

nRF52

Install the Toolchain

download [gcc-arm-none-eabi-4_9-2015q3-20150921-mac.tar.bz2](https://launchpad.net/gcc-arm-embedded/+download) toolchain from
<https://launchpad.net/gcc-arm-embedded/+download>
and move it to folder /Users/<user>/tools/

extract content:

```
host$ cd /Users/<user>/tools/  
host$ tar xf gcc-arm-none-eabi-4_9-2015q1-20150306-mac.tar.bz2  
host$ mkdir GNU_Tools_ARM_Embedded  
host$ mv gcc-arm-none-eabi-4_9-2015q1 GNU_Tools_ARM_Embedded
```

add to PATH:

in /Users/<user>/.bash_profile

add the line

```
export PATH=/Users/gabrielefilosofi/tools/GNU_Tools_ARM_Embedded/gcc-  
arm-none-eabi-4_9-2015q1/bin:$PATH
```

Test it:

```
$host arm-none-eabi-gcc --version  
arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.9.3  
20150303 (release) [ARM/embedded-4_9-branch revision 221220]
```

Install GNU make

Linux and OS X already have the necessary shell commands, but GNU make may not be a part of the standard distro. GNU make is bundled with [Xcode tools](#) if working on OS X.

Call "make -v" from the terminal to check whether it is installed or not. GNU make would need to be installed if it's not recognized as a command.

```
host$ make -v  
GNU Make 3.81
```

Nordic nRF5x SDK

download SDK 11.0.0 from <http://developer.nordicsemi.com/>

Download the repository file nRF5_SDK_x.x.x_XXXXXXX.zip (for example, nRF5_SDK_11.0.0_1a2b3c4.zip).

1. Extract the .zip file to the directory that you want to use to work

with the nRF5 SDK.

2. Install the nRF5 MDK:

- If you use Keil 5, open an example project. Keil will then prompt you to install the nRF_DeviceFamilyPack. If Keil does not prompt you, open the Pack Installer, click the Check For Updates button, and install the latest nRF_DeviceFamilyPack.
- If you use Keil 4, double-click the nRF5x_MDK_x_x_x_Keil4.msi file in the extracted directory to install the MDK.
- If you use IAR, double-click the nRF5x_MDK_x_x_x_IAR.msi file in the extracted directory to install the MDK.
- If you use GCC, the MDK is delivered with the SDK and does not need to

be

installed separately.

3. See the documentation for instructions on how to get started with the nRF5 SDK. The documentation is located at

http://infocenter.nordicsemi.com/topic/com.nordic.infocenter.sdk/dita/sdk/nrf5_sdk.html

Set GNU_INSTALL_ROOT to the toolchain path in the Makefile.posix file:

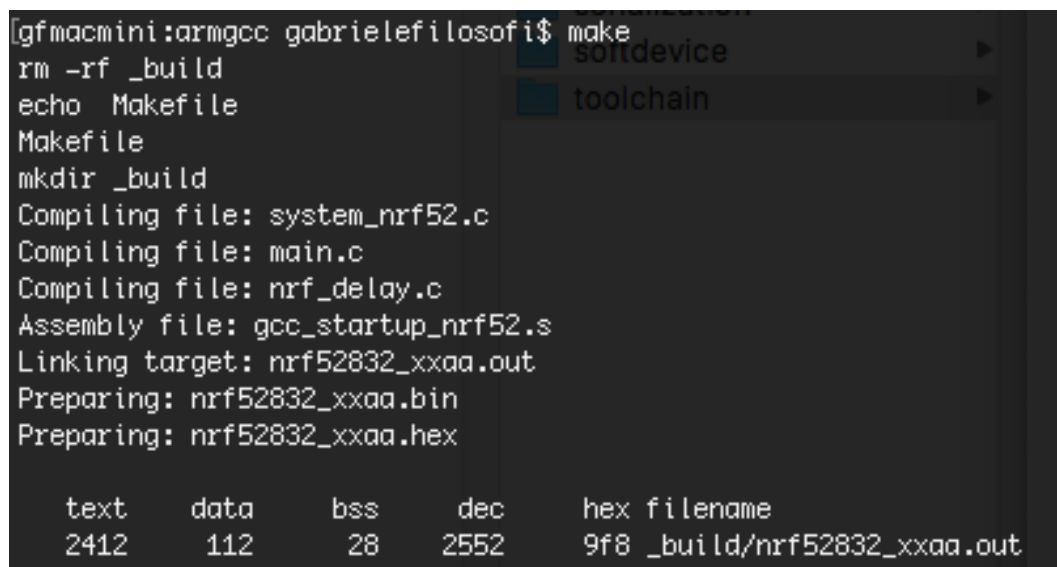
```
$host vim /Users/gabrielefilosofi/Documents/nRF52/nRF5_SDK_11/components/  
toolchain/gcc/Makefile.posix
```

```
GNU_INSTALL_ROOT := /Users/gabrielefilosofi/tools/  
GNU_Tools_ARM_Embedded/gcc-arm-none-eabi-4_9-2015q1
```

Select a project (ex. blinky) and build it for your board type (ex. pca10040):

```
$host cd /Users/gabrielefilosofi/Documents/nRF52/nRF5_SDK_11/examples/  
peripheral/blinky/pca10040/blank/armgcc
```

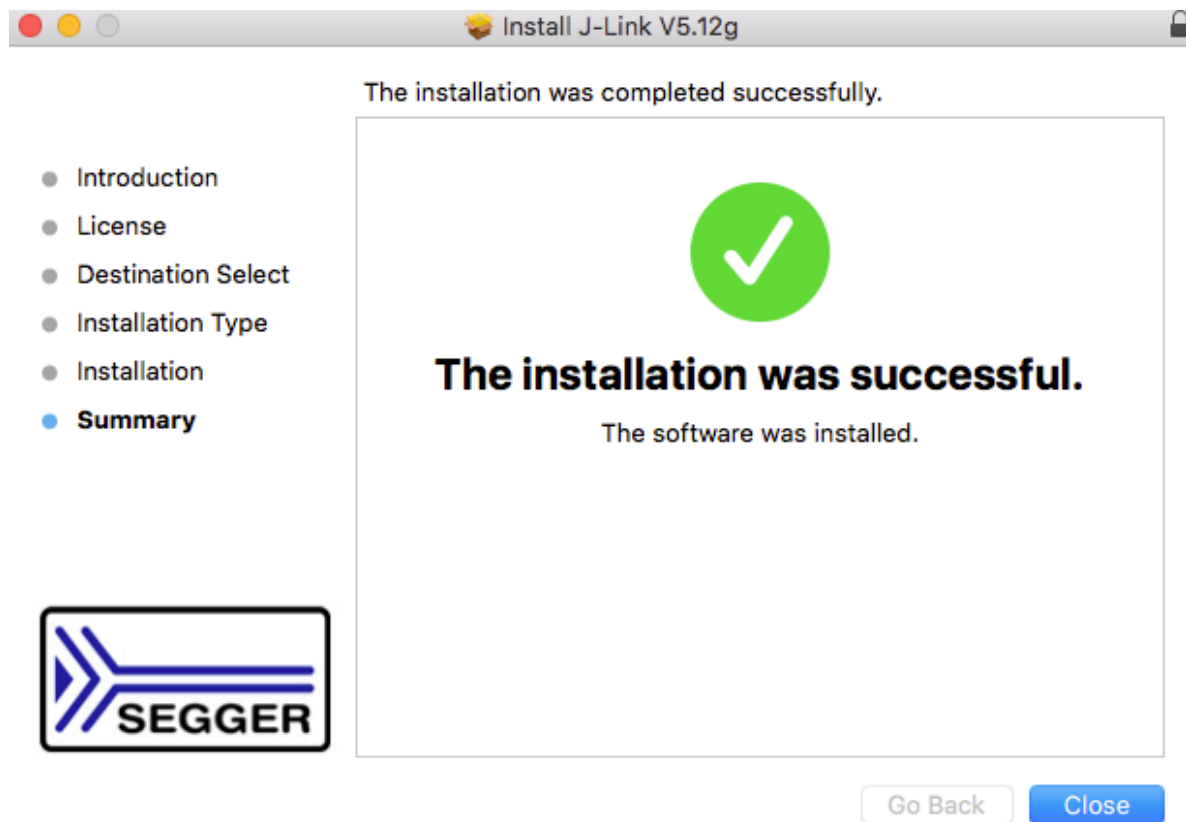
```
$host make
```



```
gfmacmini:armgcc gabrielefilosofi$ make  
rm -rf _build  
echo Makefile  
Makefile  
mkdir _build  
Compiling file: system_nrf52.c  
Compiling file: main.c  
Compiling file: nrf_delay.c  
Assembly file: gcc_startup_nrf52.s  
Linking target: nrf52832_xxaa.out  
Preparing: nrf52832_xxaa.bin  
Preparing: nrf52832_xxaa.hex  
  
text    data    bss     dec     hex filename  
2412    112     28     2552    9f8 _build/nrf52832_xxaa.out
```

J-Link Software

From <https://www.segger.com/jlink-software.html>
download and install **Software and documentation pack** for Mac OS X
V5.12g [27,042 kb]



nrfjprog

nrfjprog is the command line tool for loading FW images to target via the debug interface SWD

1. From <https://www.nordicsemi.com/eng/nordic/Products/nRF51-DK/nRF5x-Command-Line-Tools-OSX/53412>
2. Download nRF5x-Command-Line-Tools_8_5_0_OSX.tar and extract in /Users/gabrielefilosofi/Documents/nRF52/

1. Connect the target to Mac
2. Erase target
`$host nrfjprog --family nRF52 -e`
3. Load FW image to target
`$host nrfjprog --family nRF52 --program _build/nrf52832_xxaa.hex`
4. Reset and run
`$host nrfjprog --family nRF52 -r`



```
armgcc — -bash — 149x23
[gmamacmini:~ gabrielefilosofi$ cd Documents/nRF52/nRF5_SDK_11/examples/peripheral/blinkypca10040/blank/armgcc/
[gmamacmini:armgcc gabrielefilosofi$ nrfjprog -v
nrfjprog version: 8.5.0 Phone: +1-978-874-0299
JLinkARM.dll version: 5.12g Fax: +1-978-874-0599
[gmamacmini:armgcc gabrielefilosofi$ nrfjprog --family nRF52 -e
Erasing code and UICR flash areas.
Applying system reset.
[gmamacmini:armgcc gabrielefilosofi$ nrfjprog --family nRF52 --program _build/nrf52832_xxaa.hex
Parsing hex file.
Reading flash area to program to guarantee it is erased.
Checking that the area to write is not protected.
Programming device.
[gmamacmini:armgcc gabrielefilosofi$ nrfjprog --family nRF52 -r
Applying system reset.
Run.
[gmamacmini:armgcc gabrielefilosofi$
```

People
22/04/16 Franco Giannini

Xcode
22/04/16 How to install Xcode:

New Technologies
22/04/16 JSON: used for B2B API integra...

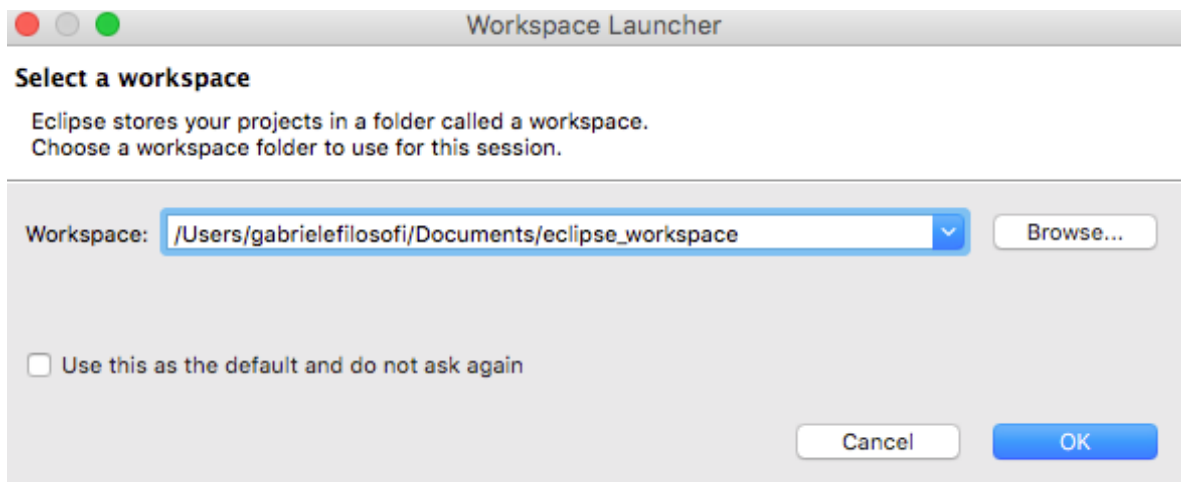
The Complete iOS 9 Developer Cour...
22/04/16 by Rob Percival

Now you should see blinking LEDs on target board.

Eclipse Mars IDE for C/C++ Developers

Download Eclipse from <https://www.eclipse.org/downloads>

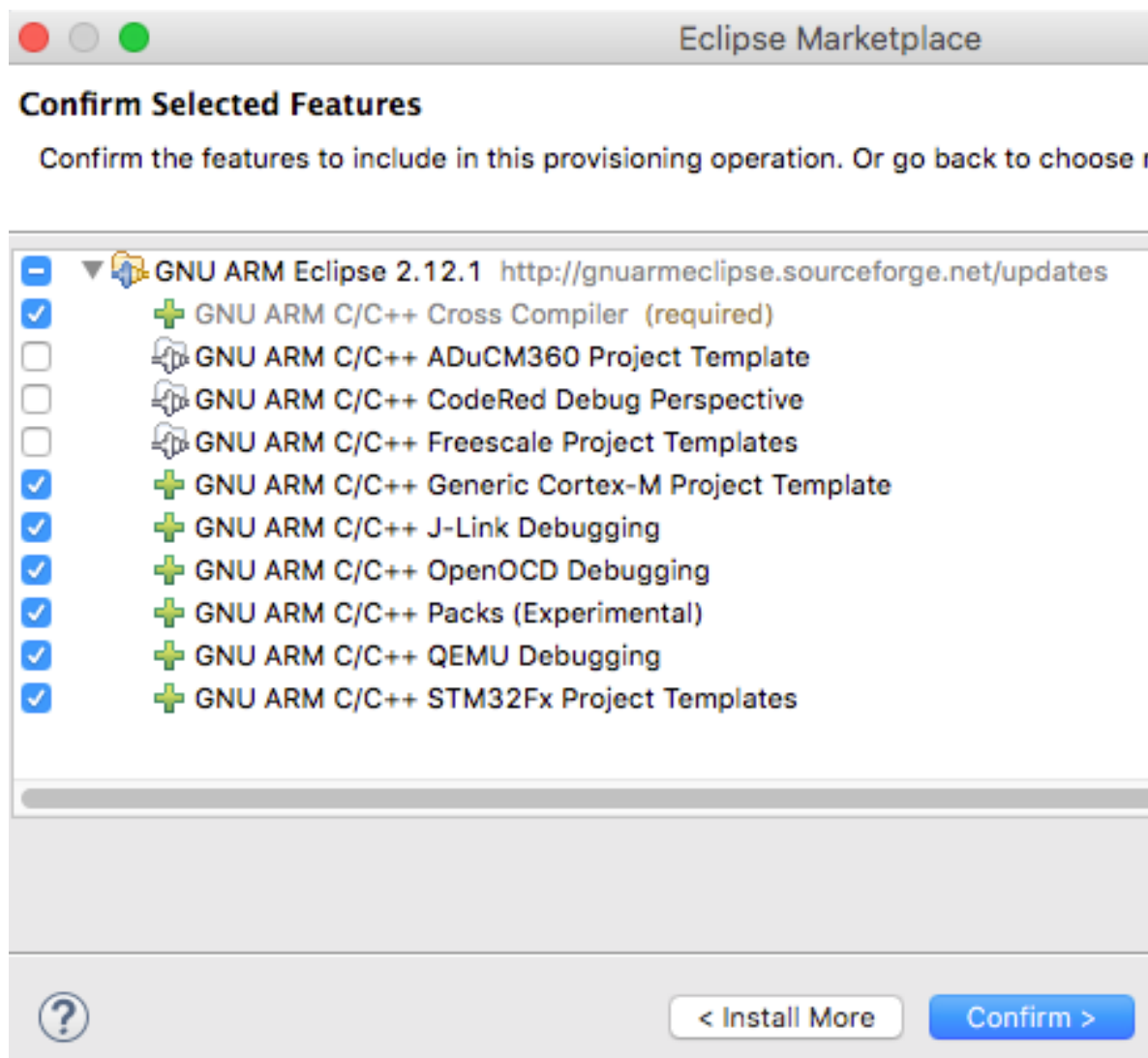
Eclipse does not need an installer, Eclipse is distributed as a plain archive. To install Eclipse, simply unpack the archive at a place of your choice and start using it.



2. Check if you really have the latest version available. For this, enter the *Eclipse* menu and go to **Help → Install New Software**

1. Install GNU ARM Eclipse plug-in

GNU ARM C/C++ Cross Compiler
GNU ARM C/C++ Packs
GNU ARM C/C++ J-link Debugging



Instruction to install:

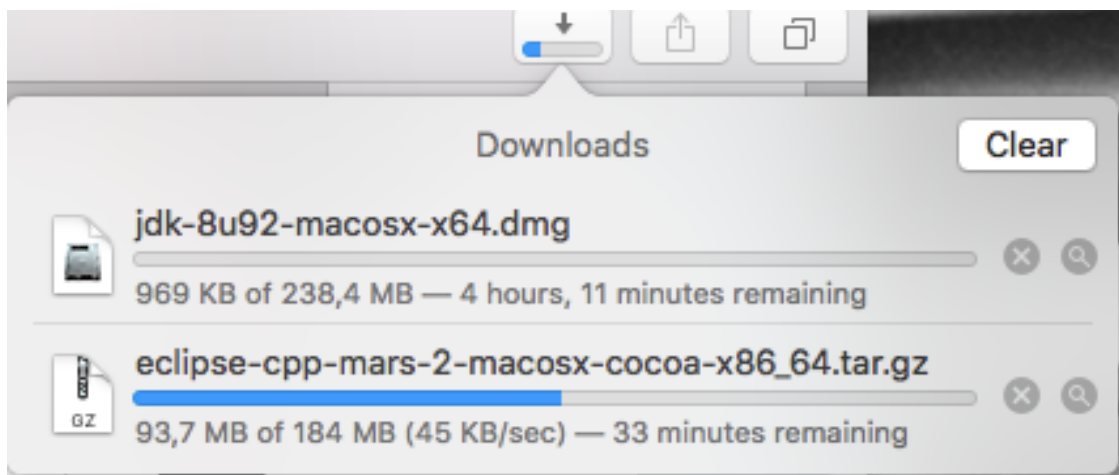
<http://gnuarmclipse.github.io/plugins/install/>

The recommended package is the official **Oracle Java SE**.

On OS X the last Apple Java implementation is 1.6, so it is required to use the Oracle 1.7 or later.

Java SE Runtime Environment 8u92

<http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>



Install device family pack for the nRF5x series

The device family pack includes CMSIS System View Description, which is basically is a memory map of all peripheral registers on the device. Although it is not required to have installed it can be quite useful for debugging purposes; it allows you to easily read and write to particular peripherals through a built in peripheral viewer (part of GNU ARM Eclipse Plug-in) rather than doing a manual lookup on the address in the datasheet, not to mention handling of bit-fields.

Steps to install:

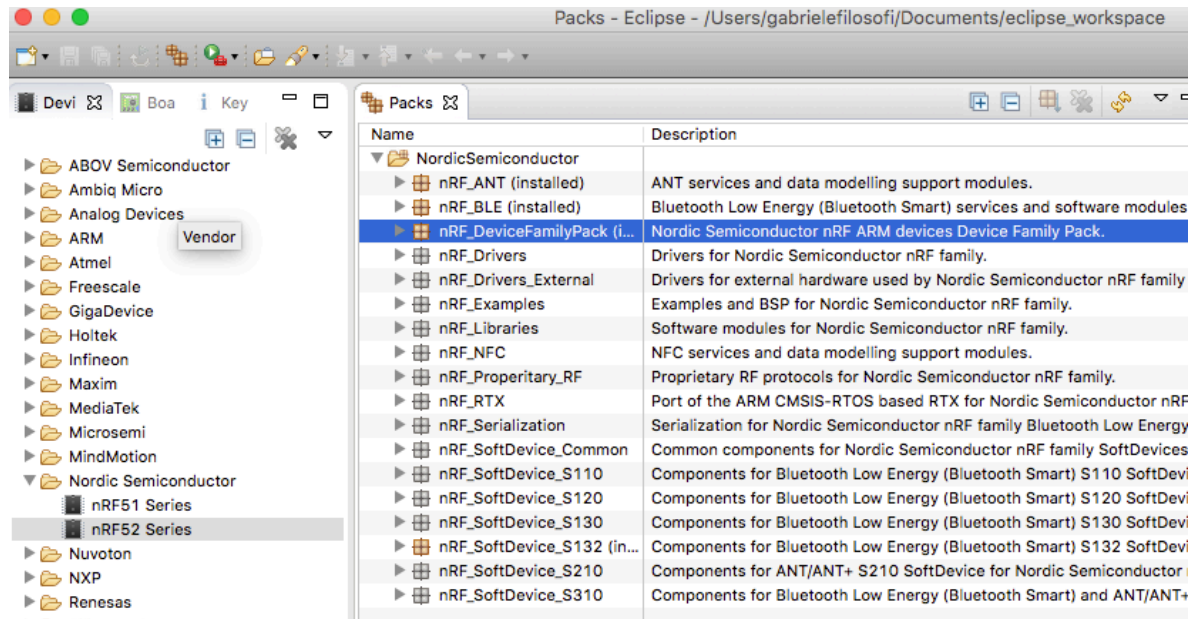
Select the Window > perspective > open perspective > other > Packs.

Click the refresh button in the top right corner of the window that got opened.

This will fetch all packs from the repositories.

Select Nordic Semiconductor in the list of vendors, and install the latest version

of Device family pack.

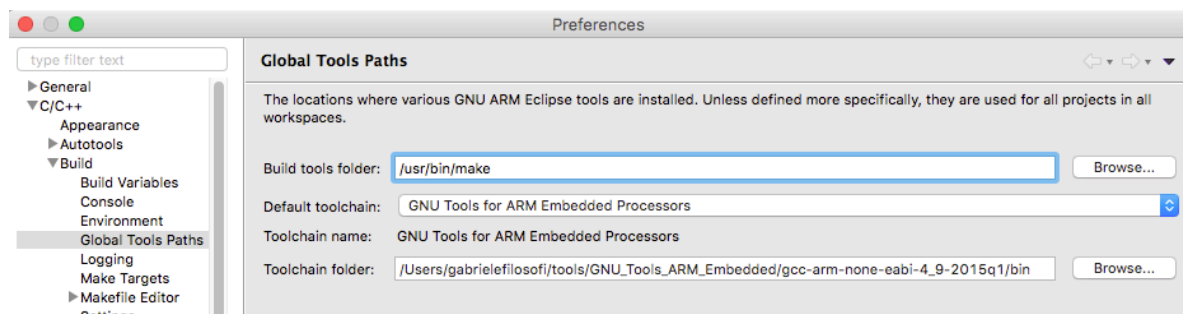


Configure environment

Here we will set the global Eclipse configurations for development on nRF5x.

From Eclipse > Preferences > C/C++ > Build > Global Tools path.

Check if the paths to your Build tools and Toolchain are set, otherwise add the paths where you installed these earlier. On this setup the following paths were used:



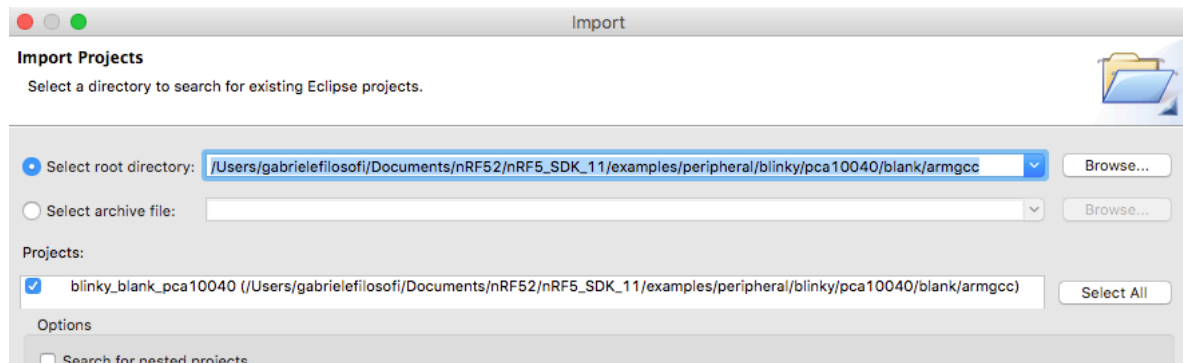
From Eclipse > Preferences > Run/Debug, click on SEGGER J-link, and make sure that it is pointing to your latest version of the Segger software.

Import existing Eclipse project to workspace

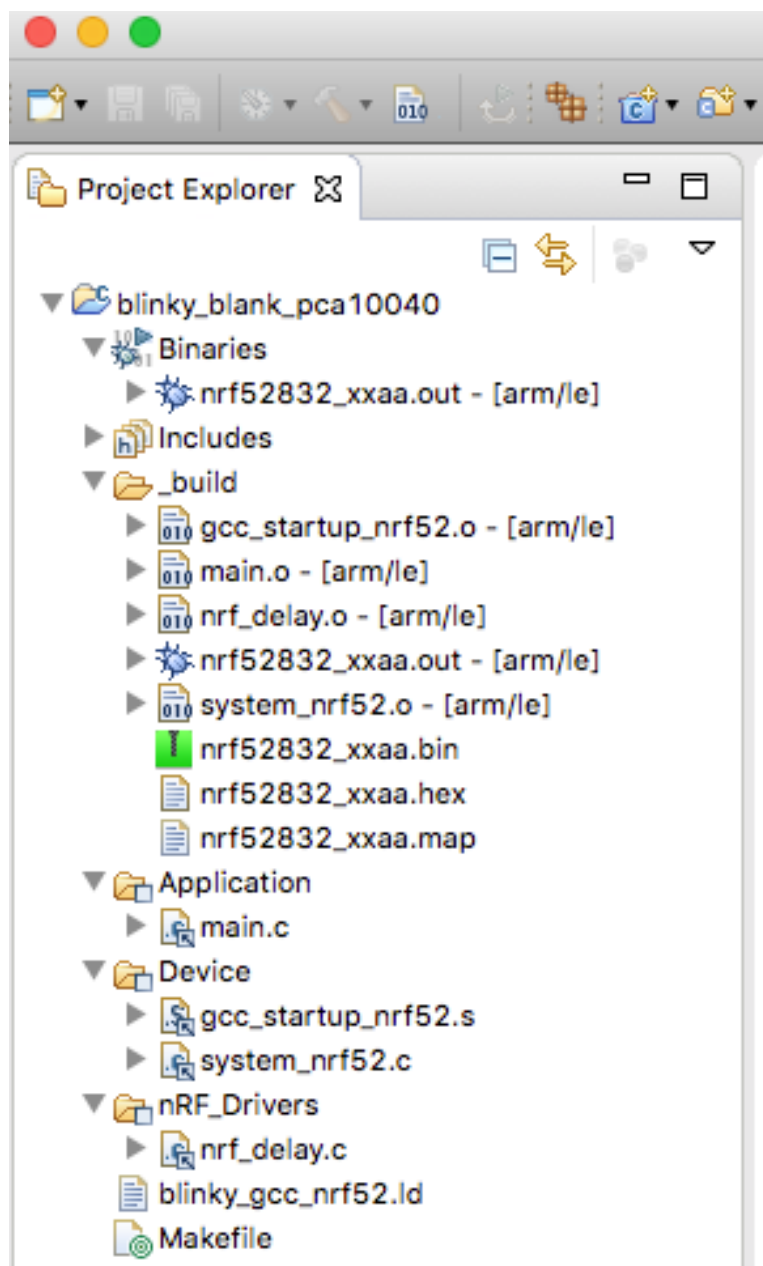
You may want to start with the blinky example first to keep it simple at first.

- Download the project you wish to use from the attachment list, extract it, copy and replace with the content into the existing ARMCC folder in the respective SDK project. E.g., `\examples\peripheral\blinky\board name\blank\armgcc` for the blinky example. Note that the directory structure **must** be kept. Otherwise it will break the paths. In other words, do not create a sub-directory in the armgcc folder.
- Open Eclipse.
- Enter file->import->General -> Existing Projects into Workspace.
- Browse ARMCC folder in your SDK

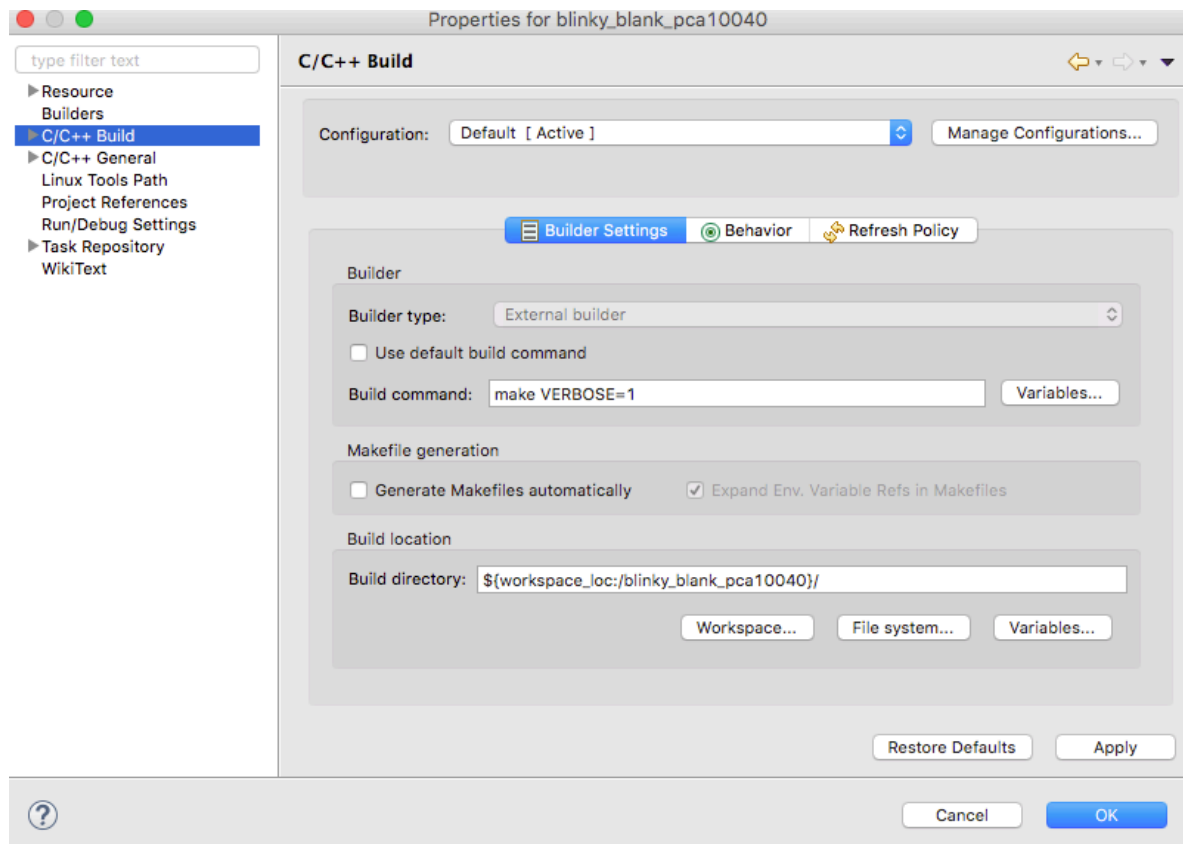
- Make sure to select the project.



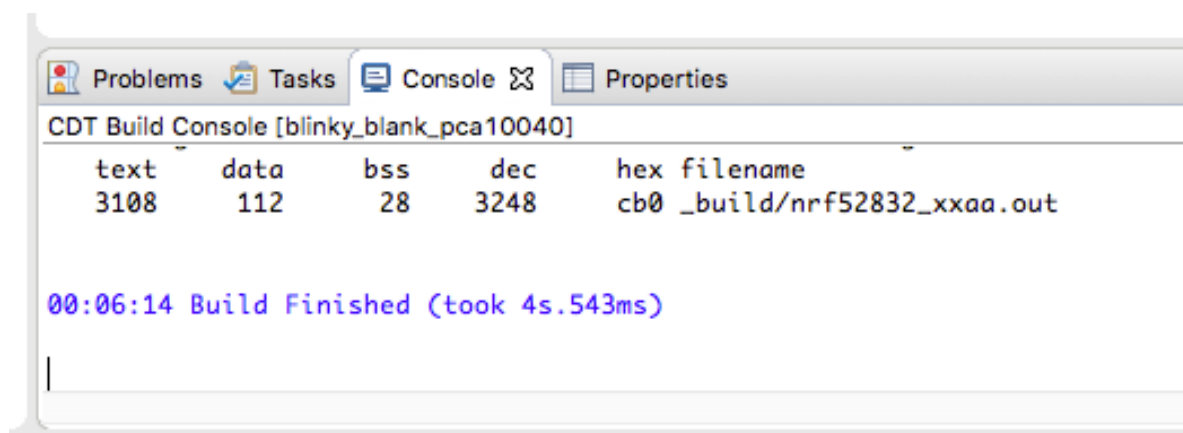
- Click finish button.
- The project should now be available in the project explorer including the linked sources.



Right click on the project folder in *project explorer*, click Properties > C/C++ then replace the default build command with *make VERBOSE=1*. In other words increase the verbosity level.



Right click on project again and click on *Build Project*. Hopefully you will see the build output in the console window, same as when you built the example via the terminal.

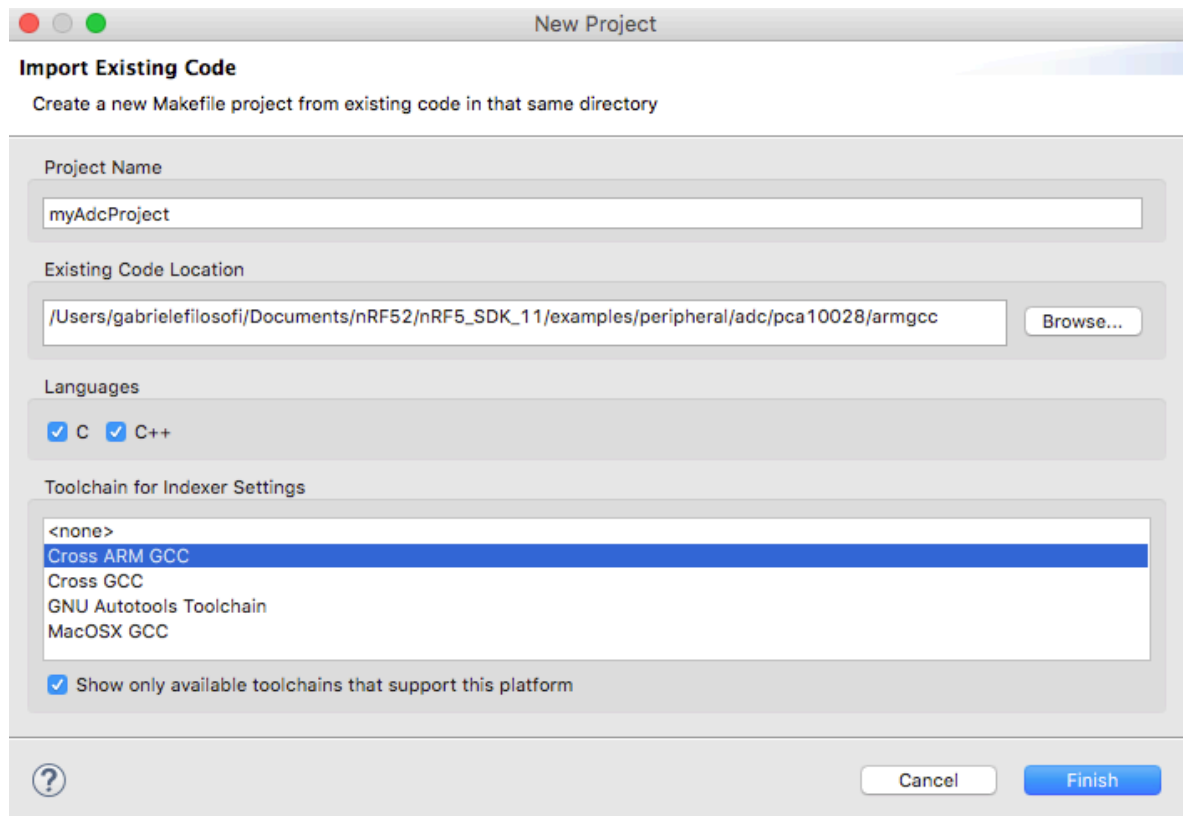


Create a new Eclipse project

Now we want to integrate an existing SDK project into Eclipse (or to create a new one).

Start Eclipse, enter File -> new Makefile project with existing code. Name your

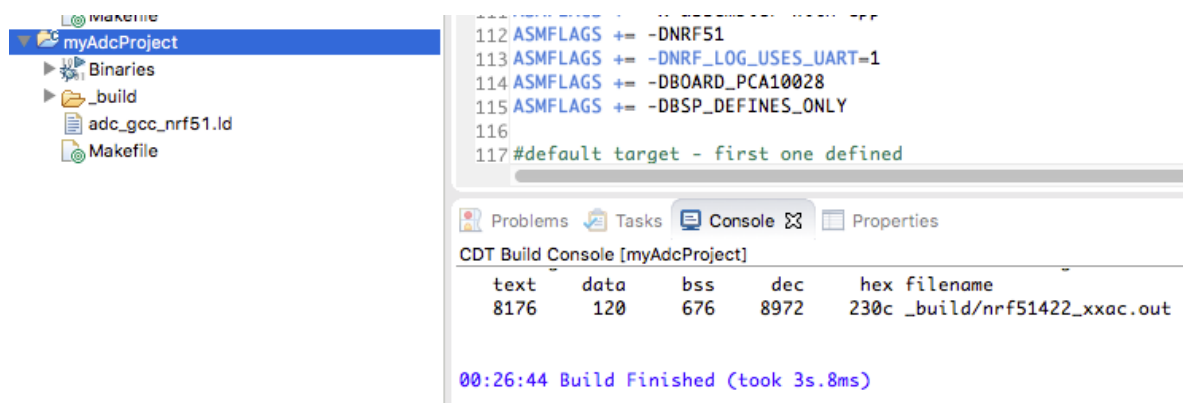
project and browse the directory of your Makefile. Click *finish* once you are done.



Open and edit the SDK Makefile to support debugging of your code. Locate the CFLAGS variables, and change '-O3' to '-O0' for debugging to produce expected results (stack variables loaded directly to CPU registers, etc). Then add '-g3' to CFLAGS to include debug symbols in the .out file ([GNU manual](#)).

```
95 CFLAGS += -mthumb -mabi=aapcs --std=gnu99
96 CFLAGS += -Wall -Werror -O0 -g3
97 CFLAGS += -mfloat-abi=soft
98 # keep every function in separate section.
```

Configure the build settings and build the project. Great.



Now you can start linking the source files into your project viewer, but you might want to add some virtual folders to obtain a more clear project view first.

Create virtual folders:

Right click on the project folder and enter New->folder.

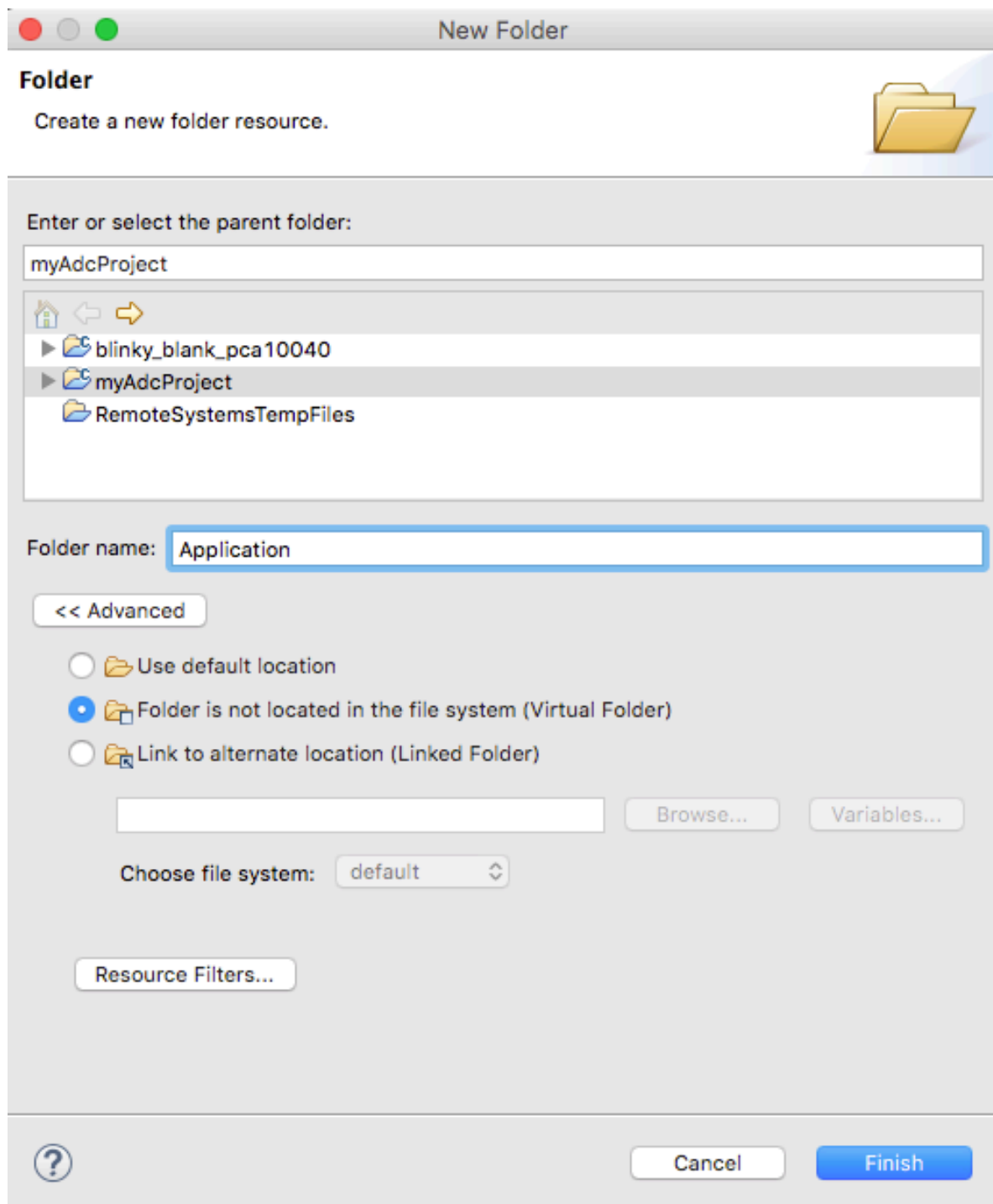
Set folder name.

Click Advanced and choose virtual folder.

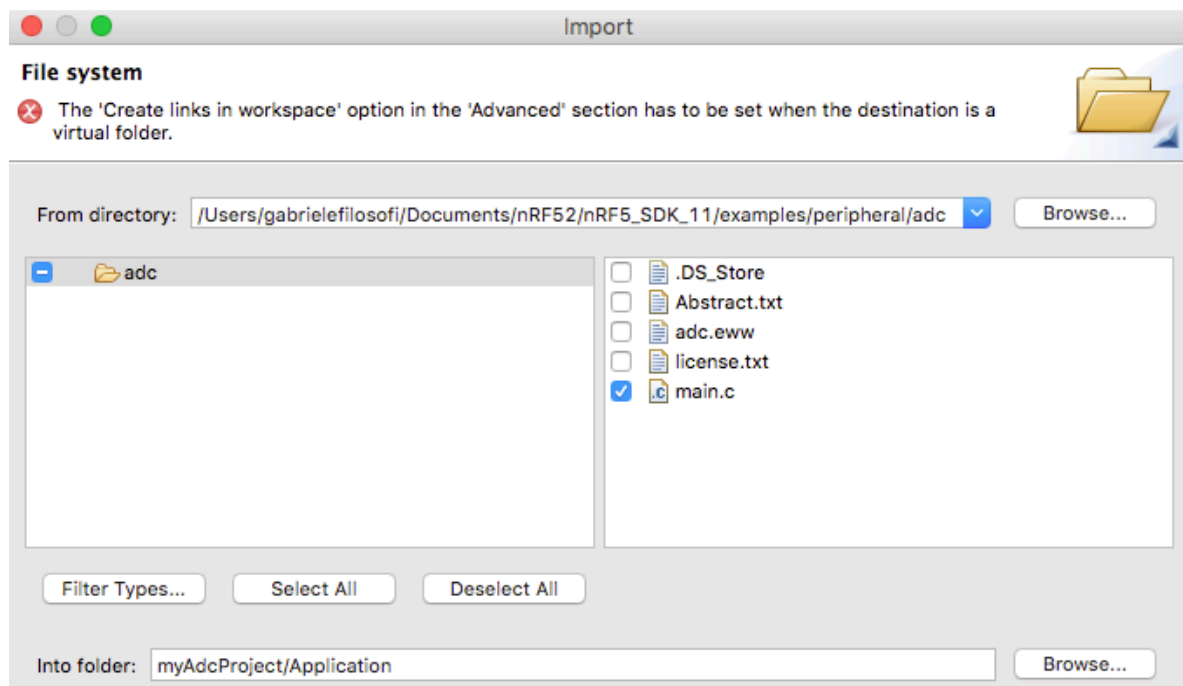
Click finish.

Repeat above steps until you have the folders you want. The naming of the folders should reflect the types of the source files you intend to link to. In the blinky it was sufficient to have Application, Device and nRF_Drivers, same as the Keil example for blinky, but, for larger projects with more source files you should add more folders. E.g., ble_app_hrs: Application, Board Support, Device, nRF_BLE, nRF_Drivers, nRF_Libraries and nRF_Softdevice.

For example, we want to create the virtual folder Application where to link the main file.



Now you can start to link the source files to the respective folders. The source files to include are listed in your Makefile->C_SOURCE_FILES, and ASM_SOURCE_FILES. Source files are then linked to the folder by right clicking on any of the virtual folders and clicking on import > General > File System and browse the source files in the SDK. The path to each source file can be a good way to determine what folder to place the file in: ../../../../../../components/**drivers_nrf**/delay/nrf_delay.c can be classified as a driver. Thus placed in nRF_Driver, etc.

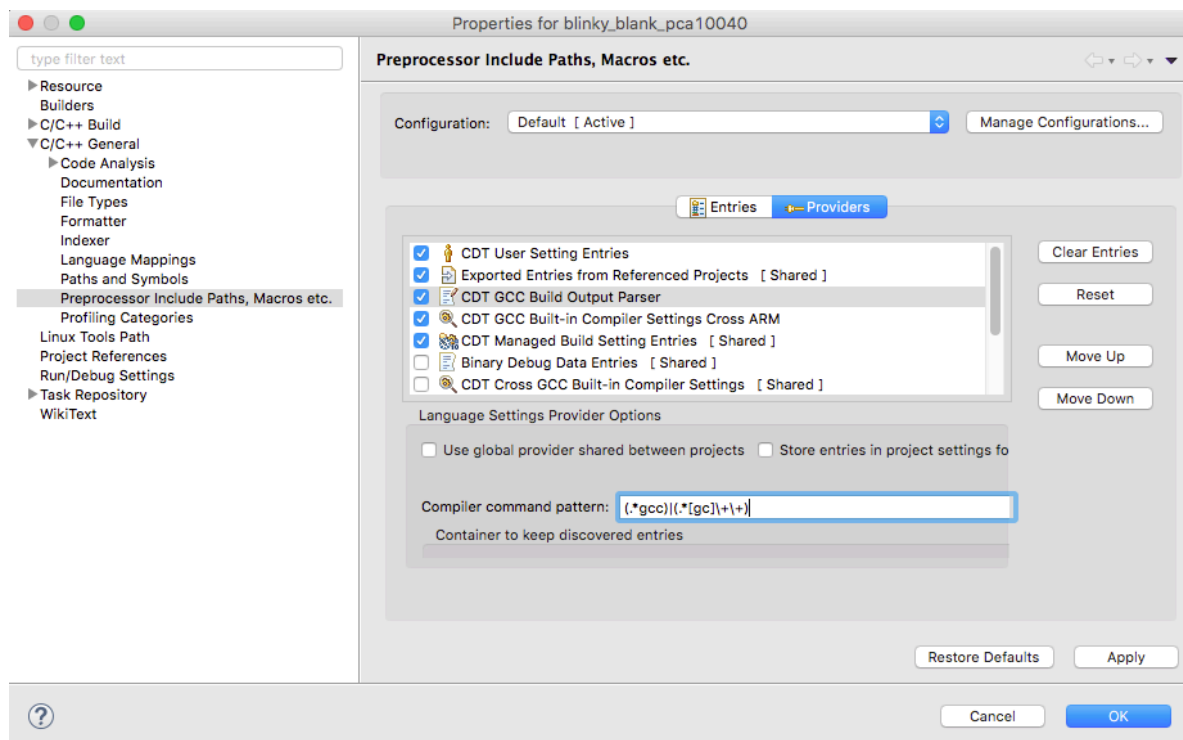


Enable auto discovery of symbols, include paths and compiler settings

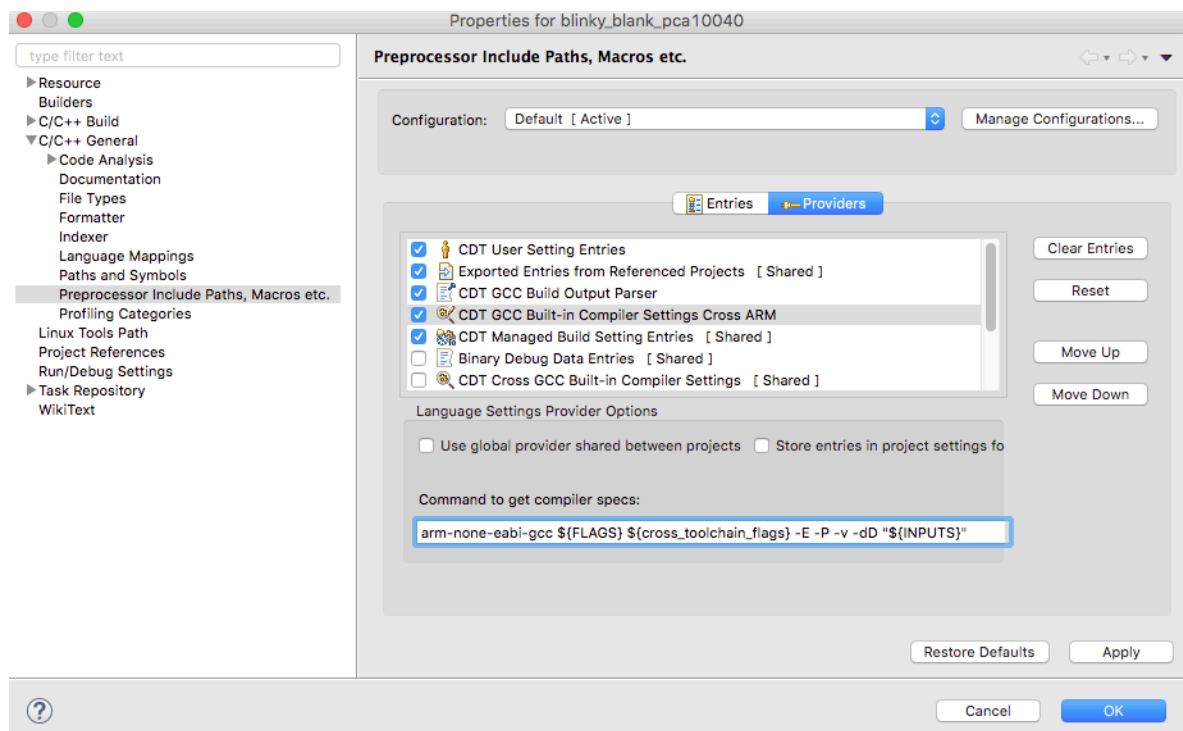
The following steps configure the **CDT build output parser** and **CDT GCC Built-in Compiler Settings Cross ARM** to automatically discover symbols, include paths and compiler settings based on the output produced by the Makefile. These steps also apply to imported projects as these configurations are not included in the project files.

From project properties > C/C++ > Preprocessor Include Paths,etc. > Providers

- Click on CDT GCC Build Output Parser and change the compiler command pattern from (gcc)|([gc]\+|+)|(clang) to (.*gcc)|(.*[gc]\+|+) then apply changes



- Click on CDT Built-in Compiler Settings Cross ARM and replace `{COMMAND}` with `arm-none-eabi-gcc` and click *Apply*.



Flash download

Flash downloading and running a program on your device without debugging is possible by using flash targets included in the Makefiles.

In order to execute the flash targets you first need to add a new target to the

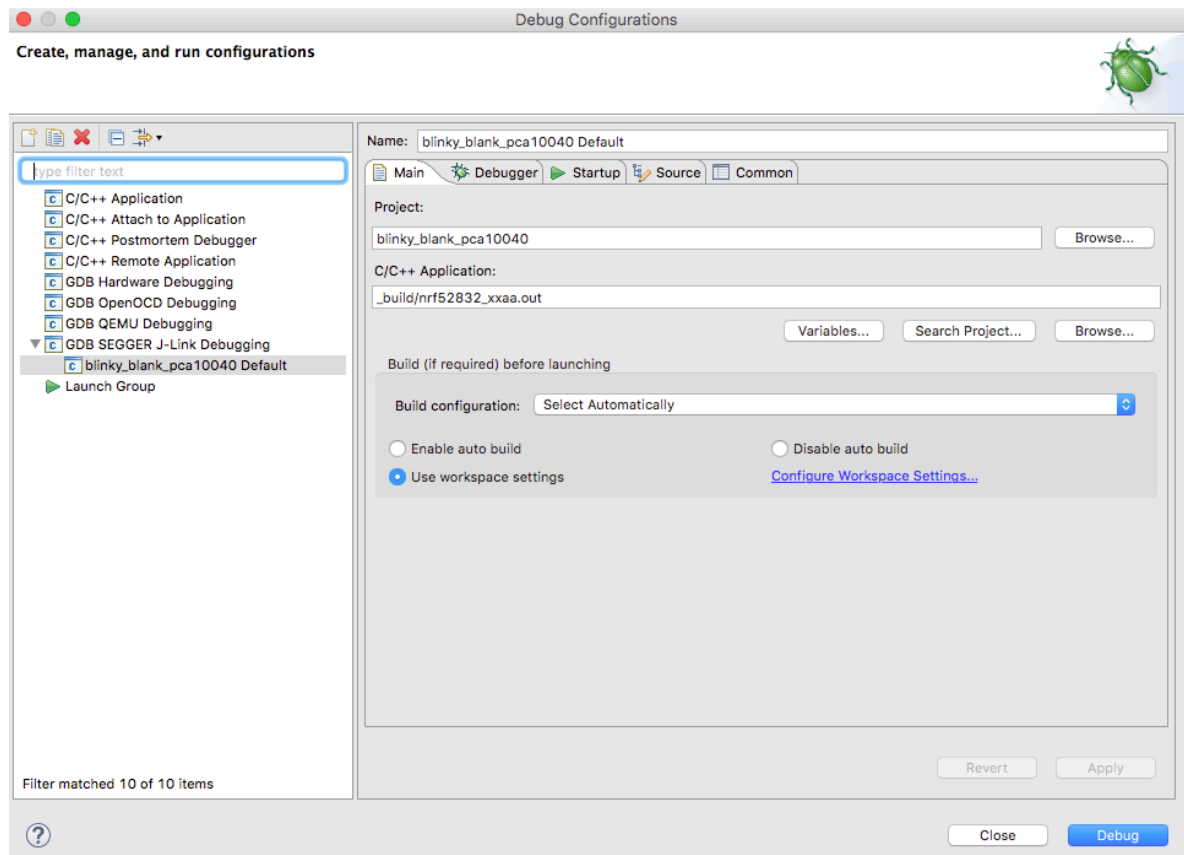
project. Here is an example with the blinky example. There are no flash SoftDevice target in this example as it runs independent of the SoftDevice. For the heart rate or other BLE examples you will need to flash the **softdevice** by executing the "flash_softdevice" target.

Setting up a project for debugging in Eclipse

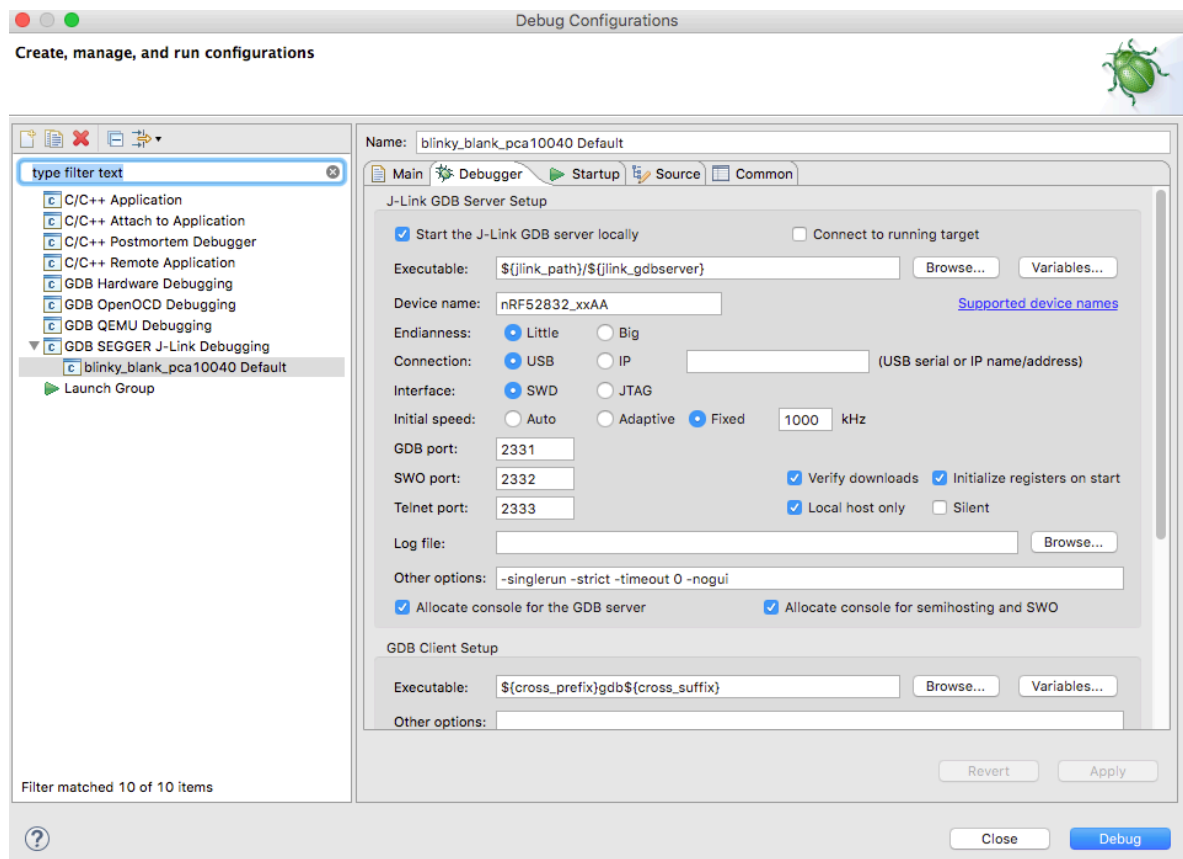
The GNU ARM Eclipse plugin we installed earlier has already integrated J-link debugging with Eclipse. We only need to set the debug configurations for our project before we can launch a debug session.

In the project explorer select your project.

From Run > Debug Configurations. Right click on GDB SEGGER J-Link Debugging and select New.



Enter the *Debugger* tab, and select the device name for the device you are targeting. Also, deselect SWO if working on nRF51 series as it is not supported.



Now you start the debug session by clicking on the *Debug* button. Eclipse will then show the debug view, and the application will break in main.

Now you can start debugging; set break points, single step, read registers, etc.

Using the peripheral viewer

Go to project properties and select C/C++ Build item. Enter settings and click on the *Devices* tab. Here you must select the chip you are using to enable the view. Start a debug session and open the peripheral viewer, and select the Peripheral you want to debug. If you select the GPIO module you can get an visual indications by controlling the board LEDs (P0 on nRF52 and GPIO on nRF51). Open the Memory view and click on PIN_CNF[24] or PIN_CNF[20] to control the LED 4 GPIO on your DK kit (PCA100028 or PCA10036/40). Changing the GPIO direction in PIN_CNF[x].DIR should toggle your board LED.