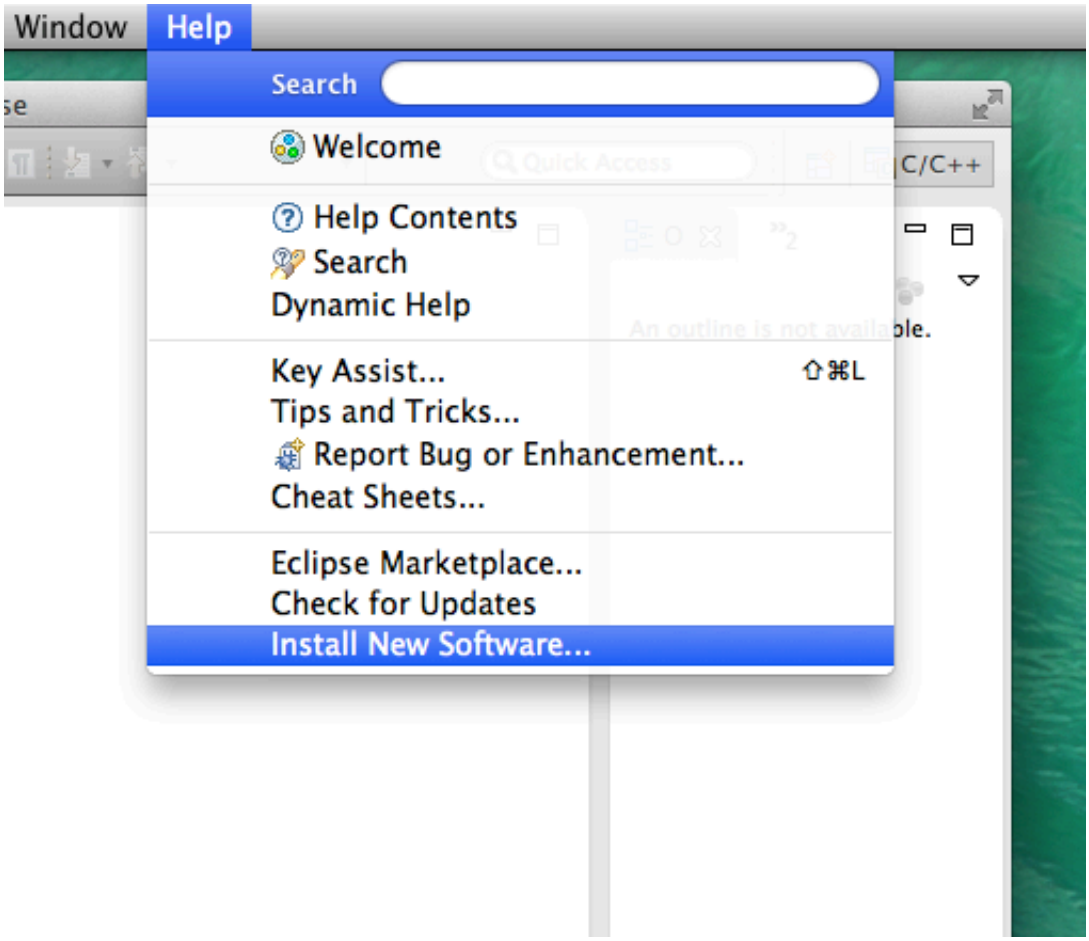
I will use the GUI debugger front-end given by Eclipse. Download Eclipse as C++ development environment from the eclipse.org website or do it directly from within the command line

```
$ cd ~/dev/
$ wget http://ftp.osuosl.org/pub/eclipse//technology/epp/downloads/release/keple
r/SR1/ eclipse-cpp-kepler-SR1-macosx-cocoa-x86_64.tar.gz
$ tar xvf eclipse-cpp-kepler-SR1-macosx-cocoa-x86_64.tar.gz
$ ./eclipse/eclipse
```

Eclipse Kepler will start and we can create a new Workspace. Choose a workspace location you prefer. Since i put everything in my ~/dev/ folder it looks like this:

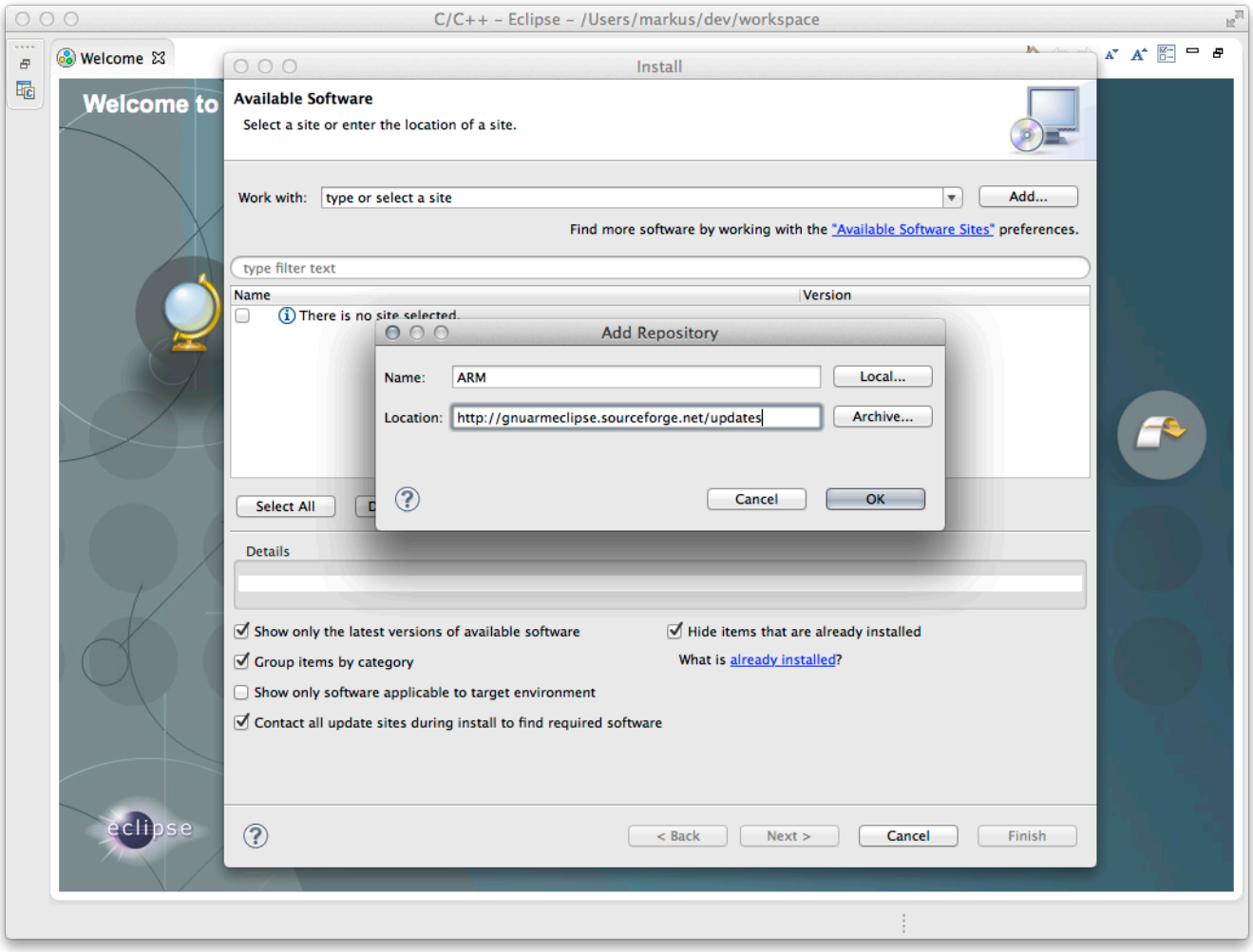


In the next step we will install some plugins in Eclipse. These plugins will handle the cross compilation and provides the libs for the stm32f32 and other boards.

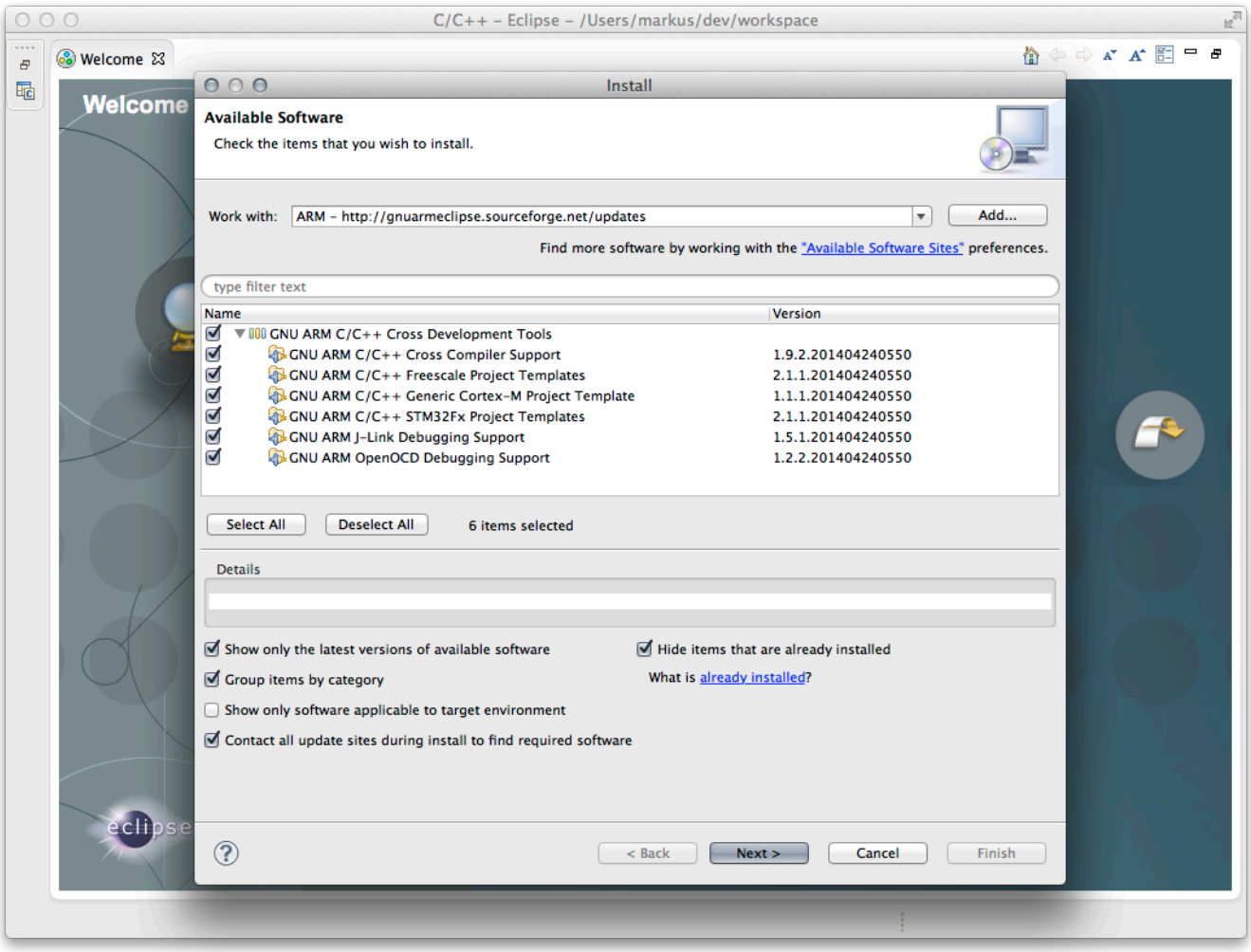


First add the source to your Eclipse. It will install templates for some boards, as well for the stm32f discovery. Add the link http://gnuarmeclipse.sourceforge.net/updates as show in the following two

screenshots.

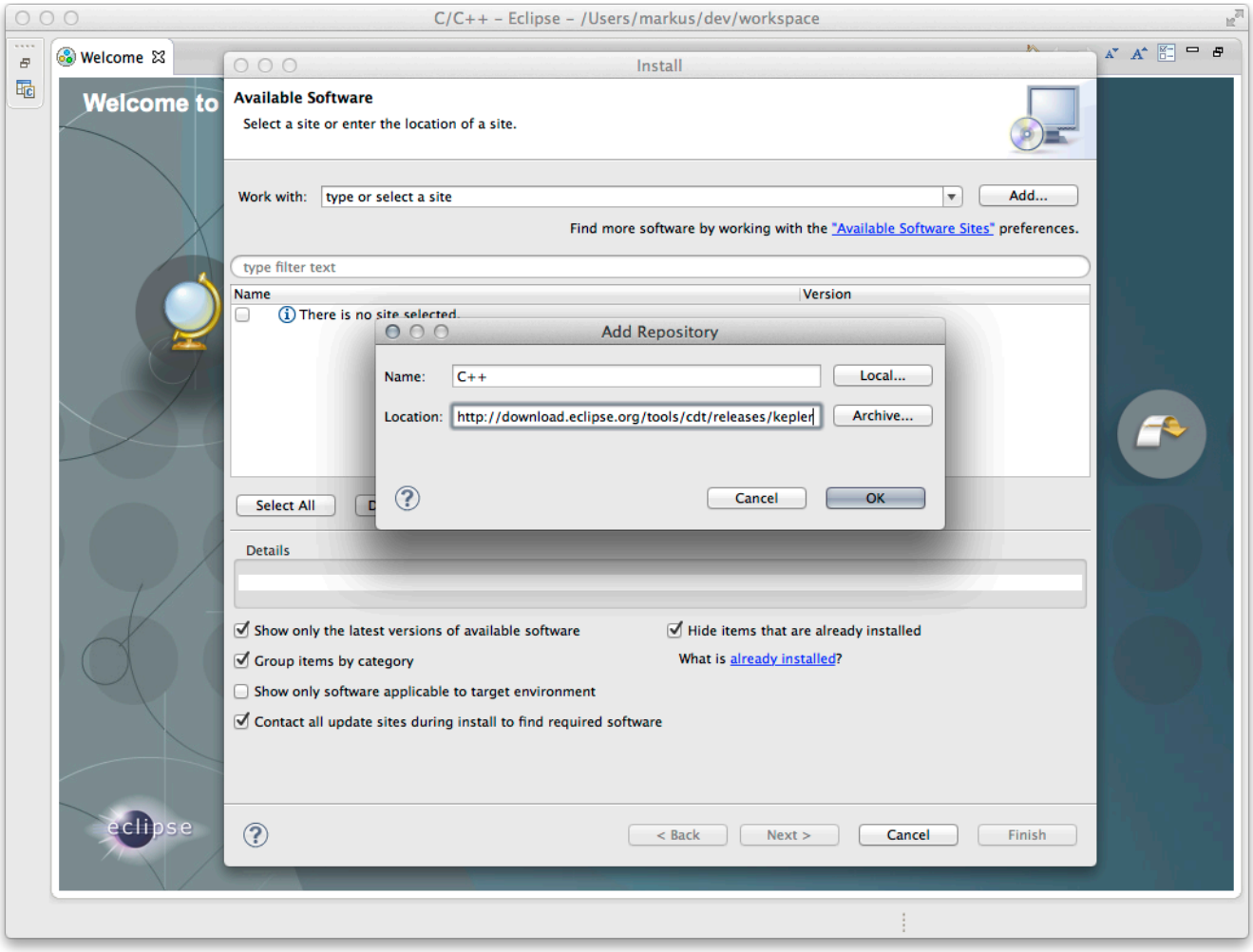


and just select all

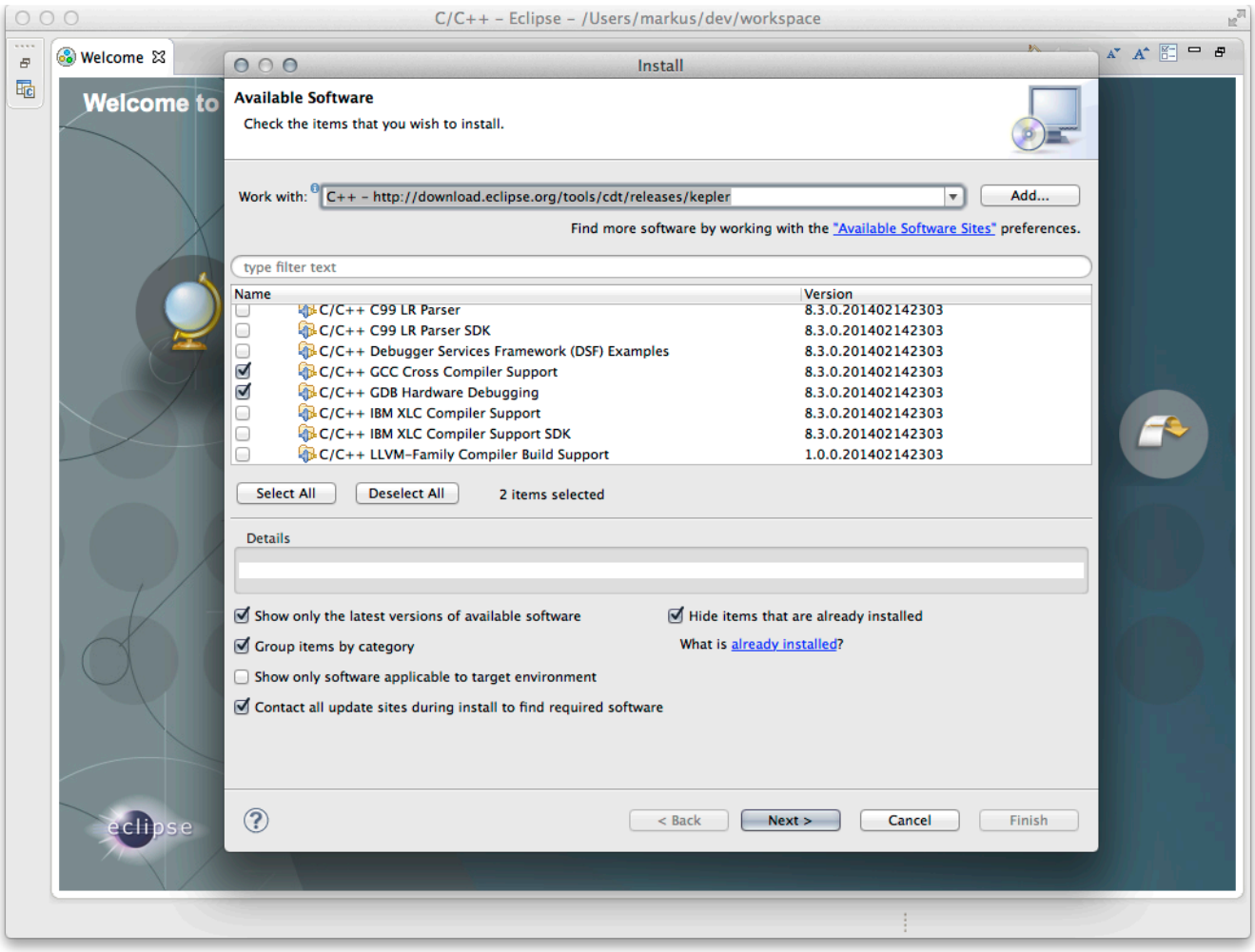


The next package is the following one

http://download.eclipse.org/tools/cdt/releases/kepler

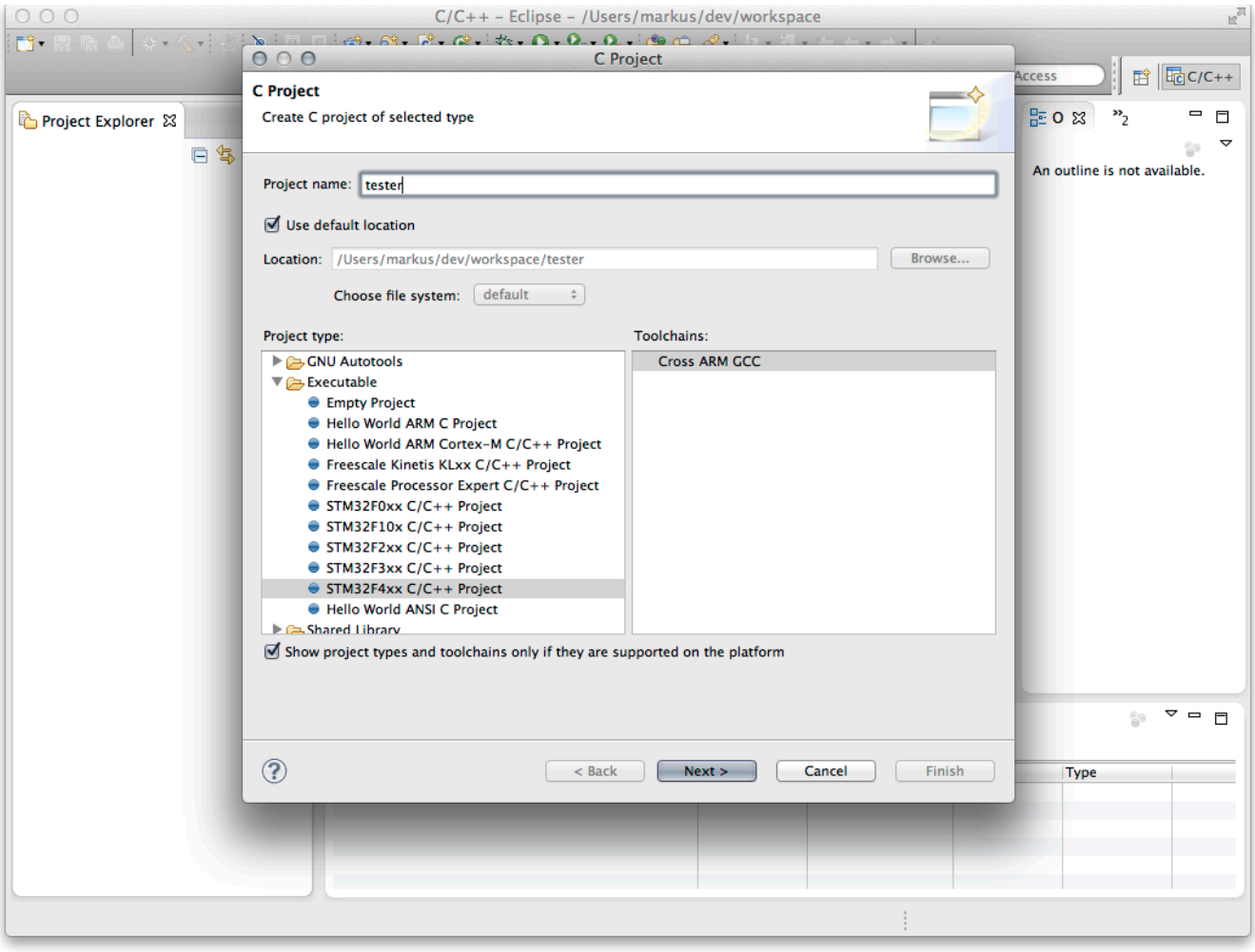


and this time just select the two packages show in the next screenshot. The other packages can be installed as well, but are not necessary in order to debug your code on the device.

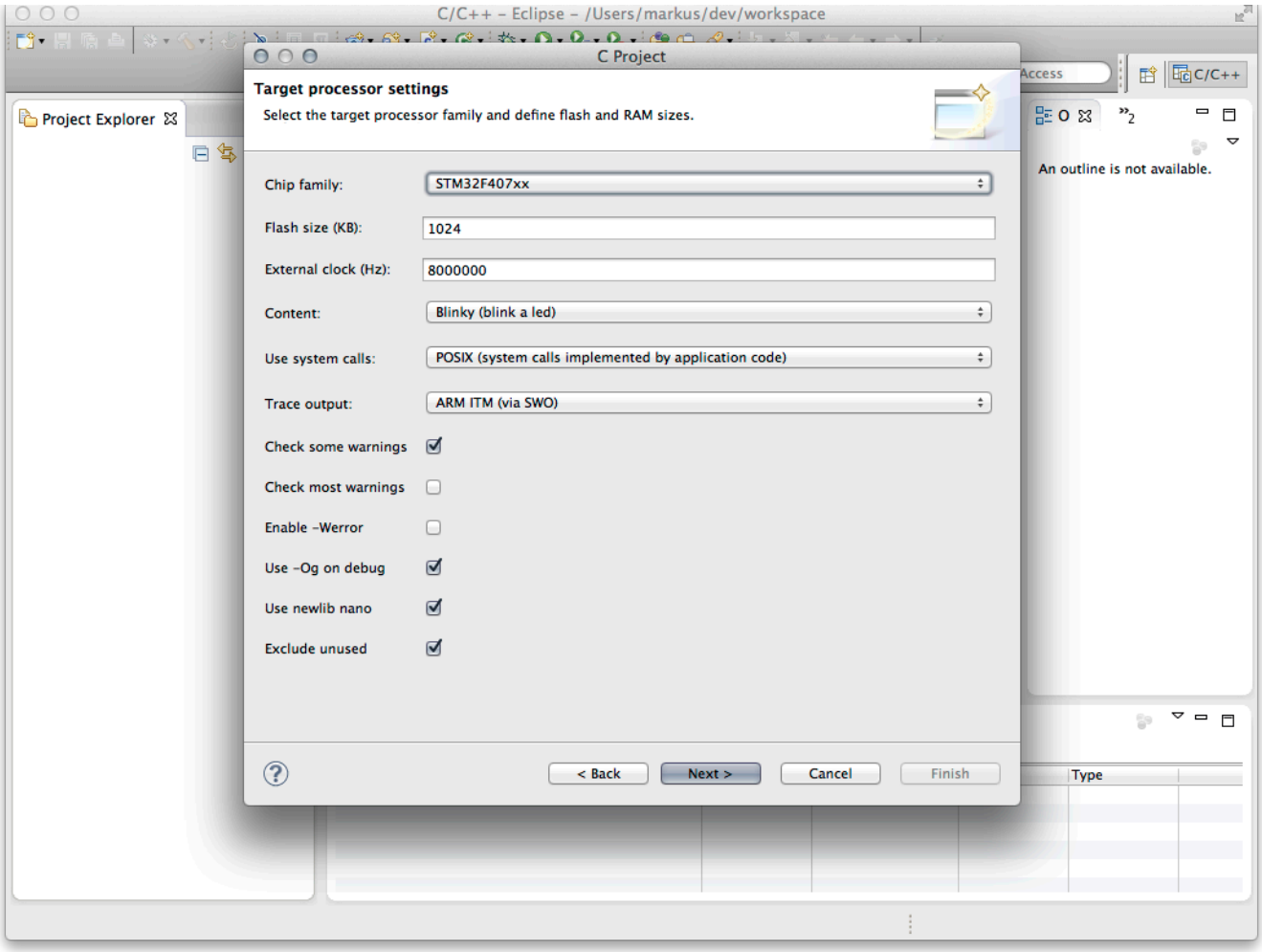


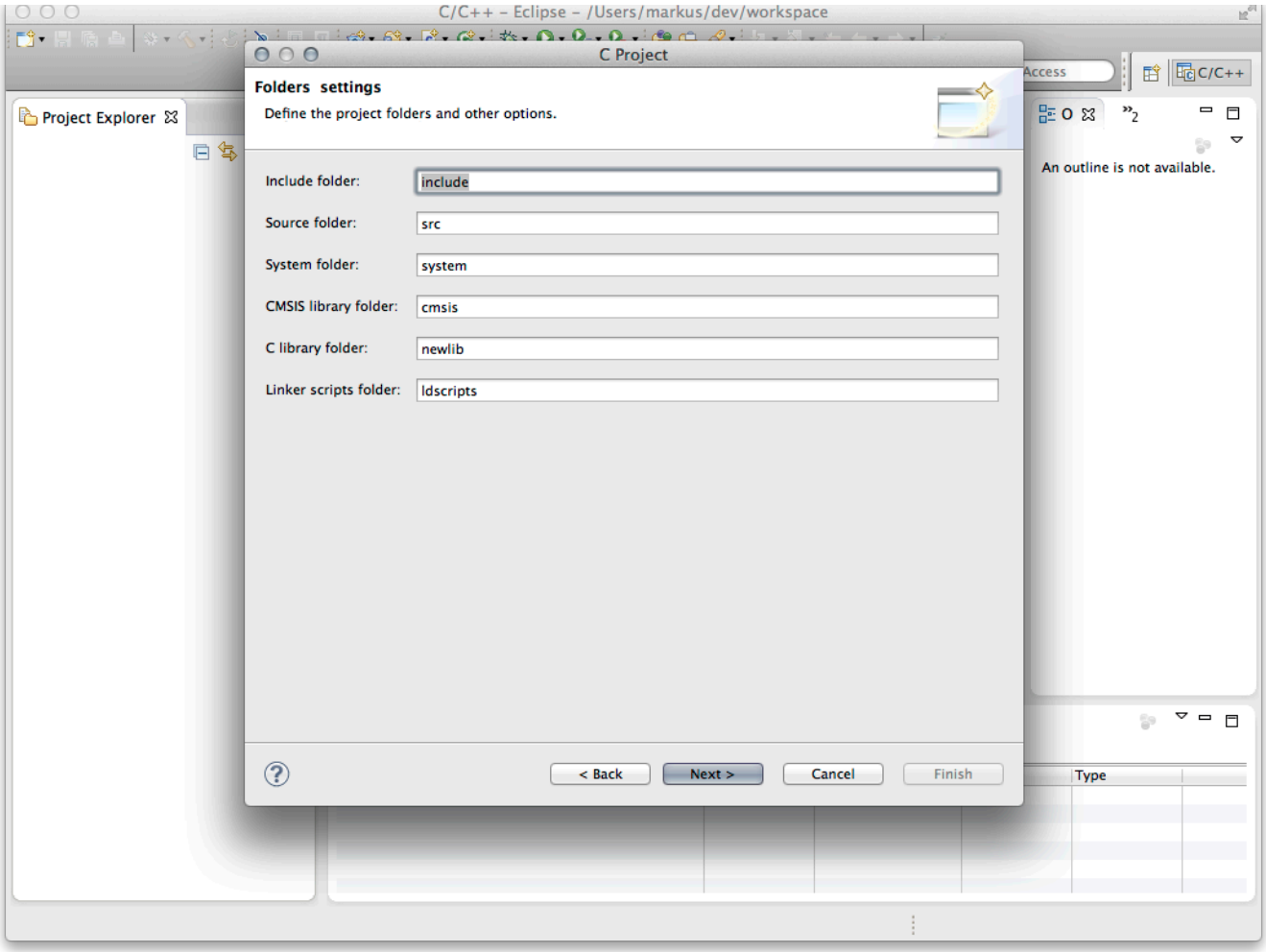
create an new blinking LED project.

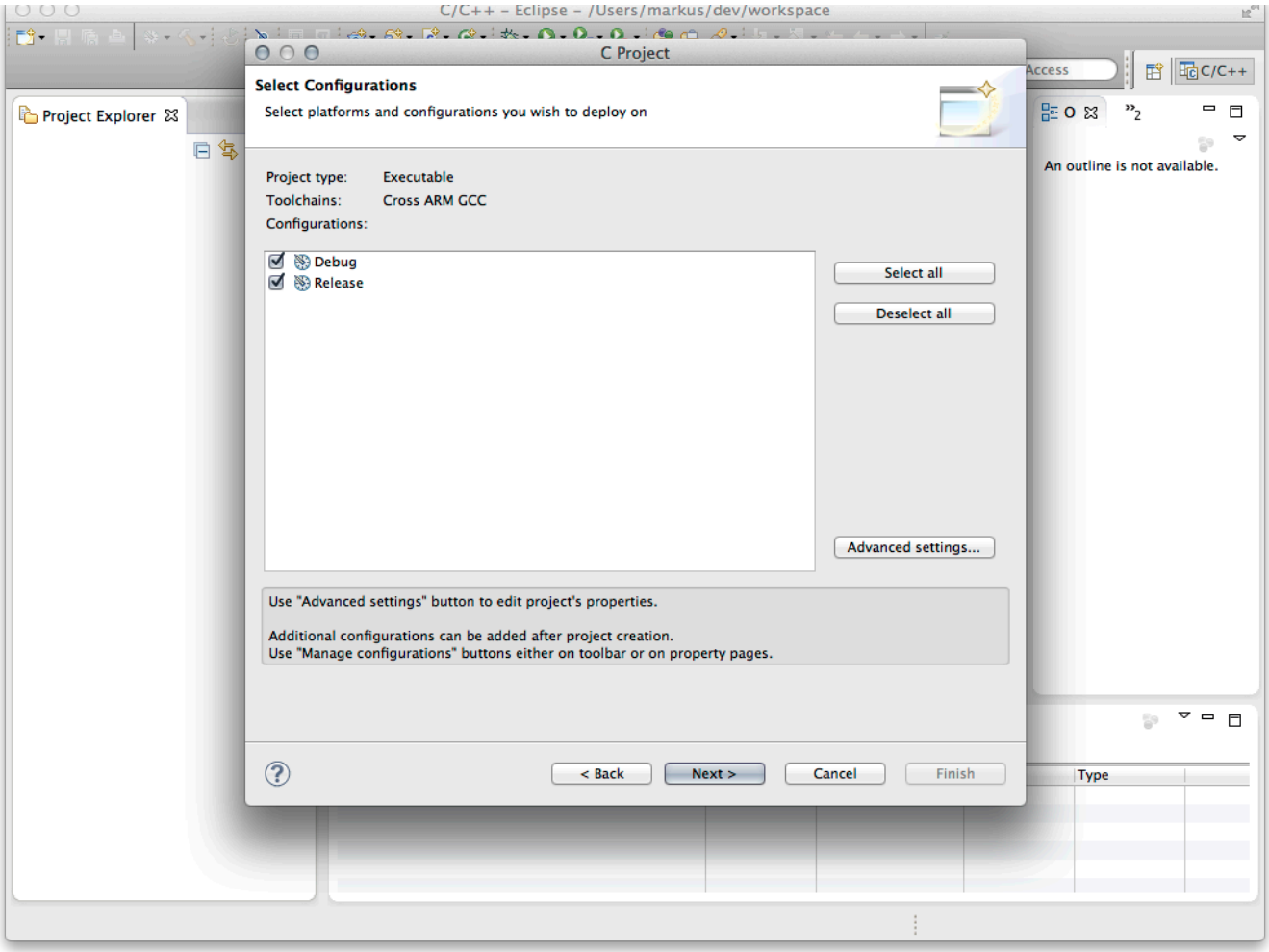
Go to the menu of eclipse and create a new C-Project. Find the folder Executable and choose the "STM32F4XX C/C++ Project".

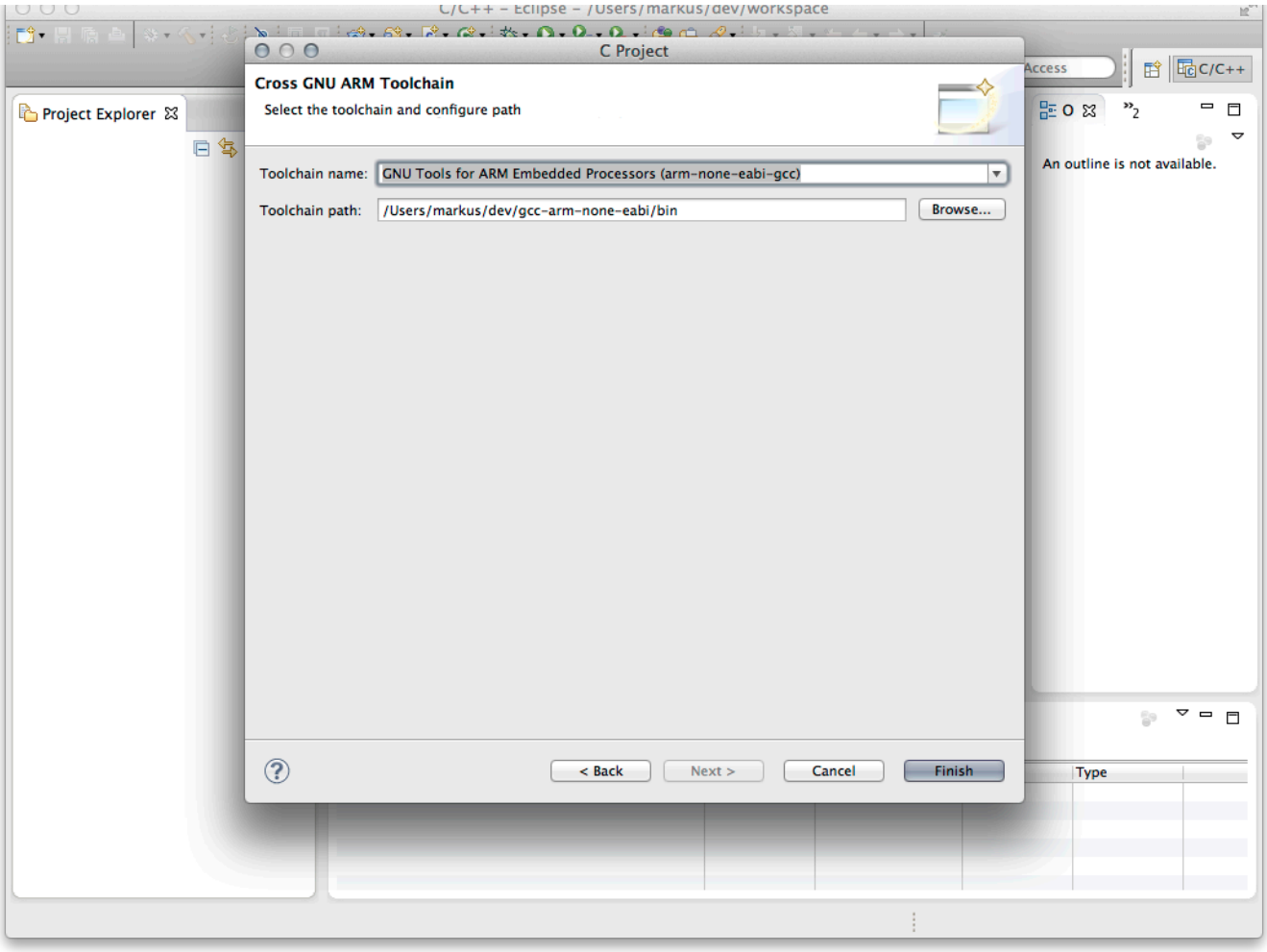


leave the rest of the setting allown except the last settings page where you point to your compiler.

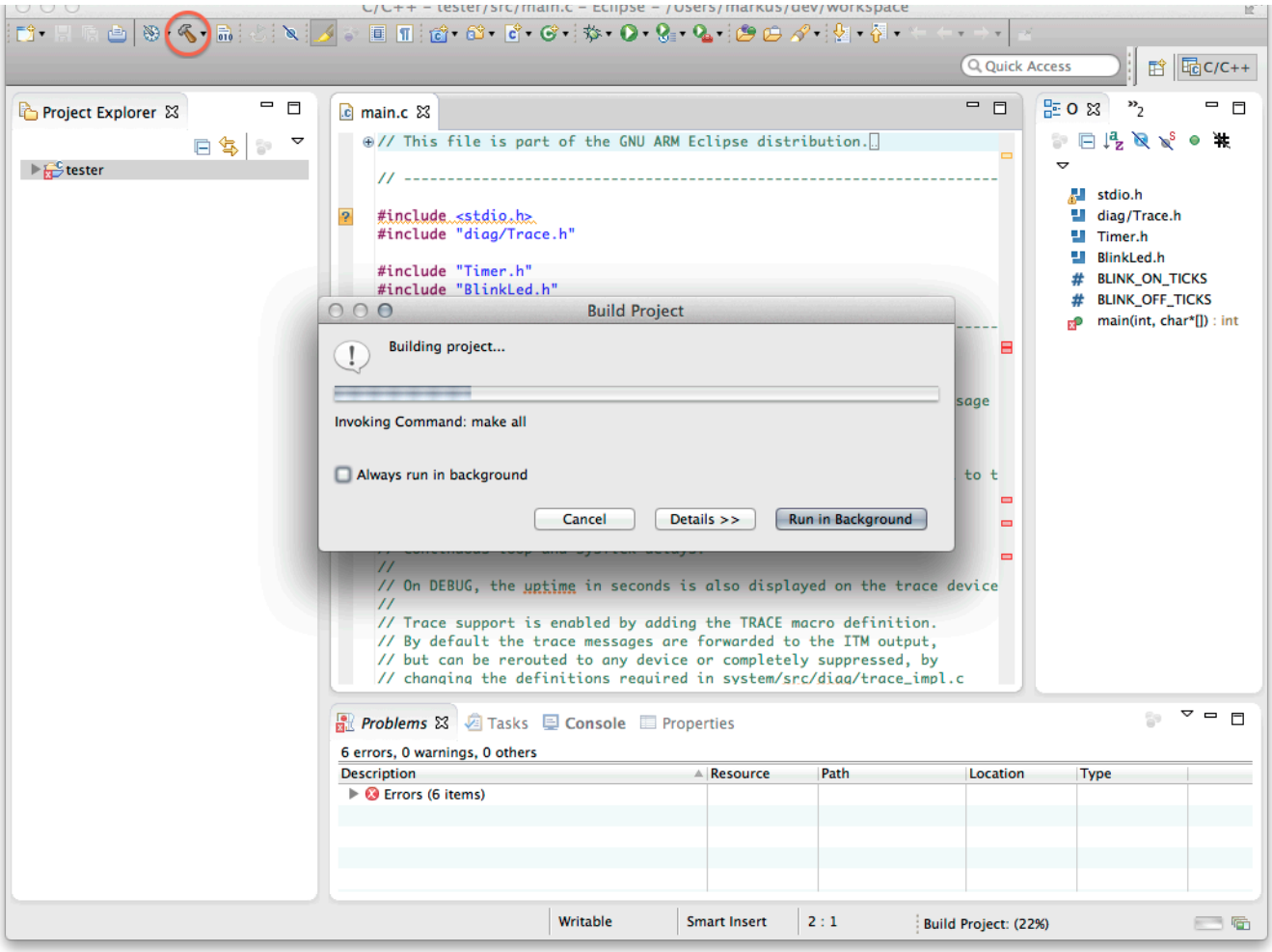

C/C++ - Eclipse - /Users/markus/dev/workspace
C Project
Target processor settings
Select the target processor family and define flash and RAM sizes.
Chip family:
STM32F407xx
Flash size (KB):
1024
External clock (Hz):
8000000
Content:
Blinky (blink a led)
Use system calls:
POSIX (system calls implemented by application code)
Trace output:
ARM ITM (via SWO)
Check some warnings
Check most warnings
Enable -Werror
Use -Og on debug
Use newlib nano
Exclude unused
< Back
Next >
Cancel
Finish
Project Explorer
An outline is not available.
Type


C/C++ - Eclipse - /Users/markus/dev/workspace
C Project
Folders settings
Define the project folders and other options.
Include folder:
include
Source folder:
src
System folder:
system
CMSIS library folder:
cmsis
C library folder:
newlib
Linker scripts folder:
ldscripts
< Back
Next >
Cancel
Finish
Project Explorer
An outline is not available.
Type


C/C++ - Eclipse - /Users/markus/dev/workspace
C Project
Select Configurations
Select platforms and configurations you wish to deploy on
Project type: Executable
Toolchains: Cross ARM GCC
Configurations:
Debug
Release
Select all
Deselect all
Advanced settings...
Use "Advanced settings" button to edit project's properties.
Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.
< Back
Next >
Cancel
Finish
Project Explorer
An outline is not available.
Type

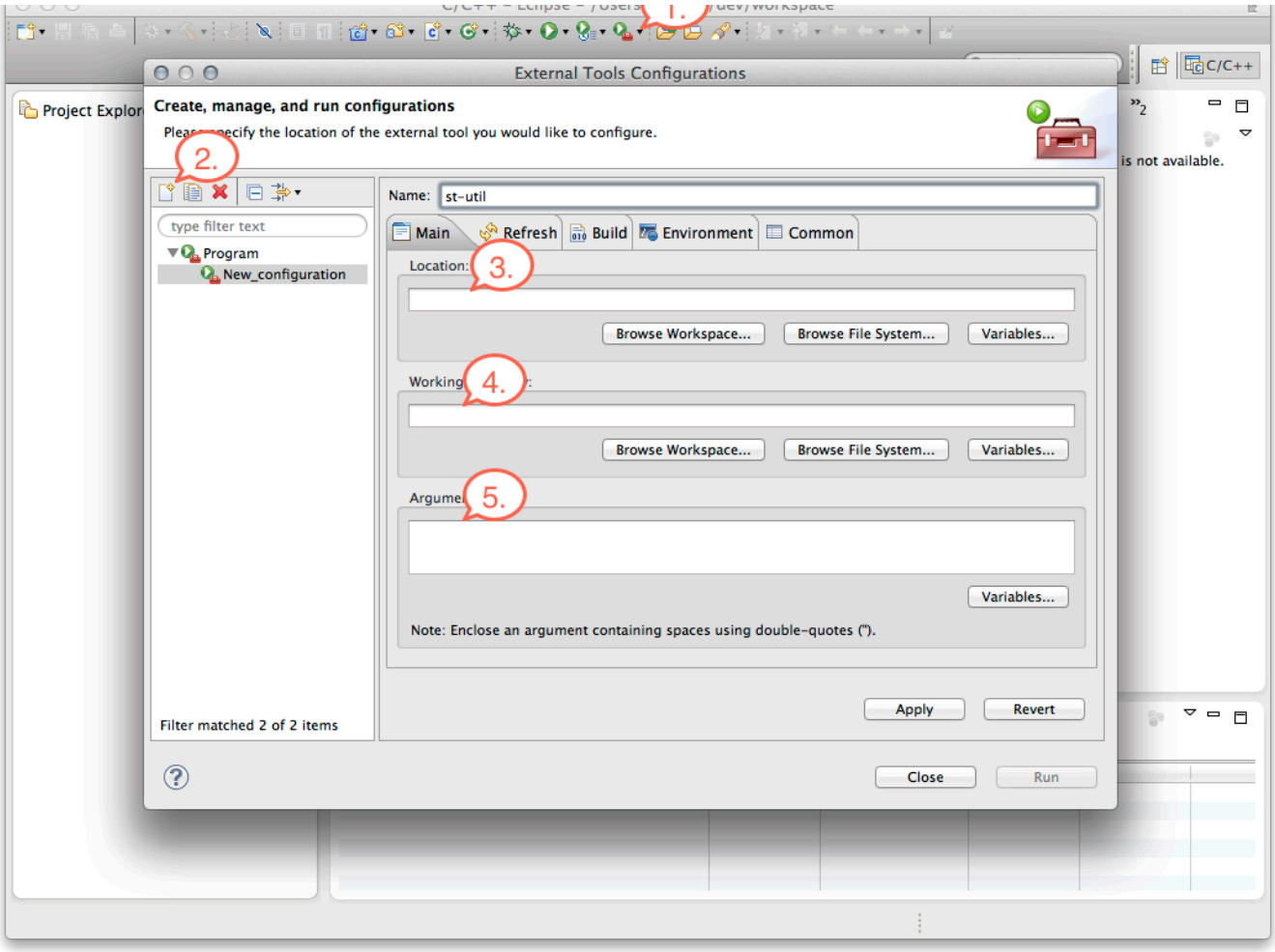


A new project is created that will show errors. Just ignore them and compile the project as shown below. The errors will disappear.

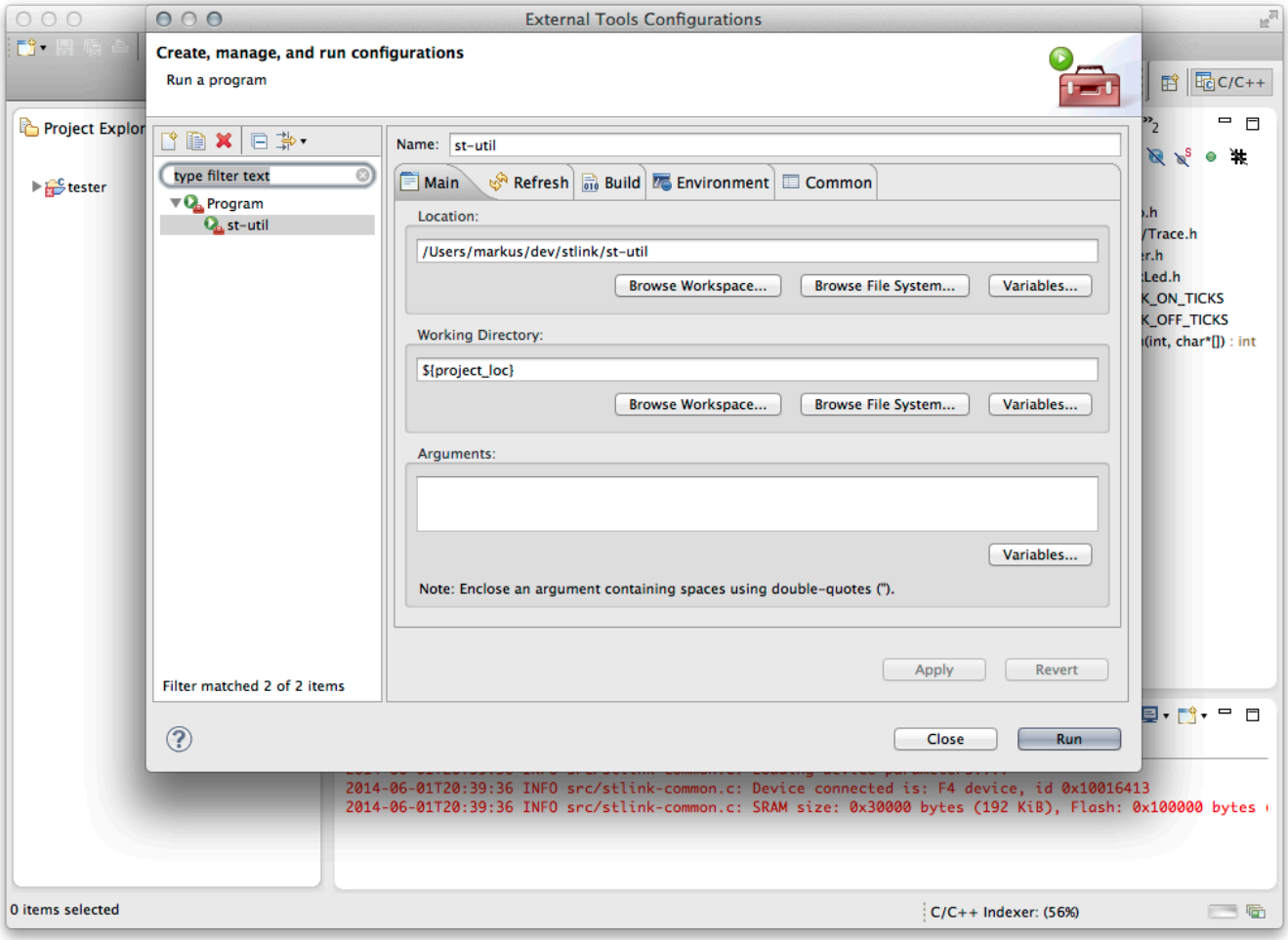


Integrate ST-Util

In the next step we will configure Eclipse to run st-util as external programm.

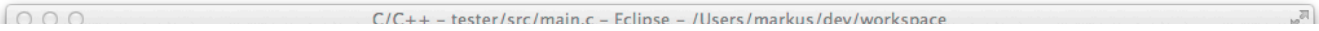


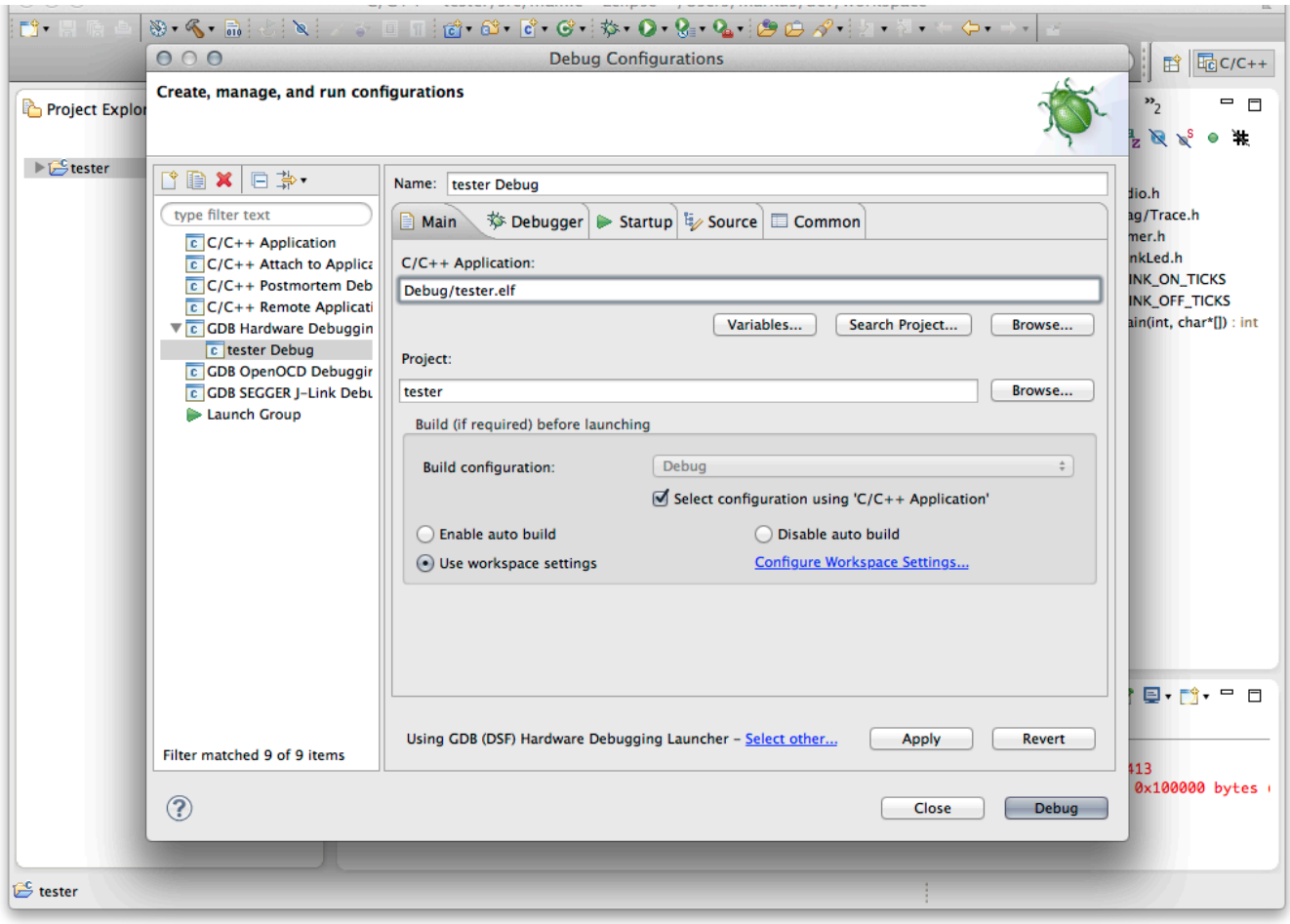
1. go to the external run configuration menu
2. create a new external run configuration
3. point to the location of the st-util in my case (/Users/markus/dev/stlink/st-util)
4. that the working directory to the variable ${project_loc}
5. leave the arguments empty.
6. done



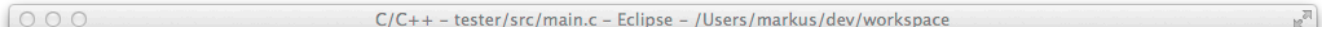
Lets Debug.

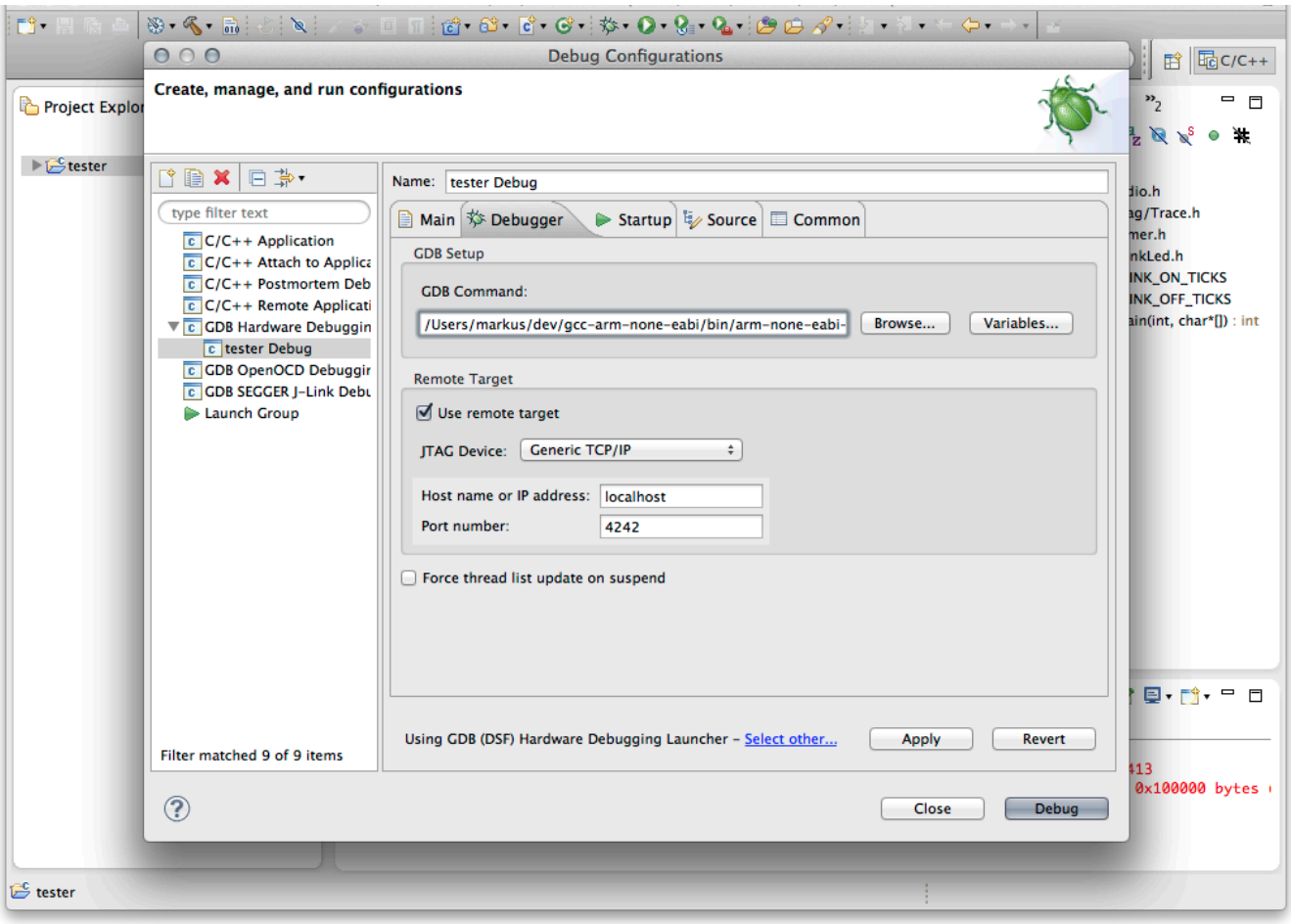
We still need to integrate the debugger with eclipse. Go to the debug configuration menu





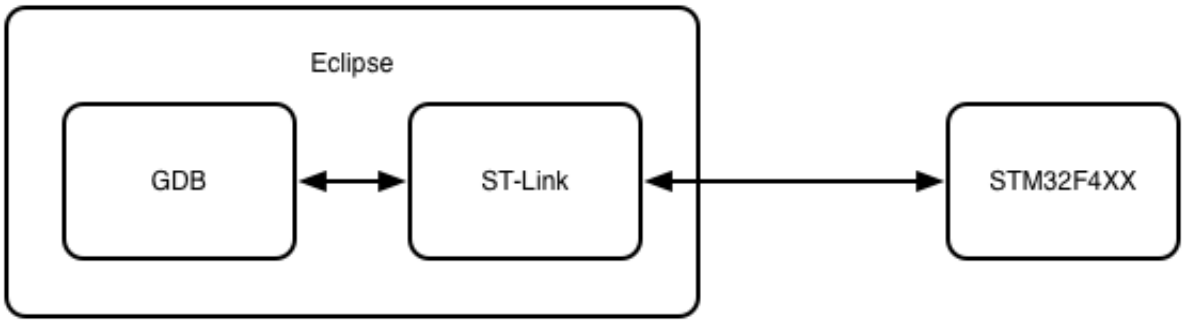
in the debugger tab, choose the location ~/dev/gcc-arm-none-eabi/bin/arm-none-eabi-gdb





and configure the JTAG device as generic TCP/IP running on localhost and port 4242. This is the already configured st-link util. That musst be started befor you think about debuging. Now hit debug and you are ready.

The setup is now a little bit like this.



Thats it.

You can now compile and debug your project. If you are not able to compile the project, close and open eclipse and try again. Befor you can debug you have to start the external tool.

se - /Use

then start the debugger and you are ready to go. If you have any questions, please ask and comment. And please notice that in the screenshots i havn't been conseqently in naming. The project is sometimes testing or tester.

Conclusion

The STM32F4xx discovery board is a cheap microcontroller that offers a lot of GPIO and is powerful enough to emulate a SEGA Master System. It can be set up with free and open source software. There are extensions boards available that brings Wifi, Ethernet and even touch screens to this board. The STM32F4 Discovery board is a great and cheap micro-controller that makes it easy to step into the world of embedded devices.

/**********

**********/

Link 2: How to switch from StmHAL to stdPeripheralLibrary for STM32F4 in eclipse.

Create a new Blinky project:

Let's delete unnecessary files, which are related to HAL Libraries and replace them with Standard Peripheral Library alternatives.

Under the project folder, '*src*' includes '*_initialize_hardware.c'* file, which configures MCU at the startup, with some functions from HAL Library. Delete this file.

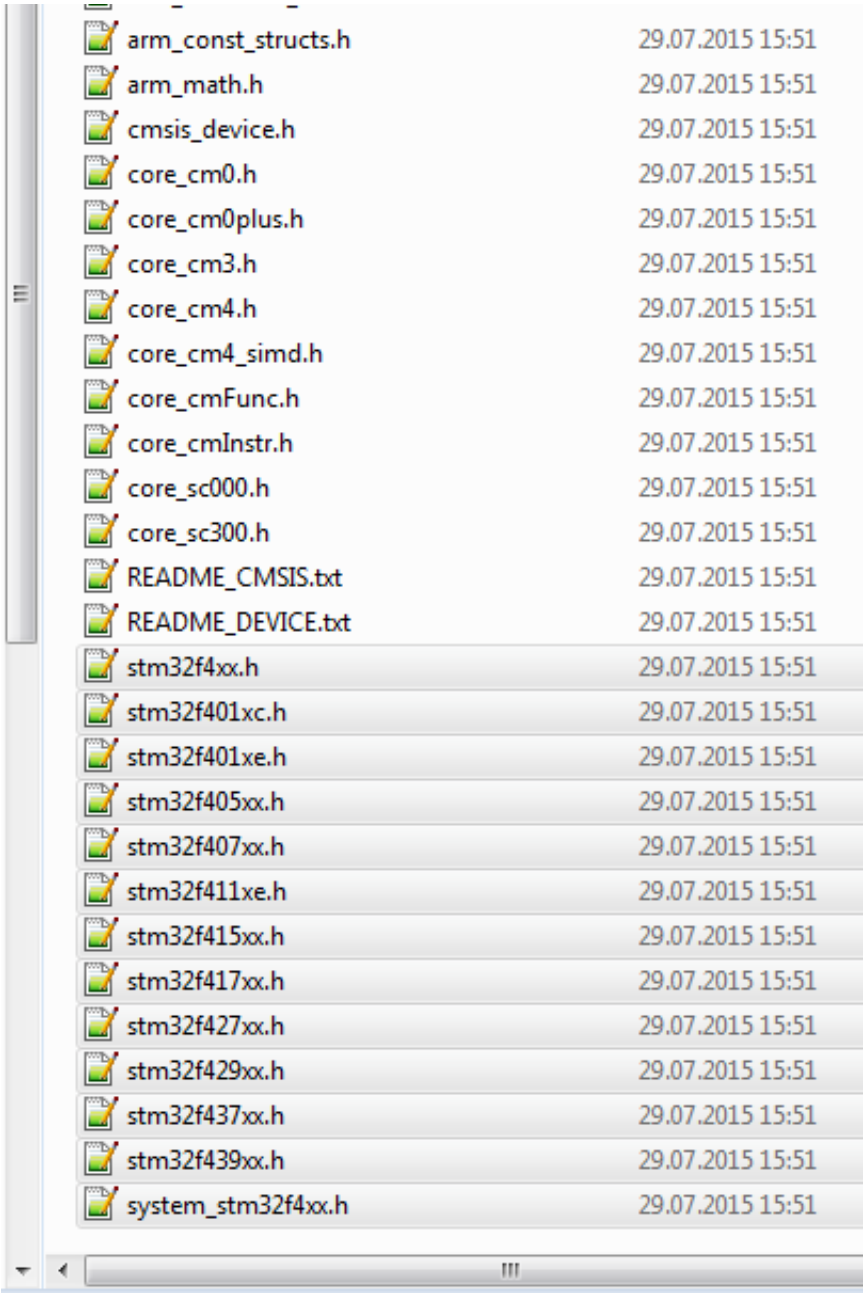
From 'include' folder, delete '*stm32f4xx_hal_conf.h'*, copy '*stm32f4xx_conf.h*'

from

from
‘STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Project\STM32F4xx_StdPeriph_Templates’.

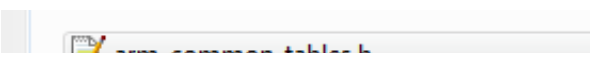
Go to ‘system/include’ folder.

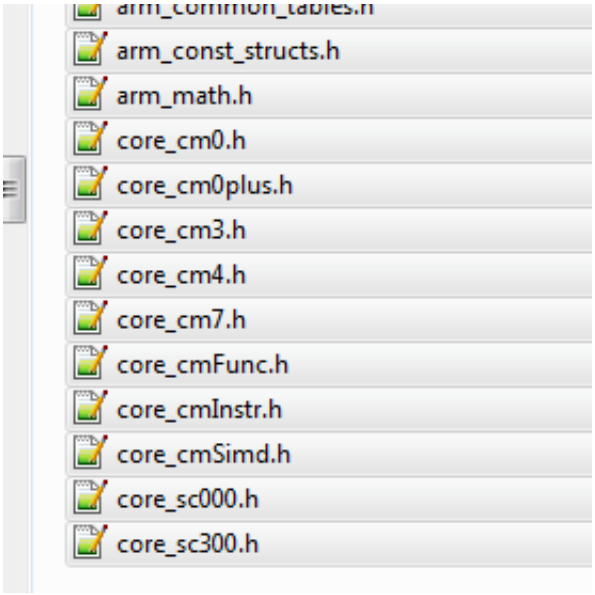
Under ‘cmsis’ folder, delete following highlighted files:



After deleting them, copy '*stm32f4xx.h*' and '*system_stm32f4xx.h*' from STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Libraries\CMSIS\Device\ST\STM32F4xx\Include to '*cmsis*' folder.

Also, copy all following files from 'STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Libraries\CMSIS\Include' to '*cmsis*' folder, and replace them.





Now, move to ‘system/include/stm32f4-hal’ folder and delete all files. Also, change the folder name to ‘stm32f4’. Copy all files from ‘STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Libraries\STM32F4xx_StdPeriph_Driver\inc’ to this folder.

System include file modifications are done. So go to ‘system/src’ folder. Under ‘cmsis’ folder, replace ‘*system_stm32f4xx.c*‘ file from the one in ‘STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Project\STM32F4xx_StdPeriph_Templates’ folder.
This ‘system_stm32f4xx.c’ file is created for a board with 25MHz crystal. If your board has a 8MHz one (like Discovery) you have to change HSE Frequency and PLL_M variables in it. Under ‘*PLL Parameters*‘ part, find ‘*#if defined(STM32F40_41xxx)*‘ and change*PLL_M* definition from *25* to *8*. Also, change ‘*#define PLL_N 360*‘ to*336*.

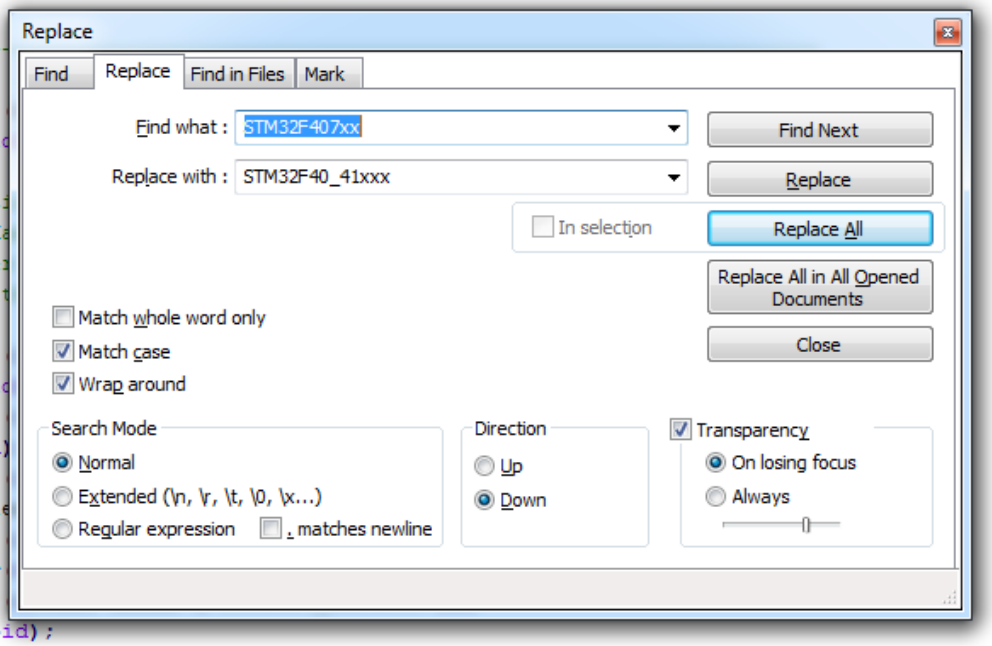
Open ‘*vectors_stm32f4xx.c*‘ file, which includes vector declarations of Interrupt

Handlers. We will change MCU family names, since HAL and Standard Library

uses different ones.

Assuming we are using STM32F407 MCU, change all '*STM32F407xx*' occurrence with '*STM32F40_41xxx*'. Standard Library uses STM32F40_41xxx definition.

```
1   //
2   // This file is part of the µOS++ III distribution.
3   // Copyright (c) 2014 Liviu Ionescu.
4   //
5
6   // ----------------------------------------------------------------------------
7
8   #include "cortexm/ExceptionHandlers.h"
9
10  // ----------
11
12  void __attribute__ (
13  Default_Handler(voi
14
15  // Forward declarati
16  // to the Default_Ha
17  // defines a handle
18  // precedence over t
19
20  void __attribute__
21  WWDG_IRQHandler(voi
22  void __attribute__
23  PVD_IRQHandler(void)
24  void __attribute__
25  TAMP_STAMP_IRQHandle
26  void __attribute__
27  RTC_WKUP_IRQHandler
28  void __attribute__
29  FLASH_IRQHandler(void);
30  void __attribute__ ((weak, alias ("Default_Handler")))
31  RCC_IRQHandler(void);
32  void __attribute__ ((weak, alias ("Default_Handler")))
33  EXTI0_IRQHandler(void);
```

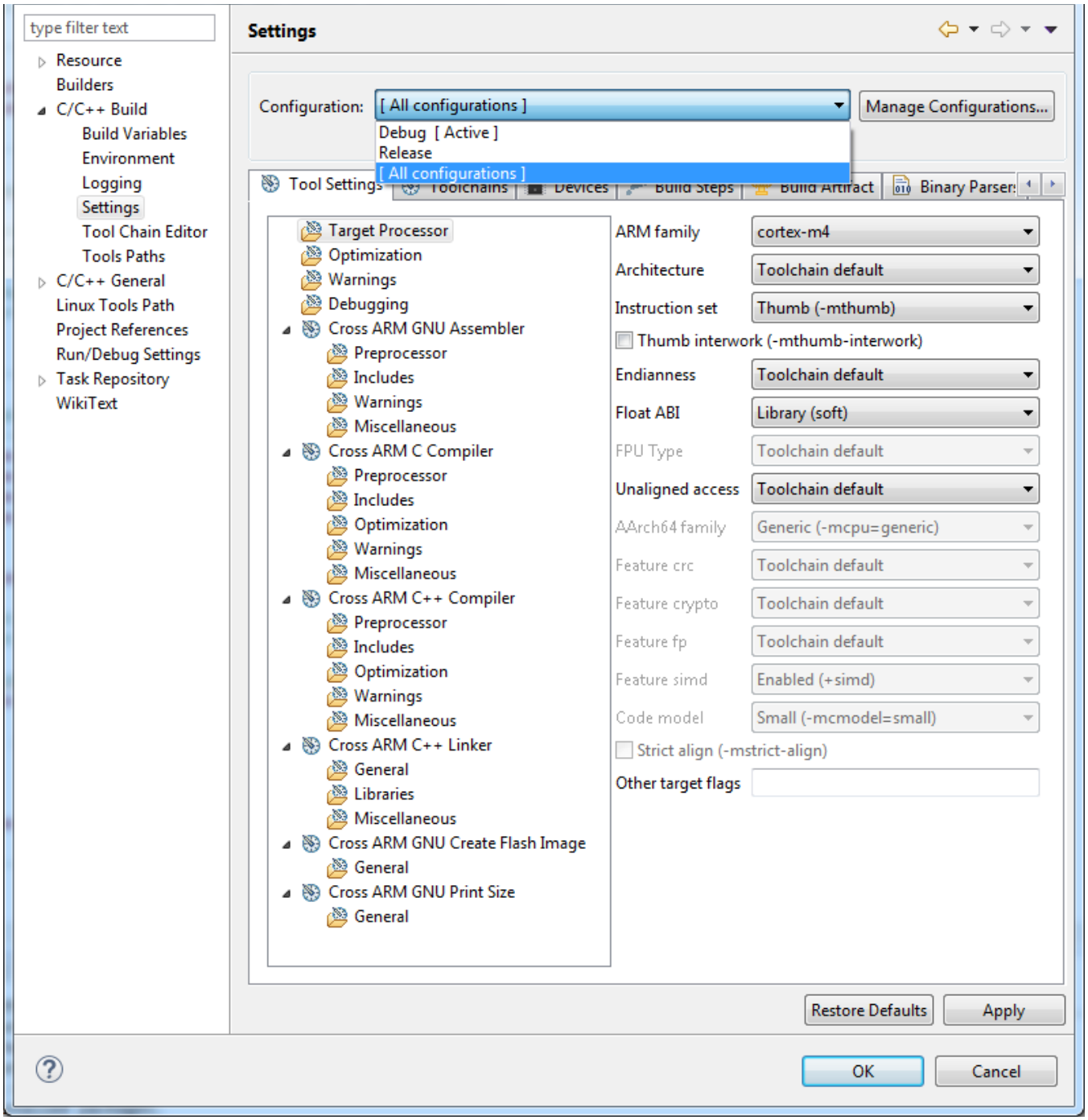


Go to 'system/src/stm32f4-hal' folder, delete all content and change its name to 'stm32f4'. Copy all files from 'STM32F4xx_DSP_StdPeriph_Lib_V1.5.1\Libraries\STM32F4xx_StdPeriph_Driver\src' to this location.

Changing files and modfying them are done. Now, we will make a few changes

from project settings.

In Eclipse, right click StdLibProject and open ‘Properties’. Select ‘Settings’ under ‘C/C++ Build’ part. From Configuratins, choose [All configurations].

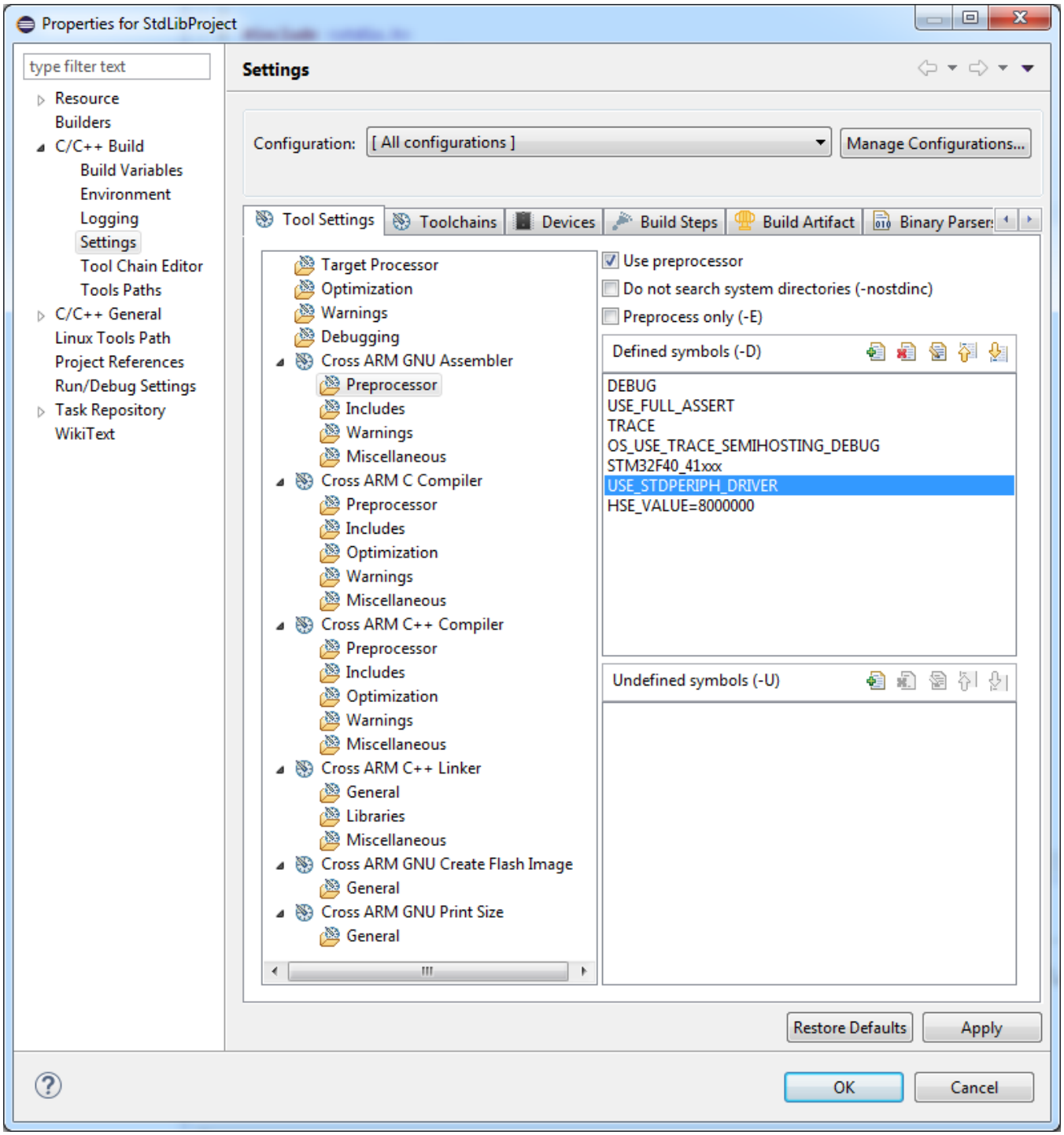


You will see 3 '*Preprocessor*' parts under 'Cross ARM GNU Assembler', 'Cross ARM C Compiler' and 'Cross ARM C++ Compiler' settings. Do the following in all these 3 parts:

-Replace '*STM32F407xx*' with '*STM32F40_41xxx*'

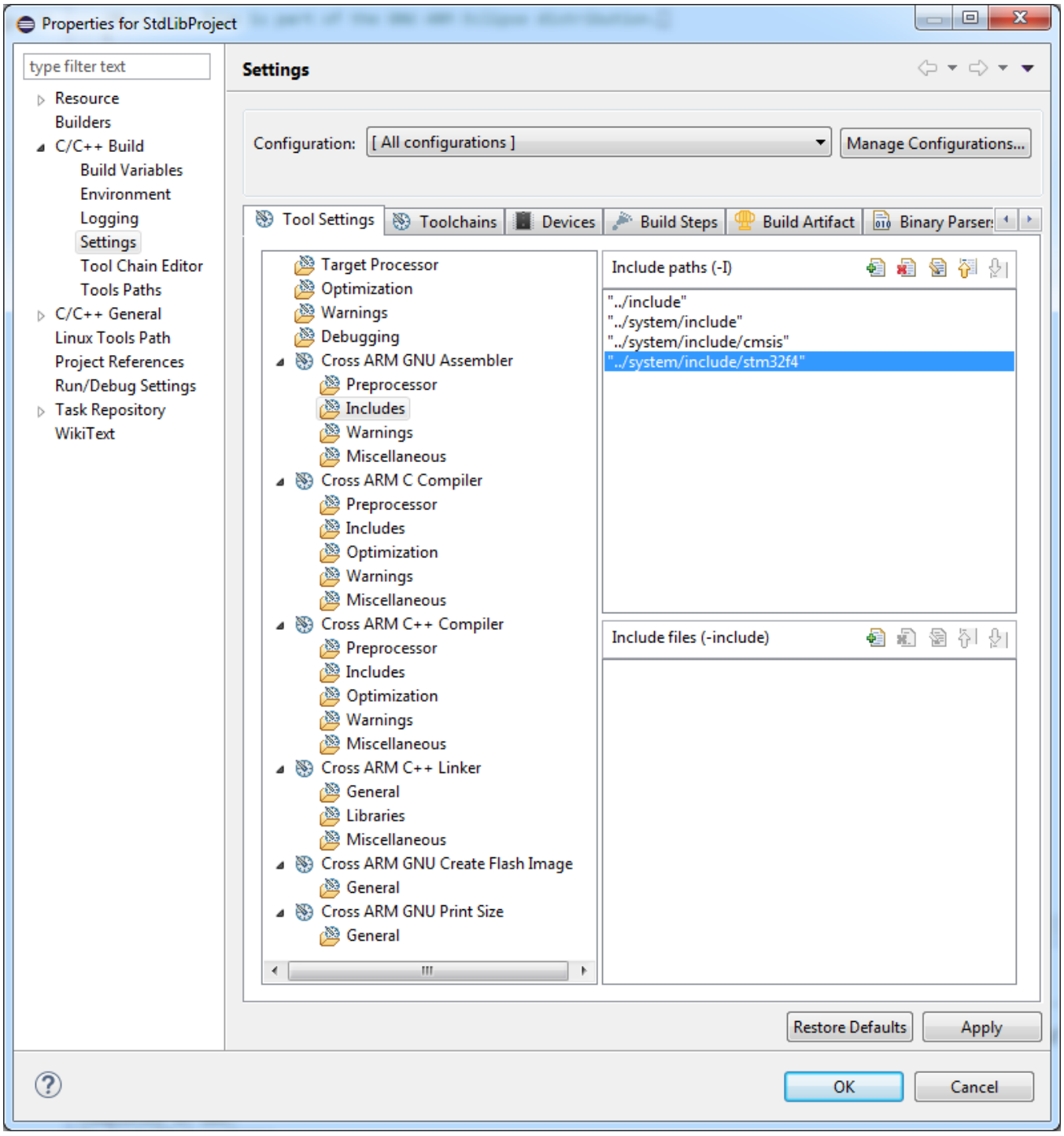
-Replace '*USE_HAL_DRIVER*' with '*USE_STDPERIPH_DRIVER*'

Preprocessor defines should look like:



Again under ‘Cross ARM GNU Assembler’, ‘Cross ARM C Compiler’ and ‘Cross

ARM C++ Compiler' settings, go to '*Includes*' parts and change "../system/include/stm32f4-hal" to "../system/include/stm32f4", since we renamed the folder.



Now, Standard Peripheral Library is ready… But, if you compile, you will get

build errors related to ‘stm32f4xx_fmc.c file’. Since you are using STM32F407, which does not have Memory Control Unit, FMC, you will get these errors. If not using STM32F429/439 series MCUs, you can
delete *stm32f4xx_fmc.c* and *stm32f4xx_fmc.h*files, or exclude from build.

After deleting these files, you can compile and run your code on MCU. Have fun!