Release 129

embedXcode

Embedded Computing on Xcode

User Manual

embedXcode.weebly.com

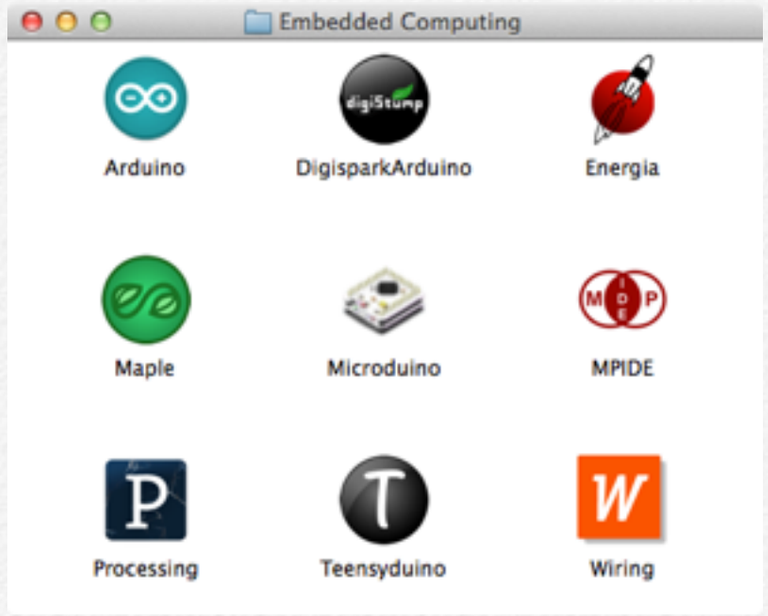
Forewords

What Is embedXcode?

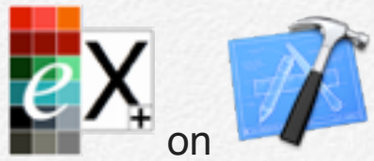
embedXcode is a template for Xcode. It eases development for the most popular embedded computing boards.

For convenience, embedXcode requires the previous installation of the IDEs of the boards, as they pack everything, including toolchains, frameworks and utilities.

Instead of dealing with multiple IDEs...



...use just one!



on

embedXcode — *Embedded Computing on Xcode*

Welcome!

Welcome!

After having played with embedded computing platforms for a while, I was looking for one single IDE and a better one.

Code-sense, coloured syntax, code-completion, check-as-you-type, click-to-error, self-documentation, tool-tip texts and debugging are some niceties Xcode brings.

*The **embedXcode** template allows to use Xcode to develop for Arduino, chipKIT, DFRobot BLuno, Digispark, LaunchPad MSP430, MSP430FR5739 Experimeter Board, LaunchPad Stellaris and Tiva C Series, Maple, Microduino, Teensy and Wiring boards. Thanks to embedXcode, I'm developing twice as fast as before.*

Because embedXcode relies on a modular design and on the boards IDEs for easier installation, virtually any board with a Processing-based Wiring-derived Arduino-like IDE can be implemented.

embedXcode comes now in two editions: embedXcode with core features and embedXcode+ with extended functionalities.

Happy coding!

Rei Vilo

Overview


Install the Template
1
Before installing the template, you need to install Xcode and at least one IDE.
Debug the Project
5
If For boards featuring a built-in debugger, define breakpoints and debug your project.
Create a New Project
2
Create and configure a new project.
Self-Document the Project
6
If In order to obtain documentation,
- Add specific comments with defined keywords to the code,
- Select the output formats,
- Build the documentation with the target Document,
- Use Quick Help to access the documentation.
This chapter requires embedXcode+.
Use the Project
3
Write the sketch using Xcode advanced features, change the board, add files and libraries, manage code for multiple boards and platforms.
Find Solutions to Issues
7
Learn more about embedXcode compatibility with Arduino, use embedXcode specific pre-processing variable and find solutions to most common issues.
Build and Upload the Project
4
Select the target, manage the libraries, build and upload the project.
Appendixes
This section includes additional information about embedXcode as well as copyright and license.

Copyright

Links

Website	http://embedXcode.weebly.com ⓦ
Tutorial	http://embedxcode.weebly.com/tutorial ⓦ
Download	http://embedxcode.weebly.com/download ⓦ
Contact	http://embedxcode.weebly.com/contact ⓦ

Conventions

This electronic book uses the following conventions:

- Keywords and folders names are in `Courier` font:

  Download and install Arduino 0023 or Arduino 1.0 or Arduino 1.5 under the `/Application` folder.

- Code is displayed with `Courier` font in a white box:

```
#include "LocalLibrary.h"
```

- Elements of the interface, as menus and keyboard shortcuts, are presented using **Helvetica Neue bold** font:

  Call the menu **File > New > New Project...** or press ⇧⌘**N**.

- Both external and internal links are in underlined red. A ⓦ mentions an external link on the web.

  The manual procedures are no longer required as it is included in the automatic procedure. Download Xcode at the Mac App Store ⓦ.

- Links to the glossary are in underlined dark grey.

  The IDEs include the toolchain and the framework.

- embedXcode+ specific features are mentioned by ⊞.

  ⊞ This section requires embedXcode+.

Install the Template

1

Before installing the template, you need to install Xcode and at least one IDE.

Install Xcode

Install Xcode

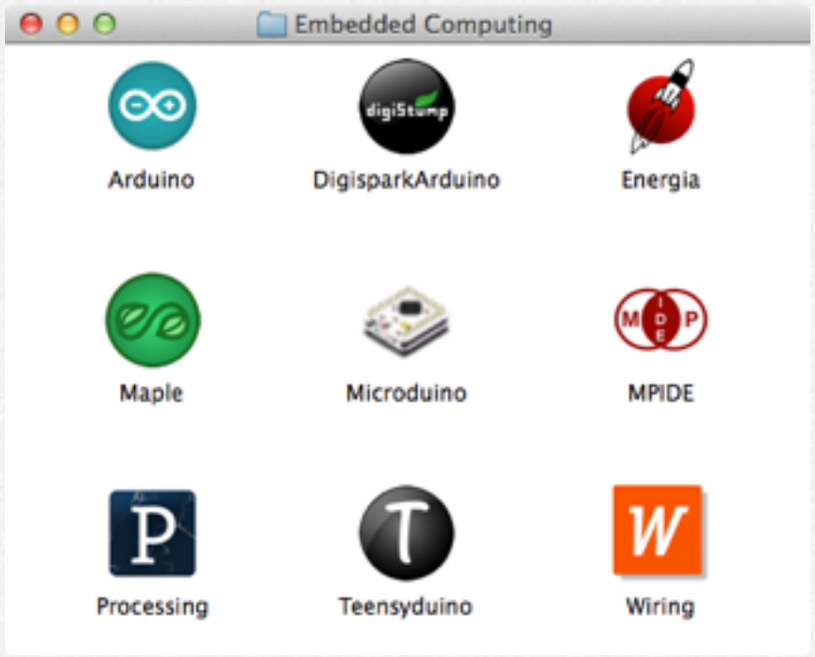
Xcode is the official IDE from Apple.

Install Xcode from the DVD or download it from the Mac App Store ⓦ.

Install the IDEs of the Boards

The IDEs of the boards bring all the tools and libraries related to them.

Using the IDEs of the boards allows an easy and safe installation for embedXcode. The IDEs include the toolchain and the framework, all the core and application libraries and the required utilities.



For each platform,

- First Install the IDEs of the boards under the `/Application` folder.
- Then launch the IDEs and define the path of the sketchbook in the menu **IDE > Preferences > Sketchbook location**.

About Frameworks

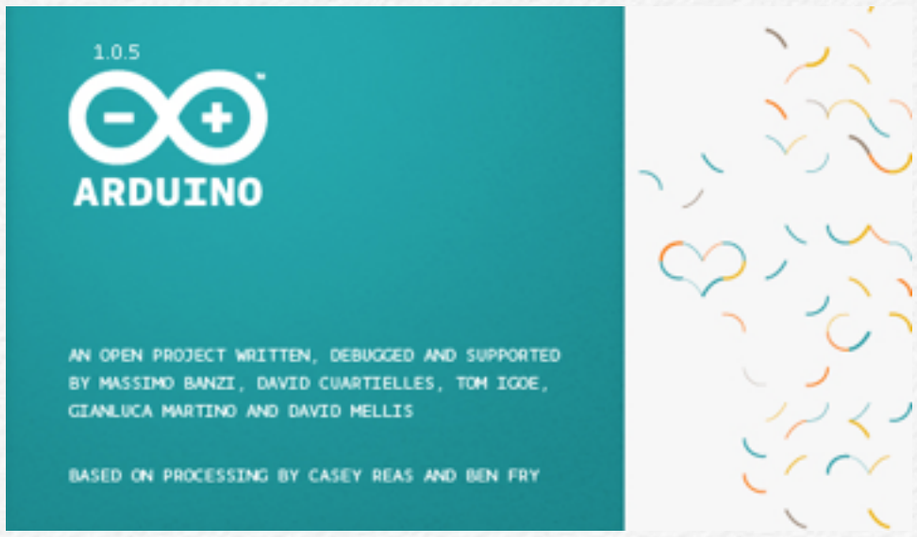
As a matter of fact, Wiring is considered as the framework of reference for embedded computing.

- Arduino 0023 is Wiring compliant, as chipKIT MPIDE 0023. It is no longer maintained
- Arduino 1.0 has introduced many small changes in the syntax which are not compatible with the previous 0023 release. Energia 1.0 is derived from Arduino 1.0. The Digispark, Microduino and Teensyduino plug-ins are designed for Arduino 1.0. This framework is still maintained.
- Arduino 1.5 adds the support for and is required by the Arduino Due and Yún boards. It also uses a new structure for the libraries, not compatible with the previous 1.0 release. However, Arduino 1.5 is still in beta and experiences issues.

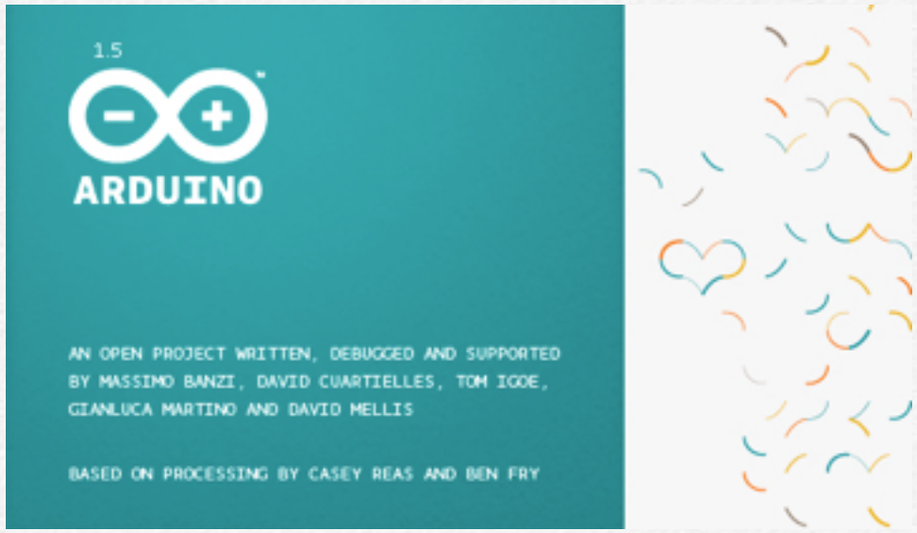
So I strongly recommend to pick the release of Arduino which is compatible with the boards and the other platforms you plan to use.

- Arduino 0023 is compatible with chipKIT MPIDE, Wiring and Maple IDEs.

- Arduino version 1.0.x is required for the Arduino Leonardo and is compatible with LaunchPad Energia IDE.



- Arduino version 1.5.x is required for the Arduino Due and still in beta.



If you plan to use the Arduino boards only, go for the latest stable release of Arduino 1.0.x.

Each release of Arduino 1.5.x includes many changes compared with the previous one and thus may prove unstable.

IDE and Plug-In Options

For some platforms, the IDE can be installed as a standalone application or as a plug-in for Arduino 1.0.

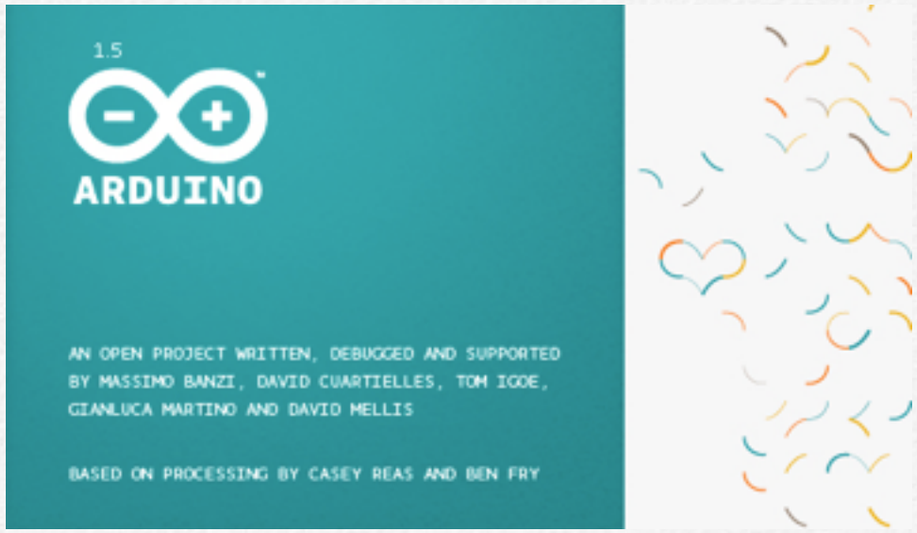
Platform	Standalone IDE	Plug-in for
Arduino	Arduino 1.0.5 or 1.5.5	
chipKIT	MPIDE 0023	
BLuno	Arduino 1.0.5	
Digispark	DigisparkArduino	Arduino 1.0.5
LaunchPad	Energia 0101E0011	
Maple	Maple 0.0.12	
Microduino	Microduino	Arduino 1.0.5
Teensy	Teensyduino 1.17	Arduino 1.0.5
Wiring	Wiring 1.0	

embedXcode finds the IDE automatically, whether it is installed as a standalone application of as a plug-in for Arduino.

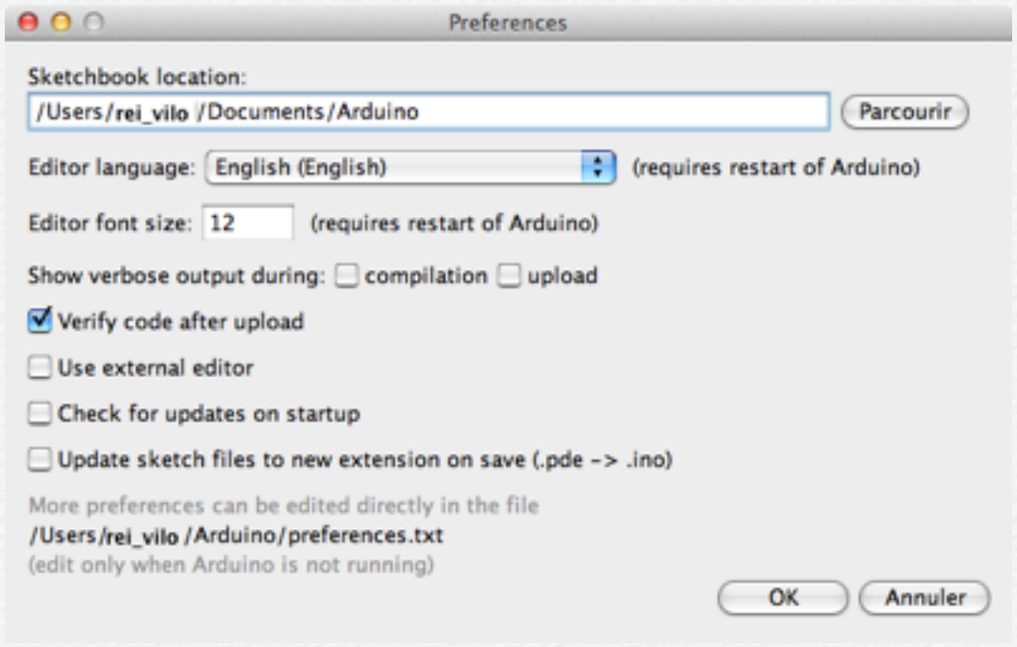
Arduino Boards

If you plan to use the Arduino boards:

- Download and install Arduino 1.0.x or Arduino 1.5.x under the `/Application` folder.



- Launch it.
- Define the path of the sketchbook in the menu **Arduino > Preferences > Sketchbook location**.



The same procedure applies for the other IDEs: chipKIT MPIDE, Energia, Maple, Teensy or Wiring.

Arduino 1.5.x manages two architectures: 8-bit ATmega and 32-bit ARM. Arduino 1.5.x is required for Arduino Due and Arduino Yún.

embedXcode identifies the version of Arduino automatically.

Please note support for Arduino 0023 is deprecated.

chipKIT Boards

If you plan to use the chipKIT boards:

- Download and install MPIDE 0023 release 20130715 under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook in the menu **Mpide > Preferences > Sketchbook location**.

MPIDE manages two architectures: 8-bit ATmega and 32-bit PIC32.

Please note support for is MPIDE put on hold, as chipKIT is currently working on the transition of its IDE from the Arduino 23 to the 1.0.x or 1.5.x framework.

DFRobot BLuno Board

If you plan to use the BLuno board:

- Download and install Arduino 1.0.x or Arduino 1.5.x under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook in the menu **Arduino > Preferences > Sketchbook location**.

The Bluetooth connection is managed by the dedicated PlainProtocol ⓦ library. Just download and install it as a standard Arduino library. Please refer to the reference pages for simple ⓦ and advanced ⓦ usage.

Select the Arduino Uno board or the DFRobot BLuno board to compile and upload the sketch to the DFRobot BLuno board.

Digispark Board

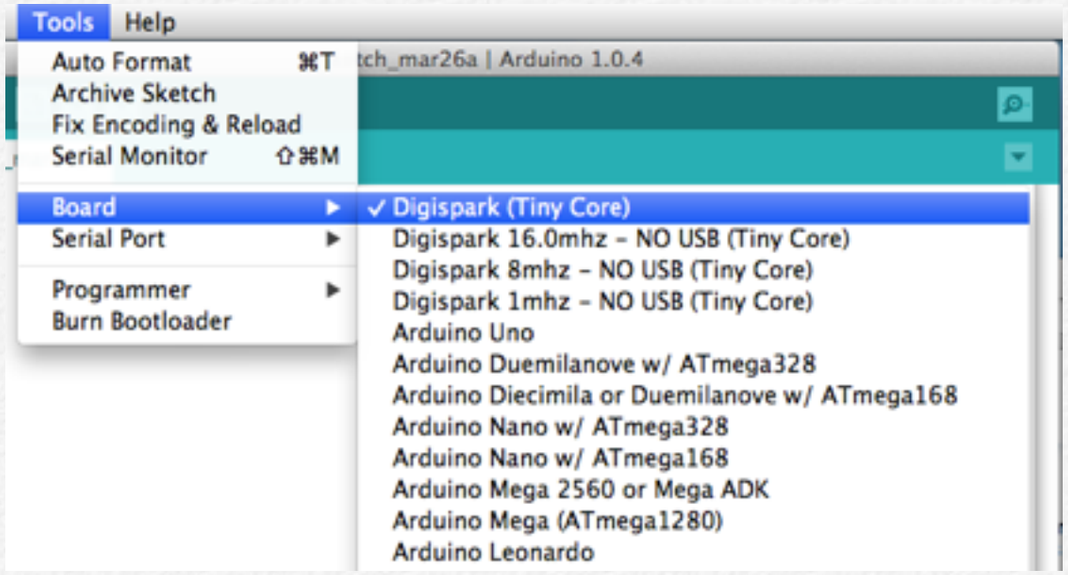
If you plan to use the Digispark board:

- Download the IDE.
- Open the `.dmg` file.
- Proceed with the installation under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook in the menu **Arduino > Preferences > Sketchbook location**.

As the IDE for Digispark boards modifies the Arduino IDE, you may want to have a separate modified Arduino IDE for the Digispark boards.

In that case, just change the name to `DigisparkArduino` and the icon to avoid any confusion with the standard Arduino.

The Digispark boards are under the menu **Tools > Board**.



LaunchPad MSP430, FR5739, Stellaris or Tiva C Series Boards

If you plan to use the MSP430, FR5739, Stellaris or Tiva C Series LaunchPad boards:

- Download and install Energia 11 under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook in the menu **Energia > Preferences > Sketchbook location**.

The Stellaris and Tiva LaunchPad boards may require the installation of additional tools. Follow the instructions provided by Energia.

If you plan to use the LaunchPad MSP430F5529,

- Download Energia release 0101E0010 and install it.
- Launch Energia.
- Create a new sketch, for example the `blinky` sketch.
- Perform one upload of the sketch with Energia, to ensure the firmware of the board is up-to-date.

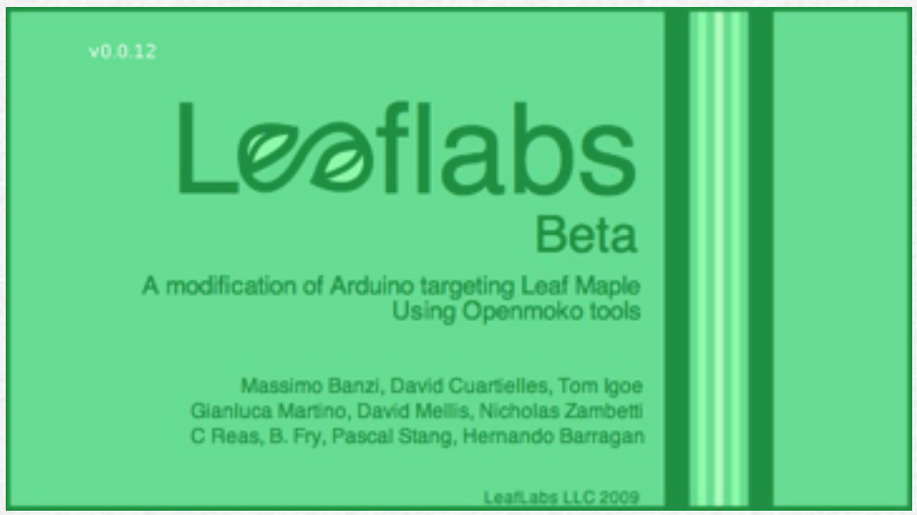
Energia manages two architectures: 16-bit MSP430 and 32-bit ARM.

⊞ All the LaunchPad boards feature a built-in hardware debugger. Learn more on how to debug at the chapter Debug the Project.

Maple Boards

If you plan to use the Maple boards:

- Download and install MapleIDE 0.0.12 under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook in the menu **MapleIDE > Preferences > Sketchbook location**.



Uploading a compiled sketch to the Maple board requires the utility `PySerial`.

To install PySerial,

- Download pyserial 2.6 ⓦ.
- Unzip it, for example in `~/Downloads/pyserial-2.6`.
- Open the **Terminal** and launch the following commands, with `~/Downloads/pyserial-2.6` being the path to `pyserial-2.6` in the example.

```
$ cd ~/Downloads/pyserial-2.6
$ sudo su
$ python setup.py install
```

Maple only knows `[u]int{8|16|32|64}` types and ignores the C99 standard `[u]int{8|16|32|64}_t` types.

If you plan to use libraries designed for another platform and want to ensure C99 compatibility,

- Open the `/Applications/MapleIDE.app/Contents/Resources/Java/hardware/leaflabs/cores/maple/libmaple_types.h` file.
- Add this `#include` to the `libmaple_types.h` file, although `stdint.h` defines other types.

```
#include <stdint.h>
```

As an alternative,

- Open the `/Applications/MapleIDE.app/Contents/Resources/Java/hardware/leaflabs/cores/maple/libmaple_types.h` file.
- And add the definitions for the `[u]int{8|16|32|64}_t` types.

```
// Added C99 Standard Types
#define uint8_t uint8
#define uint16_t uint16
#define uint32_t uint32
#define uint64_t uint64
#define int8_t int8
#define int16_t int16
#define int32_t int32
#define int64_t int64
```

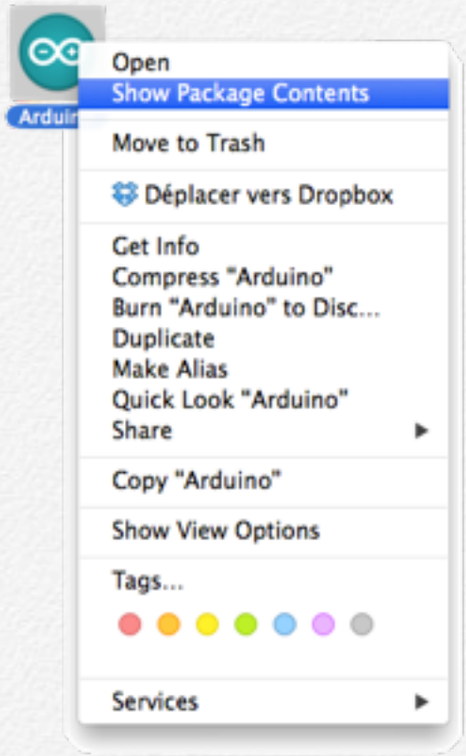
The Maple environment misses two important libraries: `strings.h` and `stream.h`.

Please note support for Maple is put on hold until a new version of the IDE or new boards are released.

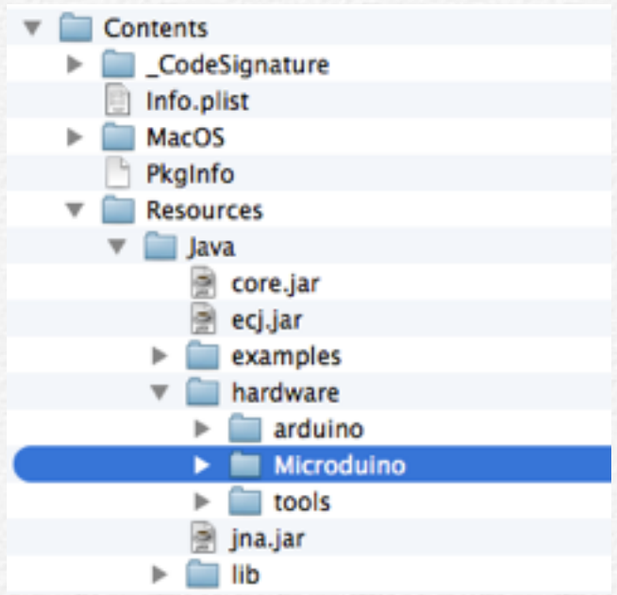
Microduino Boards

If you plan to use the Microduino boards:

- Check the Arduino 1.0.x IDE is already installed. Release 1.5.x doesn't work.
- Select the Arduino icon, call the contextual menu and select **Show Package Content**.



- Navigate to the `Content/Resources/Hardware` folder.
- Download the Microduino plug-in ⓦ.
- Open the `.zip` file.
- Copy/paste the unzipped folder `Microduino` into the `Content/Resources/Hardware` folder.

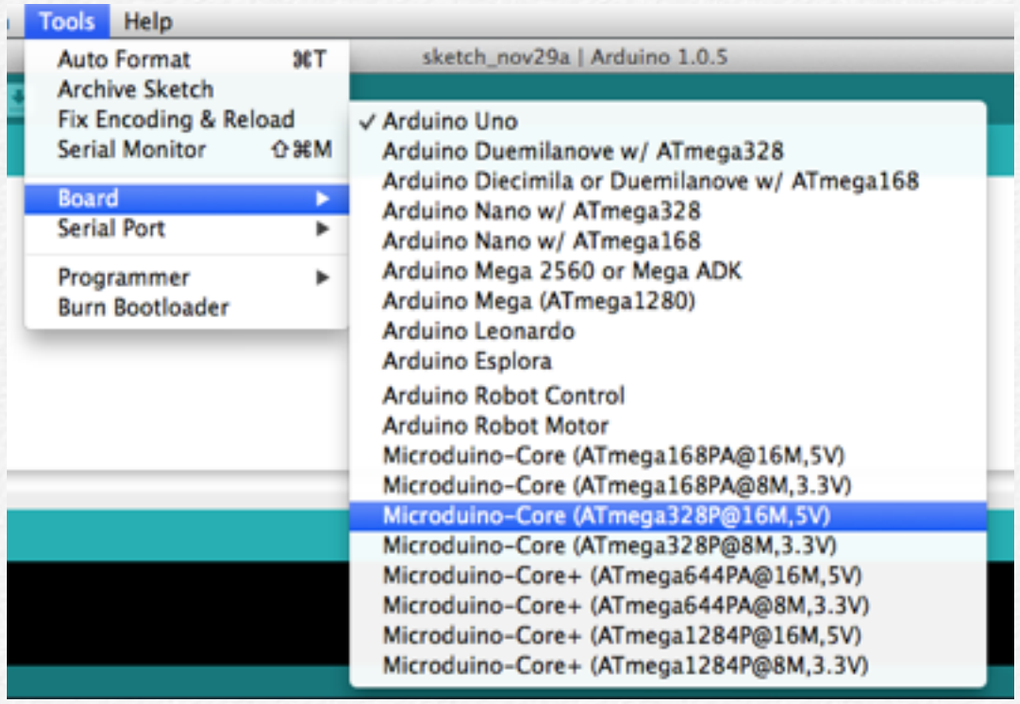


- Launch the modified Arduino IDE.
- Define the path of the sketchbook in the menu **Arduino > Preferences > Sketchbook location**.

As the IDE for Microduino boards modifies the Arduino IDE, you may want to have a separate modified Arduino IDE for the Microduino boards.

In that case, just change the name to `Microduino` and the icon to avoid any confusion with the standard Arduino.

The Microduino boards appear under the menu **Tools > Board**.



embedXcode supports two boards: the Microduino-Core with an ATmega328P at 16 MHz and 5V, and the Microduino-Core+ with an ATmega644PA at 16 MHz and 5V

Teensy Boards

If you plan to use the Teensy boards:

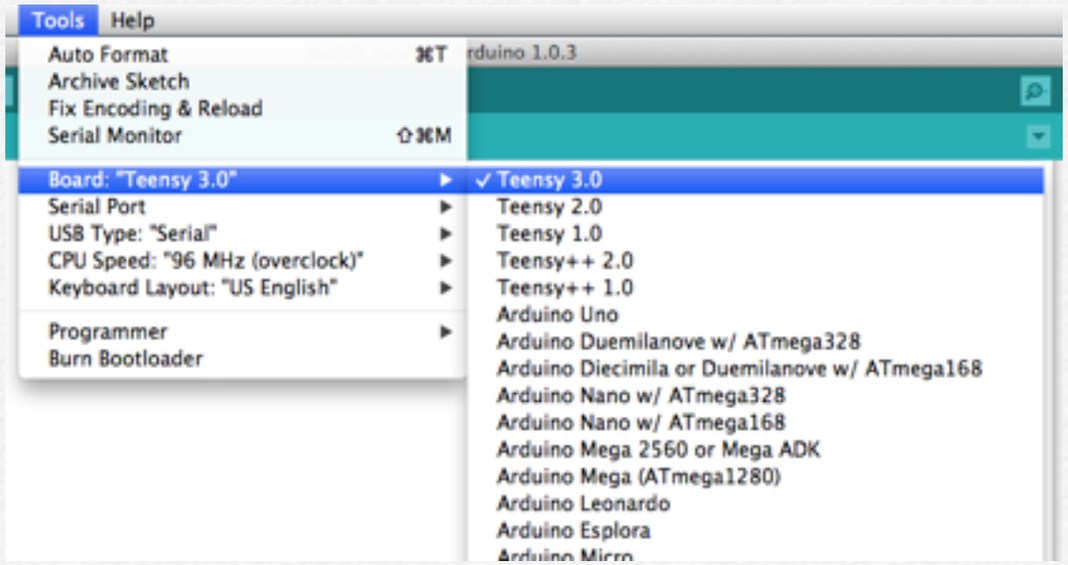
- Check the Arduino IDE is already installed.
- Download the Teensyduino plug-in.
- Open the `.dmg` file.

- Proceed with the installation.
- Launch Arduino.
- Define the path of the sketchbook in the menu **Arduino > Preferences > Sketchbook location**.

Starting release 1.13 of the Teensyduino IDE, support for the Teensy boards is now a plug-in for the Arduino 1.0.x IDE.

The Teensyduino installation adds Teensy-specific libraries into the Arduino IDE. It manages two architectures: 8-bit ATmega and 32-bit ARM.

The Teensy boards are under the menu **Tools > Board**.



As the IDE for Teensy boards modifies the Arduino IDE, you may want to have a separate modified Arduino IDE for the Teensy boards.

In that case, just change the name to `Teensyduino` and the icon to avoid any confusion with the standard Arduino.

Wiring Boards

 If you plan to use the Wiring boards:

- Download and install Wiring 1.0 under the /Application folder.
- Launch it.
- Define the path of the sketchbook in the menu **Wiring > Preferences > Sketchbook location**.

The two following files require to be deleted.

```
/Applications/Wiring.app/Contents/Resources/Java/cores/AVR8Bit/program.cpp
/Applications/Wiring.app/Contents/Resources/Java/cores/AVR8Bit/makefile
```

Please note support for Wiring is put on hold until a new version of the IDE or new boards are released.

Other Boards

 For other boards with a Processing-based Wiring-derived Arduino-like IDE, the procedure is the same:

- Download and install the corresponding Processing-based Wiring-derived Arduino-like IDE under the `/Application` folder.
- Launch it.
- Define the path of the sketchbook.

Additionally, you need to develop a specific makefile for the board, adapt the `Step1` and `Step2` makefiles, and add a specific upload procedure.

To add support for a specific board, please contact ⓦ me.

Utilities for Other AVR MCUs

⊞ This section requires embedXcode+.

Some boards aren't defined by Arduino, so you need to add them. A list is supplied by the the `avrduderc.txt` file located in the `embedXcode` folder and needs to be copied into the user's home folder.

- Open the **Terminal** and launch the following commands

```
$ cd ~/Documents/embedXcode
$ cp avrduderc.txt ~/.avrduderc
```

For more information, please refer to the AVRDUDE User Manual ⓦ.

Similarly, if you plan to use other MCUs from AVR, you may need to install a more recent version of AVRDUDE, as Arduino still uses release 5.11 dated August 2011.

To install a more recent release of AVRDUDE,

- Download AVRDUDE release 6.0.1 ⓦ or later,
- Unzip the compressed file,
- Open the **Terminal** and launch the following commands, with `path_to_avrdude` being the path to `avrdude-6.0.1`, `~/Downloads/avrdude-6.0.1` in the example.

```
$ cd ~/Downloads/avrdude-6.0.1
$ ./configure; make; make install
```

- To know where `avrdude` has been installed, launch the command

```
$ which avrdude
/usr/local/bin/avrdude
```

- To check the release and know where the configuration file is located, launch the command

```
$ /usr/local/bin/avrdude -v
avrdude: Version 6.0.1, compiled on Nov  2 2013 at 17:50:28
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2009 Joerg Wunsch
System wide configuration file is "/usr/local/etc/avrdude.conf"
```

The configuration file for AVRDUDE is located at `/usr/local/etc/avrdude.conf`.

Utilities for Programmers

⊞ This section requires embedXcode+.

If you plan to use a programmer like the USB ASP from Protostack ⓦ, the USBtinyISP from Adafruit ⓦ, the USB Spoke POV dongle or similar, you may need to install the FTDI drivers,

- Download FTDI USB Serial Driver version 2.2.18 ⓦ or later,
- Install the drivers,
- Reboot your Mac.

Rebooting the Mac is compulsory, otherwise the drivers aren't taken into account.

For more information, please refer to the FTDI Drivers Installation Guide for Mac OS X ⓦ.

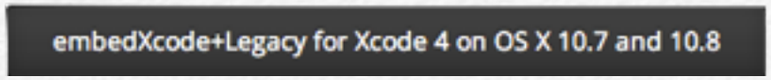
Install embedXcode

Install the Template

The installation of embedXcode is done as any other standard application thanks to the use of an installation package.

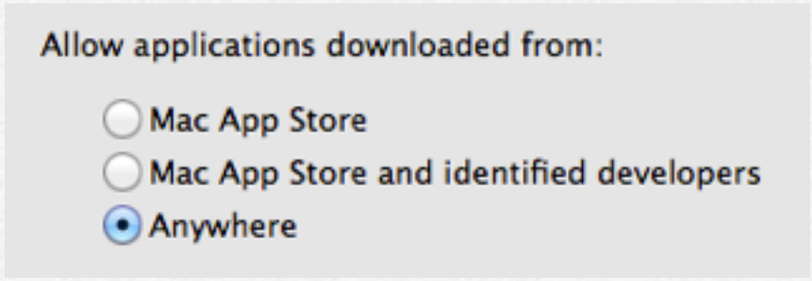
- Download the embedXcode package directly from the download page ⓦ for systems with Xcode 5 running on OS X 10.8 *Mountain Lion* or OS X 10.9 *Mavericks*.

Please note the version for legacy systems with Xcode 4 running on OS X 10.7 *Lion* or OS X 10.8 *Mountain Lion* is no longer maintained.



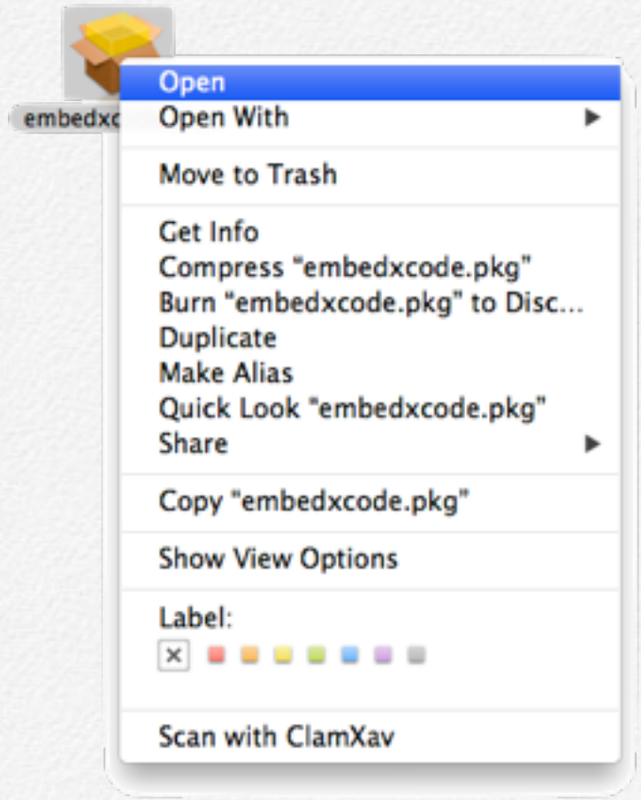
- Double-click on the `embedXcode` package icon to launch the installation.

According to the security parameters defined in **> Preferences > Security & Privacy > General**, OS X 10.8 *Mountain Lion* and OS X 10.9 *Mavericks* requires installation packages to be signed.

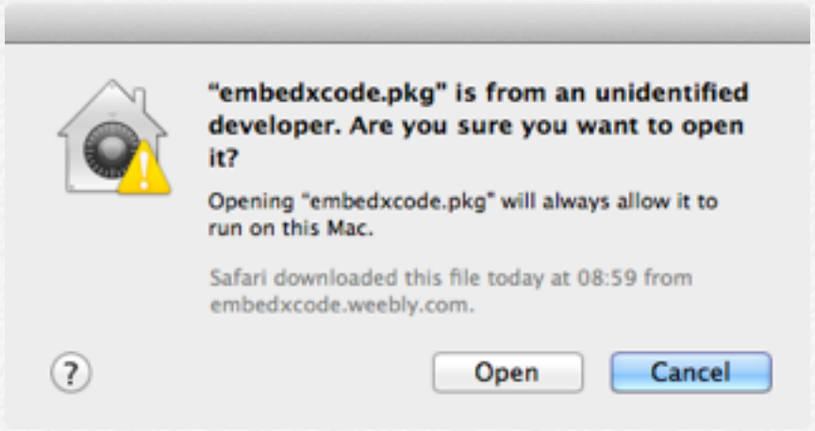


As the embedXcode installation package is not signed, the following message may appear.

- In that case, **ctrl-click** on the embedXcode package icon to display the contextual menu and select **Open**.



A windows prompts for confirmation.



- Answer **Open** to proceed.

The installation welcome screen is displayed.

- Click on **Continue** to proceed.

The license is displayed.

- Click on **Continue**.

- Click on **Agree** to proceed.

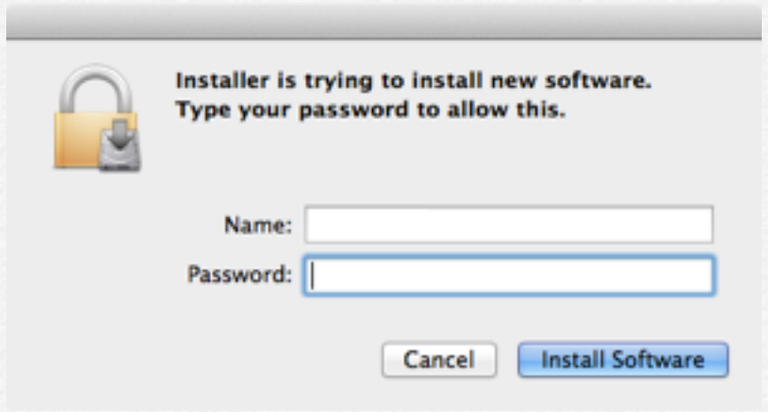
An additional window may ask for the destination.

- Select **Install for me only** and then click on **Continue** to proceed.

Next window asks for confirmation of the installation.

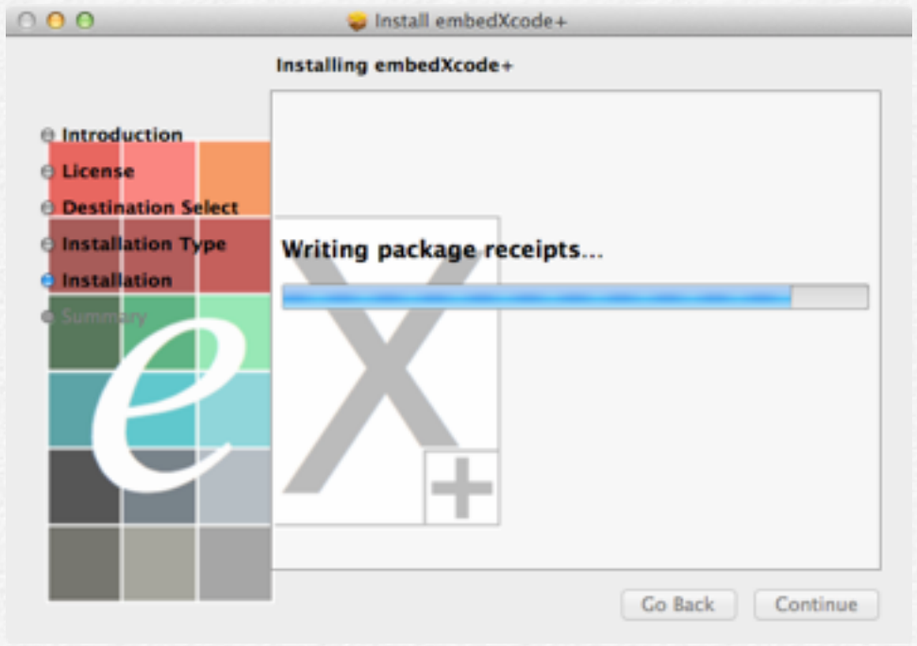
- Click on **Install** to proceed.

A window prompts for your name and password.

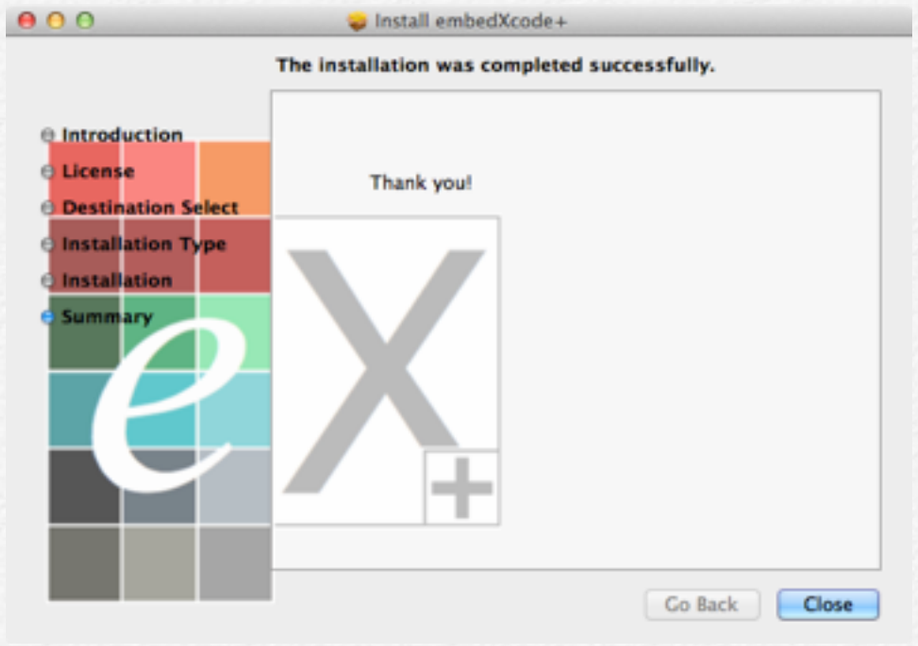


- Enter both and click on **Install Software**.

The installation goes on.



The installation has been successful.



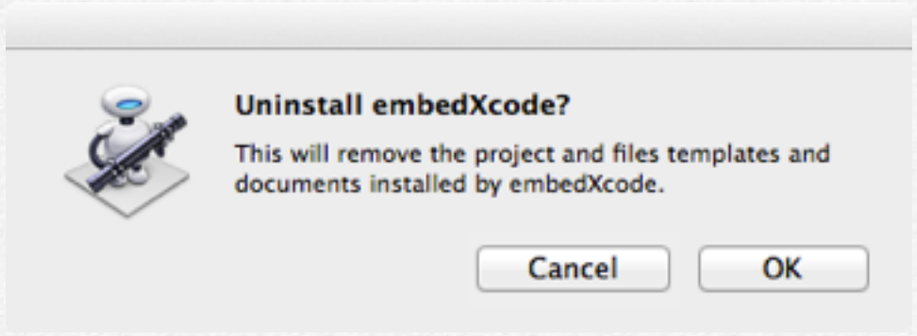
- Click on **Close** to finalise.

Uninstall the Template

There is an automatic procedure to uninstall the template.

- Click on **Uninstall** located on the `embedXcode` folder:

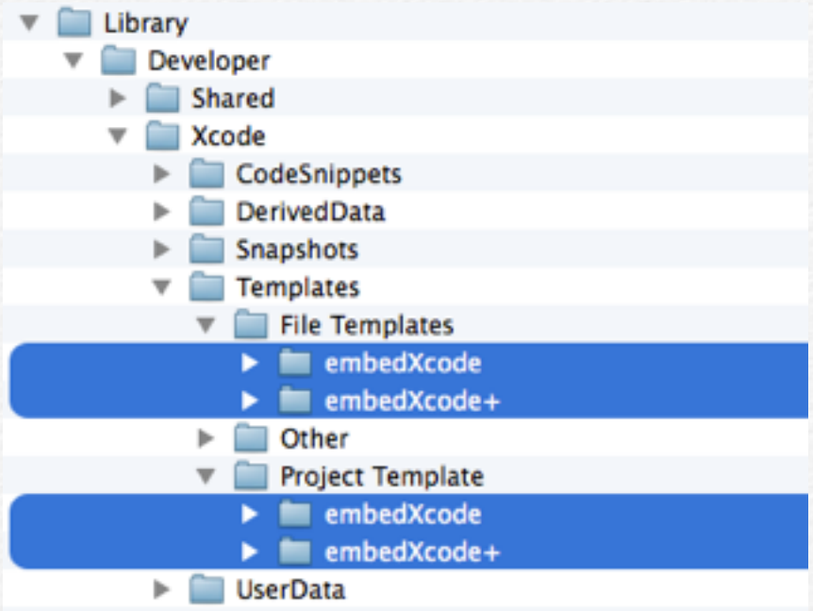
A dialog box opens and asks for confirmation:



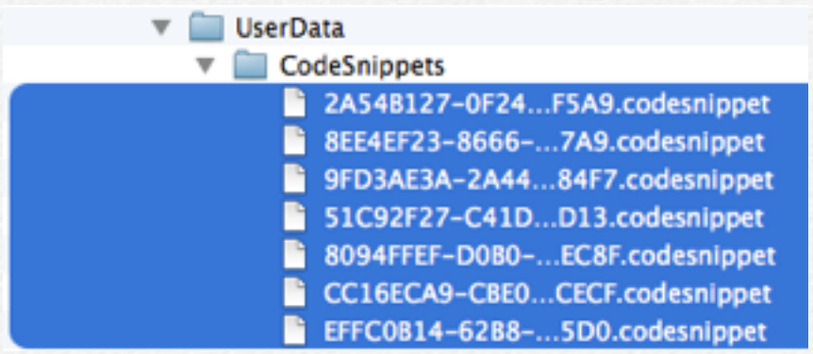
- Click on **OK** to proceed or on **Cancel** to abort.

As an alternative, the manual procedure requires to delete the following files and folders:

- Delete the folder `embedXcode` under `~/Documents`.
- Delete the folder `embedXcode` or `embedXcode+` under `~/Library/Developer/Xcode/File Templates`.
- Delete the folder `embedXcode` or `embedXcode+` under `~/Library/Developer/Xcode/Project Templates`.



- Delete the files under `~/Library/Developer/Xcode/UserData/CodeSnippets`.



For the IDEs and optional tools you may have installed, please refer to their respective procedures supplied by their editors.

Install Optional Debugging Tools

⊞ This section requires embedXcode+ and a board with a built-in hardware debugger.

Debugging requires a board with a built-in hardware debugger. Only the LaunchPad boards provide such a feature.

Debugging has been tested successfully on the following boards:

- LaunchPad MSP430G2,
- LaunchPad MSP430F5529,
- Experimenter Board MSP430FR5739,
- LaunchPad Stellaris LM4F120H5QR now Tiva C Series.

The LaunchPads MSP430G2 and MSP430F5529 and the Experimenter Board MSP430FR5739 require no additional tool as the debugger is already included in the Energia IDE.

Only the LaunchPad Stellaris LM4F120H5QR now Tiva C Series requires the installation of an additional tool, namely openOCD.

The installation of openOCD can be done using Homebrew or MacPorts and requires the previous installation of libusb ⓦ, a library that provides an easy access to USB devices.

Install openOCD with Homebrew

To install openOCD with Homebrew,

- Open Terminal and launch the following command to install Homebrew.

```
$ ruby -e "$(curl -fsSkL raw.github.com/mxcl/
homebrew/go)"
```

- Launch the following command to install `libusb`.

```
$ brew install libusb
```

- Launch the following command to install `openOCD`.

```
$ brew install openocd
```

Install openOCD with MacPorts

To install openOCD with MacPorts,

- Download and install MacPorts ⓦ.
- Open Terminal and launch the following command to install `libusb`.

```
$ sudo port install libusb
```

- Launch the following command to install `openOCD`.

```
$ sudo port install openocd +ti
```

Install Optional Self-Documentation Tools

⊞ This section requires embedXcode+.

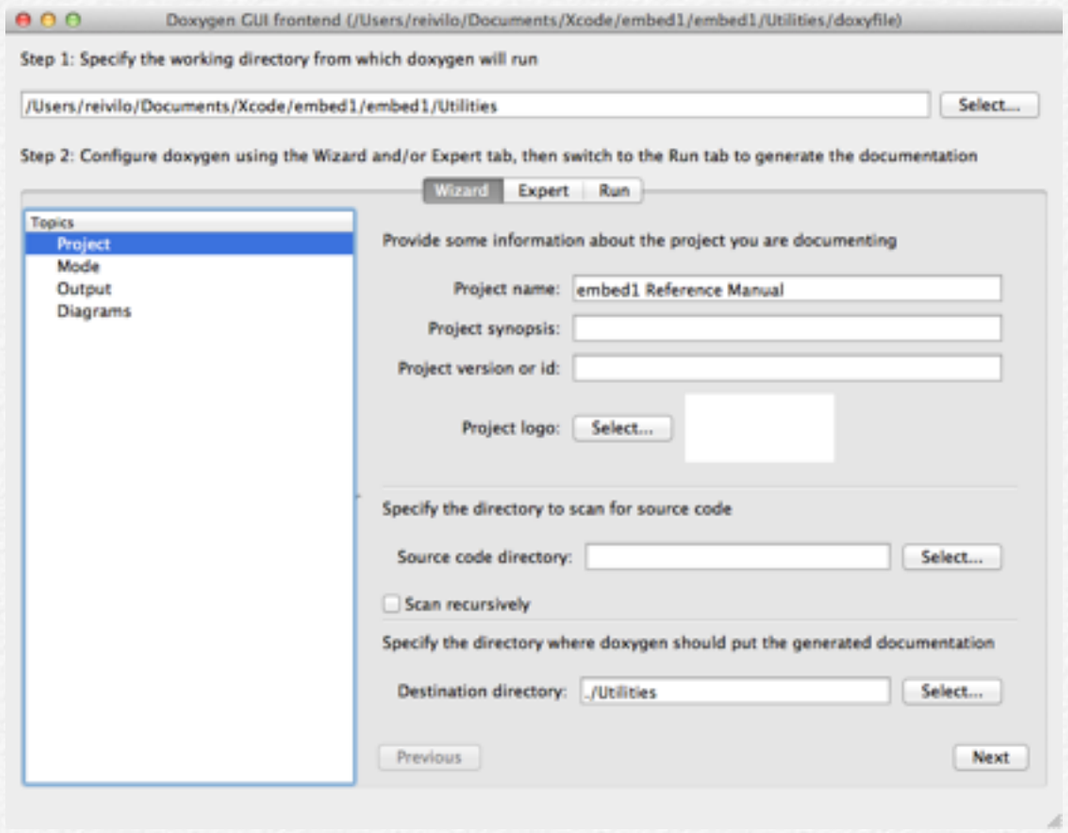
The installation of the following tools is optional and only required if you want to use the self-documentation feature.

- Doxygen generates all the help files based on comments added to the code. Doxygen requires GraphWiz to draw elaborate dependency trees.
- Doxygen Helper and ThisService provide a handy shortcut to add a ready-to-use template for comments.
- TeXShop translates the Doxygen output into PDF documents.

Doxygen

Install Doxygen ⓦ to parse the code, looks for comments and generate the HTML pages.

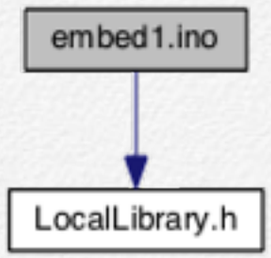
I strongly recommend to install DoxyWizard included in the package. DoxyWizard provides a GUI for an easy tweaking of the parameters.



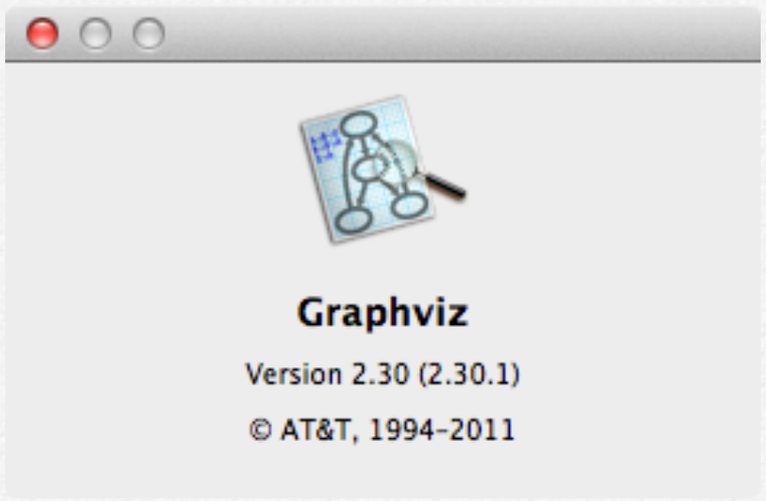
Doxygen generates HTML files, Xcode native help files and LateX files. LateX files can be converted into PDF documents using TeXShop.

GraphWiz

Install GraphWiz to add dependency trees.

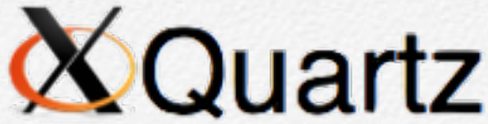


GraphWiz ⓦ 2.30 is now compatible with OS X 10.8 *Mountain Lion*. Learn more at the Mac OS dedicated page ⓦ.



However, GraphWiz requires X11, which is no longer included with OS X 10.8 *Mountain Lion*.

In a technical note ⓦ, Apple recommends to use XQuartz ⓦ.



The previous release, GraphWiz 2.28, is reported having problems with OS X 10.8 *Mountain Lion*.

A temporary solution ⓦ consists on using Homebrew ⓦ and then installing `libtool`.

- Open Terminal and launch the following commands.

```
$ ruby -e "$(curl -fsSkL raw.github.com/mxcl/
homebrew/go)"
$ brew install libtool
```

Doxygen Helper

To ease and speed up the writing of the comments, I use the Automator Service Doxygen Helper ⓦ developed by Fred McCann and Duck Rowing.

By just selecting a function and pressing ⌘⇧**D cmd-shift-D**, the helper generates a template for the comment lines.

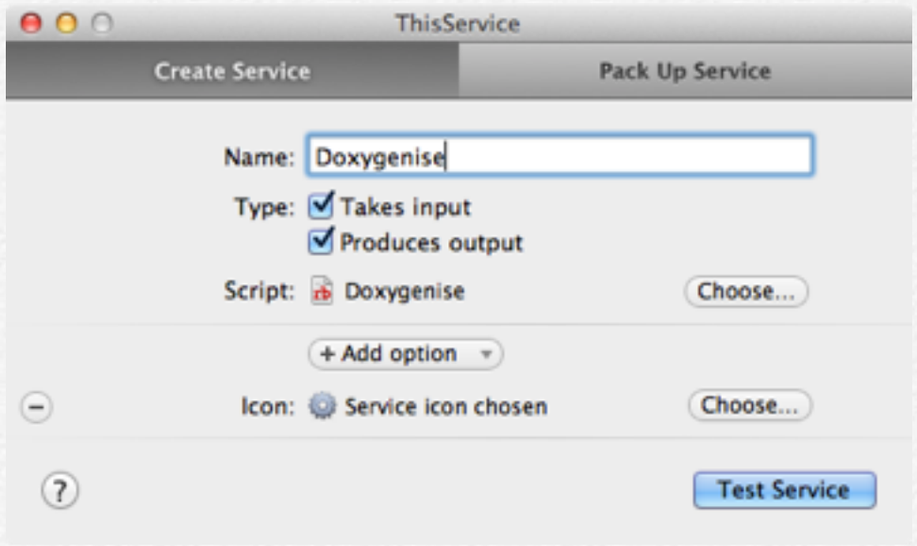
```
///
/// @brief    Blink a LED
/// @details  LED attached to pin is light on off
/// @n        Total cycle duration = ms
/// @param    pin pin to which the LED is attached
/// @param    times number of times
/// @param    ms cycle duration in ms
///
void blink(uint8_t pin, uint8_t times, uint16_t
ms);
```

There are three different procedure for the installation:

- Download the Doxygen Helper ⓦ Automator Service.
- Install it following the instructions provided on the Doxygen helper ⓦ page.

ThisService

As an option for installing the Doxygen Helper service, the very nice ThisService 3 ⓦ utility does all the job.

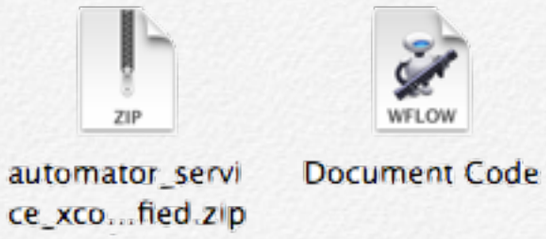


- Download the Doxygen Helper ⓦ Automator Service.
- Download the ThisService 3 ⓦ utility.
- Install the Automator Service by following the instructions provided at the Doxygen Shortcuts in Xcode 4 ⓦ page.

Automatic Installation

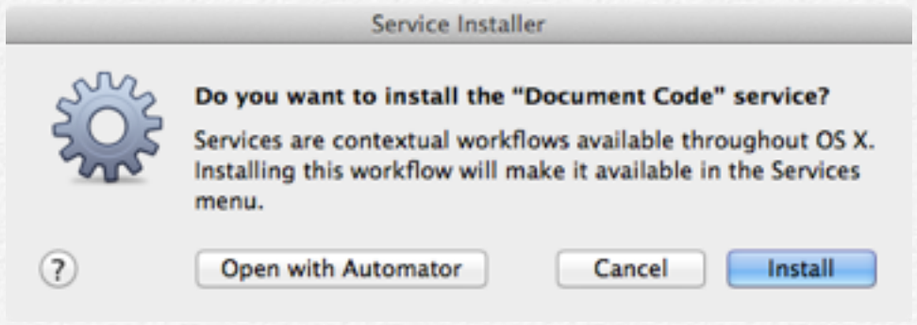
A modified version of the Automator Service is included in the `~/Documents/embedXcode` folder. To install it:

- Unzip the `automator_service_xcode_modified` file.



- Double-click on the `Document Code` workflow.

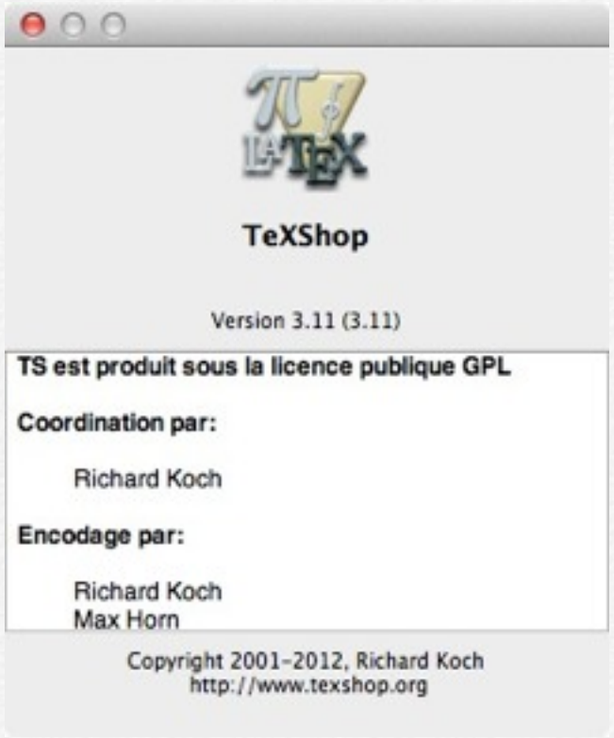
A window pops-up and asks for confirmation:



- Confirm by clicking on **Install**.
- Optionally, click on **Open with Automator** to launch Automator and edit the service.

TeXShop

Finally, TeXShop builds a PDF document from the LateX files Doxygen has generated.



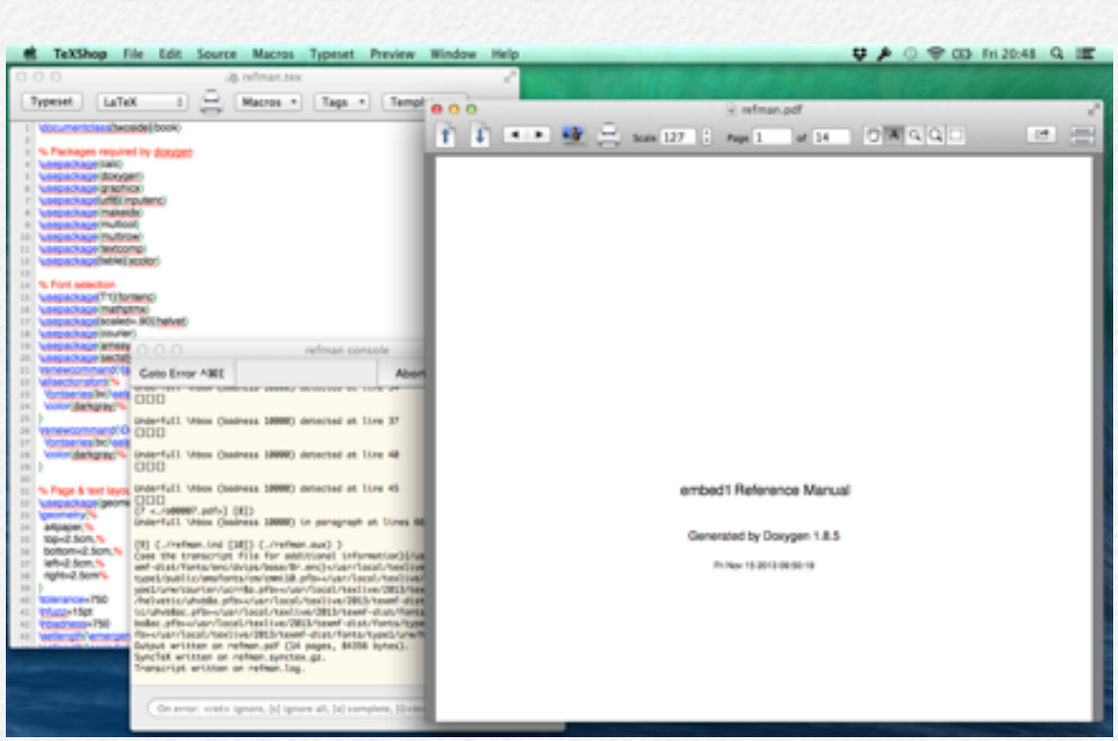
To install TeXshop,

- Download TeXShop ⓦ as a standalone application, or
- Download TeXShop and other utilities including fonts as part of the MacTeX-2013 ⓦ distribution.

embedXcode supports the MacTeX-2012 and MacTeX-2013 distributions.

- Launch the installation package and follow the instructions.

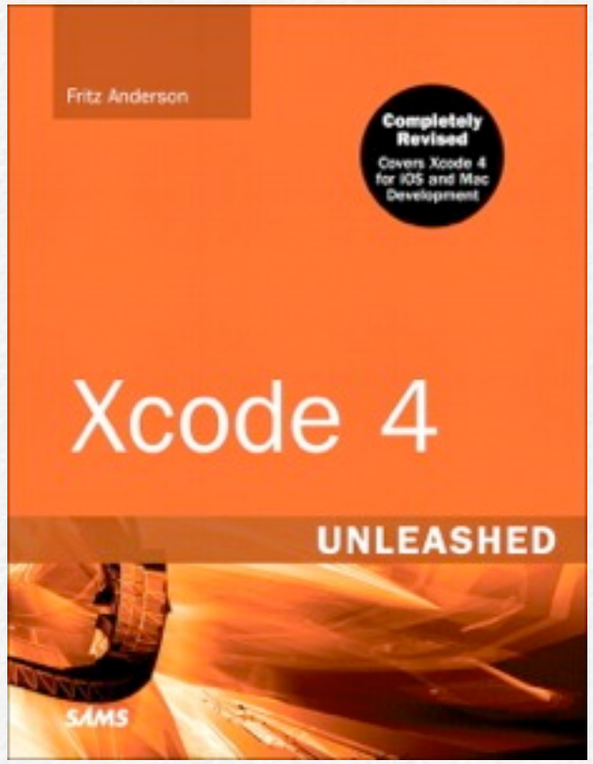
Below, an example of a LateX file translated into a PDF document.



Recommended Books

Xcode

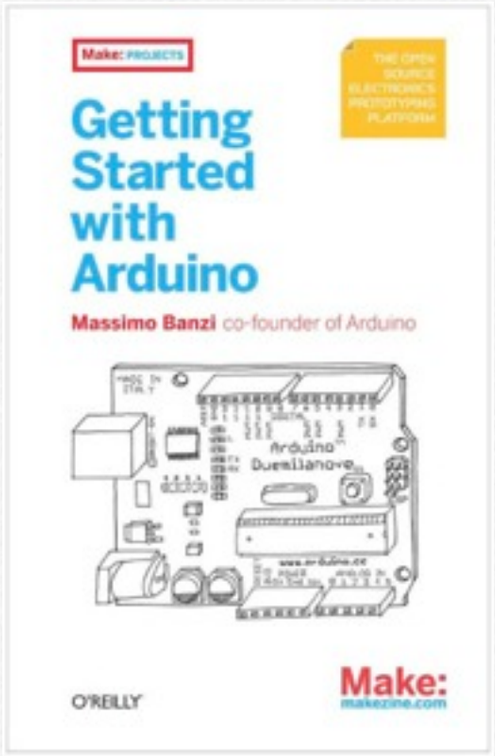
Compared to the Arduino or similar IDEs, Xcode is a whole new environment and, as other advanced and professional IDEs, requires some learning.



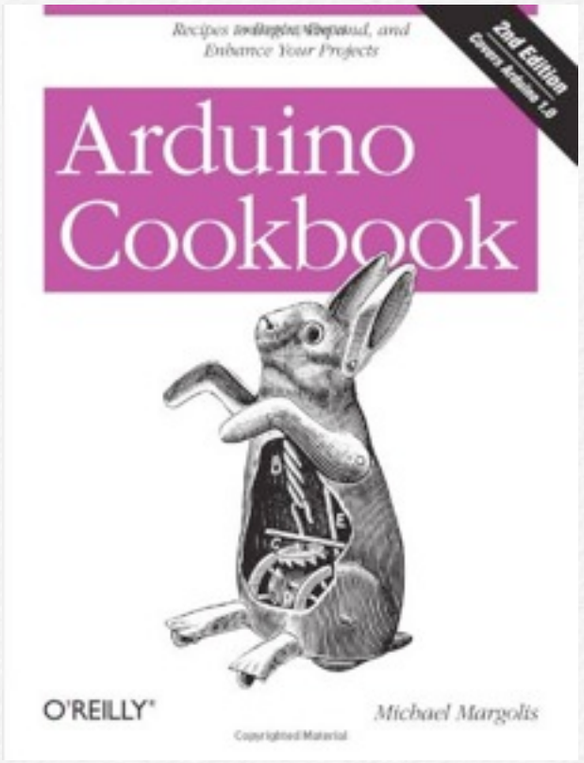
For a good introduction to Xcode, I recommend the book **Xcode 4 Unleashed** by Fritz F. Anderson.

Arduino

If you are fluent with Xcode and C/C++ programming but you discover the Arduino boards and the embedded computing world, I recommend the book **Getting Started with Arduino** by Massimo Banzi as a good first step-by-step approach.



For a more advanced exploration of Arduino possibilities, I recommend the **Arduino Cookbook** by Michael Margolis.



It is a reference manual for both the hardware and the software around Arduino.

LaunchPad

How to discover embedded computing?

One major difficulty with embedded computing is the number of fields implied: electronics, computing, programming. Another difficulty is making the connections between those different fields.

The book **Getting Started with the MSP430 LaunchPad** by Adrian Fernandez and Dung Dang offers a step-by-step introduction to embedded computing with hands-on projects.

Each chapter starts with a project, introduces the concepts and then discusses them.

I really like the progressive approach of the 26 projects, from a blinking LED to a more complex musical instrument or message displayed on a LCD.

Create a New Project

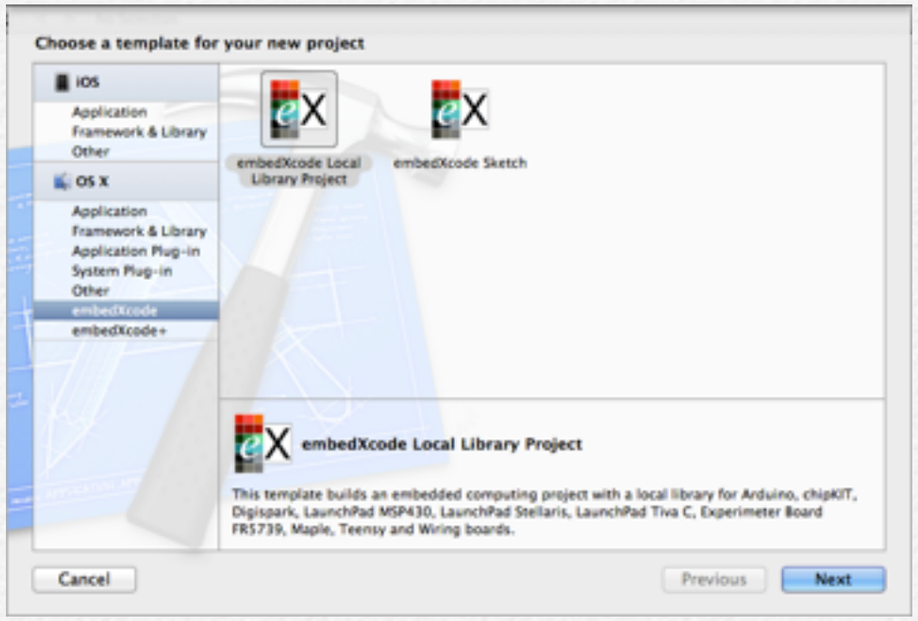
2

Create and configure a new project.

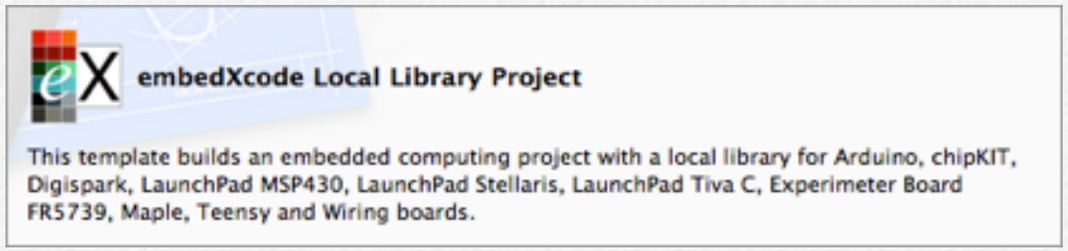
Create a New Project

To create a new project:

- Call the menu **File > New > New Project...** or press ⇧⌘**N**.
- If **embedXcode** is installed, select it on the list on the right.



- Two options are available:



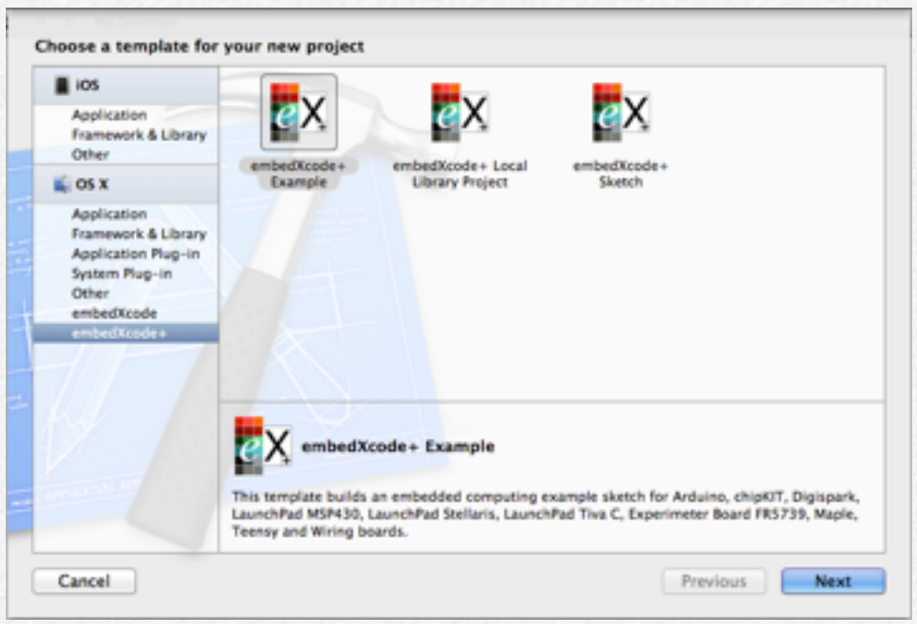
  - The template **embedXcode Local Library Project** includes a local library populated with a blinking example.

This template is recommended for developing a new library.

- The template **embedXcode Sketch** only includes a sketch populated with a blinking example.

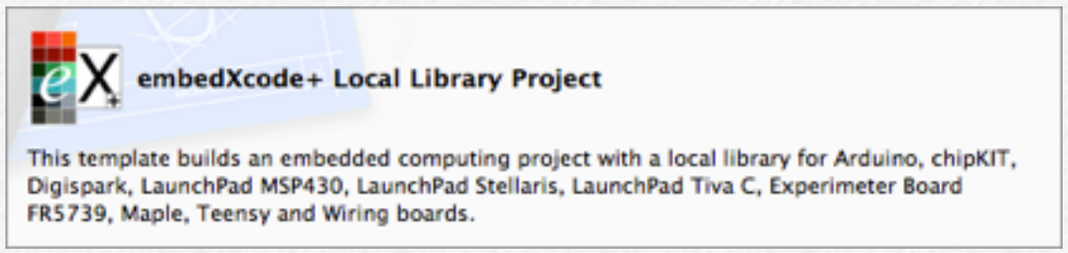
- Select one template and then click on **Next** to proceed to the next step.

- ⊞ If **embedXcode+** is installed, select it on the list on the right.



- Three options are available:

  - The template **embedXcode+ Example** provides a sketch populated with a blinking example.

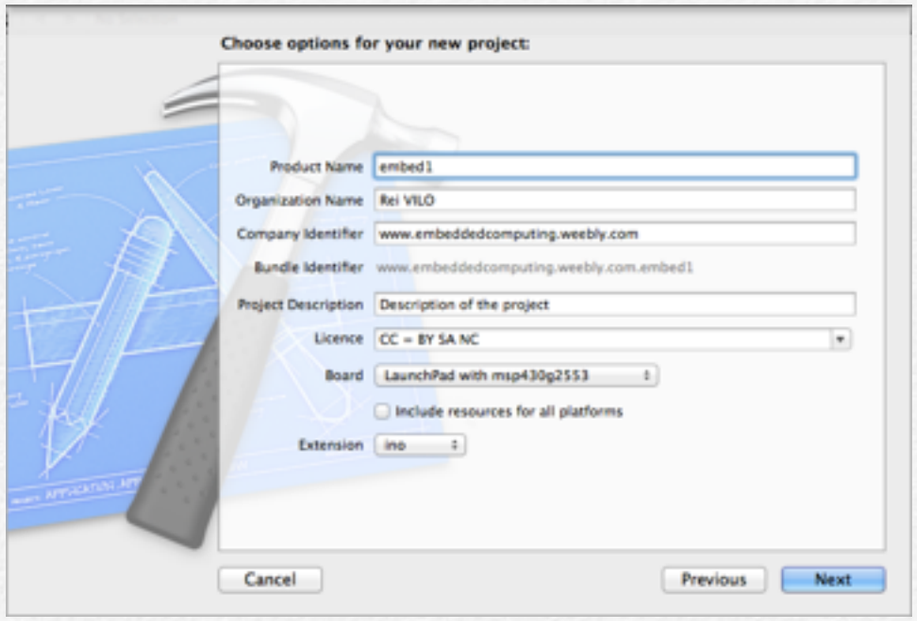


  - The template **embedXcode Local Library Project** includes an empty local library. This template is recommended for developing a new library.

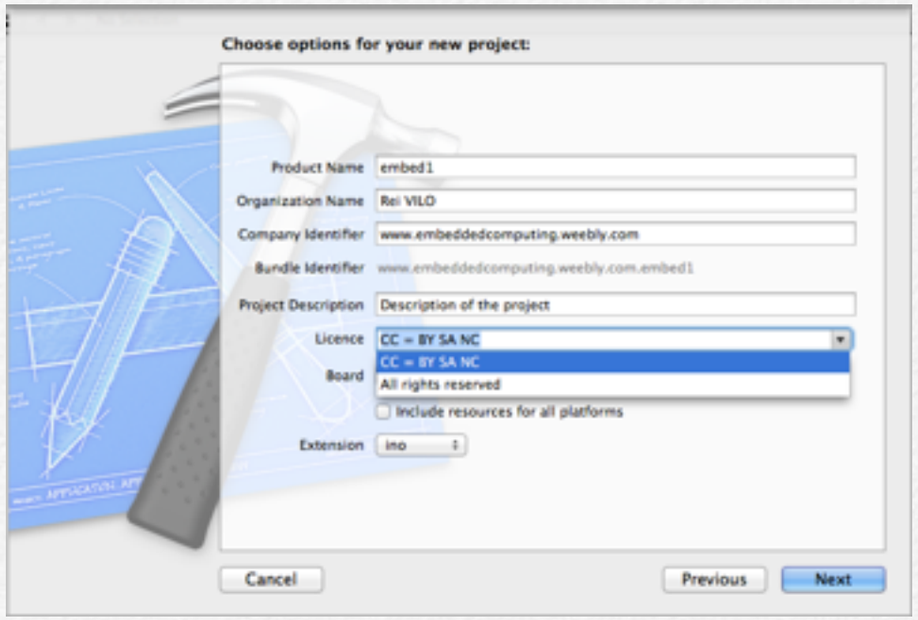
  - The template **embedXcode Sketch** only includes a empty sketch.

- Select one template and then click on **Next** to proceed to the next step.

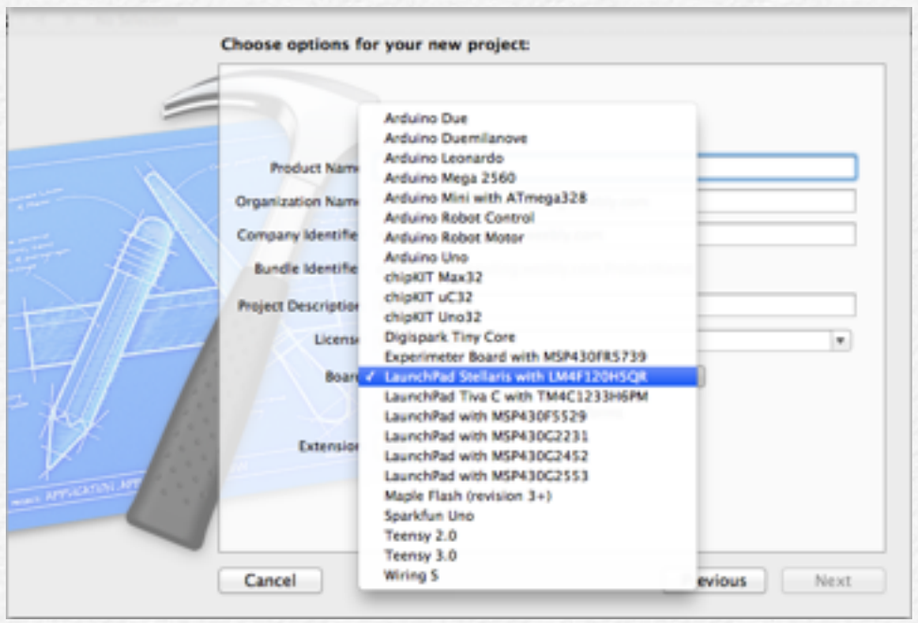
Once the option is selected, a new window asks for the details of the project:



- Type in the **Name** of the project. As for the Arduino IDEs, avoid space and special characters in the project name.
- Optionally, provide the **Organisation Name** and the **Company Identifier**. Those fields are pre-populated.
- Provide a **Description** of the project. This field is optional.
- ⊞ Define the **License** on the combo list. If the desired license isn't available on the list, type it in.



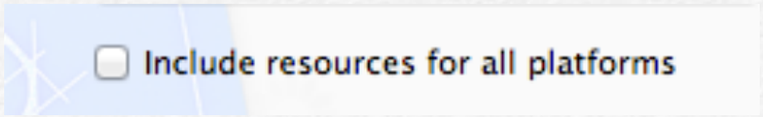
- Select the **Board** on the drop-down list.



The board can be changed afterwards. Please refer to the procedure described at the section Change the Board.

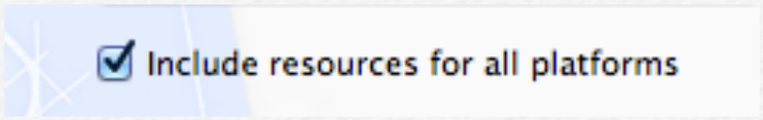
If your board isn't listed, you can create a configuration file. Please refer to the section Add a Configuration File for a New Board.

- ⊞ If you plan to use the project on one platform only, leave the box **Include resources for all platforms** unchecked.



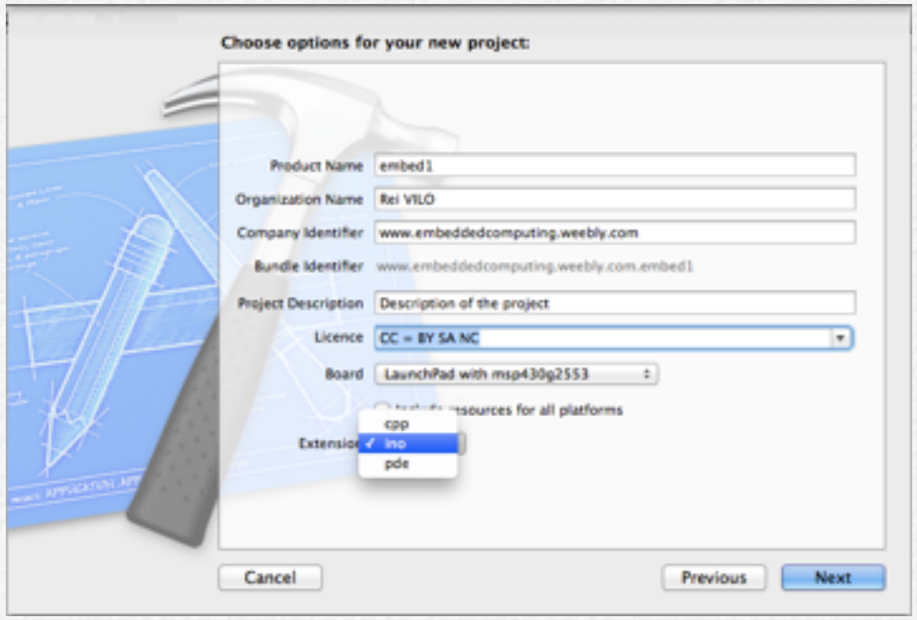
On the example, the board selected is the `LaunchPad with msp430g2553`. Only the resources for the Energia platform will be included.

- ⊞ If you plan to use the same project on different platforms, check the box **Include resources for all platforms**,



The resources are going to be included for all the platforms: Arduino, chipKIT, Digispark, Energia, Maple, Teensy and Wiring.

- Select the **Extension** on the drop-down list:

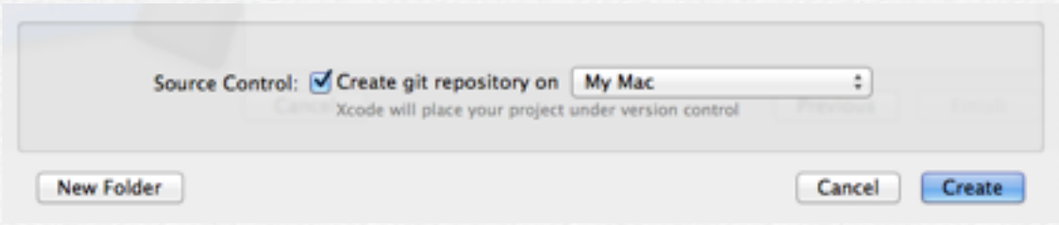


  - Choose **pde** for Arduino 0023, chipKIT MPIDE, Wiring and Maple,
  - Choose **ino** for Arduino 1.0 or 1.5, Teensy and Energia,
  - ⊞ Choose optionally **cpp** for a standard C++ file.

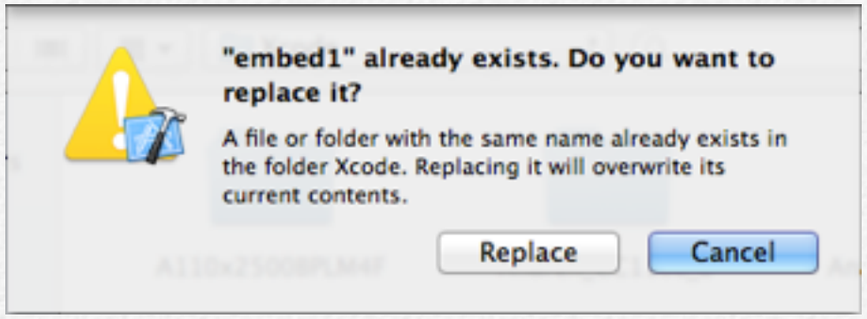
Using a `pde` or `ino` extension allows to edit the sketch with the standard IDEs of the boards.

- Click on **Next** to proceed to the last step.
- Select the folder where the project is going to be saved.

- Optionally, check **Create local git repository for this project** if you want so.



- Then click **Create** to confirm and create the project.
- If the project already exists, a window asks for confirmation:

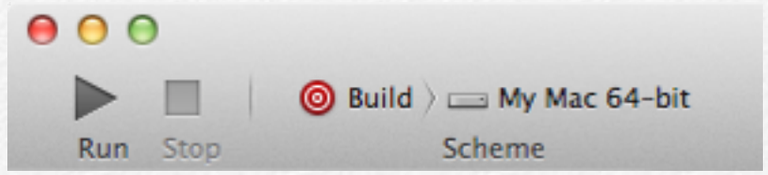


- Click on **Replace** to replace the previous project with the new one or **Cancel** to change the name.

Prepare the Project

The template doesn't define all the parameters, so the newly created project needs to be prepared.

This is done automatically during the first compilation. Simply select one target, as **Build** or **Make** and click on **Run**.



The automatic procedure provides an easy way to:

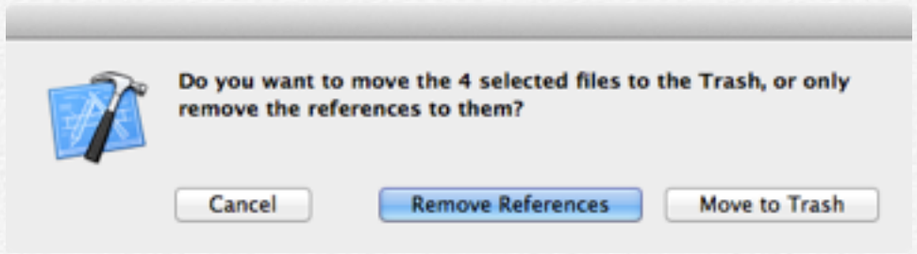
- Declare `.pde` or `.ino` file as C++ code for code-sense,
- Define the directory for the targets,
- Add all the cores, variants and libraries from the installed platforms under the group **Reference**,
- Add the user's libraries for the selected platform under the group **Sketchbook**,
- Declare all the files of the project for code-sense.
- Enter the password of the Arduino Yún board if connected through Ethernet or WiFi.

The automatic procedure is only launched once, so the manual procedures are detailed in the next section.

The manual procedures Declare Sketch .pde or .ino File as C++ File, Define the Directories for the Targets, Declare Sources for Code-Sense and Add User's Libraries are no longer required.

The manual procedure Declare User's Sketchbook is optional.

When removing files from the **Resources** group, always select **Remove References** and not **Move to Trash**.

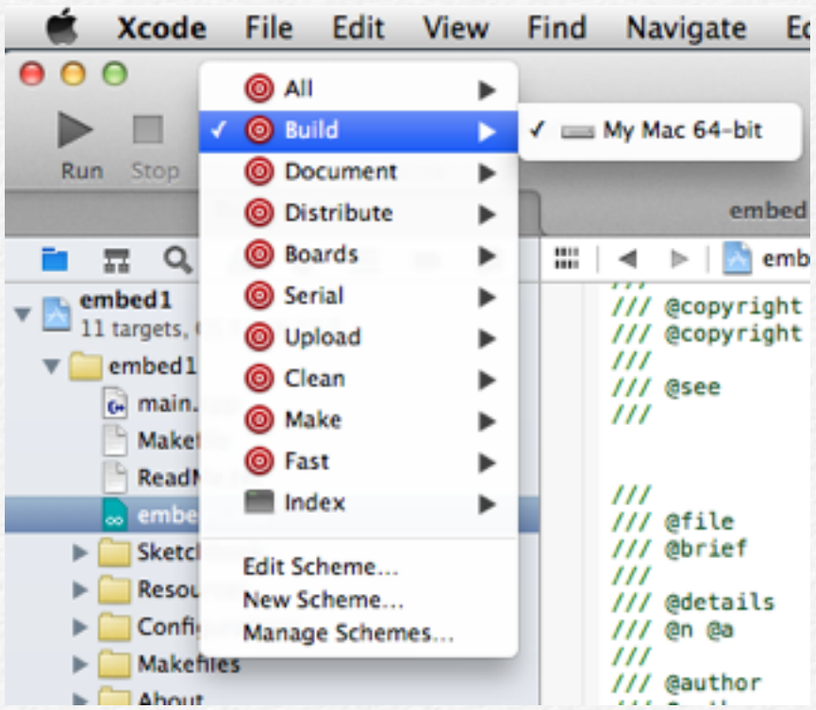


The option **Remove References** only removes the reference from the Xcode project while the option **Move to Trash** deletes the original file.

Launch the Procedure

To proceed with the automatic configuration procedure,

- Select any target, as **Build** or **Make**.



- Check no other process is running on Xcode, like indexing.
- Click on **Run** to launch it.

The utility starts and prepares the project.

Please note the user's libraries are taken from the platform corresponding to the board selected during the creation of the new project.

In this example, the `LaunchPad with MSP430G2553` board was selected for the new project, and the corresponding platform is `Energia`.

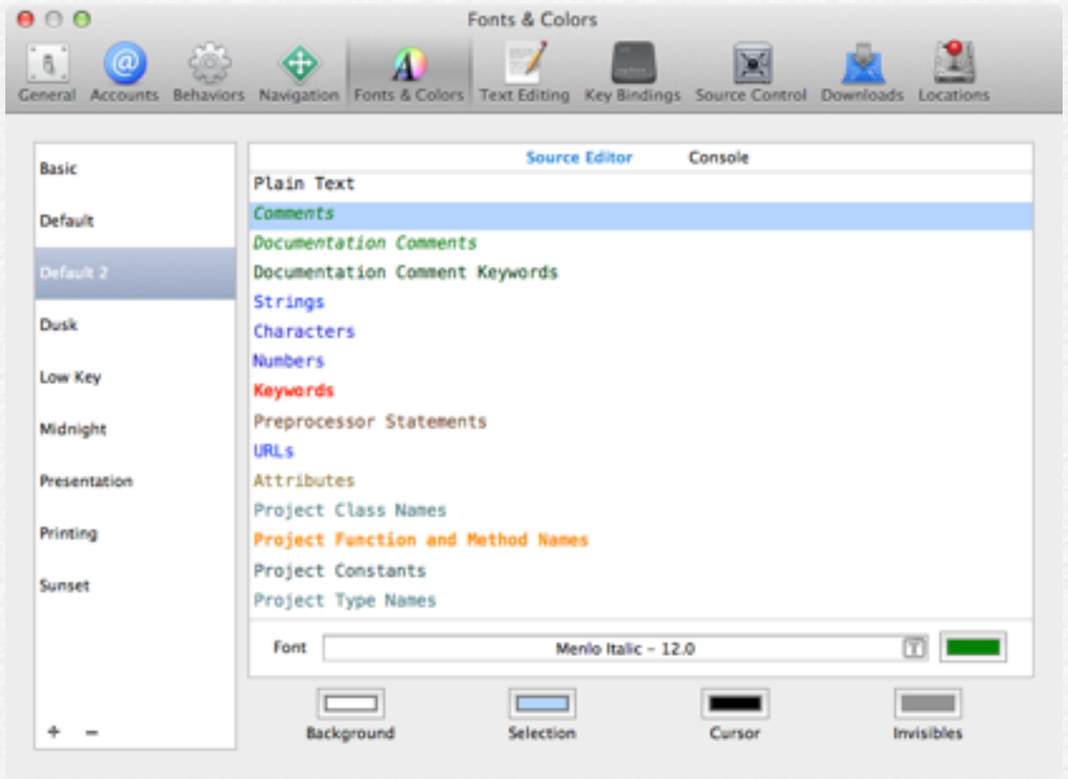
The sketchbook folder is defined in the Energia preferences, and the user's libraries are under the `libraries` or `Libraries` folder.

```
embedXcode_prepare
----------------------------------
Embedded Computing on Xcode
© Rei VILO, 2010-2013
All rights reserved
http://embedXcode.weebly.com/


      embedXcode_prepare • Nov 14, 2013  • 2.22 • Support for Xcode 5

==== Prepare embed1 ====
Project /Users/ReiVilo/Documents/Projets/Xcode/embed1/embed1.xcodeproj found
Sketchbook /Users/ReiVilo/Documents/Projets/Energia/Libraries found
---- Core, application and user libraries ----
Arduino 1.0     214 files
Energia         123 files
Energia         335 files
Maple           134 files
Teensyduino     511 files
Wiring      122 files
Sketchbook      128 files
---- 1567 libraries collected ----
Build file section updated
File reference section updated
Group section updated
Sources build phase section updated
Libraries listed in makefile
==== embed1 prepared ====
```

The following message may appear if there are too many folders and libraries. In such a case, some user's libraries may be missing as they are parsed after the core and application libraries.

```
WARNING  Libraries overflow reached
```

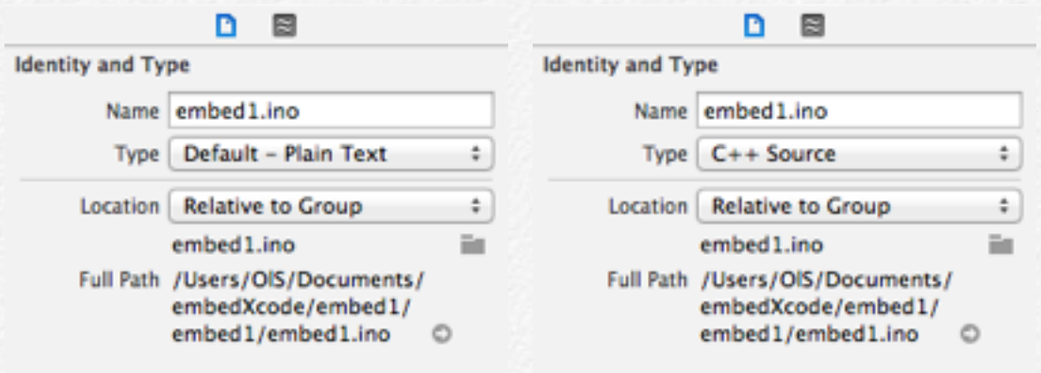
The colours are defined on the **Fonts & Colours** pane available on the preferences. To display the preferences, call the menu **Xcode > Preferences** or press ⌘**,**.



Updated Parameters

Here's the list of the parameters updated during the automatic preparation procedure, with the corresponding manual procedures:

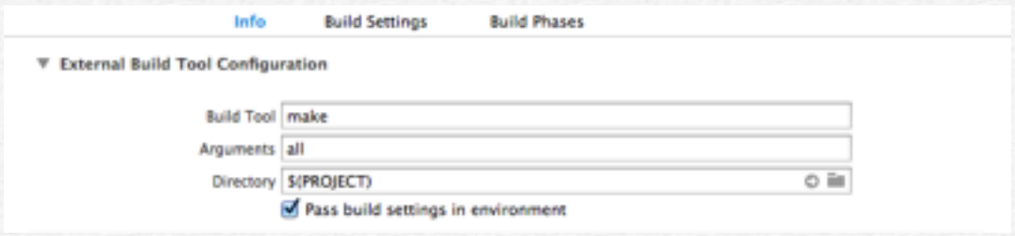
- The `.pde` or `.ino` files are declared as C++ code for code-sense.



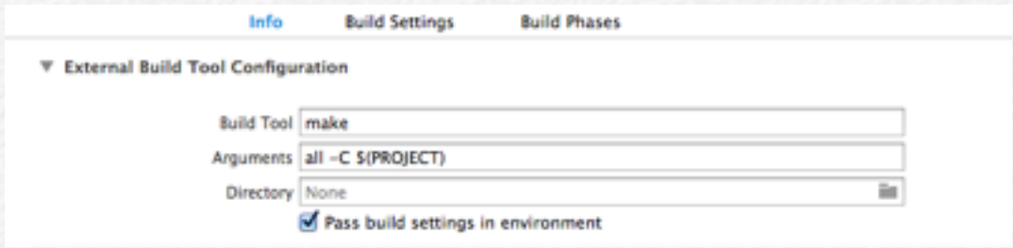
Before and after

The corresponding manual procedure is detailed at Declare Sketch .pde or .ino File as C++ File.

- The directories for the targets are set to allow the click-to-error.

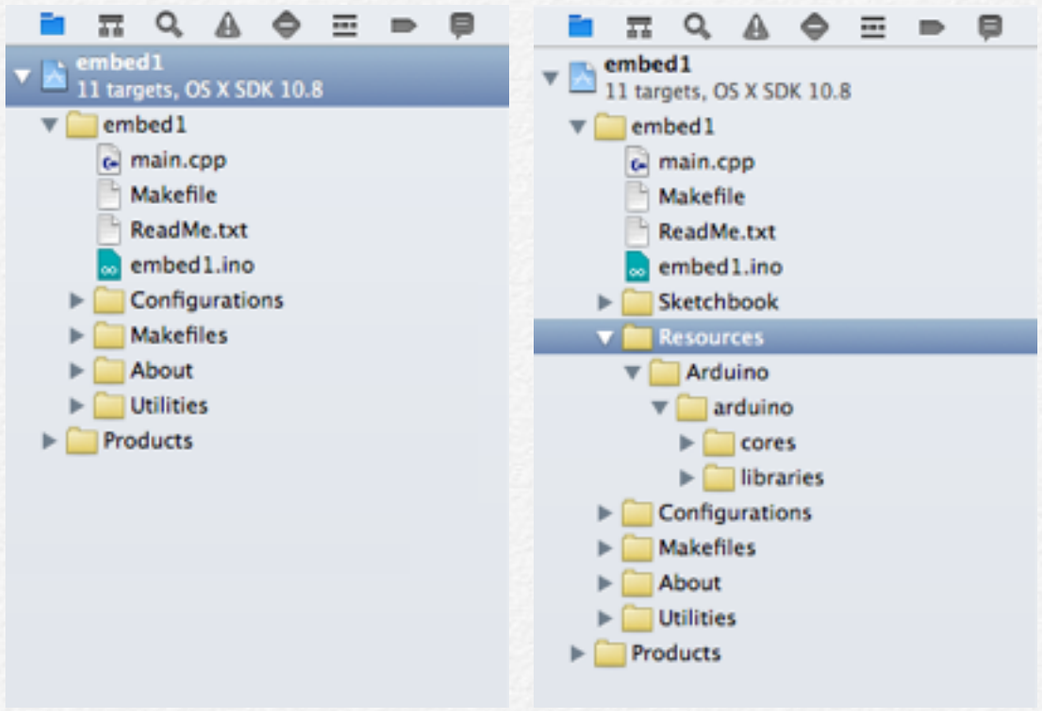


Before and after



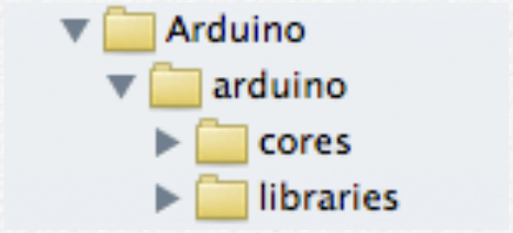
The corresponding manual procedure is detailed at Define the Directories for the Targets.

- A new **Resources** group is created and includes all the cores, variants and libraries of the installed platforms.

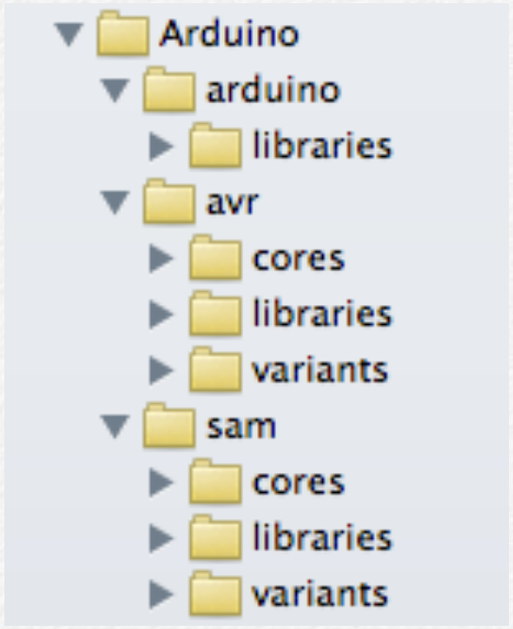


Before and after

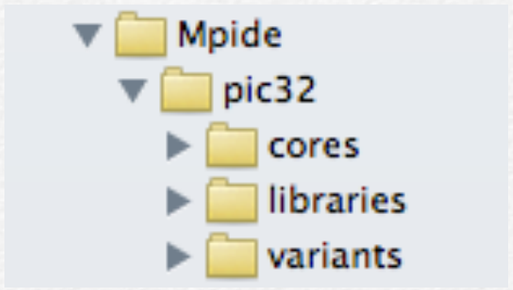
Here's the detail for each platform:



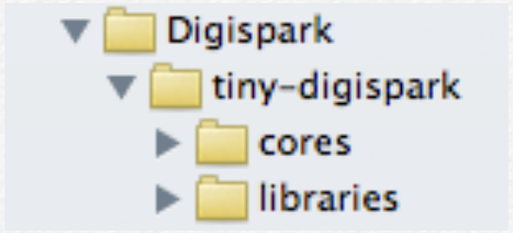
Arduino 1.0



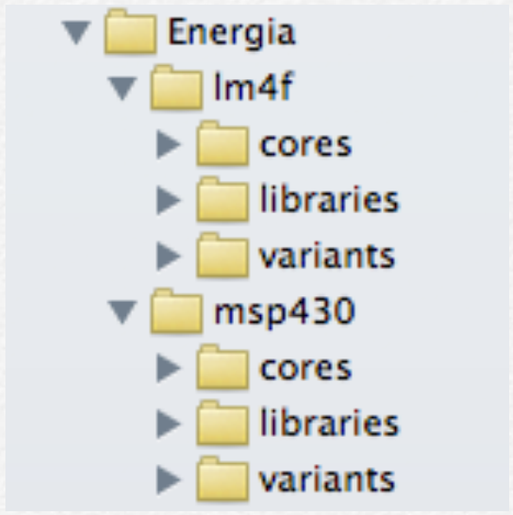
Arduino 1.5



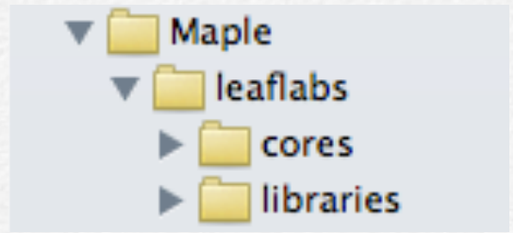
chipKIT MPIDE



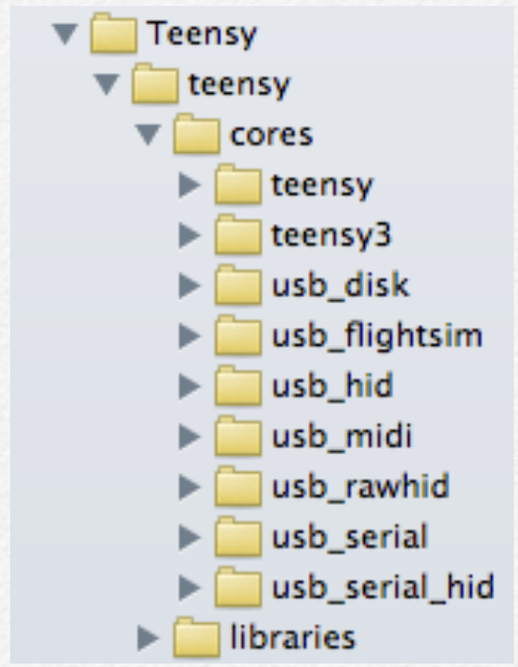
Digispark



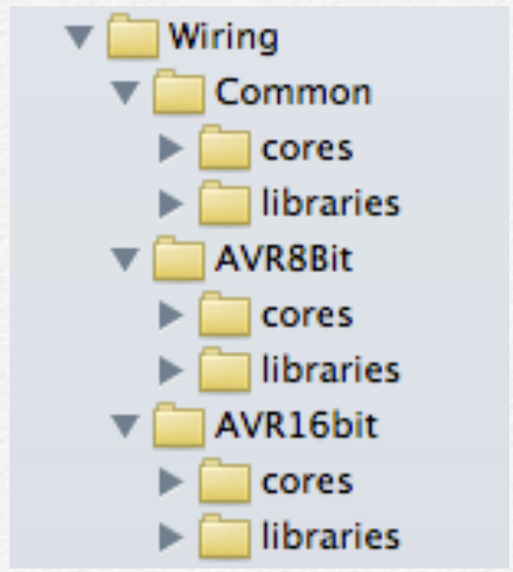
Energia



Maple

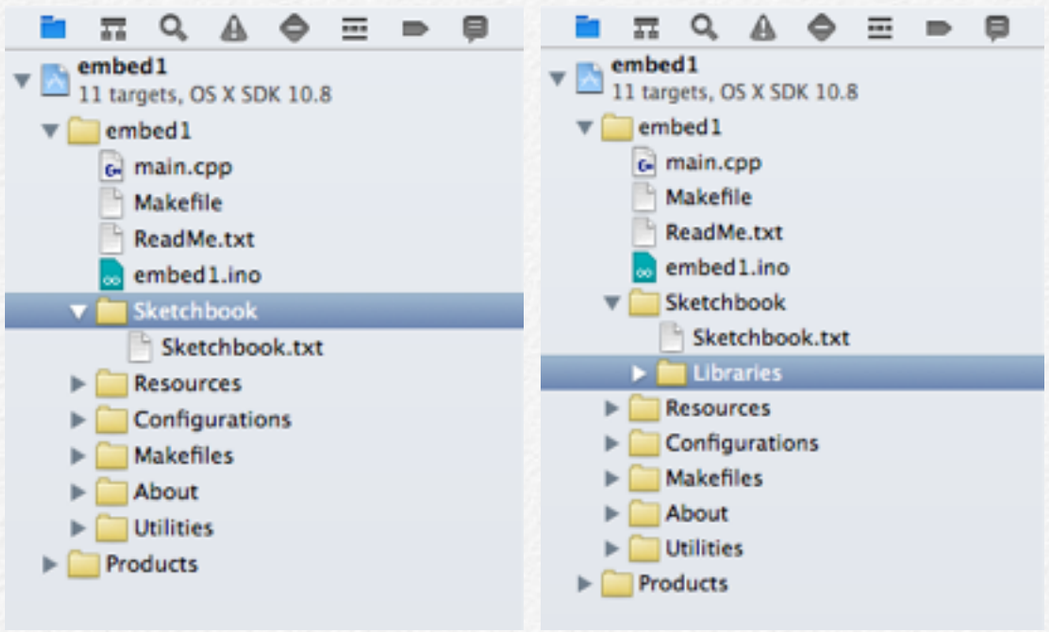


Teensy



Wiring

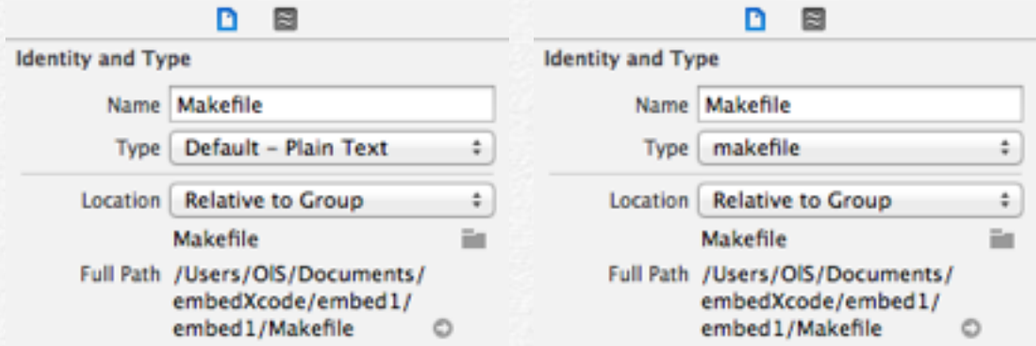
All the libraries for the selected platform are under the **Sketchbook** group.



Before and after

The corresponding manual procedure is detailed at Add User's Libraries.

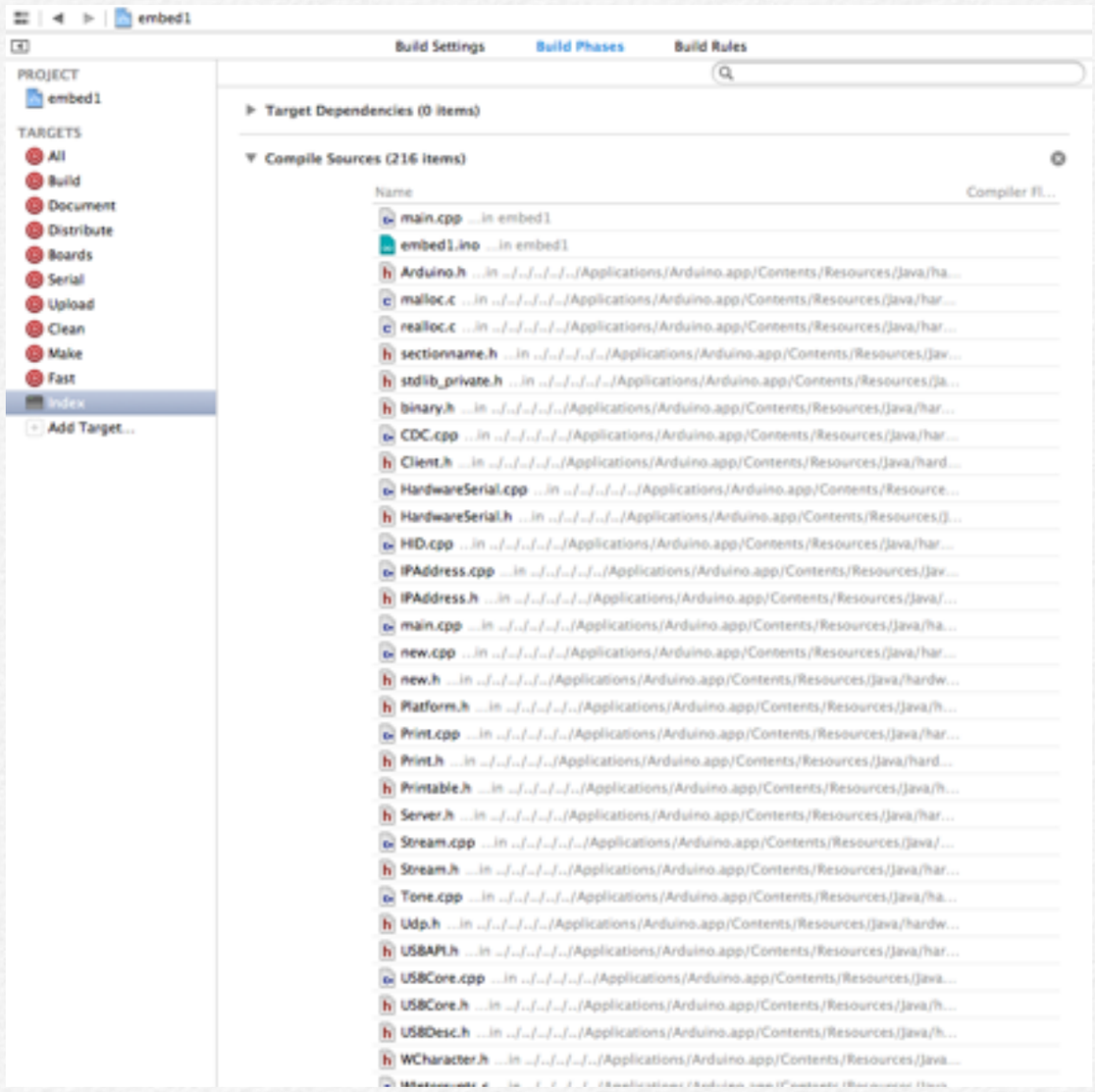
- Additionally, the makefiles are updated with the type `makefile`.



Before and after

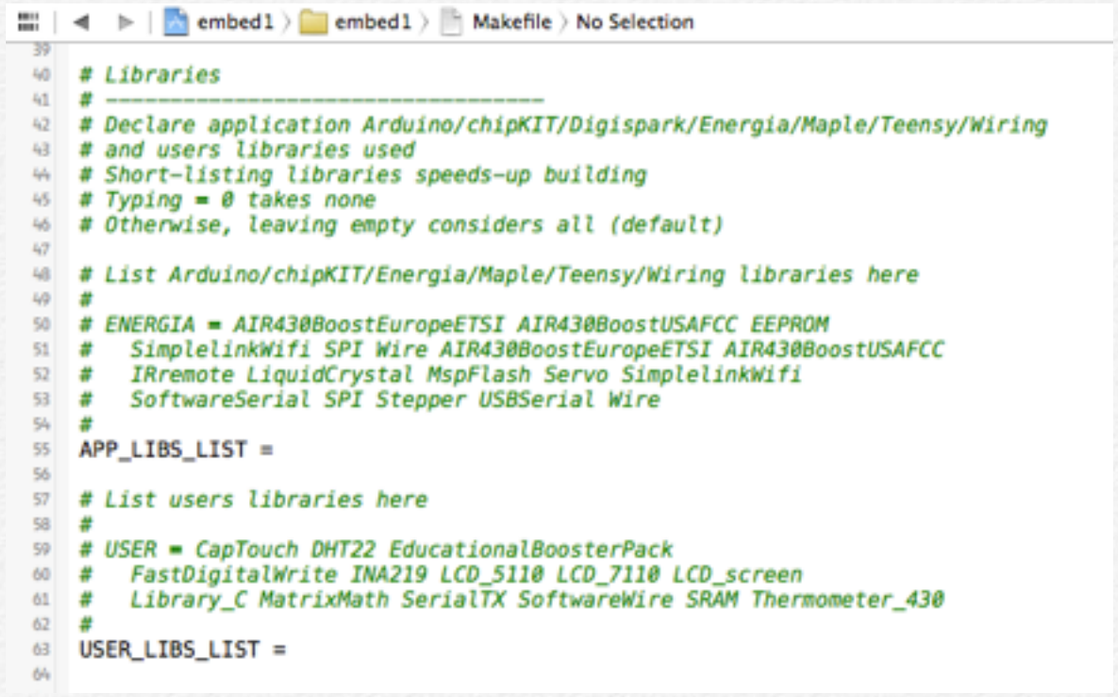
The corresponding manual procedure is the same as Declare Sketch .pde or .ino File as C++ File.

- All the sources for code-sense are declared.



The corresponding manual procedure is detailed at Declare Sources for Code-Sense.

- ⊞ Finally, all the libraries are listed in the main `makefile` for an easier management.

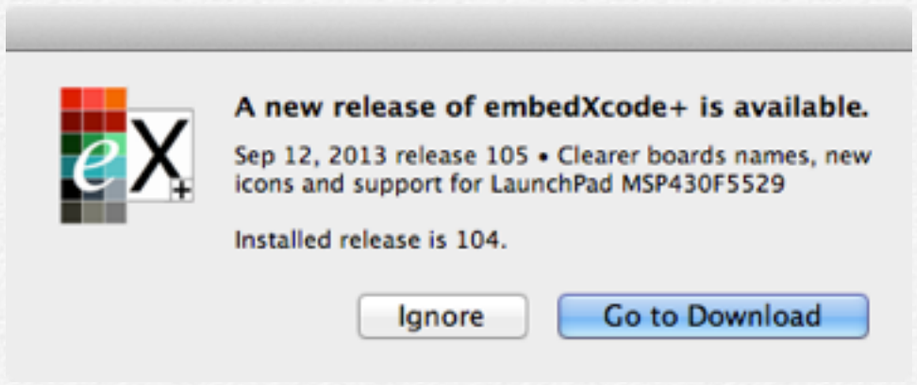


You're ready now!

New Release Message

During the first compilation of the project, a dialog box may appear if a new release of embedXcode is available.

The dialog box closes automatically after 5 seconds.



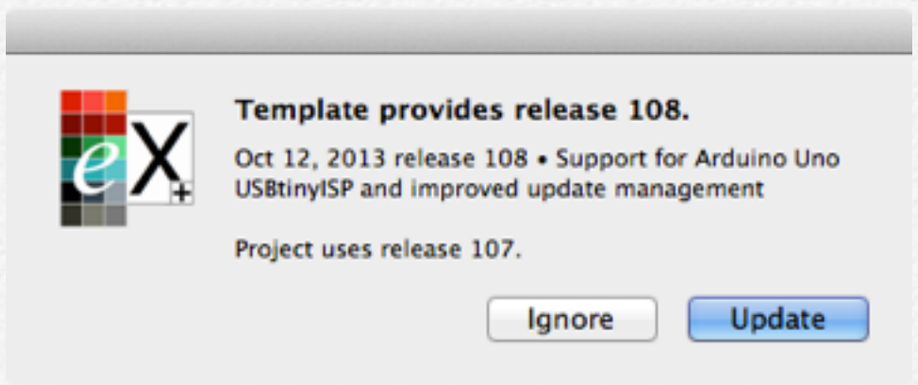
- Click on **Go to Download** to download the new release or **Ignore** to ignore it.

The new release of the template only applies for new projects. Existing projects are not updated with the new template.

Automatic Update Message

⊞ This section requires embedXcode+.

During the compilation of the project, a dialog box may prompt if the release of the template is more recent than the project.



- Click on **Update** to update the project or **Ignore** to ignore it.

The dialog box closes automatically after 5 seconds.

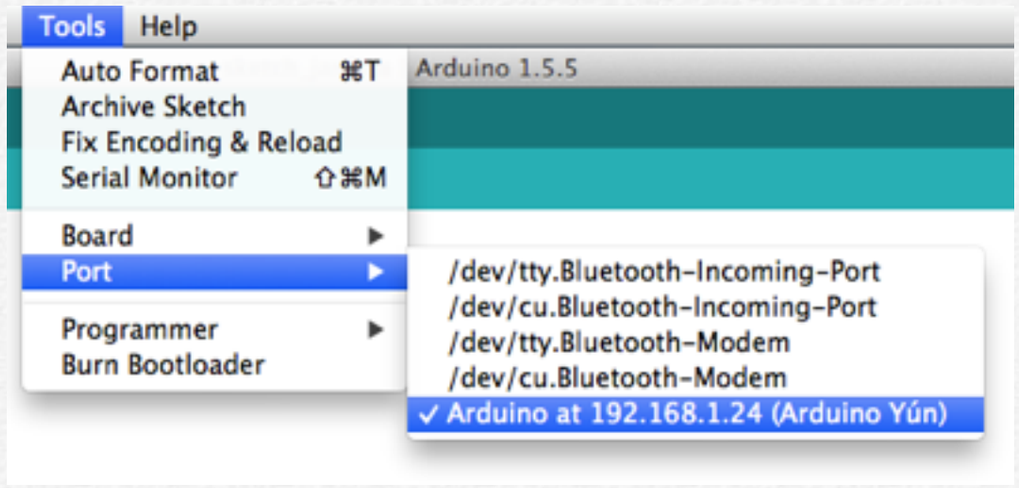
The automatic update requires the project to have been created with embedXcode release 108 or more recent. Projects created with an earlier release don’t feature the automatic update.

Yún Password Message

⊞ This section requires embedXcode+.

The Arduino Yún board allows to use an over-the-air connection through Ethernet or WiFi for the upload and console. The connection is protected by a password defined during the installation of the board.

In the Arduino IDE, the Arduino Yún board appears with its IP address on the **Tools > Port** menu.



For more information about the Arduino Yún installation and over-the-air upload, please refer to the Guide to the Arduino Yún ⓦ on the Arduino website.

On embedXcode+, just select **Arduino Yun (WiFi Ethernet)** among the other boards.

A message box asks for the password:

- Enter the password of the Arduino Yún board.
- Click on **OK** to validate or **Cancel** to cancel.

When validated, the password is saved on the board configuration file **Arduino Yun (WiFi Ethernet)** in the project.

```
48  // Yun password
49  //
50  YUN_PASSWORD = password
51
```

Please note the password is not encrypted. The password is only asked once. To erase the password, just delete the corresponding line. Similarly, you can add or edit the password after the `YUN_PASSWORD` variable.

Manual Procedures

The following procedures are provided in case important changes have been done to the project, as the automatic procedure can only be launched once.

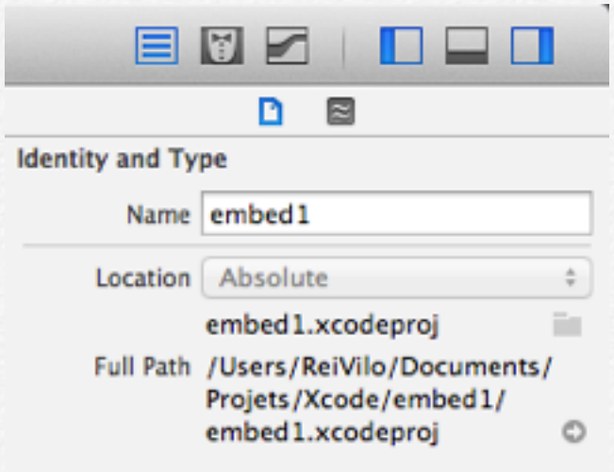
Declare Sketch .pde or .ino File as C++ File

This manual procedure is no longer required as it is included in the automatic procedure.

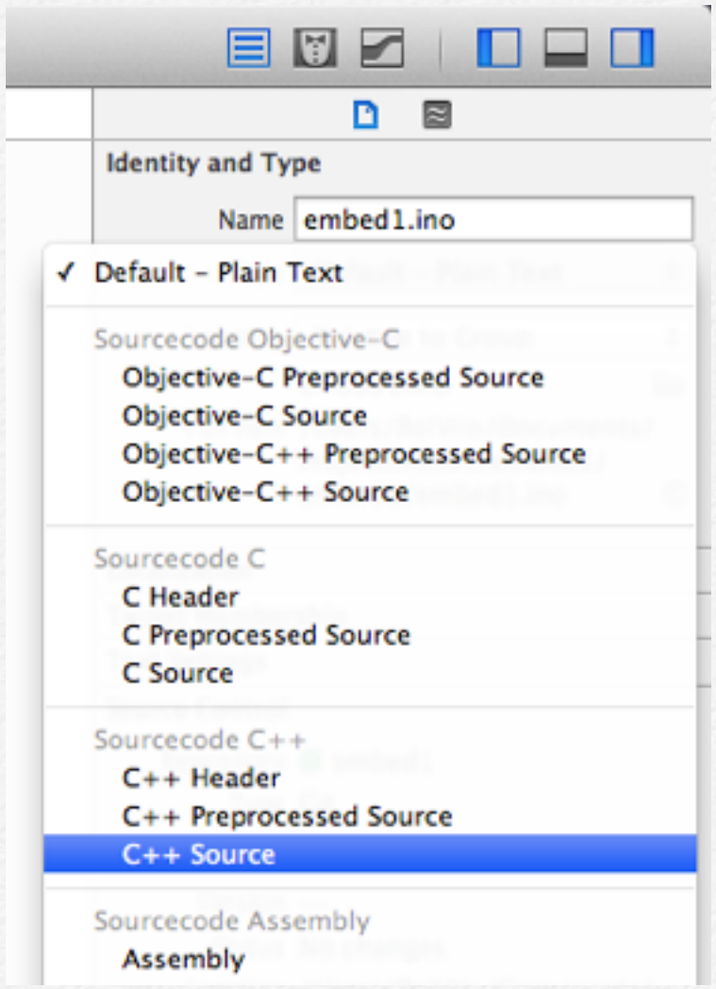
The sketch `.pde` or `.ino` file is considered as plain text. For code-sense, it should be declared as C++ file.

- Select the sketch `.pde` or `.ino` file.

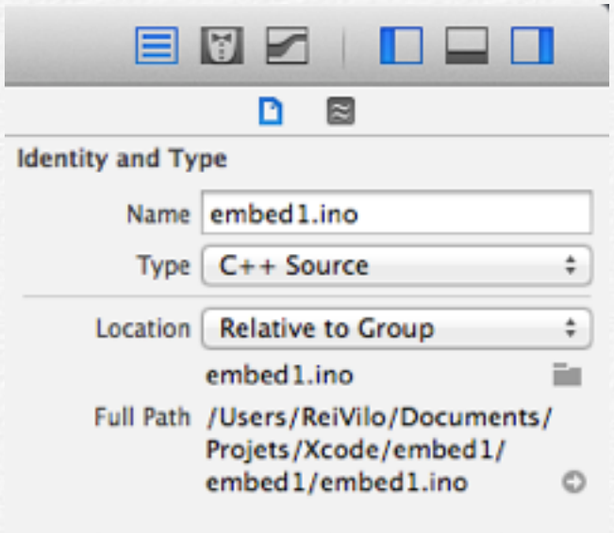
In the rightmost column,



- Click on the drop-down list of **File Type**.



- Select **C++ source**.



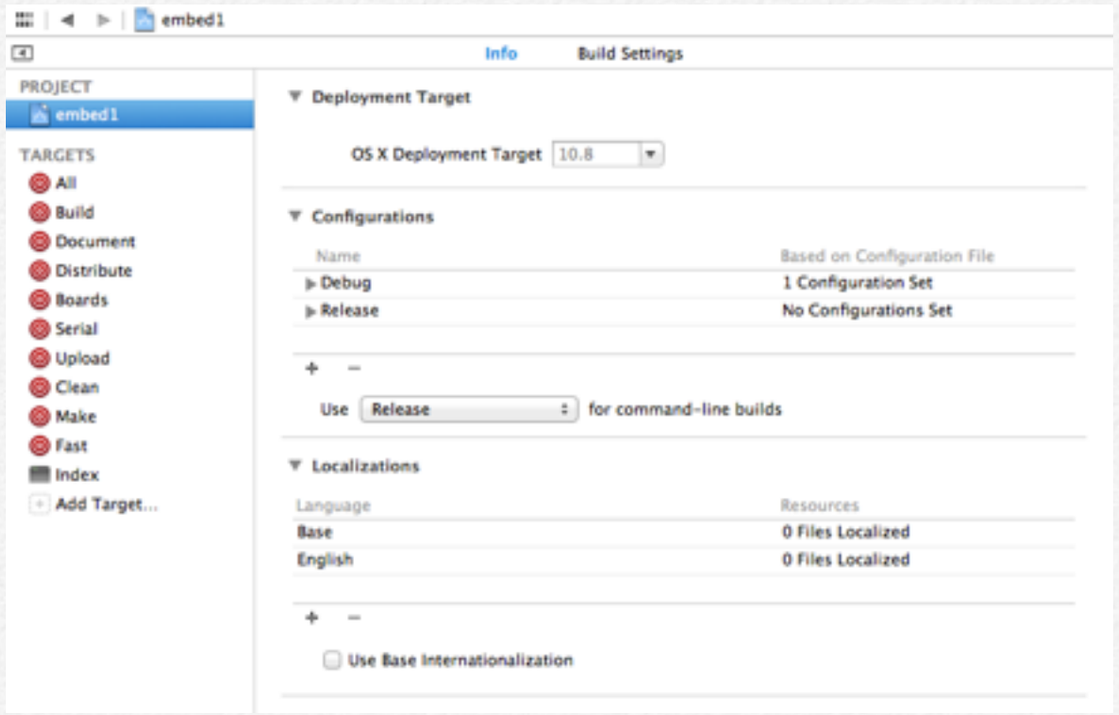
Now, the sketch is considered as C++ code for code-sense.

Declare User's Sketchbook

This is an optional procedure as, by default, embedXcode reads the path automatically from the configuration file of each IDE.

The user's sketchbook is a folder where the user's sketches are saved, among them the libraries in a dedicated sub-folder `Libraries`.

It is defined on the IDE of each board, Arduino, chipKIT MPIDE, Energia, Maple, Teensy or Wiring.

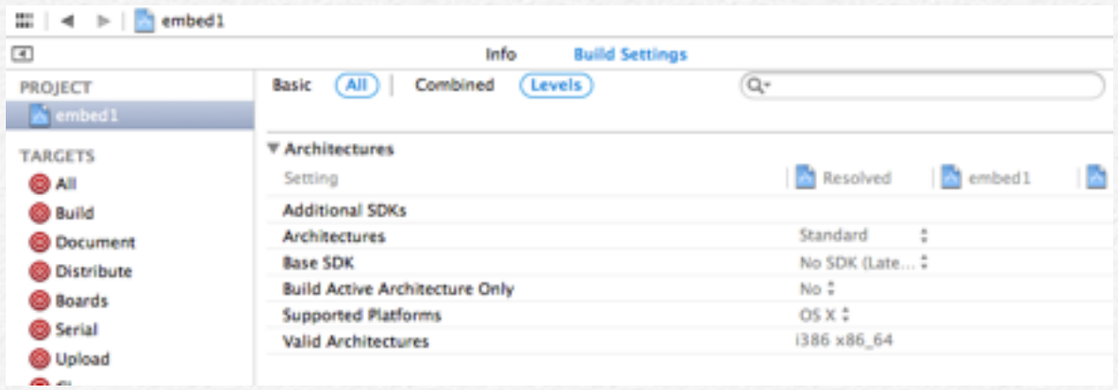


By default, embedXcode takes the value for `SKETCHBOOK_DIR` automatically from the configuration of Arduino, chipKIT MPIDE, Digispark, Energia, Maple, Teensy or Wiring.

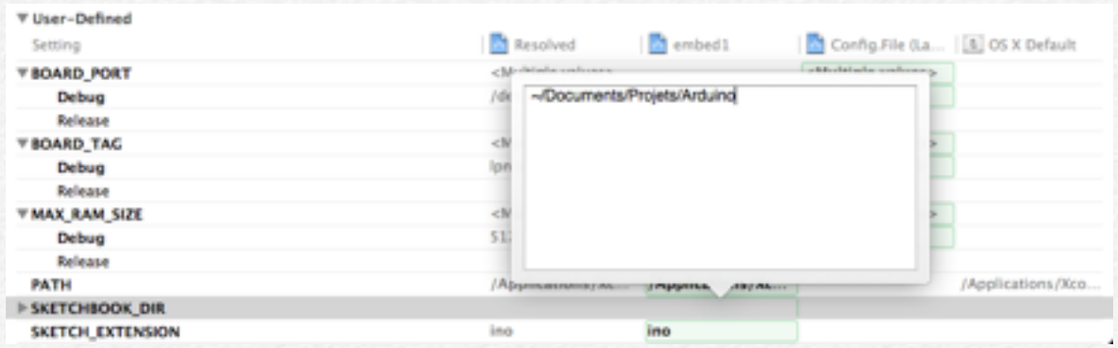
Please note that each IDE has its own sketchbook directory. So changing the board also changes the sketchbook.

To set one sketchbook directory for all boards, proceed as follow:

- Select the project on top of the leftmost list,
- Select the project under the **PROJECT** list,
- Select the **Build Settings** pane,



- At the very bottom, double-click on `SKETCHBOOK_DIR`,
- Either type in the name of the folder
- Or drag-and-drop it from a **Finder** window.



The ~ character, shortcut for the `Home` folder, is accepted.

Add User's Libraries

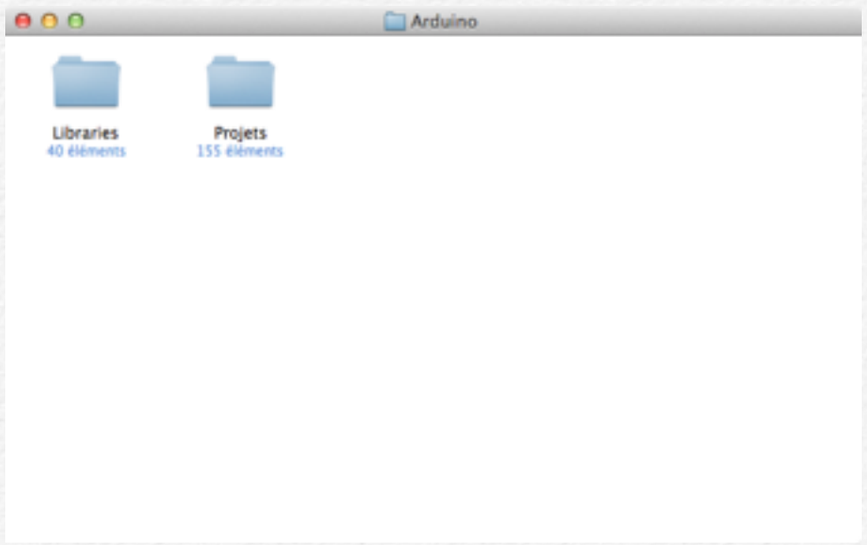
The manual procedures are no longer required as it is included in the automatic procedure.

However, if you change the platform, you may need to add manually the user's libraries for the new platform.

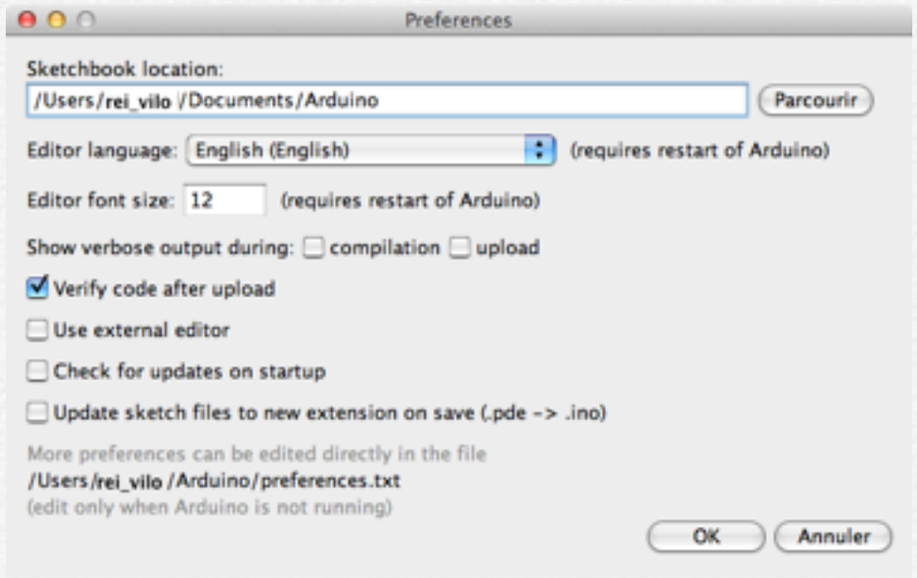
- First, make sure the sketchbook folder has been created with a `Libraries` or `libraries` subfolder.

Please refer to Installing Additional Arduino Libraries ⓦ on how to install the `Sketchbook/Libraries or Sketchbook/libraries` folder.

The name of the folder for the libraries can have a capitalised first letter `Libraries` or not `libraries`.

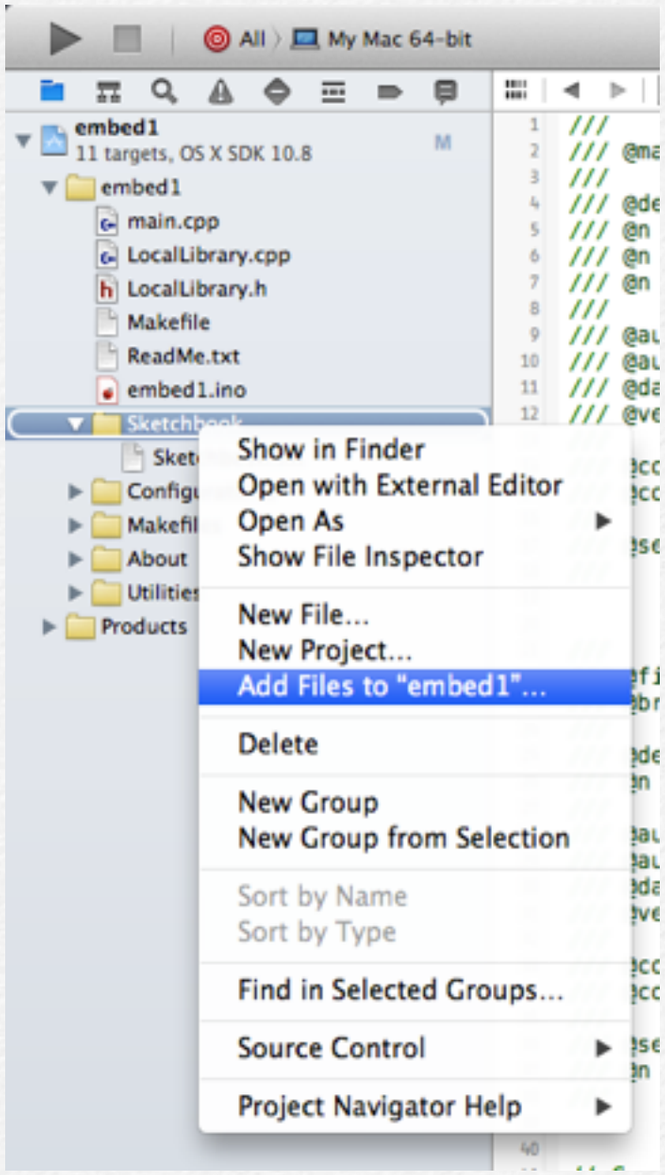


- Then, for each board, open the corresponding IDE and define the sketchbook path in the Arduino in the menu **Arduino > Preferences > Sketchbook location**.



The same procedure applies for all the other IDEs: chipKIT MPIDE, Digispark, Energia, Maple, Teensy or Wiring.

- Back on Xcode, open the **Sketchbook** group on the project hierarchy.
- Right-click to obtain the contextual menu.
- Choose **Add file to...**

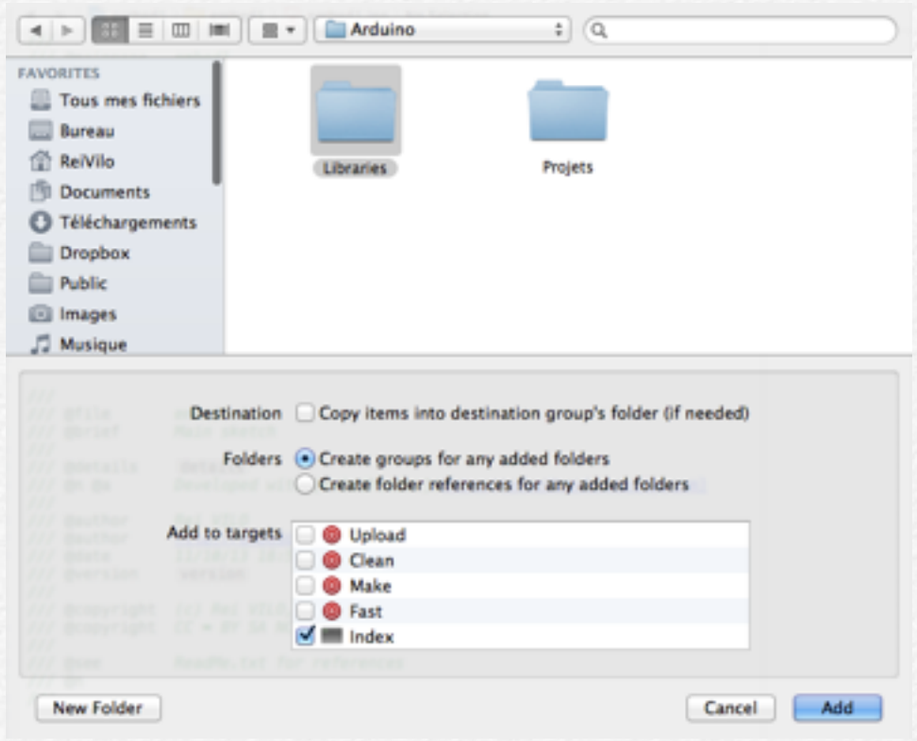


Still in Xcode, select the `Library` sub-folder on the sketchbook folder.

- Check **Index** under **Add to target**.

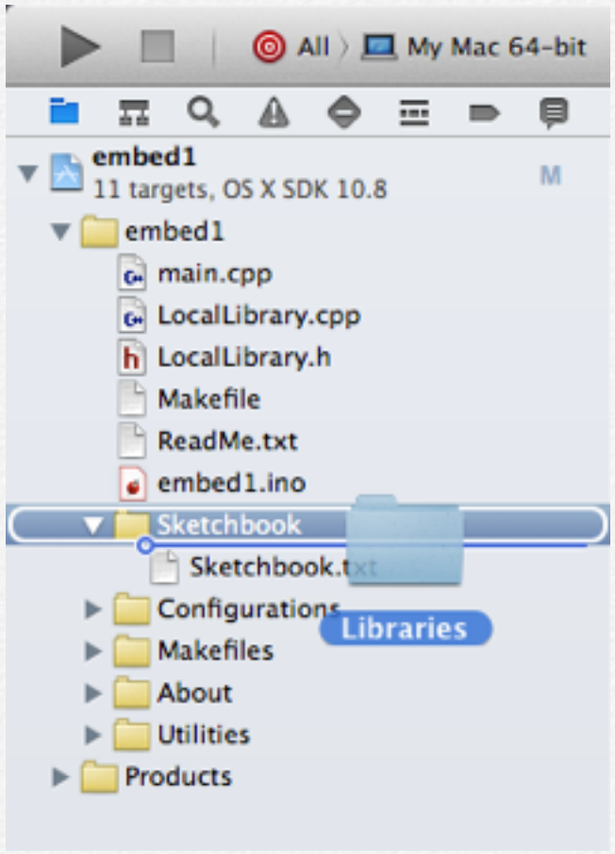
Both **Create groups for any added folders** and **Create folder references for any added folders** are relevant.

- Don't check **Copy items into destination group's folder (if needed)** to avoid duplicating files.
- Finally, validate with **Add**.



As an alternative,

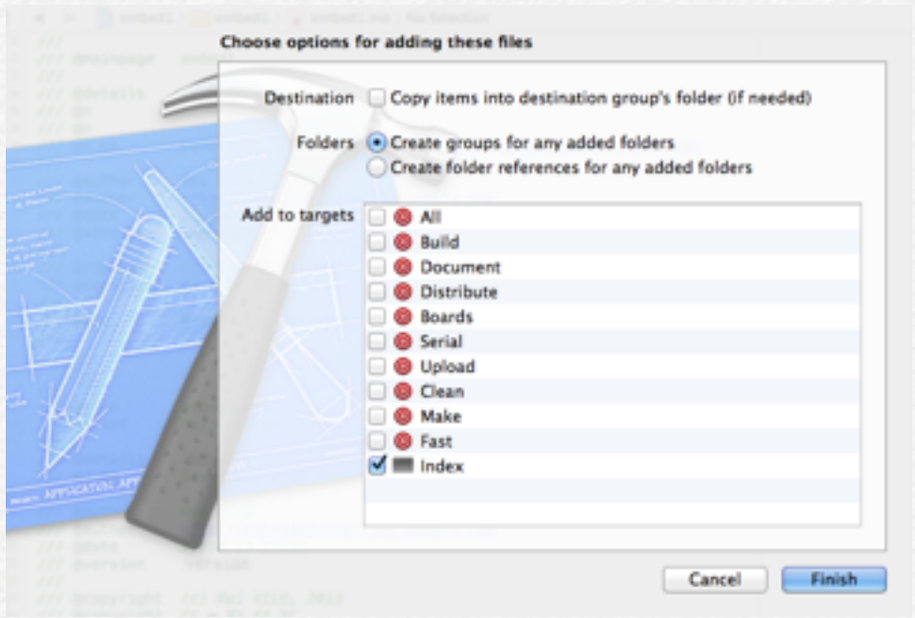
- Drag and drop the `Libraries` folder under the **Sketchbook** group.



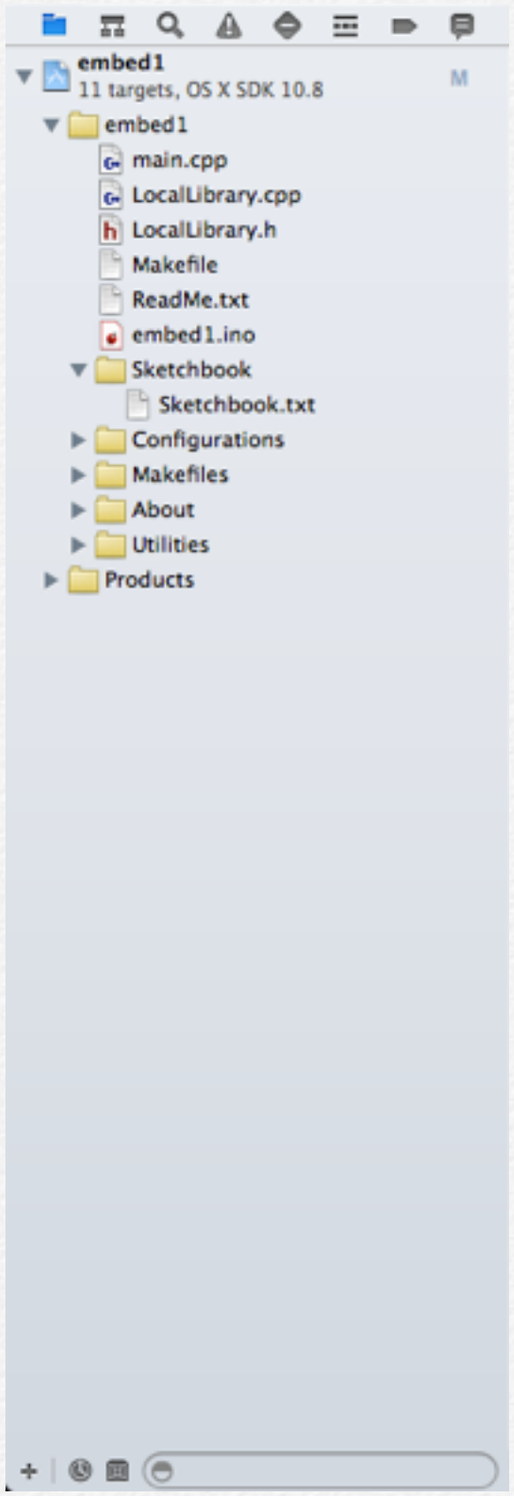
- Check **Index** under **Add to target**.

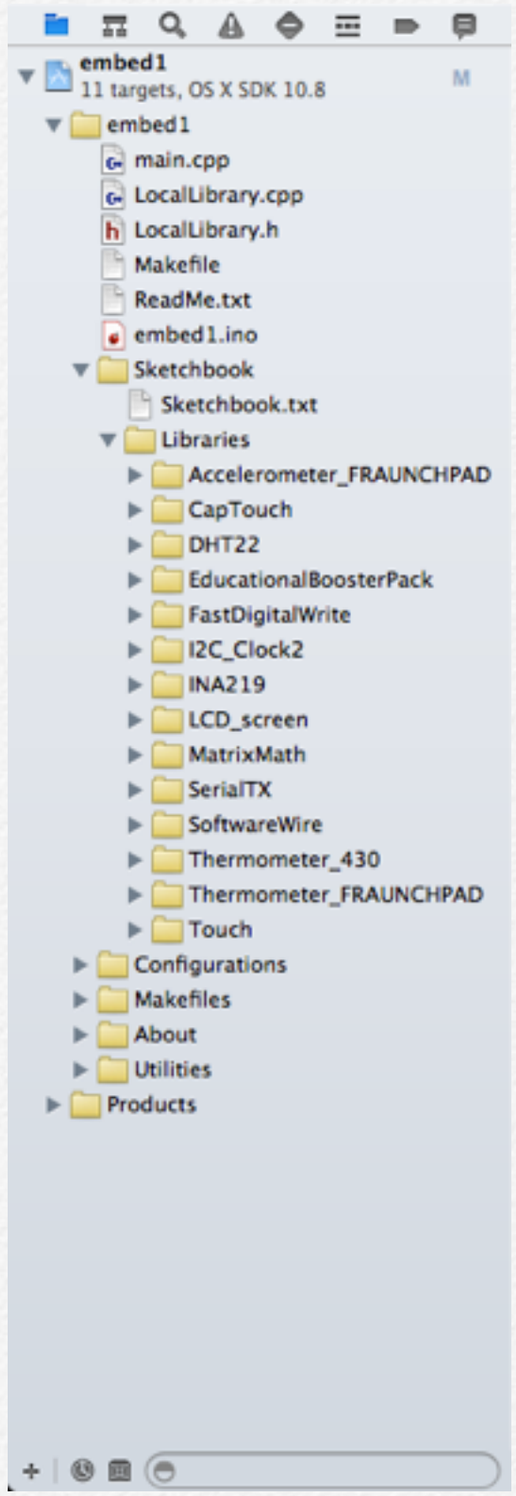
Both **Create groups for any added folders** and **Create folder references for any added folders** are relevant.

- Don't check **Copy items into destination group's folder (if needed)** to avoid duplicating files.
- Finally, validate with **Finish**.



The project hierarchy snow hows all your libraries.





Xcode adds the `.cpp` files of the libraries but omits the `.h` headers.

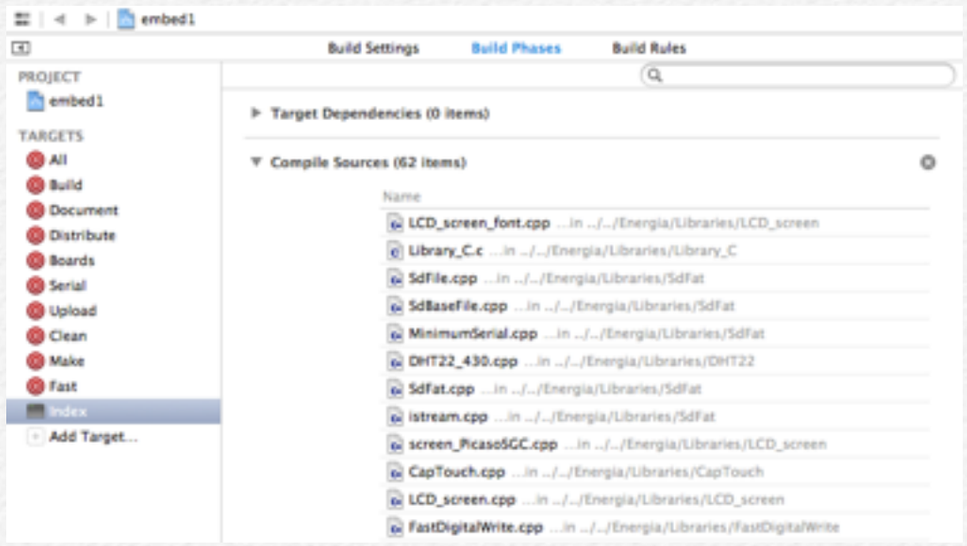
This needs to be done manually with the procedure detailed at Declare Sources for Code-Sense.

This is an optional procedure.

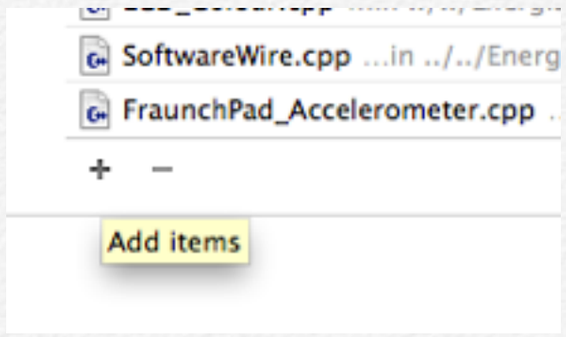
Declare Sources for Code-Sense

The selection of a board defines the headers for code-sense. So Xcode needs to be taught where to find them.

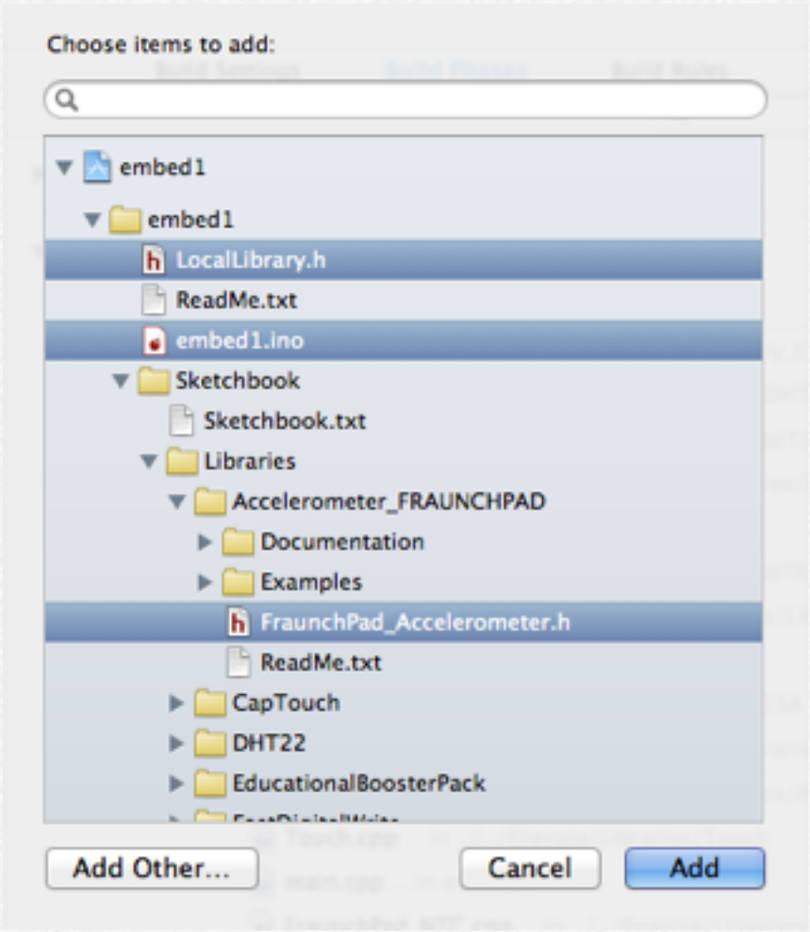
- Select the target **Index** and the **Build Phases** pane.



- Go a the bottom of the list and click on the **+** button.



A list shows up.



- Select all the `.h` and `.cpp` files and click on **Add**.

Define the Directories for the Targets

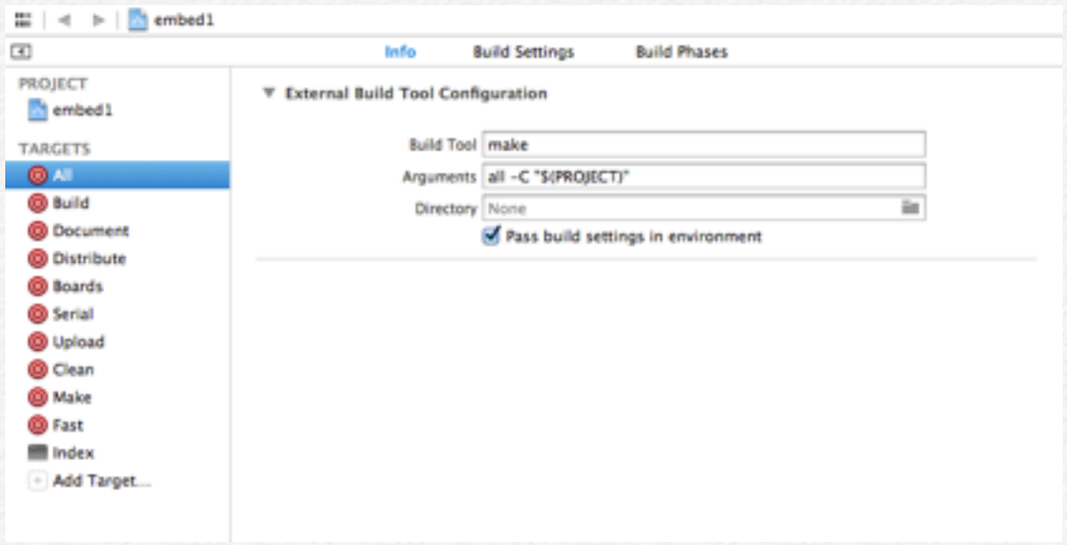
This manual procedure is no longer required as it is included in the automatic procedure.

The template doesn't allow to specify the exact directory for the targets. In case of a building error, the click-to-error feature may not work properly for the main sketch and the local libraries.

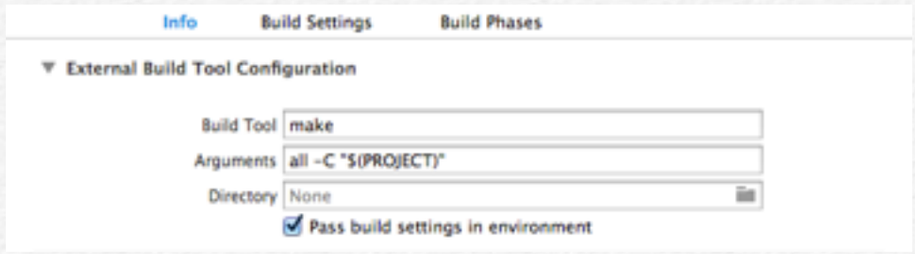
So Xcode needs to be taught where to find them.

This is an optional procedure.

- Select the target **All**.



By default, the template mentions `all -C $(PROJECT)` as arguments.



- Remove `-C $(PROJECT)` from the arguments and add `$(PROJECT)` into directory.

Feel free to update other targets you use often, as **Build** and **Make**.

Use the Project

3

Write the sketch using Xcode advanced features, change the board, add files and libraries, manage code for multiple boards and platforms.

Write the Sketch

The newly created project contains various files:

- The file `main.cpp` calls the appropriate core libraries, initialises the board, includes the sketch. The `main()` function calls the `setup()` and `loop()` functions from the sketch. Do not alter this file.
- The `embed1.pde` or `embed1.ino` file is where you write the sketch, with the `setup()` and `loop()` functions and all the additional ones.
- The files `LocalLibrary.h` and `LocalLibrary.cpp` are the header and the code for the `LocalLibrary` library. They are provided as example.
- The file `makefile` is the entry for the compilation processes. Do not alter this file.

The files you can play with are the `embed1.pde` or `embed1.ino` files for the sketch, and the `LocalLibrary.h` header and `LocalLibrary.cpp` code for the `LocalLibrary` library.

Include Libraries to the Project

Just like any other IDE, including a library to the project is done by specifying an `#include` statement on the main sketch.

Here, for example, the `Wire` library is added:

```
64  // Include application, user and local libraries
65  #include "Wire.h"
66
```

Additionally, the library needs to be mentioned on the main makefile to be compiled.

There are two kinds of libraries with their corresponding variables:

- The standard libraries are provided by the Arduino / chipKIT / Digispark / Energia / Maple / Microduino / Teensy / Wiring IDEs, with the variable `APP_LIBS_LIST`;
- The other libraries are developed by the user and stored under the `Library` sub-folder on the sketchbook folder, with the variable `USER_LIBS_LIST`.

In the following example, as the `Wire` library is a standard library, the `Wire` folder is mentioned on the `APP_LIBS_LIST` variable for the `Wire` library.

```
# List Arduino/chipKIT/Digispark/Energia/Maple/
Microduino/Teensy/Wiring libraries here
#
APP_LIBS_LIST = Wire
# List users libraries here
#
USER_LIBS_LIST = 0
```

⊞ With embedXcode+, the makefile comes with the list of the libraries. This list is added during the preparation and for the selected platform of the project:

```
# List of Arduino/chipKIT/Digispark/Energia/
Maple/Microduino/Teensy/Wiring libraries
#
# ARDUINO = Bridge EEPROM Esplora Ethernet
#   Firmata GSM LiquidCrystal SD Servo
#   SoftwareSerial SPI Stepper WiFi Wire
#
APP_LIBS_LIST = Wire
# List users libraries here
#
USER_LIBS_LIST = 0
```

For more information about how the libraries are managed, please refer to the section Manage the Libraries for Compilation.

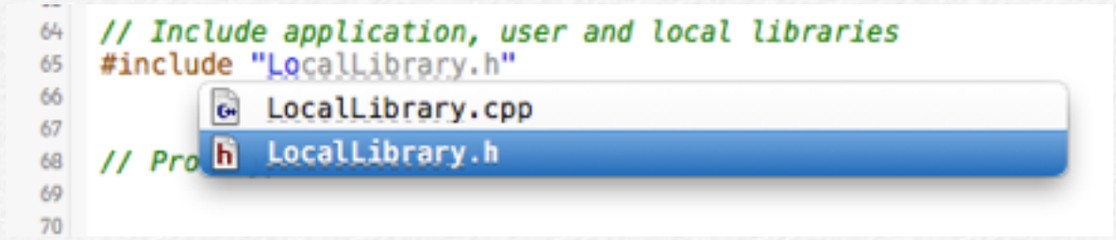
Use Code-Sense

Code-sense is a major feature of Xcode.

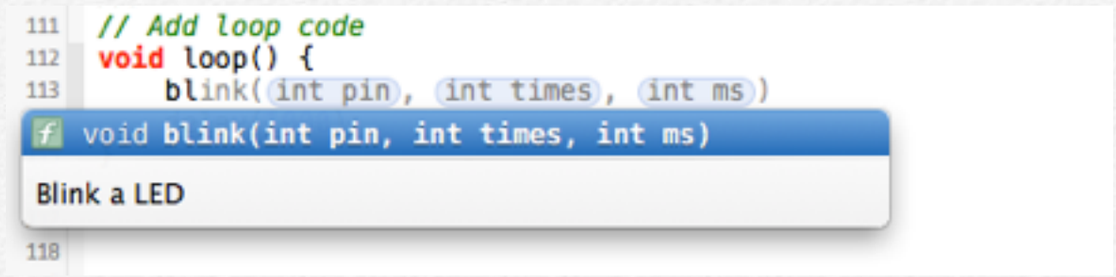
Apart from pretty colours on the code and enhanced visibility, code-sense brings many features that speed up coding.

- Auto-completion

Type the first letters and Xcode completes. Select one item of the list and press **tab**.



- Code-snippets and check-as-you-type code monitoring

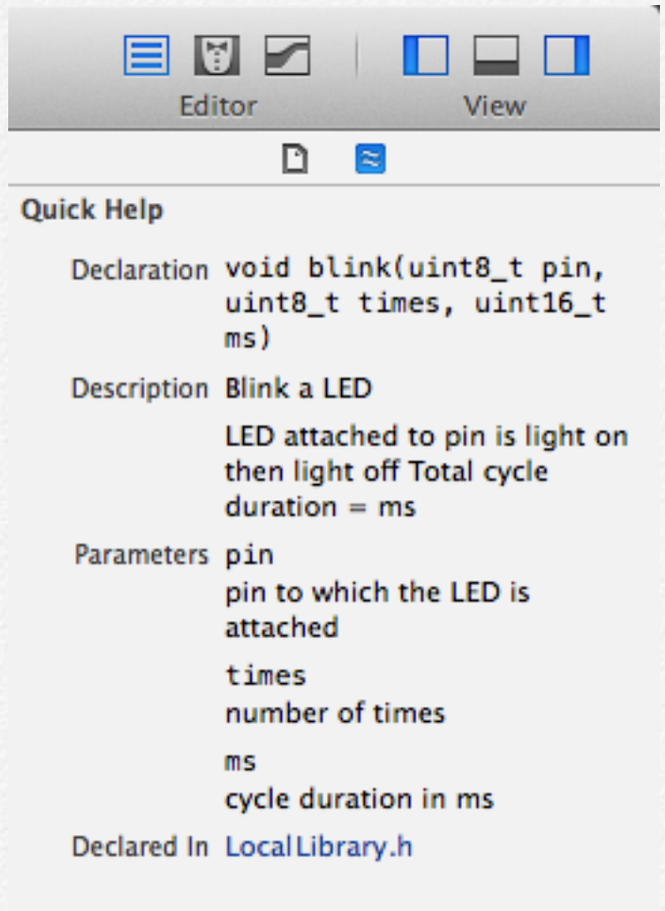


In this example, the function name is completed with all the parameters.

Fill-in each parameter and press **tab** to go to the next one.

- Quick Help

Based on the function under the cursor, the **Utilities panel** displays the available documentation for the function:



- Click-to-definition

Press **cmd-click** to access a function definition.

```
111  // Add loop code
112  void loop() {
113      blink(myLED, 3, 333);
114      delay(1000);
115  }
116
```

Select the function and press **cmd-click** to access directly its definition. Notice the hand.

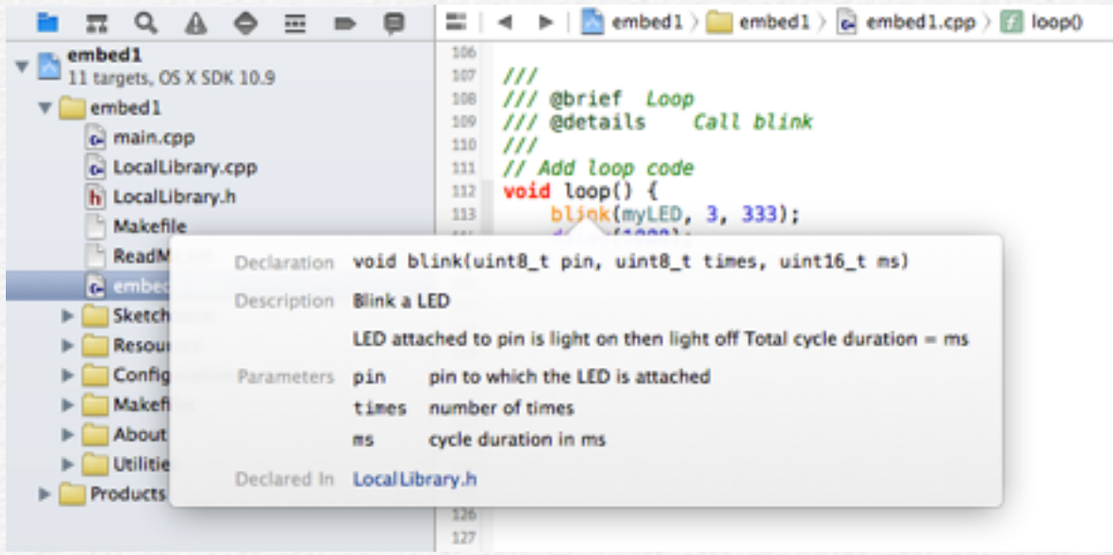
```
20
21  #include "LocalLibrary.h"
22
23  void blink(uint8_t pin, uint8_t times, uint16_t ms) {
24    for (uint8_t i=0; i<times; i++) {
25      digitalWrite(pin, HIGH);
26      delay(ms >> 1);
27      digitalWrite(pin, LOW);
28      delay(ms >> 1);
29    }
30  }
```

- Contextual help

Press **alt-click** to display the definition.

```
111  // Add loop code
112  void loop() {
113      blink(myLED, 3, 333);
114      delay(1000);
115  }
116
```

Select the function and press **alt-click** to display its definition. Notice the question mark.

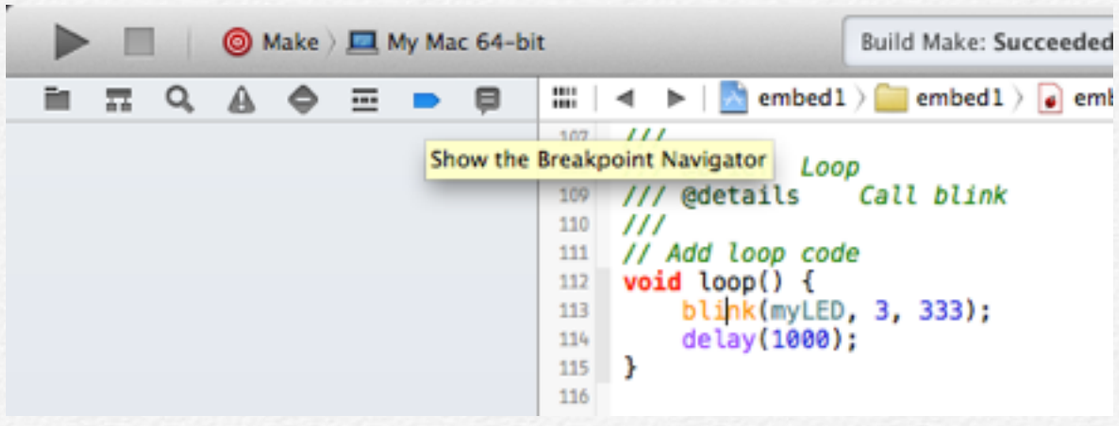


Sometimes, the index of keywords needs to be re-indexed. Please refer to Re-Index the Keywords for Code-Sense.

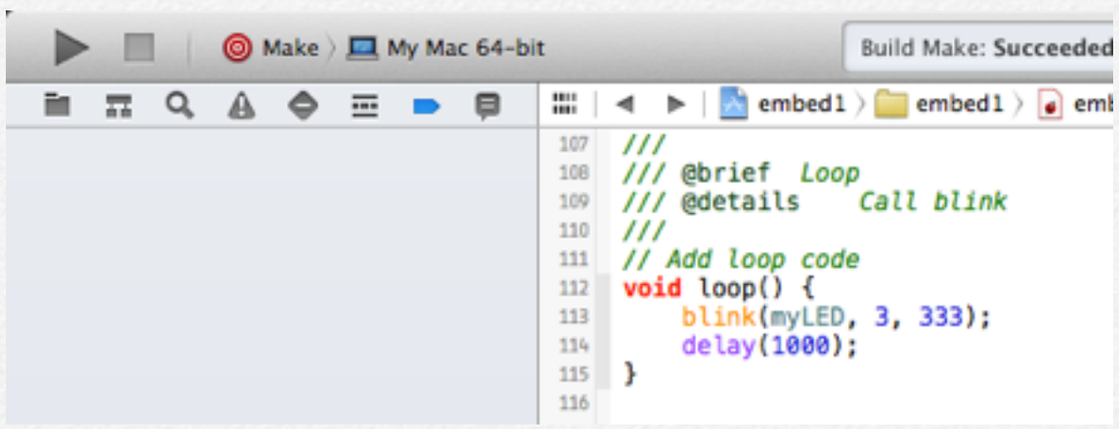
Use Bookmarks

We are using the breakpoints as bookmarks.

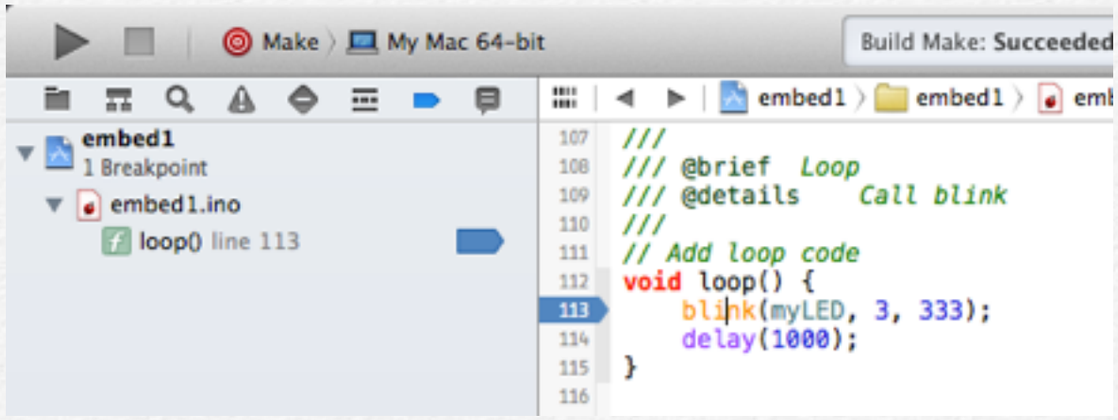
- To display the bookmarks, click on the **Breakpoint Navigator** button:



- To add a bookmark, select the line:

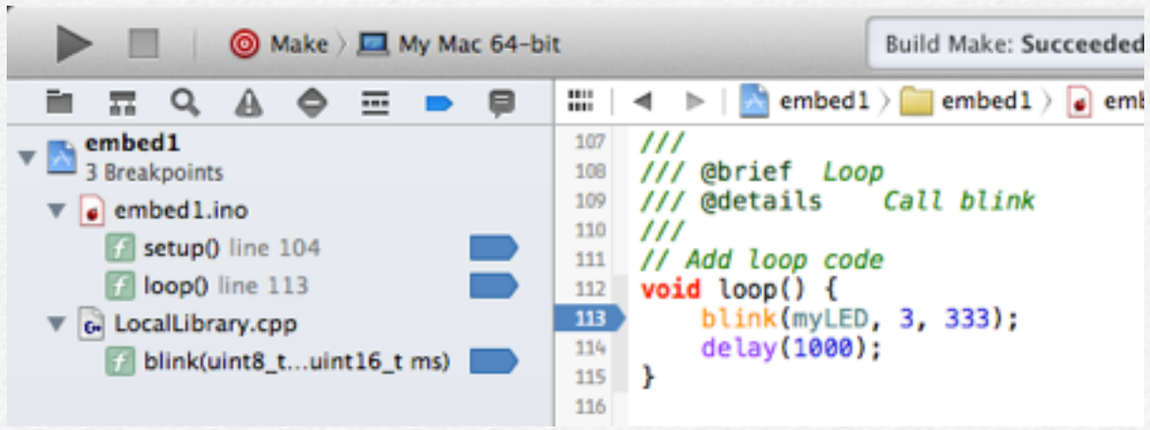


- Click on the line number, here **113** of function **loop()**. The bookmark is added.

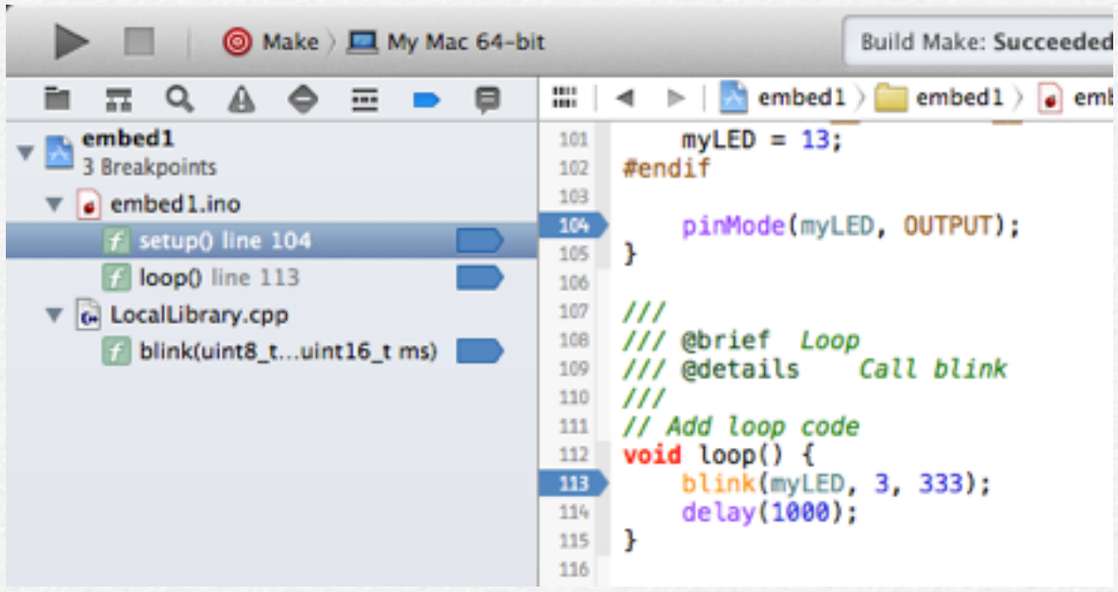


Multiple bookmarks can be added.

They are grouped by file within the project.



- To reach a position, just click on the bookmark you want to go, here line **104** of **setup()**.

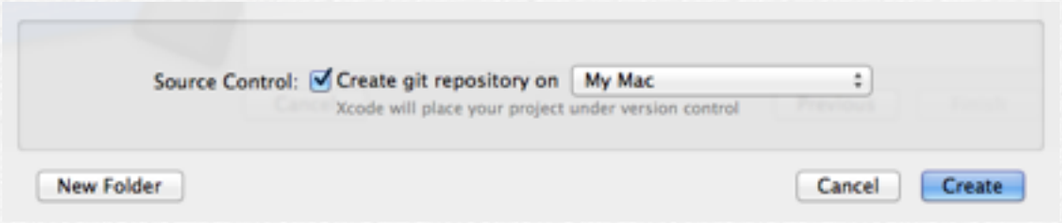


Please note bookmarks are actually breakpoints. None of the Processing-based Wiring-derived Arduino-like IDEs has implemented debugging yet.

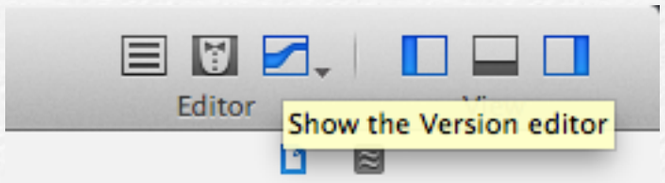
Use Other Xcode Advanced Features

Xcode is packed with many advanced features.

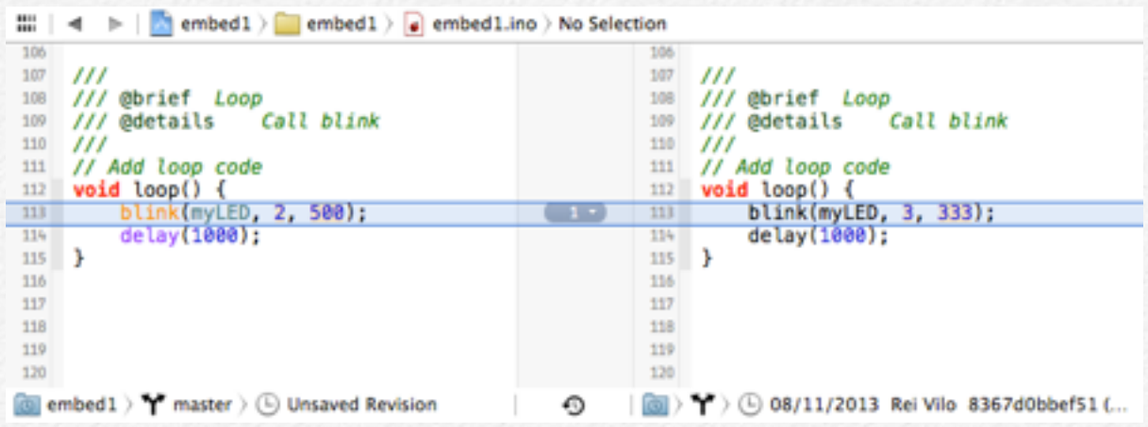
For example, Xcode manages versions if source control has been checked when creating the new project.



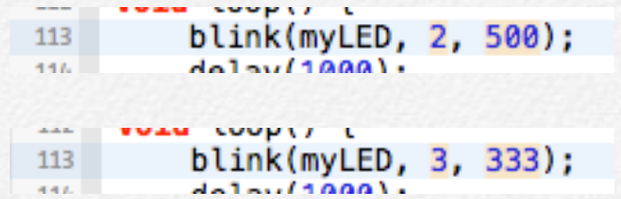
To display the version editor, click on the button:



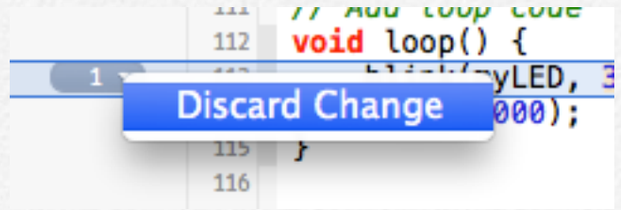
The actual and previous versions are displayed alongside:



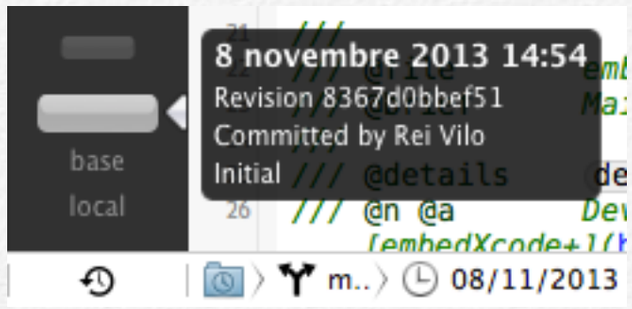
In this example, line 103 has been changed:



The button allows to discard changes:



The versions can be selected among the previously saved versions.



For more informations on this feature and on others, please refer to the Xcode help.

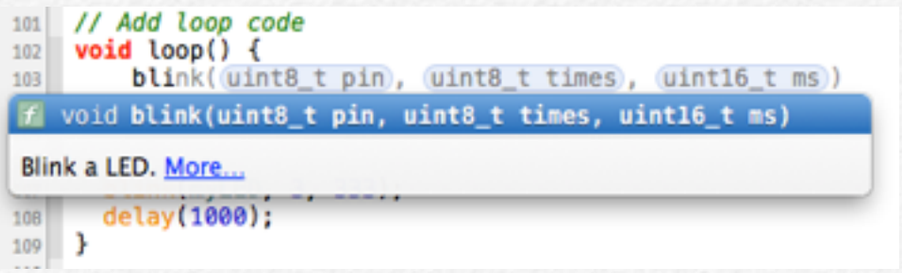
Re-Index the Keywords for Code-Sense

Code-sense is a major feature of Xcode. Sometimes, keywords need to be re-indexed.

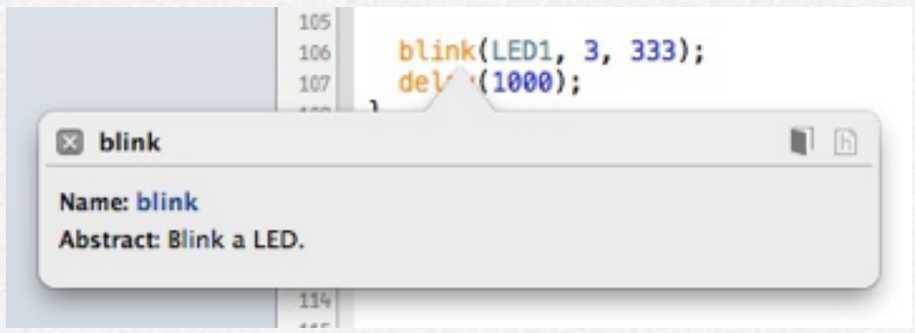
Apart from pretty colours on the code and enhanced visibility, code-sense brings:

- auto-completion,

- code-snippets and check-as-you-type code monitoring,



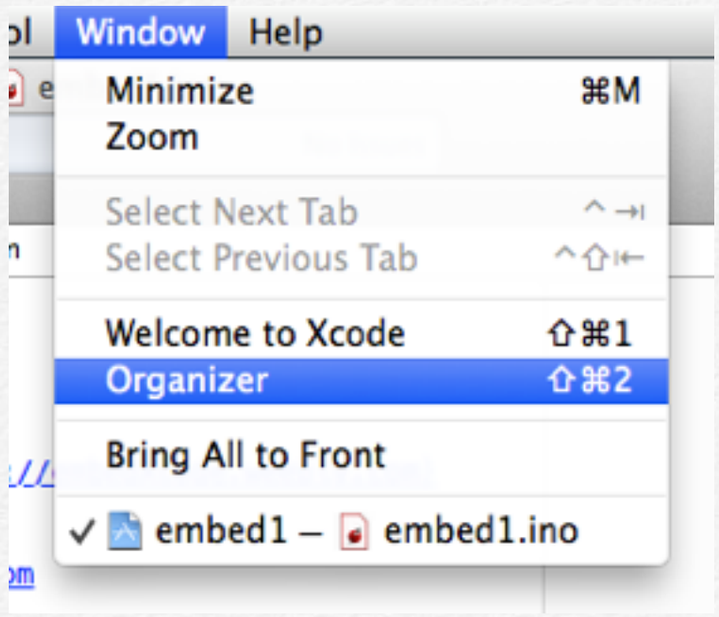
- click-to-definition



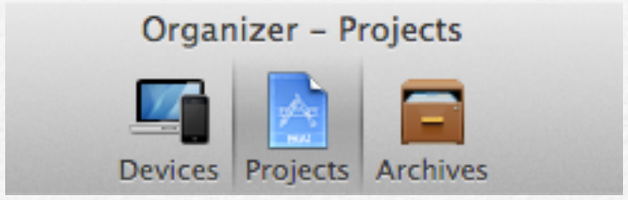
If code-sense doesn't work, we need to force a re-indexing of the key words.

To do so,

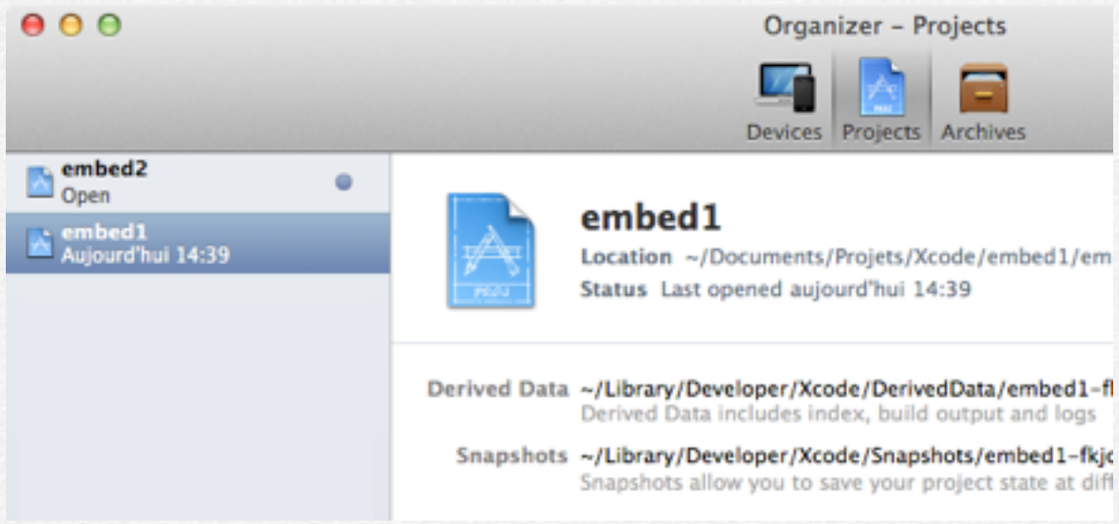
- First close the project.
- Call the menu **Window > Organiser**.



- The Organiser window opens. Select the **Projects** pane.



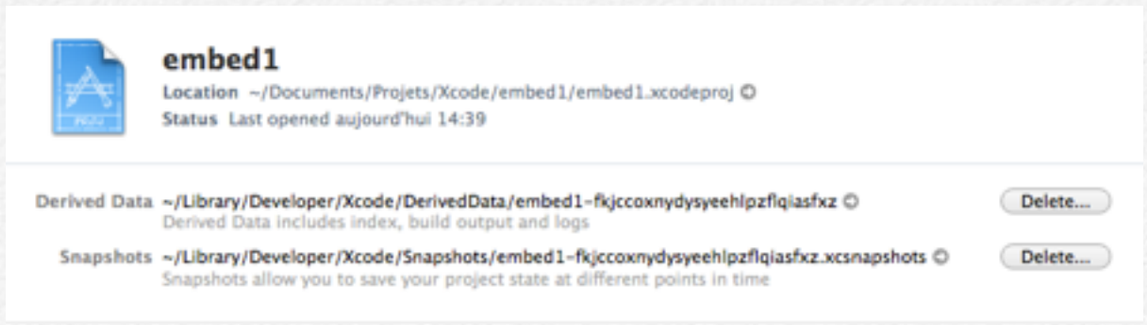
- Select then the project.



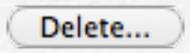
The blue dot close to **embed2** means this project is open.

If a blue dot appears close to **embed1**, close the project first.

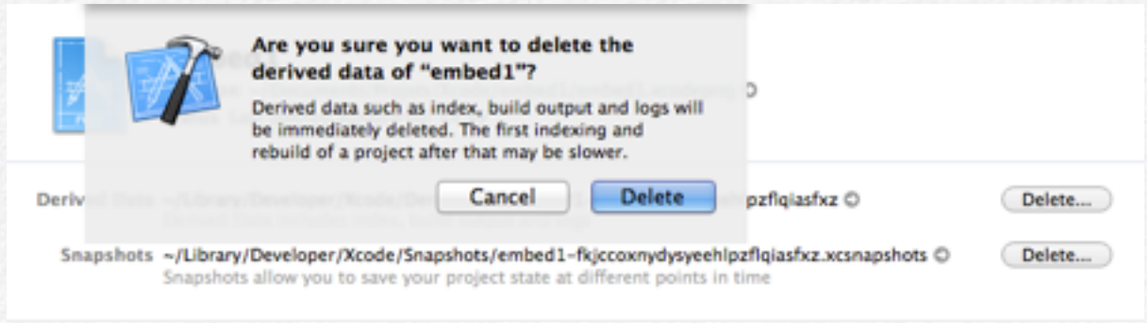
The index is saved within the `Derived Data` folder.



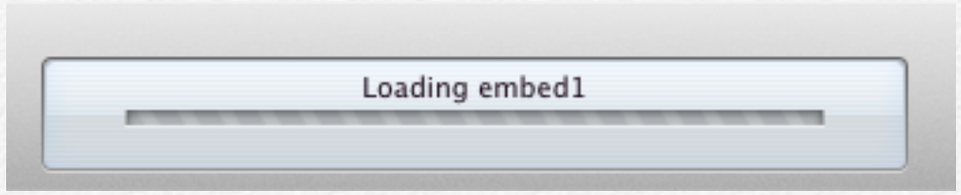
- Click on the **Delete** button to delete the index.



- Confirm the deletion with **Delete**.



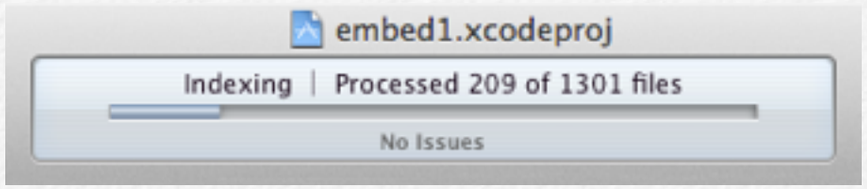
- Load the project.



There's no code-sense yet: everything is in black-and-white, except standard C++ keywords in red, pre-processing statements in brown, comments in green and numbers in blue.

```
101 ///
102 /// @brief  Loop
103 /// @details    Call blink
104 ///
105 // Add loop code
106 void loop() {
107   blink(myLED, 3, 333);
108   delay(1000);
109 }
```

The index is being built again.



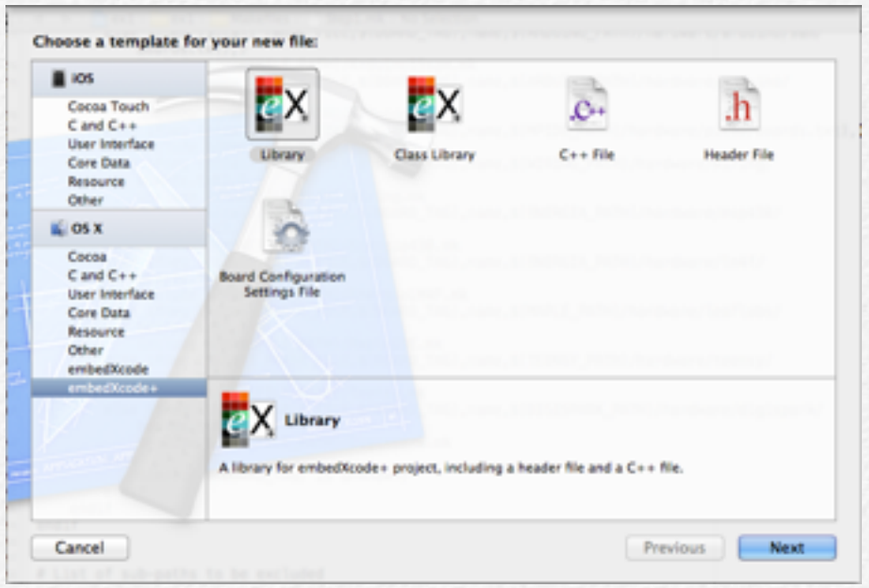
When the index is built, code-sense shows pretty colours.

```
107  ///
108  /// @brief  Loop
109  /// @details    Call blink
110  ///
111  // Add loop code
112  void loop() {
113      blink(myLED, 2, 500);
114      delay(1000);
115  }
```

Add Files

To add a file,

- Call the menu **File > New > New File...** or press ⌘**N**.



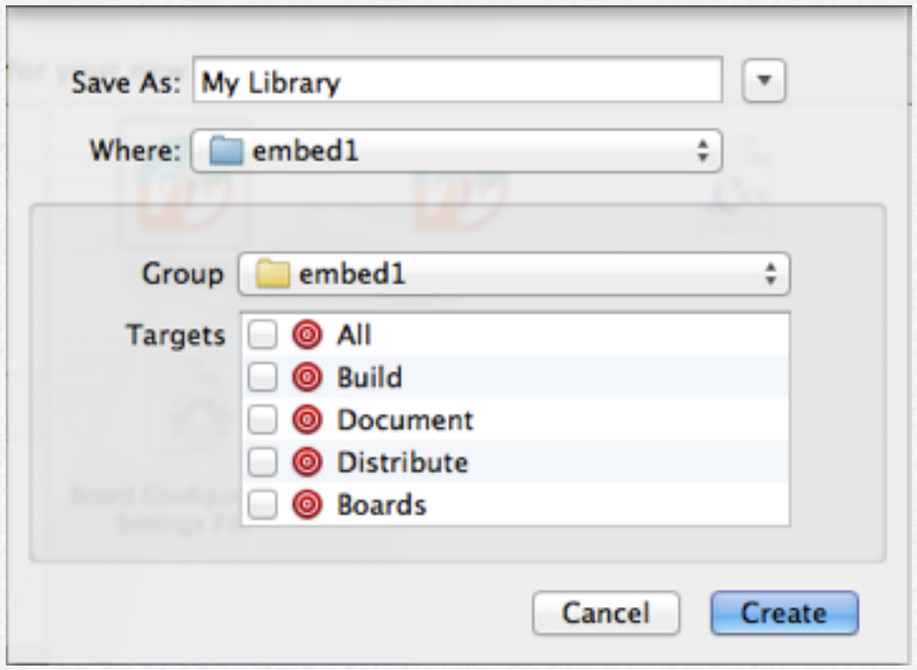
- Select **embedXcode** or **embedXcode+** and then one of the following options: **Header File**, **C++ file**, **Library**, **Class Library** or ⊞ **Board Configuration Settings File**.

New File Name

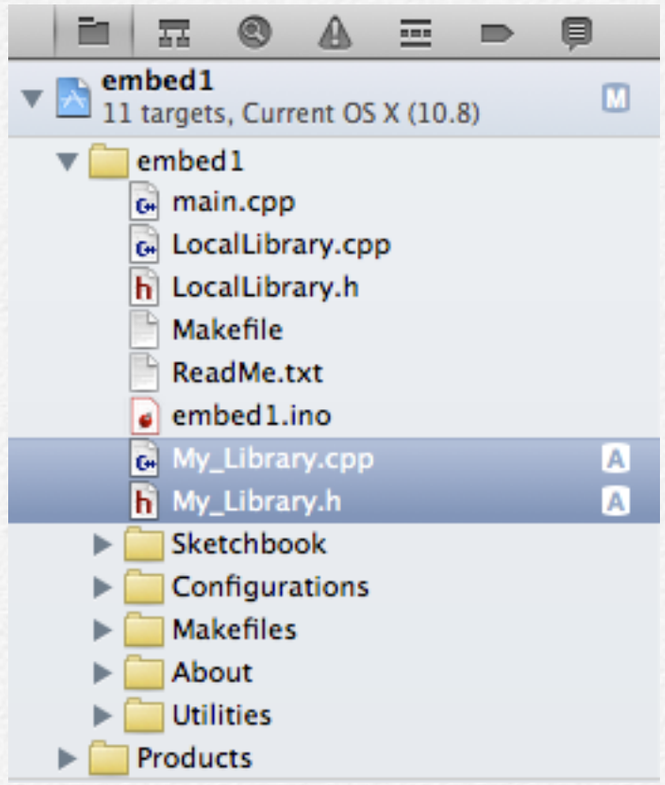
As a general rule and as for the Arduino IDEs, avoid space and special characters in the files name.

However, embedXcode manages the entered name and transforms it into a legal C-style name.

For example, if the name entered for a new library is `My Library`.



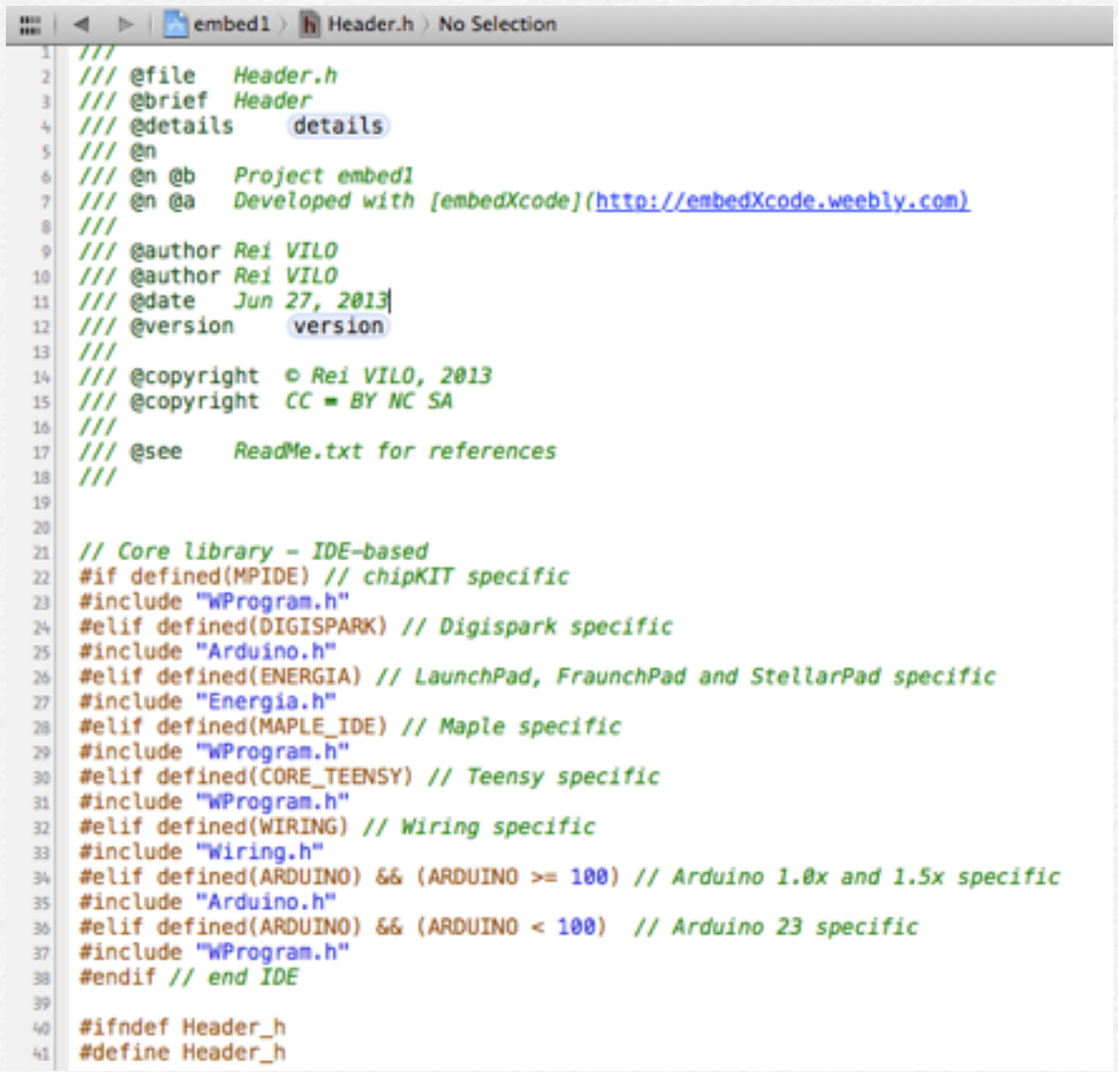
However, embedXcode renames it into **My_Library**. for the two files, `My_Library.cpp` and `My_Library.h`.



Add a C++ Code or a Header File

The **C++ File** and **Header File** options are standard files.

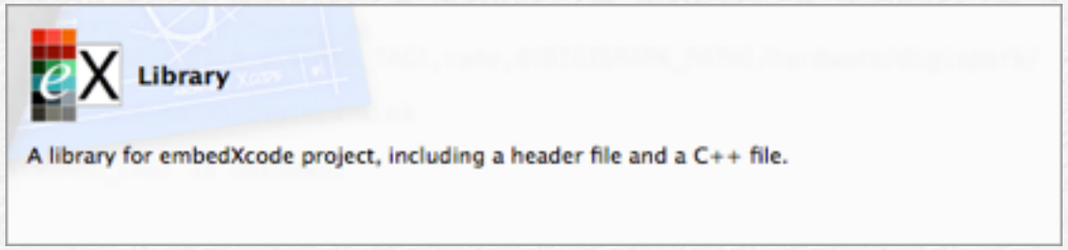
The header file comes with the `#include` pre-processing statements.



Add a Library or a Class Library

To add a library or a class library

- Call the menu **File > New > New File...** or press ⌘**N**.
- Select **embedXcode** and then **Library** or **Class Library**.



`Library` creates a header file and a C++ code file with the `#include` pre-processing statements ready.

```
#include "LocalLibrary.h"
```

`Class Library` creates a header file with the `#include` pre-processing statements and a C++ code file, including a pre-populated basic snippet for a class!

```
///
/// @class  Description
///
class ClassLibrary {

public:
  ///
  /// @brief    Cosntructor
  ///
  ClassLibrary();

  ///
  /// @brief    Initialisation
  ///
  void begin();

  ///
  ///   @brief Who am I?
  ///   @return Who am I? string
  ///
  String WhoAmI();

  ///
  ///   @brief  Description
  ///   @param  data data description
  ///
  void set(uint8_t data);

  ///
  ///   @brief  Description
  ///   @return return value description
  ///
  uint8_t get();

private:
  uint8_t _data;
};
```

The Class Library header file

```
20
21  // Library header
22  #include "ClassLibrary.h"
23
24  // Code
25  ClassLibrary::ClassLibrary() {
26
27  }
28
29  void ClassLibrary::begin() {
30      _data = 0;
31  }
32
33  String ClassLibrary::WhoAmI() {
34      return "ClassLibrary";
35  }
36
37  void ClassLibrary::set(uint8_t data) {
38      _data = data;
39  }
40
41  uint8_t ClassLibrary::get() {
42      return _data;
43  }
```

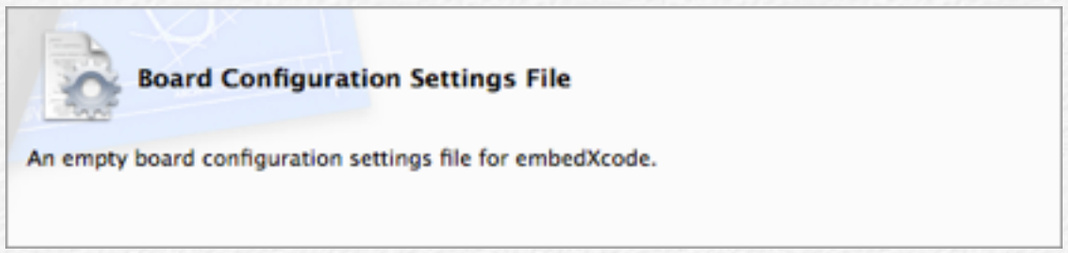
The Class Library C++ code file

Add a Configuration File for a New Board

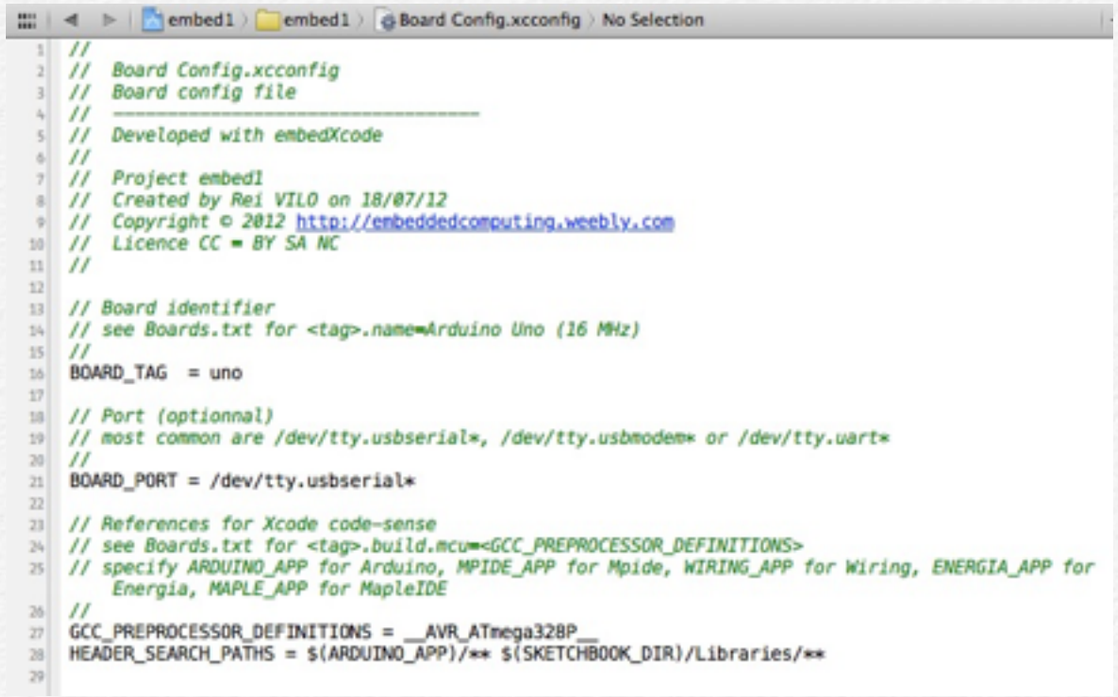
⊞ This section requires embedXcode+.

To add a configuration file for a new board,

- Call the menu **File > New > New File...** or press ⌘**N**,
- Select **embedXcode** and then **Board Configuration Settings File**.



The specific `Board Configuration Settings File` allows to define the settings for a new board, including a specific programmer.



The Board Configuration Setting file

- Specify:
  - `BOARD_TAG` is the unique identifier of the board, found in the `Boards.txt` file.

```
BOARD_TAG  = uno
```

  - `BOARD_PORT` defines the USB port to be used.

```
BOARD_PORT = /dev/tty.usbmodem*
```

This parameter is optional. To know the name of the USB port, proceed as follow:

- Open the `Terminal` window,
- Plug the board,
- Launch the command `ls /dev/tty.usb*`

```
$ ls /dev/tty.usb*
/dev/tty.usbmodem0E100631
```

- Read the answer, here `/dev/tty.usbmodem0E100631`
- Change the value of `BOARD_PORT` accordingly.

```
BOARD_PORT = /dev/tty.usbmodem*
```

The generic character `*` allows other values for the port, for example `/dev/tty.usbmodem0E100641`.

  - `GCC_PREPROCESSOR_DEFINITIONS` is the name of the micro-controller of the board, found in the `Boards.txt` file.

```
GCC_PREPROCESSOR_DEFINITIONS = __AVR_ATmega328P__
ARDUINO
```

  - `HEADER_SEARCH_PATHS` needs to be updated with the reference of the IDE, `ARDUINO_APP` for Arduino, `DIGISPARK_APP` for DigiSpark, `MPIDE_APP` for MPIDE, `ENERGIA_APP` for Energia, `MAPLE_APP` for Maple, `TEENSY_APP` for Teensy or `WIRING_APP` for Wiring.

```
HEADER_SEARCH_PATHS = $(ARDUINO_APP)/**
$(SKETCHBOOK_DIR)/Libraries/**
```

Those two parameters improve the selection of the headers for code-sense.

  - `MAX_RAM_SIZE` gives the number of bytes of SRAM. Read the specification sheet of the MCU to find the value.

```
MAX_RAM_SIZE = 2048
```

Define a Specific Programmer for a New Board

Additionally, embedXcode+ offers options to use a different programmer on the same board configuration settings file.

- Only specify the parameters when the values are different from the default ones:
  - `AVRDUDE_SPECIAL` states that a specific configuration is set for the programmer. Otherwise, comment the line.

```
AVRDUDE_SPECIAL = 1
```

  - `AVRDUDE_PROGRAMMER` provides the name of the specific programmer. Otherwise, the variable isn't required: just comment the line.

```
AVRDUDE_PROGRAMMER = usbtiny
```

  - `AVRDUDE_OTHER_OPTIONS` provides a free variable, for example for selecting verbose output or erasing flash.

```
AVRDUDE_OTHER_OPTIONS = -v
```

  - If the programmer doesn't feature a serial port, set `AVRDUDE_NO_SERIAL_PORT` to `1`.

```
AVRDUDE_NO_SERIAL_PORT = 1
```

Otherwise, set `AVRDUDE_NO_SERIAL_PORT` to `0` or comment the line. The port to be used is defined by `BOARD_PORT`.

Many ATmega MCUs are compatible and Arduino provides support to most of them. For example, the ATmega328 is compatible with the ATmega328P. However, Arduino doesn't support the ATmega328 but supports the ATmega328P.

If the speed is different, just define `F_CPU`, otherwise comment the line to use the default value provided by the IDE.

```
F_CPU = 20000000L
```

However, each MCU has a unique signature required by the programmer. For example, the signature of the ATmega328 is `0x1e 0x95 0x14` while the compatible ATmega328P has `0x1e 0x95 0x0f`.

The programmer checks the signature of the MCU, so the exact reference of the MCU needs to be specified to AVRDUDE.

  - In that case, specify `AVRDUDE_MCU` as the MCU for the programmer only:

```
AVRDUDE_MCU = atmega328
```

- Optionally, fuses can be set, including `ISP_LOCK_FUSE_PRE`, `ISP_LOCK_FUSE_POST`, `ISP_HIGH_FUSE`, `ISP_LOW_FUSE` and `ISP_EXT_FUSE`.

If those variables aren't defined on the board configuration file, default values are provided by the IDE.

```
ISP_LOCK_FUSE_PRE = 0x3f
ISP_LOCK_FUSE_POST = 0x0f
ISP_HIGH_FUSE = 0xde
ISP_LOW_FUSE = 0xff
ISP_EXT_FUSE = 0x05
```

- To by-pass the fuses, set `AVR_IGNORE_FUSES` to `1`, otherwise set the value to `0` or comment the line.

```
AVR_IGNORE_FUSES = 1
```

Please refer to the documentation of the MCUs for the correct values. Incorrect values may damage the MCU.

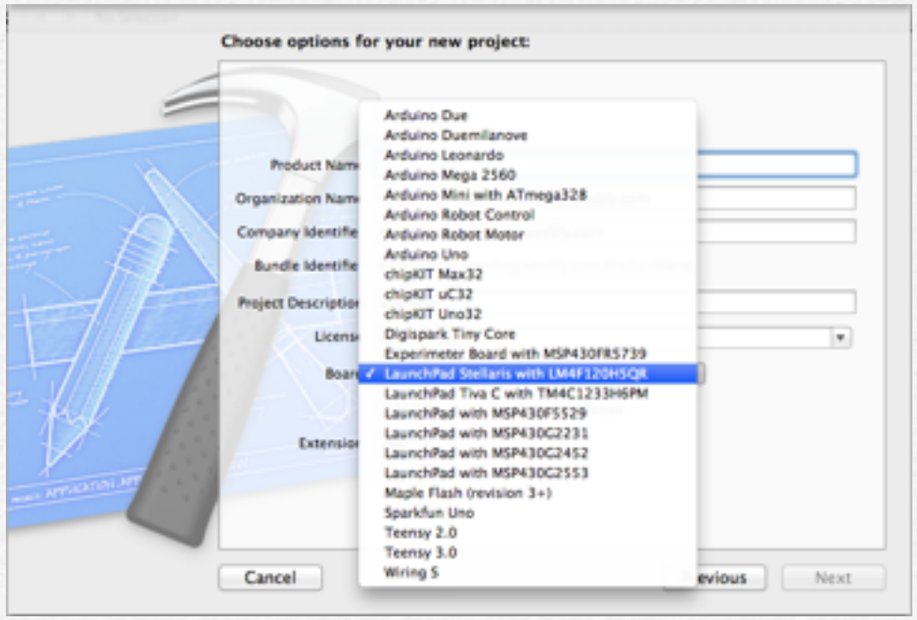
These options have been tested with the USBtinyISP AVR programmer kit ⓦ from Adafruit and the USB ASP programmer ⓦ from Protostack.

You may also need to update the FTDI drivers to use a programmer. Please refer to Utilities for Programmers.

Please refer to the documentation of the programmer for the available options, for example the AVRDUDE - AVR Downloader Uploader ⓦ manual.

Change the Board

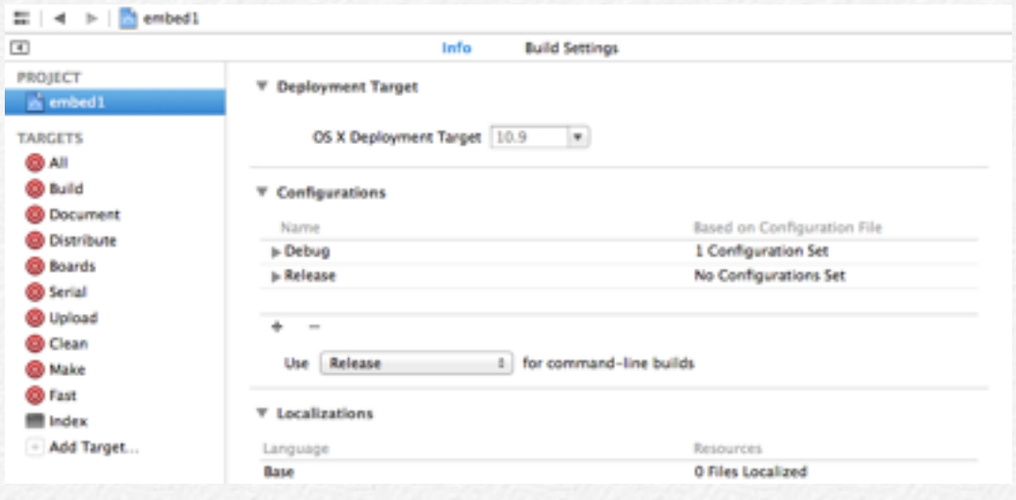
The board has been defined initially during the creation of the project.



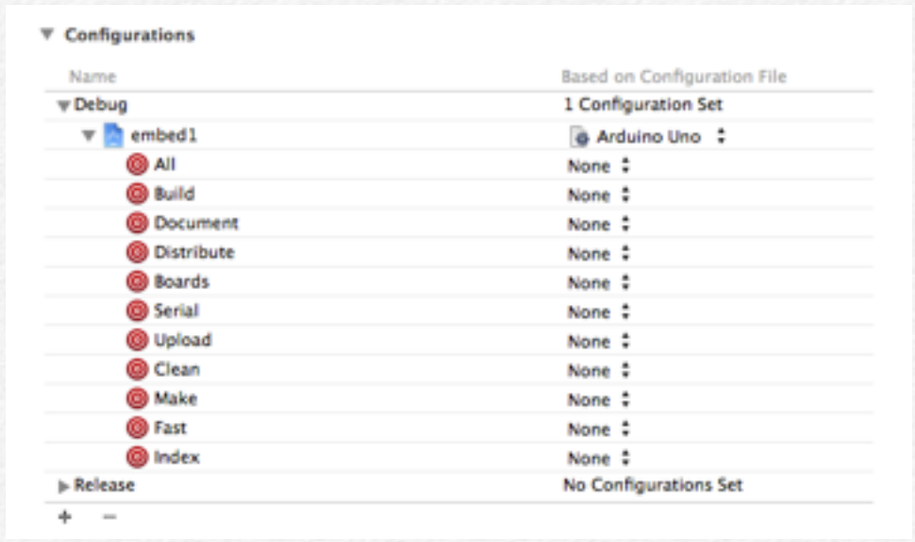
However, it can be easily changed.

To change the board,

- Select the project under **Project** and then the **Info** pane.

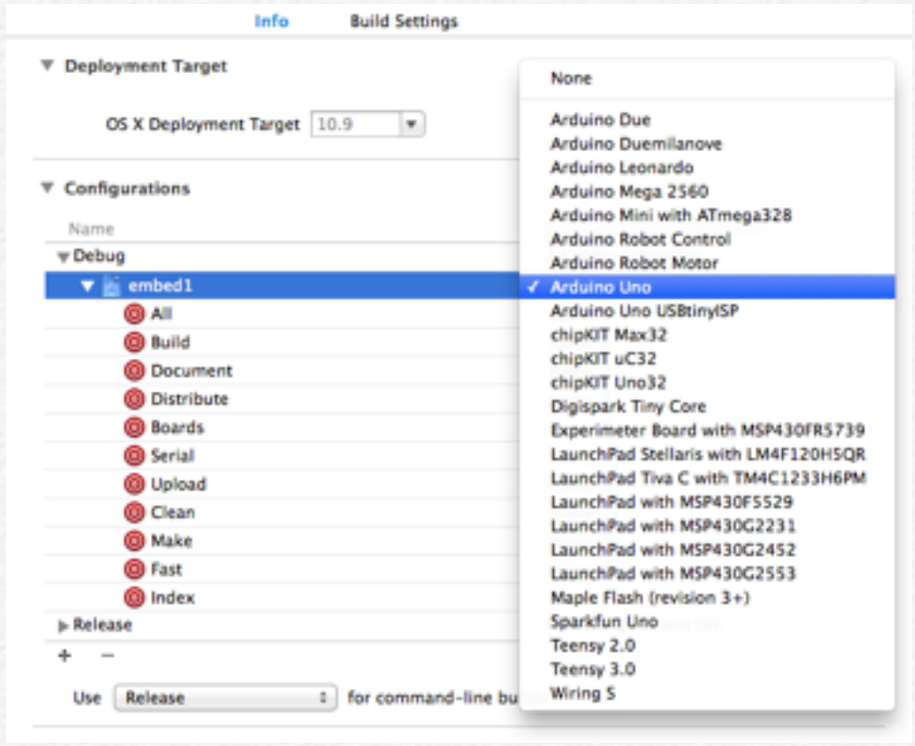


- Select **Configuration > Debug**, then the name of the project.



All the targets are displayed under the name of the project. In this example, the name of the project is `embed1` and the active board `Arduino Uno`.

- Click on the name of the board: a drop-down list shows the boards available.



- Just select one.

If your board isn't listed, you can create a configuration file. Please refer to the section Add a Configuration File for a New Board.

Manage Code for Multiple Boards

⊞ This section requires embedXcode+.

Managing code for multiples boards and platforms is a real issue. Why start a project from scratch for a new board?

We need to take into account different dimensions:

- the boards, as Arduino Uno or LaunchPad MSP430,
- the architectures, as AVR and SAM for Arduino, or MSP430 and LM4F for Energia,
- the platforms, as Arduino or Energia,
- and the frameworks, some of them with incompatible releases, as Arduino 0023, 1.0 or 1.5, Wiring.

One platform can handle different architectures, and one architecture includes different boards. For example, the same Arduino platform manages two architectures, AVR and SAM. The Arduino AVR architecture includes many different boards, as the Arduino Uno, Arduino Mega2560, ...

I've designed embedXcode with the aim of sharing the same code with different boards and platforms.

This can be done in two ways, MCU-based or IDE-based.

Both are valid from an embedXcode point of view and compatible with all the IDEs.

Write Specific Code for Multiple Platforms

The different platforms share most of the framework in common, except limited but annoying differences. Most of the code is the same, except a small number of lines. So we use pre-processing statements to write code for different platforms.

The most pre-processing used statements are `#if #elif #endif` and `#defined()`.

One example is the number of the pin for the LED,

```
    // myLED pin number
#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega2560__) ||  \
defined(__AVR_ATmega32U4__) || defined(__SAM3X8E__) // Arduino specific
    myLED = 13;
#elif defined(__PIC32MX__) // chipKIT specific
    myLED = 13;
#elif defined(__AVR_ATtinyX5__) // Digispark specific
    myLED = 1; // assuming model A
#elif defined(__AVR_ATmega644P__) // Wiring specific
    myLED = 15;
#elif defined(__MSP430G2452__) || defined(__MSP430G2553__) || \
defined(__MSP430G2231__) || \
defined(__MSP430FR5739__) // LaunchPad and FraunchPad specific
    myLED = RED_LED;
#elif defined(__LM4F120H5QR__) // StellarPad specific
    myLED = RED_LED;
#elif defined(MCU_STM32F103RB) || defined(MCU_STM32F103ZE) || \
defined(MCU_STM32F103CB) || defined(MCU_STM32F103RE) // Maple specific
    myLED = BOARD_LED_PIN;
#elif defined(__MK20DX128__) // Teensy 3.0 specific
    myLED = 13;
#endif
```

The name of the board is queried to select the right pin number. This example uses the MCU-based approach.

Another example is the function for writing a byte to the I²C bus. The function is `send()` or `write()` depending on the platform.

```
// Write or read according to framework
static void _send(uint8_t ui) {
#if defined(WIRING) // Wiring specific
  Wire.write(ui);
#elif defined(ARDUINO) && (ARDUINO >= 100)
  Wire.write(ui);
#elif defined(ENERGIA)
  Wire.write(ui);
#else
  Wire.send(ui);
#endif
}
```

This example uses the IDE-based approach, as it queries the IDE.

MCU-Based Approach

The first approach is MCU-based and relies solely on the micro-controller type.

This approach is compatible with the respective IDEs, as no new environment variable is created or required.

In the Arduino case, two frameworks exist so the IDE variable `ARDUINO` is required for disambiguation.

```
// Core library for code-sense
#if defined(__PIC32MX__) // chipKIT specific
#include "WProgram.h"
#elif defined(__AVR_ATtinyX5__) // Digispark specific
#include "Arduino.h"
#elif defined(__AVR_ATmega644P__) // Wiring specific
#include "Wiring.h"
#elif defined(__MSP430G2452__) || defined(__MSP430G2553__) || \
defined(__MSP430G2231__) || defined(__MSP430FR5739__) || \
defined(__LM4F120H5QR__) \ // LaunchPad, FraunchPad and StellarPad specific
#include "Energia.h"
#elif defined(MCU_STM32F103RB) || defined(MCU_STM32F103ZE) || \
defined(MCU_STM32F103CB) || defined(MCU_STM32F103RE) // Maple specific
#include "WProgram.h"
#elif defined(__MK20DX128__) // Teensy 3.0 specific
#include "WProgram.h"
#elif defined(__AVR_ATmega328P__) || defined(__AVR_ATmega2560__) ||  \
defined(__AVR_ATmega32U4__) || defined(__SAM3X8E__) // Arduino specific
#if defined(ARDUINO) && (ARDUINO >= 100) // Arduino 1.0 and 1.5 specific
#include "Arduino.h"
#elif defined(ARDUINO) && (ARDUINO < 100) // Arduino 23 specific
#include "WProgram.h"
#else // error
#error Platform not defined
#endif
```

IDE-Based Approach

The second approach is IDE-based.

Each IDE defines a specific environment variable which includes the boards type it supports, and optionally the framework version.

For example, the Arduino IDE defines `ARDUINO=23`, `ARDUINO=101` or `ARDUINO=150`, depending on the version installed.

The variable is then passed on to the tool-chain with `-D`, as `-DARDUINO=101` or `-DARDUINO=150`.

```
// Core library - IDE-based
#if defined(WIRING) // Wiring specific
#include "Wiring.h"
#elif defined(MAPLE_IDE) // Maple specific
#include "WProgram.h"
#elif defined(MPIDE) // chipKIT specific
#include "WProgram.h"
#elif defined(DIGISPARK) // Digispark specific
#include "Arduino.h"
#elif defined(ENERGIA) // LaunchPad, FraunchPad and StellarPad specific
#include "Energia.h"
#elif defined(CORE_TEENSY) // Teensy specific
#include "WProgram.h"
#elif defined(ARDUINO) && (ARDUINO >= 100) // Arduino 1.0 and 1.5 specific
#include "Arduino.h"
#elif defined(ARDUINO) && (ARDUINO < 100) // Arduino 23 specific
#include "WProgram.h"
#else // error
#error Platform not defined
#endif
```

The Arduino, Wiring and Maple IDEs set one single environment variable: `ARDUINO=23` or `ARDUINO=101`, `WIRING=100` and `MAPLE_IDE`, respectively.

The remaining three IDEs, MPIDE, Teensy and Energia defines two environment variables, their own on top of the default one: `MPIDE=23` and `ARDUINO=23`, `CORE_TEENSY` and `ARDUINO=101`, `ENERGIA=10` and `ARDUINO=101`, respectively.

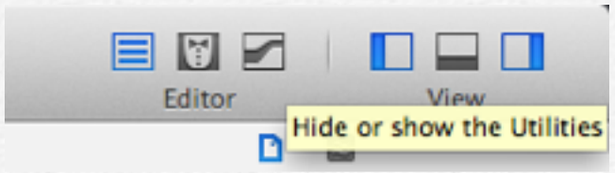
So embedXcode tests `ARDUINO` after the specific variables.

Insert Pre-Processing Statements From Code Snippet

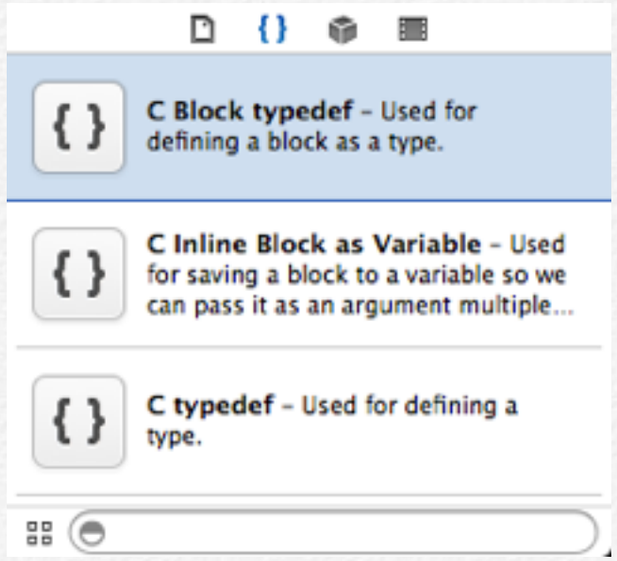
A code snippet includes all the pre-processing statements for selecting the core libraries. There are two versions: one MCU-based and another IDE-based.

For more information, please refer to Write Specific Code for Multiple Platforms.

To display the code snippets, click on the right button of the **View** selector.

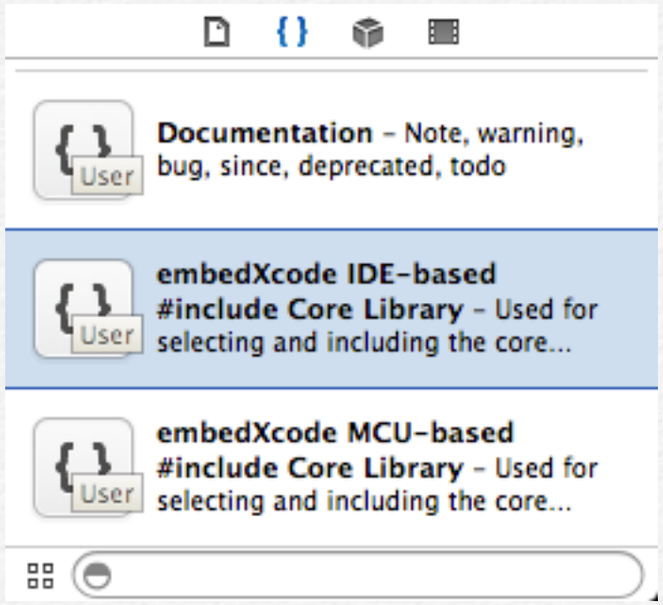


The library of code snippets is at the bottom of the pane.



- Select **User** on the drop-down list.

- Select the **embedXcode #include Core Library** snippet.



- Click and drop to the destination.

The pointer changes for  .

The cursor appears on the code.

The code is then inserted.

embed1 〉 embed1 〉 embed1.ino 〉 No Selection

```
///
/// @file       embed1.ino
/// @brief      Main sketch
///
/// @details    details
/// @n @a       Developed with [embedXcode+](http://embedXcode.weebly.com)
///
/// @author     Rei VILO
/// @author     http://embeddedcomputing.weebly.com
/// @date       11/10/13 18:56
/// @version    version
///
/// @copyright  (c) Rei VILO, 2013
/// @copyright  CC = BY SA NC
///
/// @see        ReadMe.txt for references
/// @n
///

// Core library - IDE-based
#if defined(WIRING) // Wiring specific
#include "Wiring.h"
#elif defined(MAPLE_IDE) // Maple specific
#include "WProgram.h"
#elif defined(MPIDE) // chipKIT specific
#include "WProgram.h"
#elif defined(DIGISPARK) // Digispark specific
#include "Arduino.h"
#elif defined(ENERGIA) // LaunchPad MSP430, Stellaris and Tiva, Experimeter Boar
#include "Energia.h"
#elif defined(CORE_TEENSY) // Teensy specific
#include "WProgram.h"
#elif defined(ARDUINO) && (ARDUINO >= 100) // Arduino 1.0 and 1.5 specific
#include "Arduino.h"
#elif defined(ARDUINO) && (ARDUINO < 100) // Arduino 23 specific
#include "WProgram.h"
#else // error
#error Platform not defined
#endif


// Include application, user and local libraries
#include "LocalLibrary.h"
```

Build and Upload the Project

4

Select the target, manage the libraries, built and upload the project.

Select a Target

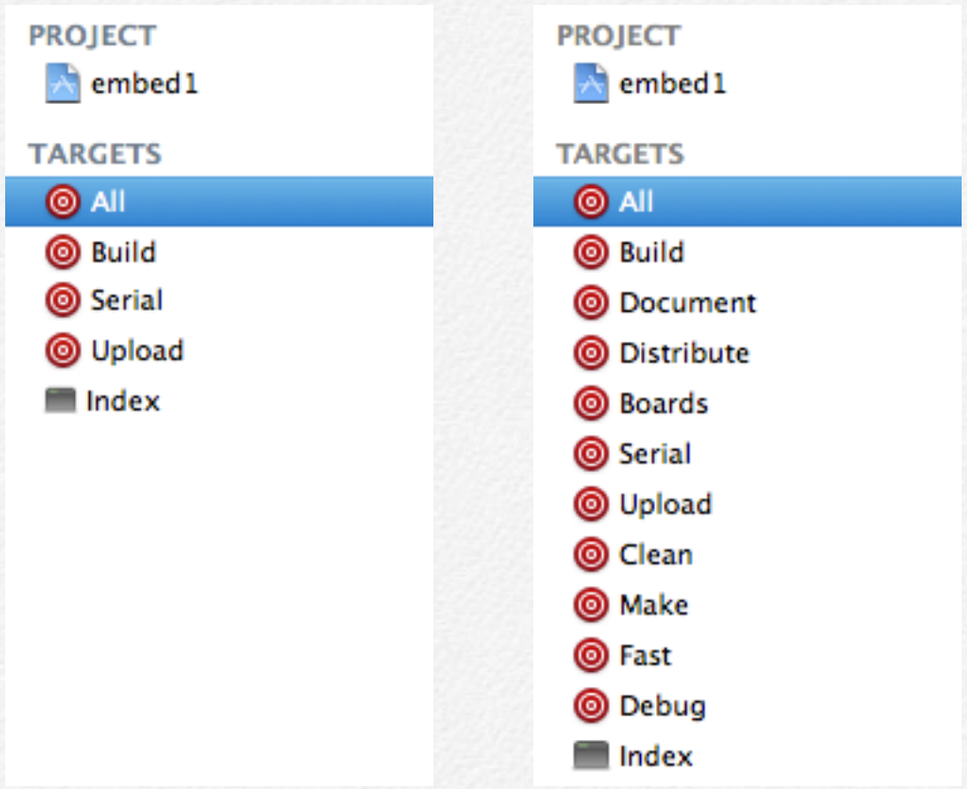
Eleven Targets

Eleven targets are available:

- **All** cleans the files from a previous compilation, compiles and links, uploads and opens a serial window in Terminal.
- **Build** compiles all the files, changed and unchanged, and links them.
- **Boards** lists all the boards with their tags and their names.
- **Serial** open the serial console in a `Terminal` window.
- **Upload** uploads the resulting HEX or BIN executable file to the board.
- **Clean** erases the files generated from a previous compilation.
- ⊞ **Fast** compiles only the main sketch and the local libraries, links and uploads them, and opens a serial window in Terminal.
- ⊞ **Make** compiles only the main sketch and the local libraries, and links them.
- ⊞ **Debug** compiles only the main sketch and the local libraries, links and uploads them, and opens a debugging session in a `Terminal` window.

- ⊞ **Document** builds the documentation.
- ⊞ **Distribute** prepares a folder for distribution.

The additional target **Index** is a proxy target solely used for code-sense. Do not launch it.



The targets for embedXcode (left) and embedXcode+ (right)

What Does What?

The following table shows the scope of each target:

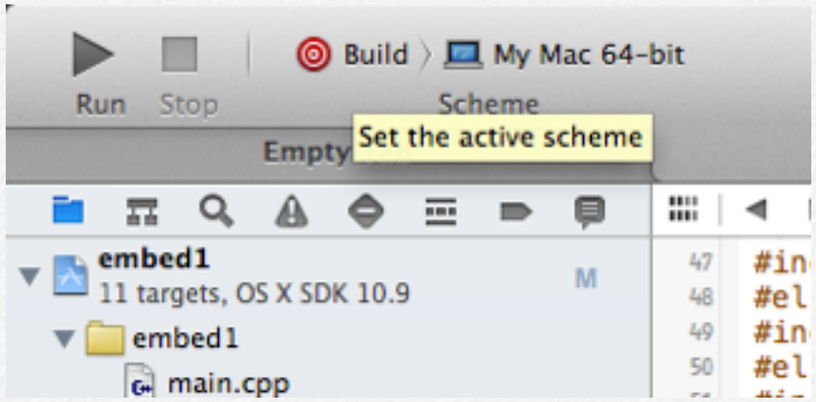
Target...	Cleans	Compiles	Uploads	Opens Terminal	Debugs
All	■	■	■	■	
Build	■	■			
Serial				■	
Upload			■		
Clean	■				
⊞ **Fast**		■	■	■	
⊞ **Make**		■			
⊞ **Debug**		■	■	■	■

For example, the **All** target cleans, compiles, uploads and opens a `Terminal` window.

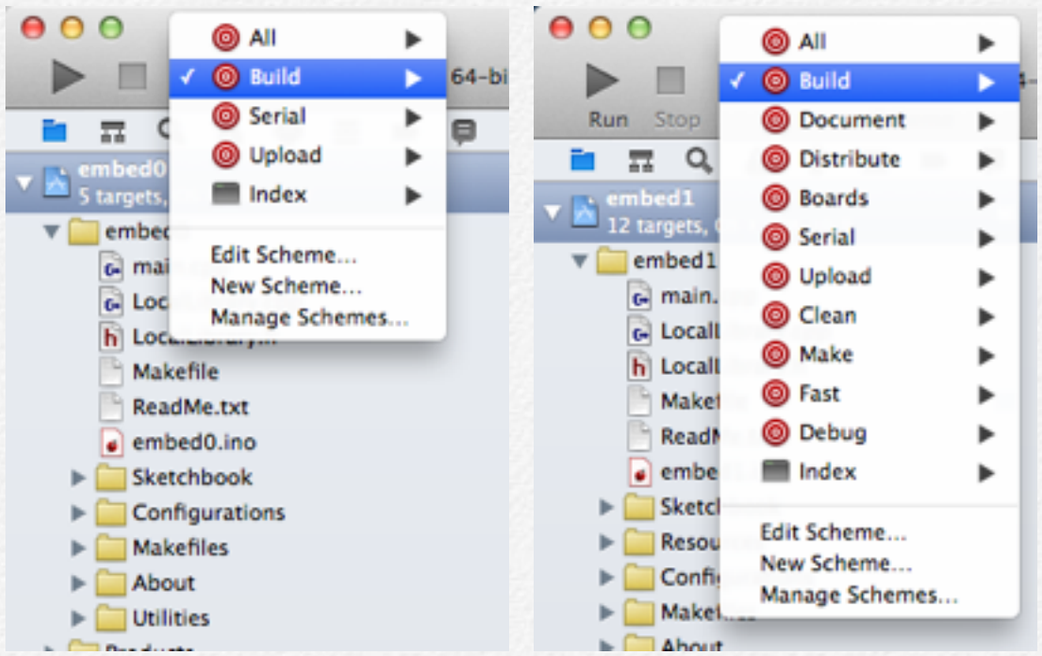
Select a Target

To select a target,

- Click on the list on the top right of the Xcode window.



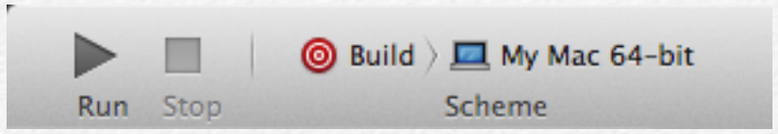
- Then select the target you want.



The targets for embedXcode (left) and embedXcode+ (right)

Once the target is selected,

- Click on the **Run** button to launch it.



As a matter of fact, I mainly use four of them:

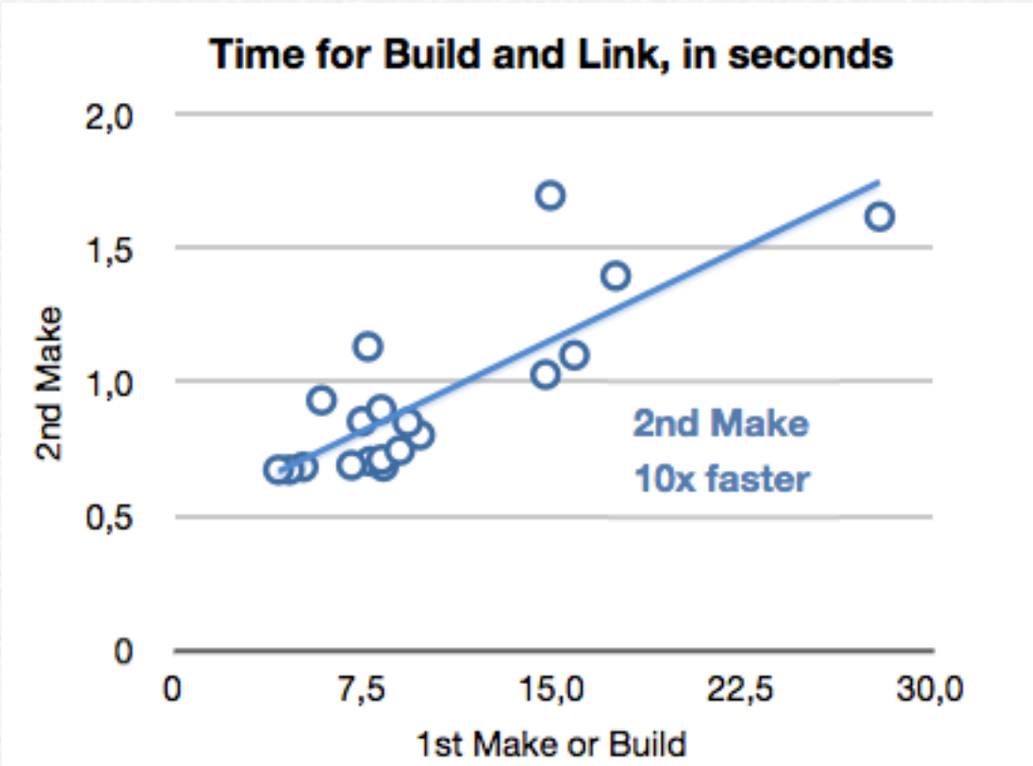
- **Build** and ⊞ **Make** for developing the code,
- **All** and ⊞ **Fast** for compiling and uploading the code.

Other targets may be useful for specific needs.

Faster Targets

⊞ embedXcode+ offer additional faster targets.

The ⊞ **Make** and ⊞ **Fast** targets feature an optimisation for the build of the foundation libraries, which include the core, application and user libraries.



The first time the **Make** and **Fast** targets are launched, they build all the foundation libraries into an archive for later reuse.

On their first launch, the **Make** and **Fast** targets act exactly the same way as the corresponding **Build** and **All** targets.

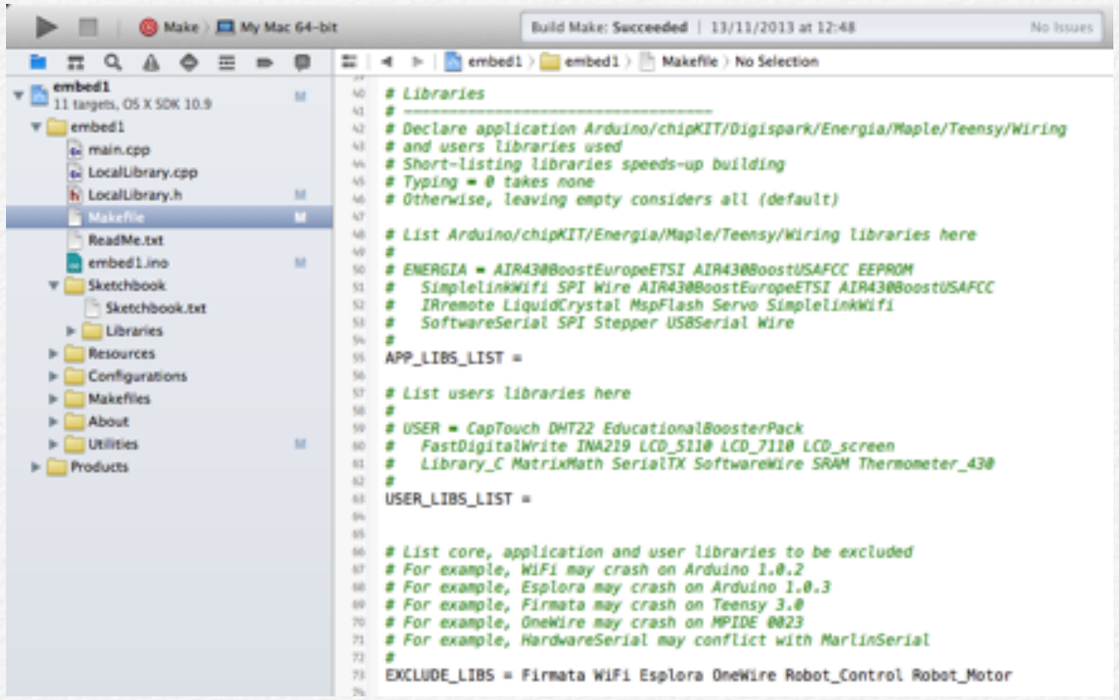
The second time the **Make** and **Fast** targets are launched, they only compile the main sketch and the local libraries, and link them reusing the foundation archive built previously. The **Fast** target also uploads and opens a serial window in Terminal.

So on their second launch, the **Make** and **Fast** targets are on average 10 times faster than the corresponding **Build** and **All** targets.

Manage the Libraries for Compilation

This procedure allows to manage the libraries:

- Select only the libraries to be taken into account to speed compilation, and
- Exclude some libraries to avoid conflicts.



For more information on the installation of the libraries, please refer to the tutorial Installing Additional Arduino Libraries ⓦ.

⊞ The main `makefile` is populated with the list of all the libraries during preparation.

Select the Libraries for Compilation

On the makefile, two variables are used:

- `APP_LIBS_LIST` for the Arduino / chipKIT / Digispark / Energia / Maple / Microduino / Teensy / Wiring libraries
- and `USER_LIBS_LIST` for the user's libraries under the `Library` sub-folder on the sketchbook folder.

By default, both variables are set to `0`: no library is considered and compiled.

```
# List Arduino/chipKIT/Wiring/Energia/Maple
libraries here
#
APP_LIBS_LIST = 0

# List users libraries here
#
USER_LIBS_LIST = 0
```

This is the fastest option, as no library is compiled.

If a library is required, just specify the name of its folder. Only the specified libraries are compiled.

For example, if the `Wire` library is required in the project:

- Specify the `Wire` folder after the `APP_LIBS_LIST` variable in the main makefile:

```
# List Arduino/chipKIT/Wiring/Energia/Maple
libraries here
#
APP_LIBS_LIST = Wire
# List users libraries here
#
USER_LIBS_LIST = 0
```

- Add use the `#include` statement in the main sketch:

```
64  // Include application, user and local libraries
65  #include "Wire.h"
66
```

finally, if you don't know which libraries are needed, just leave the two lines empty after the variables.

```
# List Arduino/chipKIT/Wiring/Energia/Maple
libraries here
#
APP_LIBS_LIST =
# List users libraries here
#
USER_LIBS_LIST =
```

This is the slowest option. Additionally, it may rise errors as some libraries are conflicting with others.

Exclude Libraries

Some libraries are specific to a platform but are included in a folder shared with other platforms, and some libraries may conflict with other ones.

For example, the `WiFi` library included among the core libraries is solely designed for the Arduino WiFi Shield.

With the Arduino 1.0.x and 1.5.x IDE, the `WiFi` library is included by default in the compilation process, and raises error and warning messages even with the official IDE.

To exclude the `WiFi` library and avoid any unnecessary error, just mention its name after the variable `EXCLUDE_LIBS`.

```
# List the libraries to be excluded
# For example, WiFi may crash on Arduino 1.0.2
#
EXCLUDE_LIBS = WiFi
```

Same happens for user's libraries. If a serial library conflicts with `SoftwareSerial`, use `EXCLUDE_LIBS` to prevent that.

```
# List the libraries to be excluded
# For example, WiFi may crash on Arduino 1.0.2
#
EXCLUDE_LIBS = SoftwareSerial
```

If you need to use all the libraries, leave the line empty after the `EXCLUDE_LIBS` variable.

```
EXCLUDE_LIBS =
```

By default, the `EXCLUDE_LIBS` is empty and the line has a leading # to comment it:

```
# List the libraries to be excluded
#
#EXCLUDE_LIBS =
```

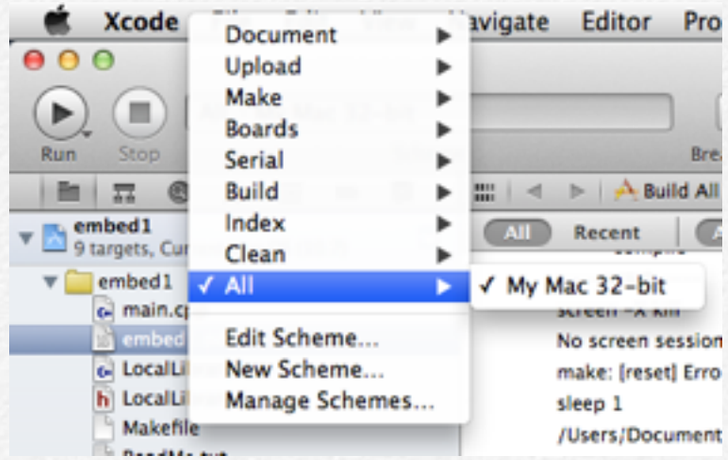
The following libraries may rise issues:

- `WiFi` on Arduino 1.0.2,
- `Esplora` on Arduino 1.0.3,
- `Firmata` on Teensy 3.0,
- `OneWire` on MPIDE 0023.
- `ArduinoRobot` on Arduino 1.0.5

The parameter impacts all libraries, core, application and user.

Build and Upload

Select the target **All** or ⊞ **Fast** and then press the button **Run**.



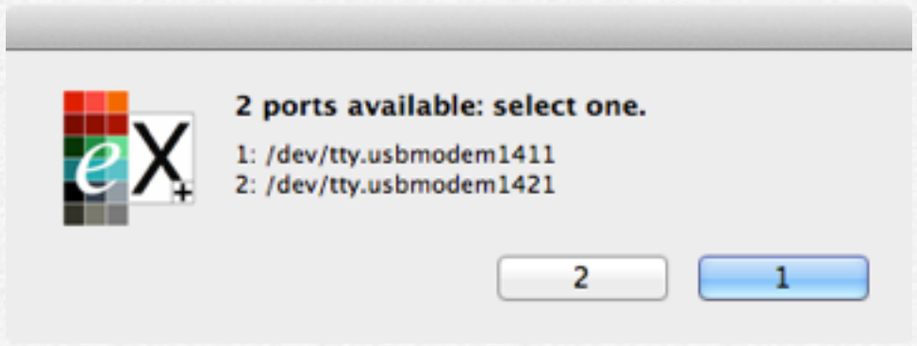
The sketch is going to be built and linked, and the resulting executable uploaded to the board and open.

At the end, the `Terminal` application opens a serial console.

Select Among Two Connected Boards

⊞ This section requires embedXcode+.

In case two similar boards are connected, for example two Arduino boards, a windows lists the connections and ask to select one:



The first port is the default option.

- To select the default port, press **Enter**, click on the button **1** or wait 10 seconds.
- To select the second port, click on the button **2**.

The following message is added to the **Log navigator**:

```
==== 2 ports available ====
1: /dev/tty.usbmodem1411
2: /dev/tty.usbmodem1421
==== /dev/tty.usbmodem1411 port selected ====
```

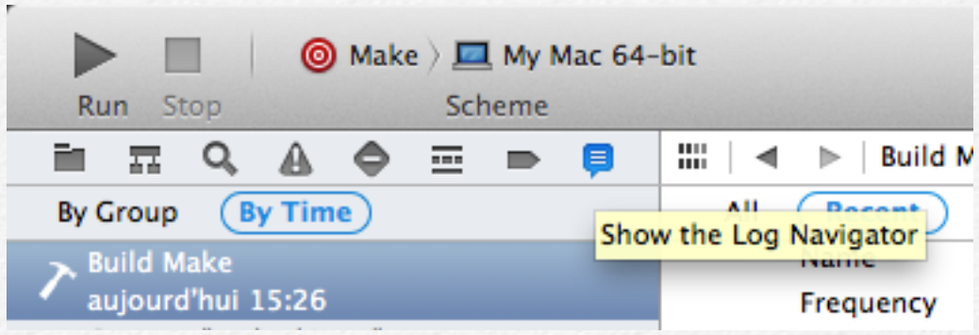
This feature works with the Arduino boards and the boards using a serial through USB connection.

It is not available for the LaunchPads, as they use a different management that combines different services under a single USB port.

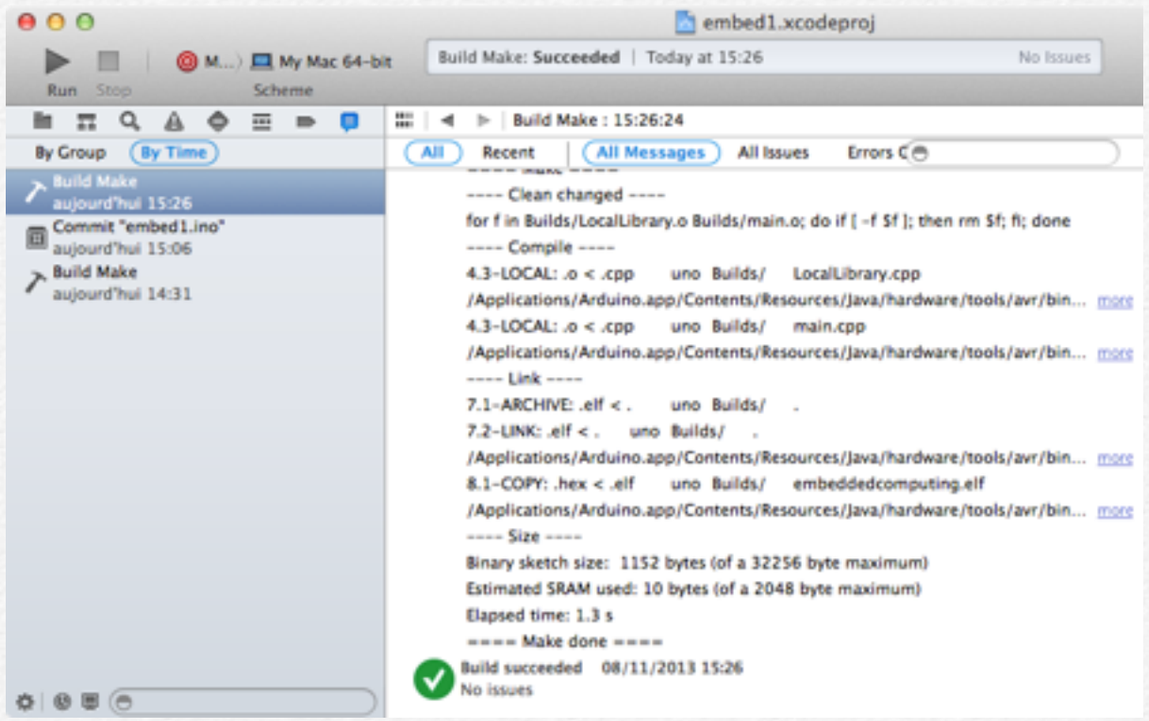
Display the Log Navigator

The **Log navigator** provides information about the compilation and the size of the final binary file.

To display the **Log navigator**, click on the rightmost icon:



The **Log navigator** provides the detail of the whole process.



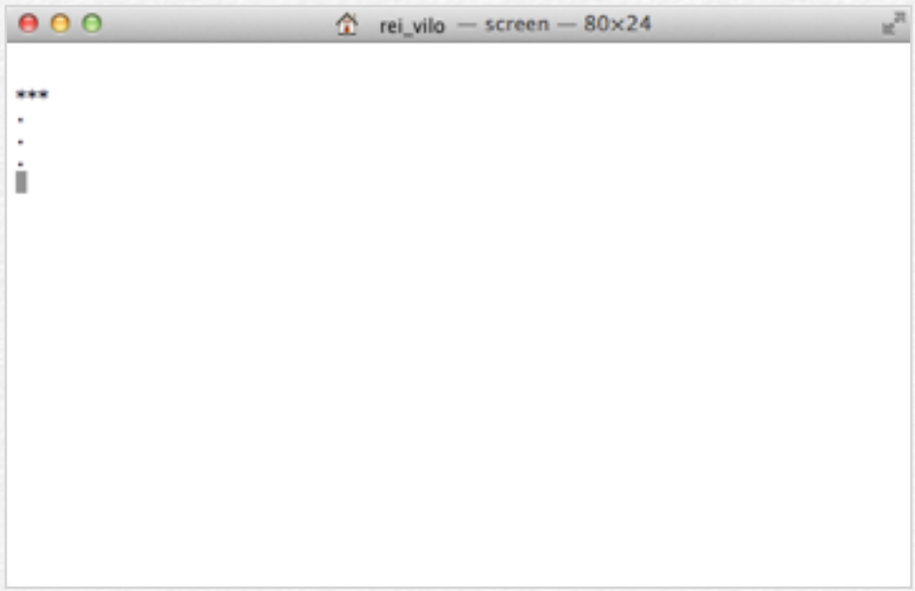
Once compilation is completed, the **Log Navigator** displays the total size of the sketch on the flash memory and an estimation of the RAM usage.

It also provides the elapsed time spent for building and linking.

```
---- Size ----
Binary sketch size:  1158 bytes (of a 32256 byte
maximum)
Estimated SRAM used: 10 bytes (of a 2048 byte
maximum)
Elapsed time: 0.3 s
==== Make done ====
```

Display the Serial Console on a Terminal Window

If the target **All** or ⊞ **Fast** is selected, once the sketch is built, linked and uploaded to the board, the `Terminal` application opens a serial console.



The serial console can be opened in a `Terminal` window by selecting the target **Serial** and then **Run**.

To close the serial connection,

- Use the **ctlr-K ctrl-A** key sequence.

- Then press **Y**.

For the moment, the serial console isn't displayed on the debug pane in the main Xcode window.

Upload Specific Procedures

The following boards and programmers require specific procedures for uploading the sketch:

- Digispark,
- Teensy 3.0 and 3.1,
- LaunchPad MSP430F5529,
- and Arduino Yún for over-the-air upload with WiFi or Ethernet connection.

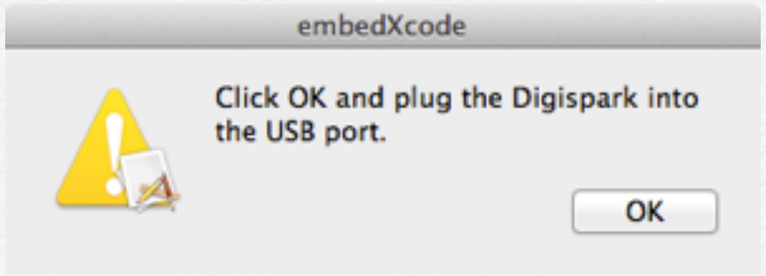
An additional section provides the procedure for using standard USB upload after having used a programmer.

Digispark Upload Specific Procedure

The Digispark board requires a specific procedure.

Proceed as follow:

- Unplug the Digispark board.
- Launch any of the targets **All**, **Upload** or **Fast**.
- Wait for the message window:

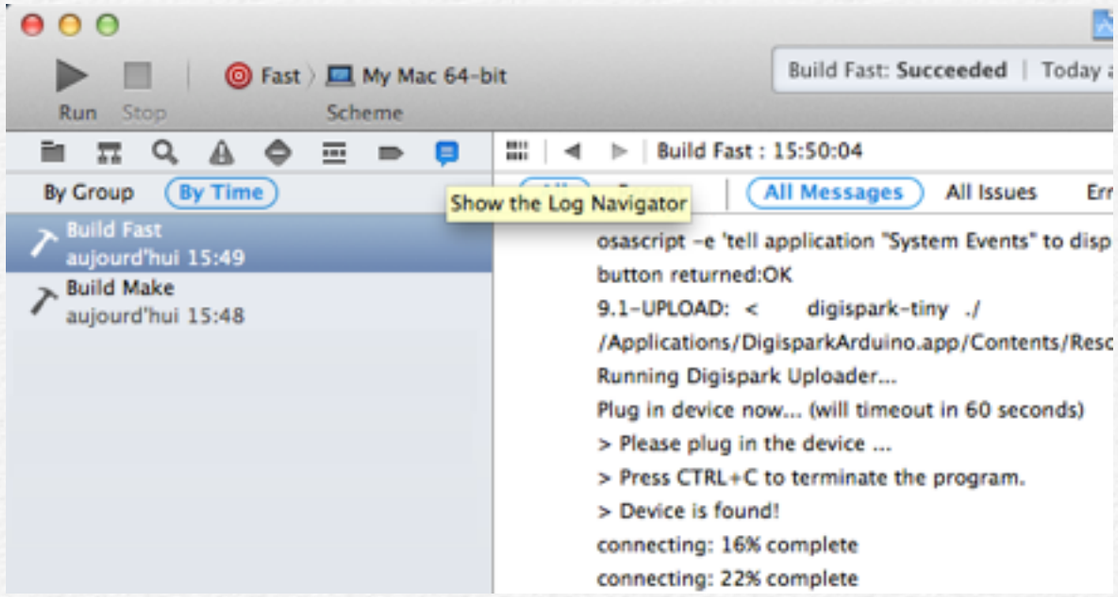


- Plug the cable from the Digispark board into the USB port.
- Wait for the final message.

You can also monitor the upload with the **Log Navigator**.

Proceed as follow:

- Open the **Log Navigator**.



Wait for the message `Plug in device now...`

```
Running Digispark Uploader...
Plug in device now...
```

- Plug the cable from the Digispark board into the USB port.

```
> Press CTRL+C to terminate the program.
> Device is found!
```

The uploader connects, erases and then writes the sketch on the Digispark board.

```
connecting: 20% complete
erasing: 80% complete
> Starting to upload ...
writing: 80% complete
```

- Wait for the final message:

```
Writing: 100% complete
>> Micronucleus done. Thank you!
```

For more information, please refer to the tutorial Connecting and Programming Your Digispark ⓦ available at the Digistump wiki.

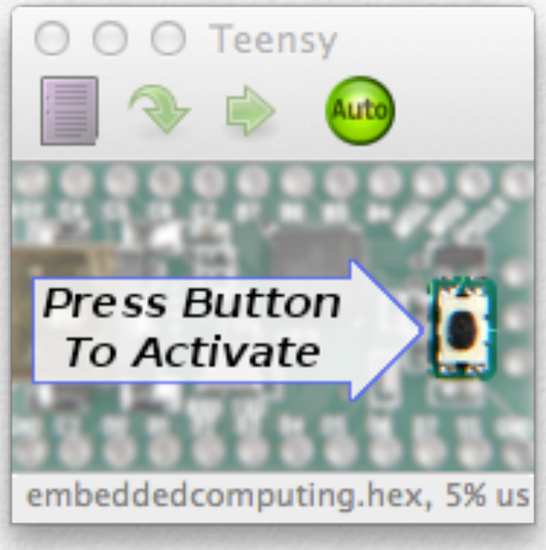
Digistump Wiki

Teensy 3.0 and 3.1 Upload Specific Procedure

The Teensy 3.0 and 3.1 boards require a specific procedure.

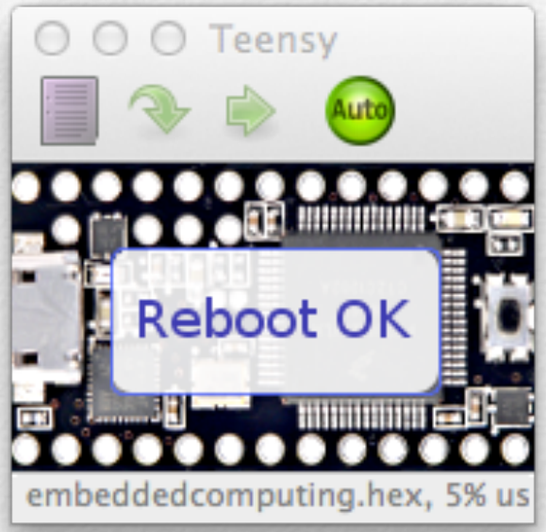
During the first upload, a new window asks for the button on the board to be pressed in order to start the process.

The following uploads no longer require this manual operation.



- Press the button on the Teensy 3.0 board.

A final message confirms the end of the operation.



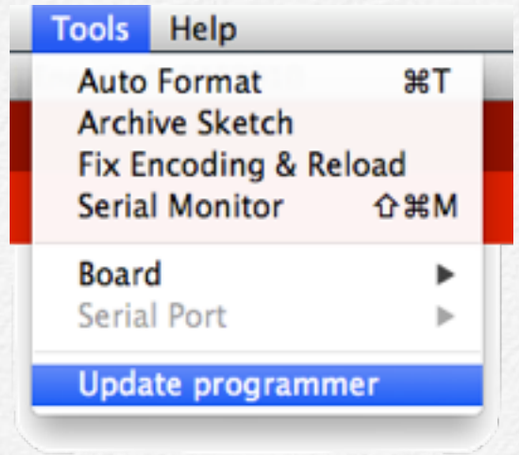
LaunchPad MSP430F5529 Upload Specific Procedure

The LaunchPad MSP430F5529 requires Energia release 10.

Some versions of the LaunchPad MSP430F5529 may require an update of the firmware.

To do so,

- Download Energia release 10.
- Launch Energia.
- Create a new basic sketch, for example the `blinky` sketch.
- Select the board **LaunchPad with MSP430F5529**
- Upload the sketch to the LaunchPad.
- If the firmware needs to be updated, a windows pops-up.
- In that case, follow the instructions.
- Call the menu **Tools > Update programmer**.



- Close Energia.

Arduino Yún Ethernet or WiFi Upload Specific Procedure

⊞ This section requires embedXcode+.

Although the Arduino Yún requires no specific procedure, the Ethernet or WiFi network needs to be installed and configured successfully before any upload.

Please check that the router has discovered the Arduino Yún and note the IP address. Also, keep the password of the Arduino Yún at hand.

During the first compilation, a message box asks for the password:

- Enter the password of the Arduino Yún board.
- Click on **OK** to validate or **Cancel** to cancel.

When validated, the password is saved on the board configuration file Arduino Yun (WiFi Ethernet) in the project.

```
48  // Yun password
49  //
50  YUN_PASSWORD = password
51
```

Please note the password is not encrypted. The password is only asked once. To erase the password, just delete the corresponding line.

For more information about the Arduino Yún installation and over-the-air upload, please refer to the Guide to the Arduino Yún ⓦ on the Arduino website.

Using Standard USB Upload After a Programmer

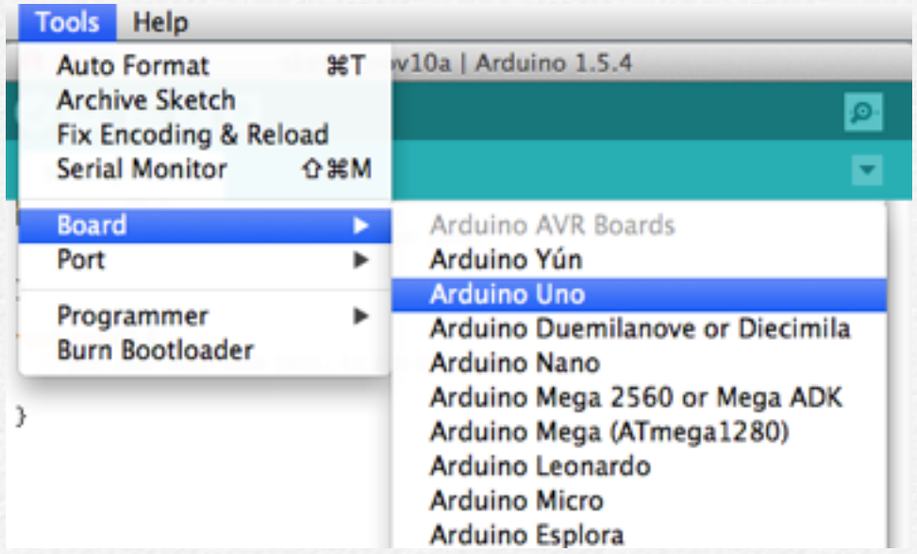
The programmer erases all the flash, including eventually the boot-loader.

After having used a programmer, you may need to burn to boo-loader again to upload a sketch through the standard USB connection.

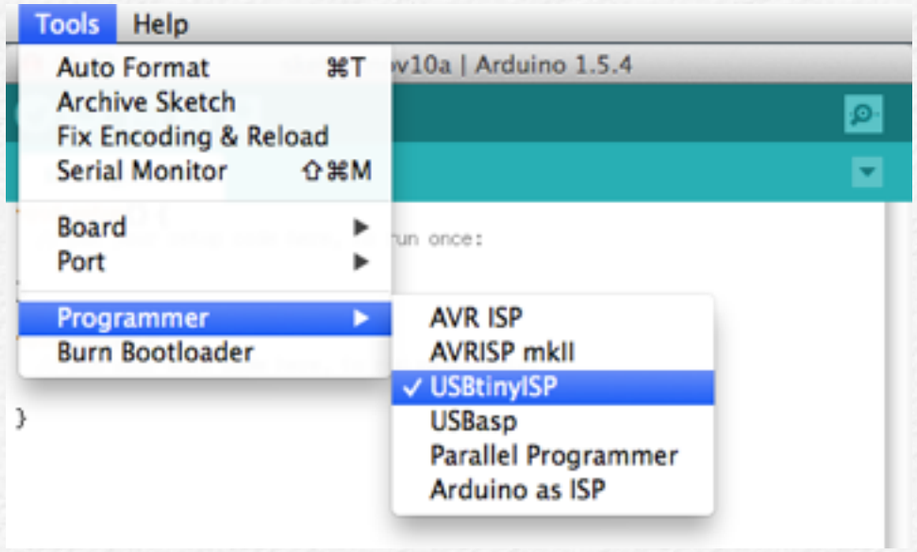
For more informations, please refer to the Boot-Loader Development ⓦ page at the Arduino website.

To burn the boot-loader again,

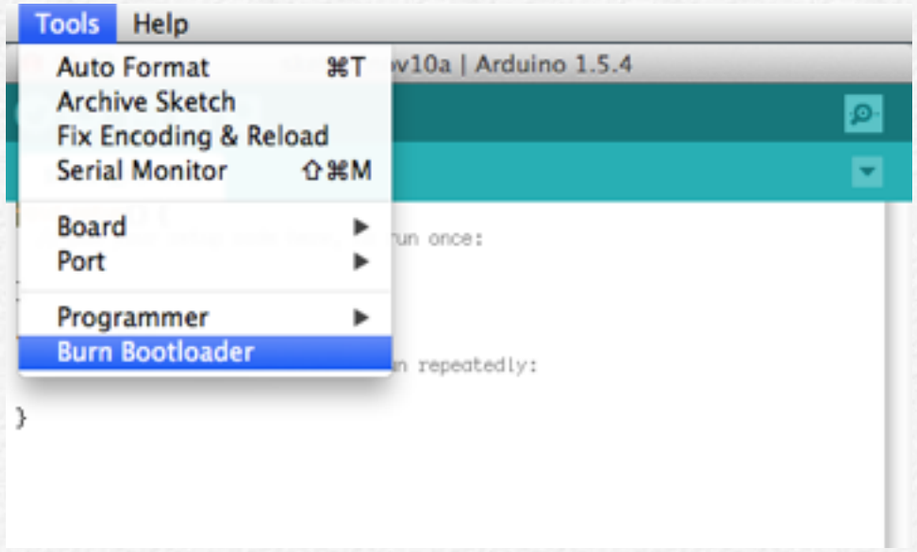
- Launch the Arduino IDE,
- Select the board, here the Arduino Uno board.



- Select the programmer, here the USBtinyASP programmer.



- Click on **Burn Bootloader**.



You can upload the sketch to the board using the standard USB connection again.

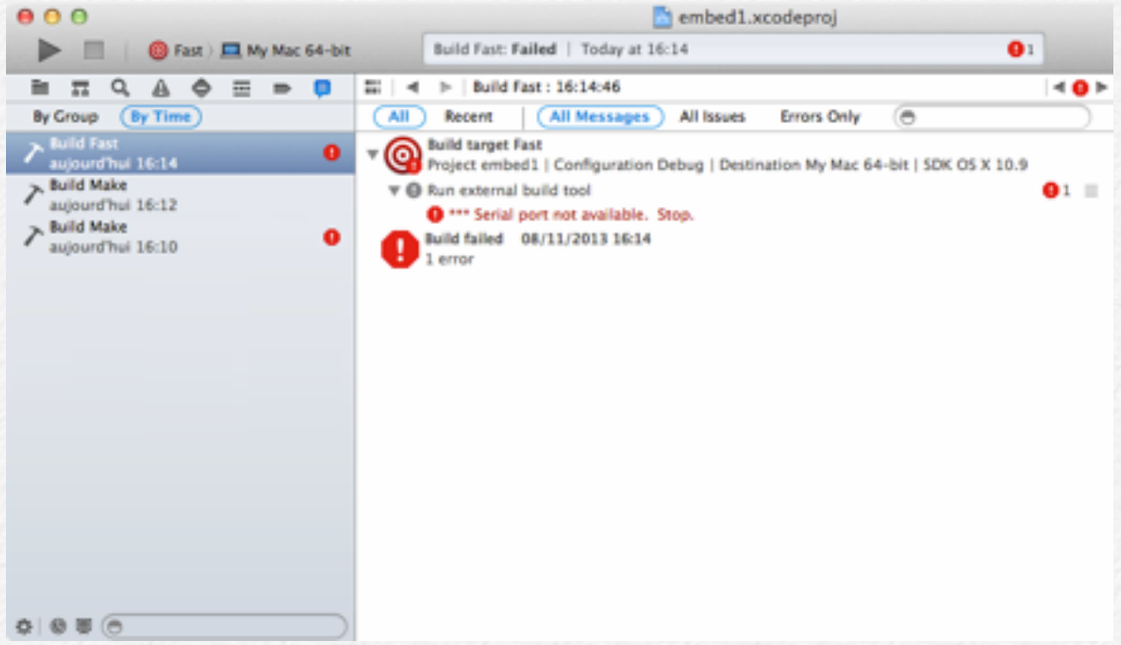
Messages

Error Messages

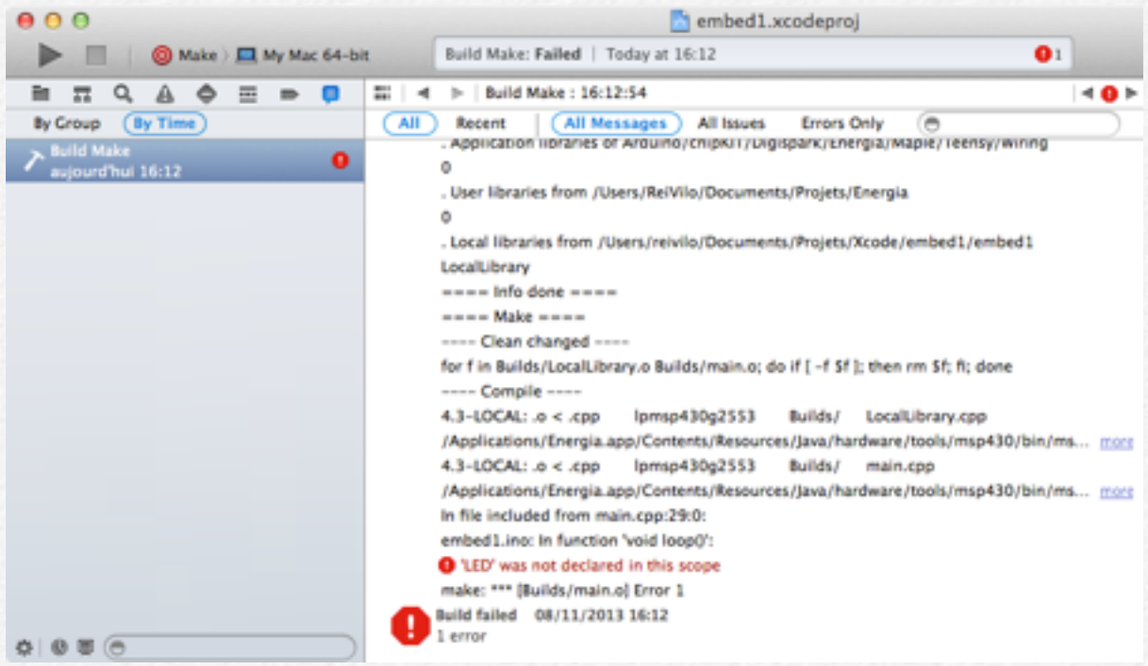
In case of a failed build, the **Issue navigator** display a message, either warning or error.

A warning is highlighted in yellow and doesn't stop the compilation. An error is highlighted in red and stops the compilation.

Here, the board isn't connected and thus the serial port is not available.

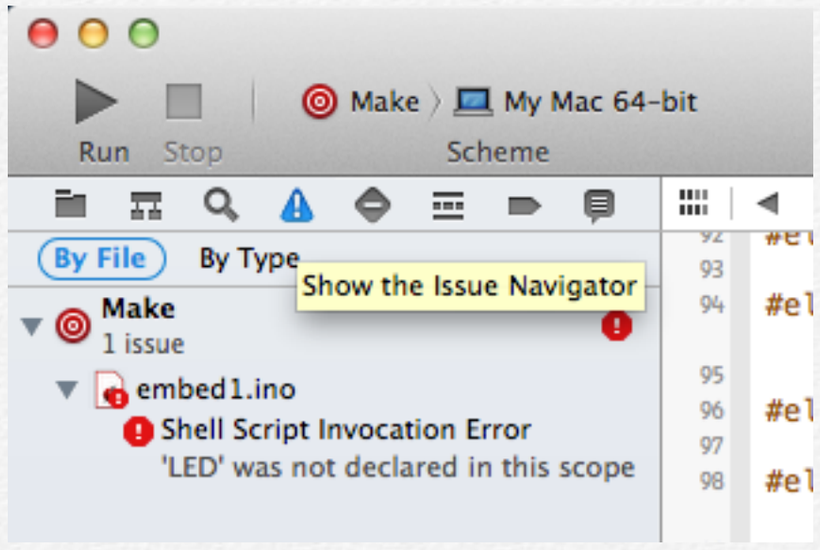


Here, `LED` was not declared in this scope.

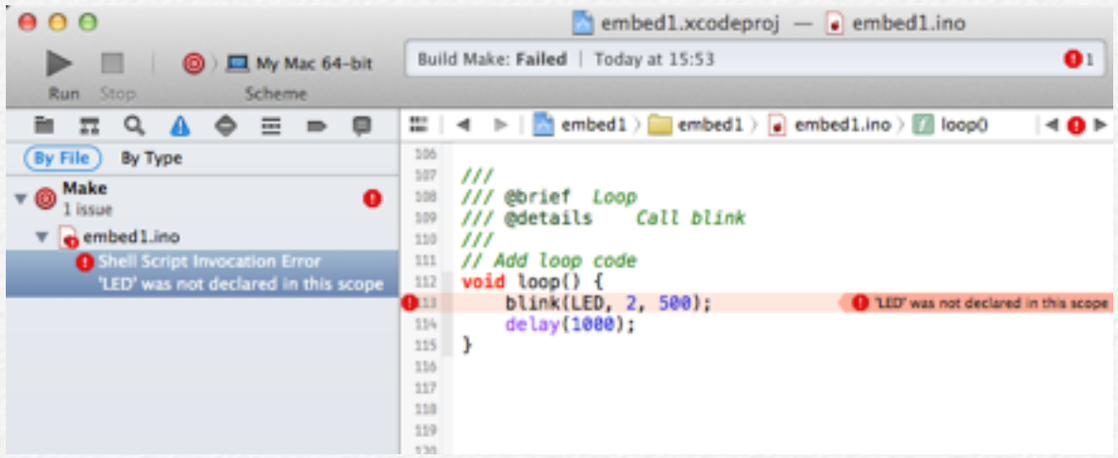


The **Issue navigator** provides a list of all the errors found during compilation.

- To display the **Issue navigator**, click on the icon of the middle:



- Click once on the message to display the line where the error is located.



The cursor points at `LED` line `113`: the correct variable name is `myLED`. The error is just a misspelling.

```
107  ///
108  /// @brief  Loop
109  /// @details    Call blink
110  ///
111  // Add loop code
112  void loop() {
113      blink(LED, 2, 500);          'LED' was not declared in this scope
114      delay(1000);
115  }
116
```

- Change `LED` for `myLED`. The line is coloured back.

```
107  ///
108  /// @brief  Loop
109  /// @details    Call blink
110  ///
111  // Add loop code
112  void loop() {
113      blink(myLED, 2, 500);
114      delay(1000);
115  }
116
```

This feature may require an optional manual setting as explained in Define the Directories for the Targets.

New Release Message

During the first compilation of the project, a dialog box may prompt if a new release of embedXcode is available.

The dialog box closes automatically after 5 seconds.

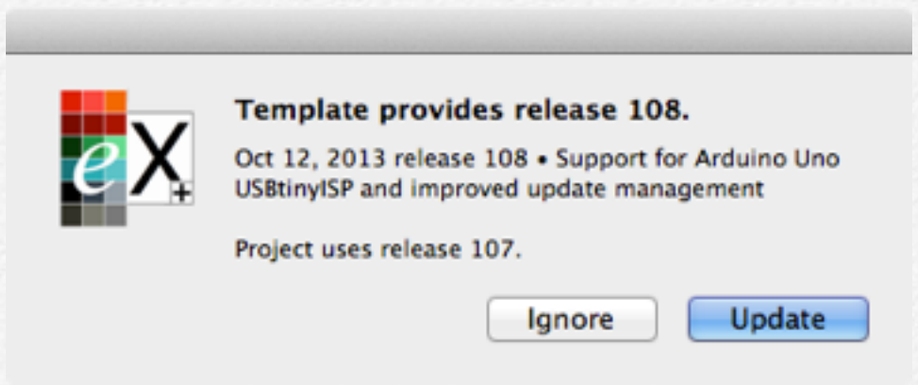
- Click on **Go to Download** to download the new release or **Ignore** to ignore it.

The new release of the template only applies for new projects. Existing projects are not updated with the new template.

Automatic Update Message

⊞ This section requires embedXcode+.

During the compilation of the project, a dialog box may prompt if the release of the template is more recent than the project.



- Click on **Update** to update the project or **Ignore** to ignore it.

The dialog box closes automatically after 5 seconds.

The automatic update requires the project to have been created with embedXcode release 108 or more recent. Projects created with an earlier release don’t feature the automatic update.

Debug the Project

5

⊞ For boards featuring a built-in debugger, define breakpoints and debug your project.

Check the Configuration

⊞ This section requires embedXcode+ and a board with a built-in hardware debugger.

Debugging requires boards with a built-in hardware **debugger** and has been tested successfully on the following boards:

- LaunchPads MSP430G2 and MSP430F5529,
- Experimenter Board MSP430FR5739,
- LaunchPad Stellaris LM4F120H5QR now Tiva C Series, with openOCD installed. For more information, please refer to Install Optional Debugging Tools.

Debugging involves two phases:

- Breakpoints with the associated conditions and actions are defined within the standard Xcode interface.
- The debugging session is performed in a `Terminal` window while the sketch is running on the board.

Contrary to some debuggers, it is not possible to change breakpoints interactively during a debugging session, unless you use the command line interface within the `Terminal` window.

Please note that, on the MSP430G2 LaunchPad, the debugging session interferes with the output to the serial console. On the MSP430F5529 and the Stellaris LaunchPads, the serial console is not affected by the debugging session.

Define the Break-Points

⊞ This section requires embedXcode+ and a board with a built-in hardware debugger.

Breakpoints are defined within the standard Xcode interface. Multiple options are possible including conditional breakpoints and associated actions.

To add a breakpoint,

```
114     for (i=1; i<10; i++) {
115         blink(myLED, 2, 1000/i);
116         delay(1000);
```

- Click once on the line number: a dark blue arrow appears over the line number and the breakpoint is defined and enabled.

```
114     for (i=1; i<10; i++) {
115         blink(myLED, 2, 1000/i);
116         delay(1000);
```

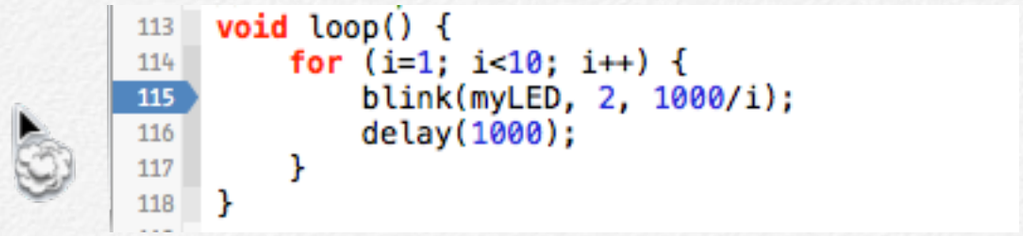
To disable a breakpoint,

- Click again on the line number: the arrow turns light blue and the breakpoint is disabled.

```
114     for (i=1; i<10; i++) {
115         blink(myLED, 2, 1000/i);
116         delay(1000);
```

To remove a breakpoint,

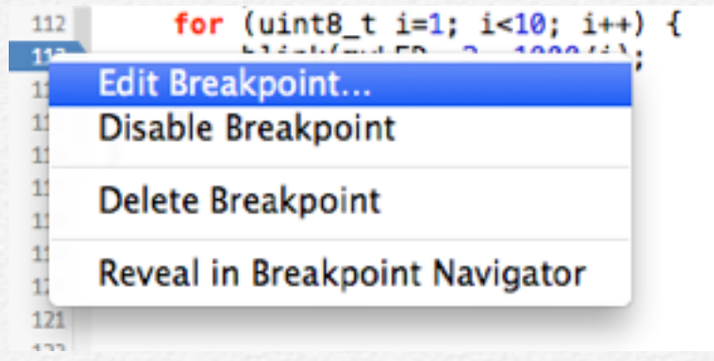
- Click again on the arrow and move it outside till a cloud appears. Then release the button; the breakpoint is removed.



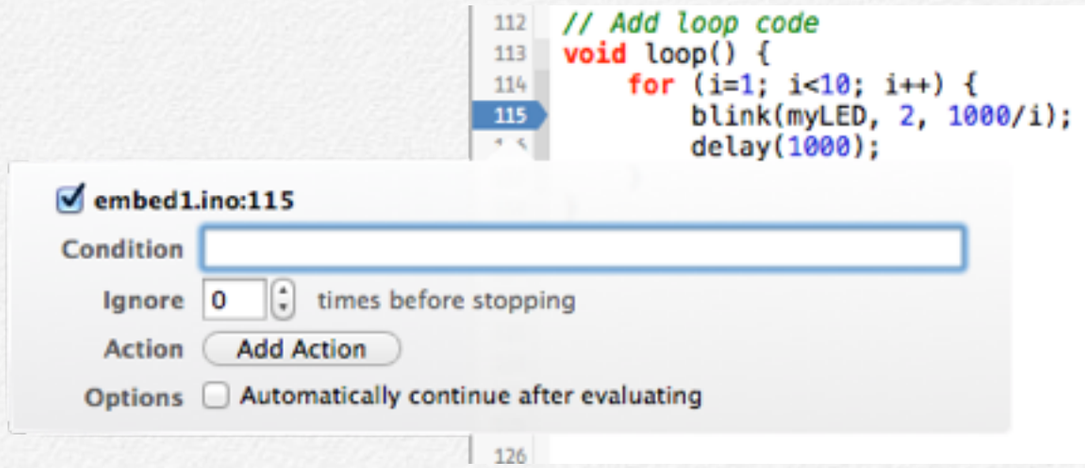
Edit a Breakpoint

To edit the breakpoint,

- Left-click on the line number and choose **Edit Breakpoint** on the menu.



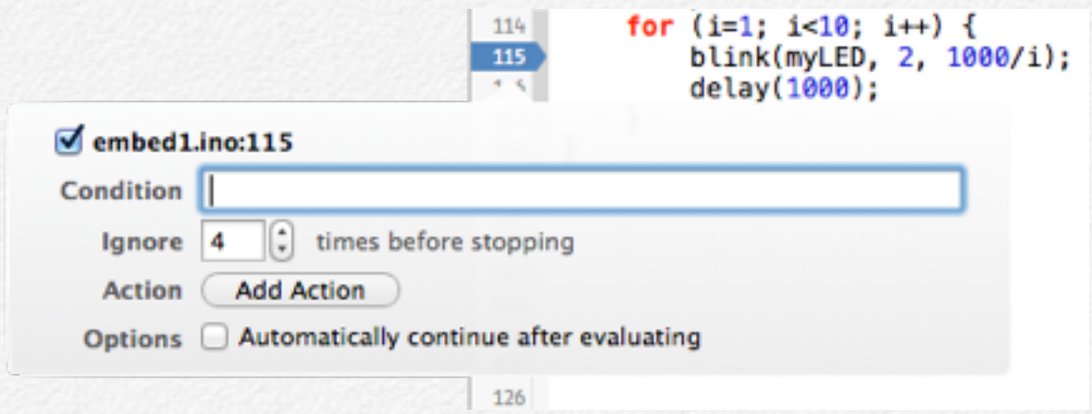
A window shows all the options for the breakpoint.



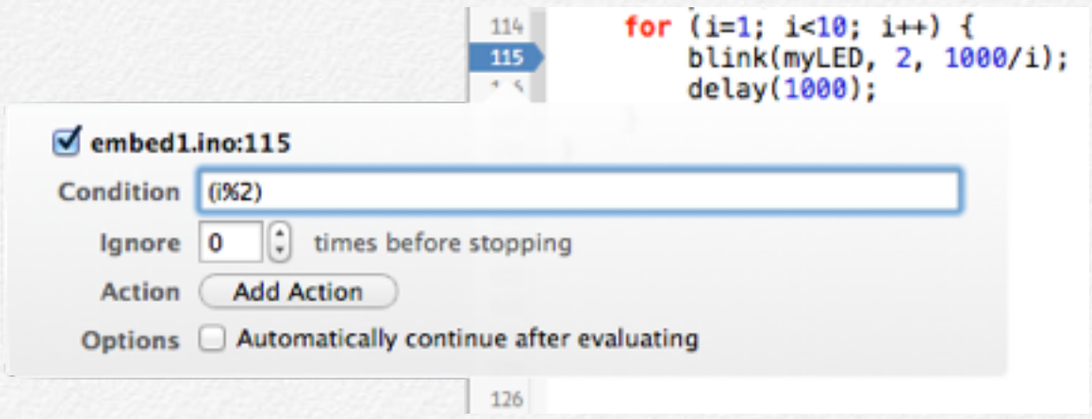
Define a Conditional Breakpoint

The breakpoints can stop the program only when conditions are met.

The breakpoint is ignored a certain number of times before stopping. In this example, the breakpoint is ignored `4` times before stopping.

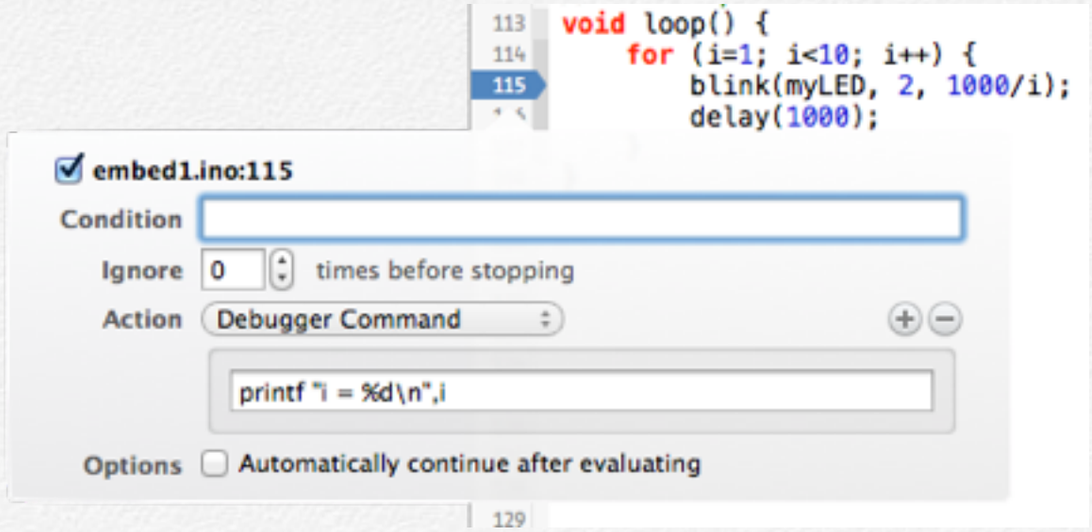


The breakpoint stops only if the `(i%2)` condition is met.

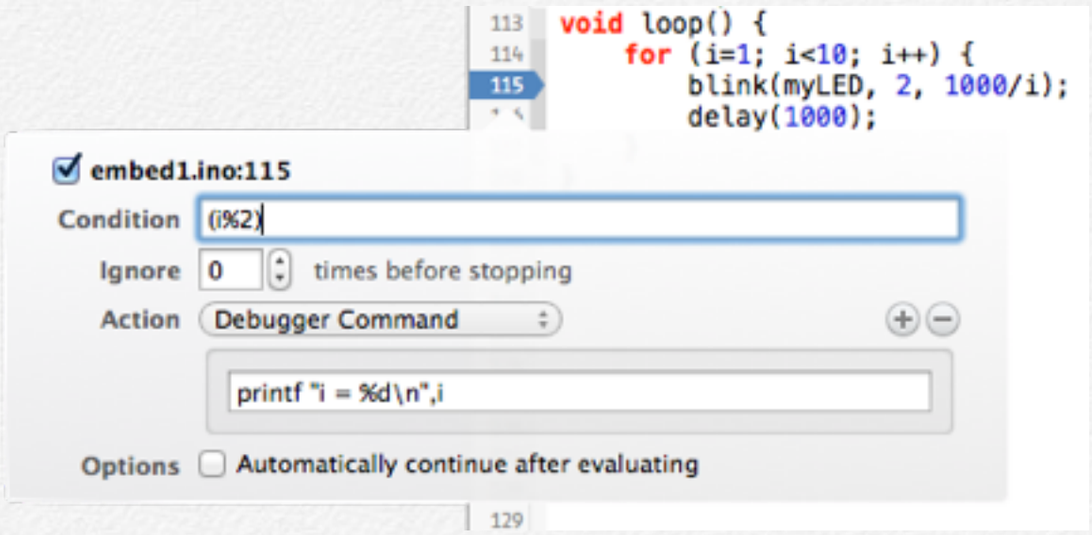


Associate an Action to a Breakpoint

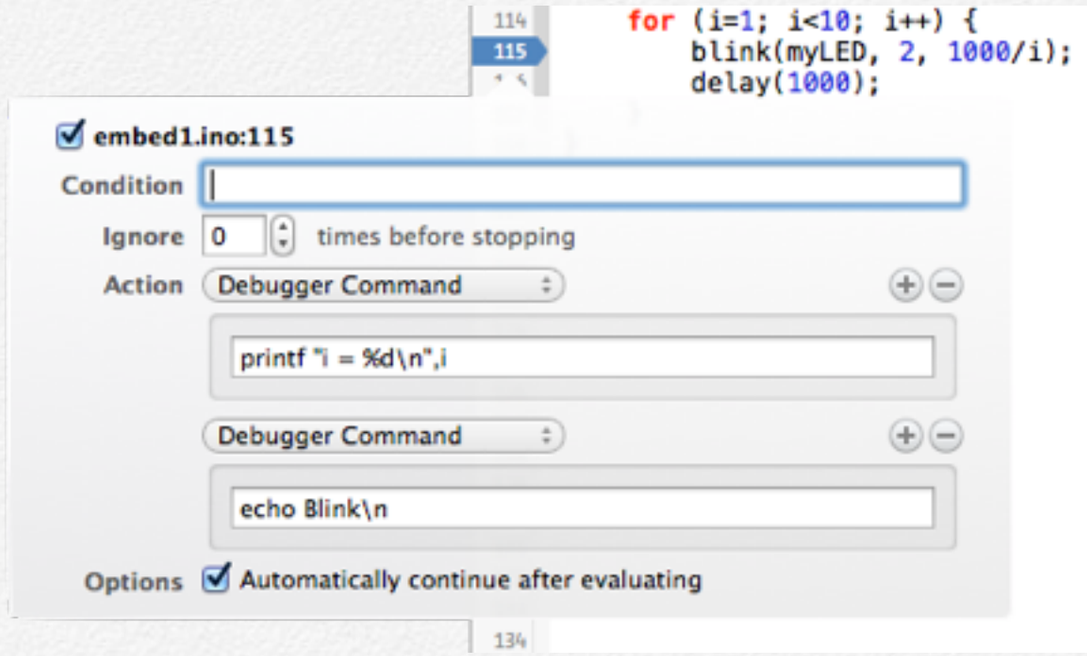
A breakpoint can perform actions. Always select **Debugger Command** as type of action. Here, the debugger prints the value of `i` as `i = 2`.



Actions and conditions can be used together.



Multiple actions can be performed. Here, the debugger prints the value of `i` as `i = 2` and displays the message `Blink`.

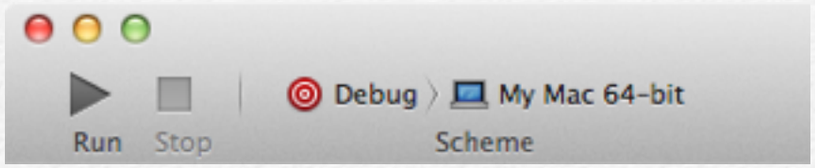


If **Automatically continue after evaluating** is checked, the action is performed but the program doesn't stop.

Use the Debugger

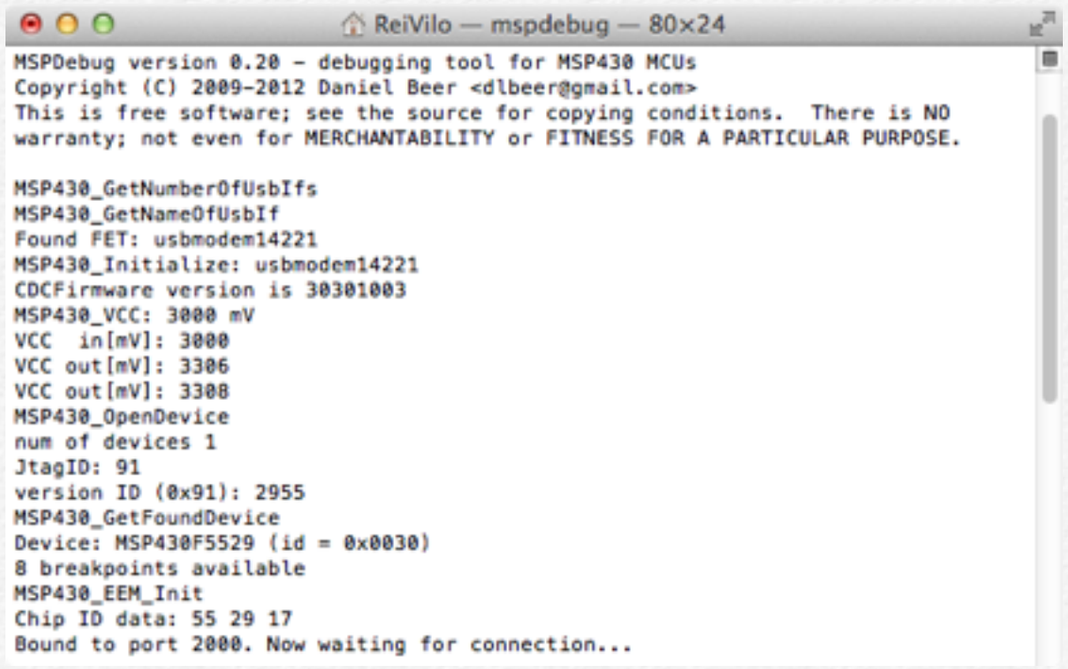
⊞ This section requires embedXcode+ and a board with a built-in hardware debugger.

To launch a debugging session, just select the **Debug** target and press **Run**.

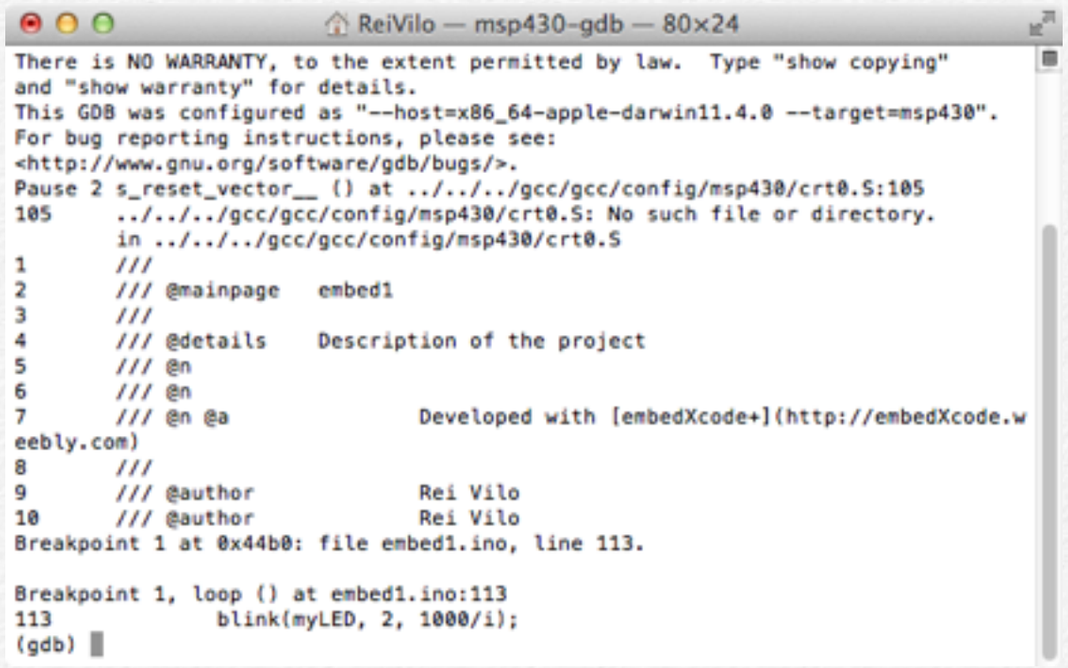


embedXcode compiles, builds and links the project, uploads it to the board, and then opens two `Terminal` windows.

- The first `Terminal` window hosts the server. The server connects to the board and answers the requests from the debugger.



- The second `Terminal` window is the active debugger. It downloads all the breakpoints defined previously and starts running the project.



Enter the commands after the `(gdb)` prompt and validate it by pressing the **Enter** ⏎ key.

```
(gdb)
```

When the debugger stops, type `continue` or simply `c` to continue till the next breakpoint:

```
(gdb) continue
(gdb) c
```

To run the sketch one step to the next line of code to be executed, including the sub-functions called by the current function, type `step` or simply `s`:

```
(gdb) step
(gdb) s
```

To run the sketch to the next line of the current function, without stopping at the sub-functions, type `next` or simply `n`:

```
(gdb) next
(gdb) n
```

To stop the sketch, press **Ctrl-C**.

To list the breakpoints defined and loaded, type `info break`:

```
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000044b2 in loop() at embed1.ino:115
        breakpoint already hit 2 times
        silent
        printf "i = %d\n",i
        echo Blink\n
2       breakpoint     keep y   0x00004468 in blink(uint8_t, uint8_t, uint16_t)
                                           at LocalLibrary.cpp:25
        breakpoint already hit 2 times
(gdb)
```

For more information about the GDB commands, please refer to the GDB Documentation ⓦ.

Change the Value of a Variable

To get the value of a variable, type the command `print` and the name of the variable:

```
(gdb) print i
$1 = 7 '\a'
```

To change the value of a variable, type first the command `set variable` or `set var`, then the name of the variable and the = sign, and finally the new value:

```
(gdb) set var i = 4
(gdb) print i
$2 = 4 '\004'
```

Display the Call Stack

To display the stack of the calls, type the command `backtrace` or `bt`:

```
(gdb) backtrace
#0  blink (pin=43 '+', times=2 '\002', ms=500) at
LocalLibrary.cpp:25
#1  0x000044c8 in loop () at embed1.ino:115
#2  0x0000444e in main () at main.cpp:164
```

The `blink()` function is called by the function `loop()`, and `loop()` is called by the main function `main()`.

The backtrace command also provides the details of the parameters and values passed by `loop()` on to `blink()`.

Here, values are `pin=43`, `times=2`, `ms=500`.

```
(gdb) print RED_LED
$1 = 43 '+'
```

Value `43` does correspond to constant `RED_LED`.

End and Quit a Debugging Session

To end and quit the debugging session, type `quit` or simply `q` and confirm by `y`:

```
(gdb) quit
(gdb) q
A debugging session is active.

    Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y
```

Always quit and close the two `Terminal` windows: the active debugging session and the server session.

Self-Document the Project

⊞ In order to obtain documentation,

- Add specific comments with defined keywords to the code,
- Select the output formats,
- Build the documentation with the target **Document**,
- Use Quick Help to access the documentation.

This chapter requires embedXcode+.

Comment the Code

⊞ This section requires embedXcode+.

First step consists on adding specific comments with defined keywords right into the code.

Comments for self-documentation start with `///` instead of the standard `//` and include keywords with a `@` prefix.

```
///
/// @mainpage   embed1
/// @details    Main sketch
/// @n
```

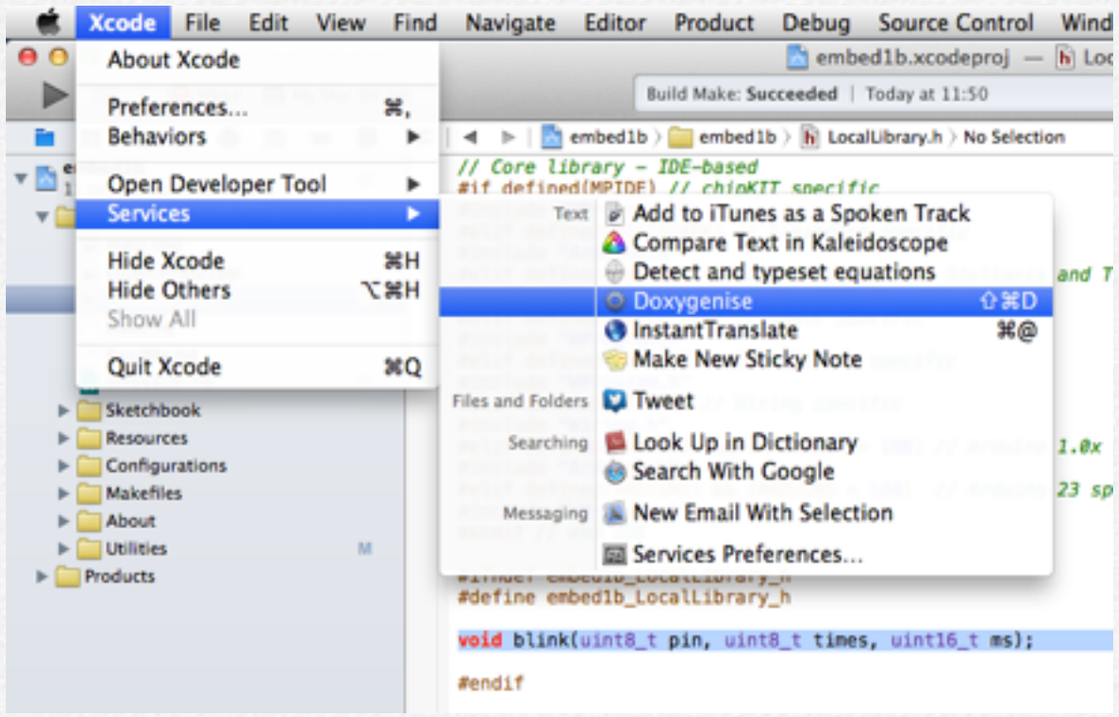
This means that standard comments starting with the standard `//` aren't included in the documentation.

Use the Doxygen Helper to speed up and ease the writing of comments for the functions.

- Just select the code of a function and

```
62
63  void blink(uint8_t pin, uint8_t times, uint16_t ms);
64
```

- press ⌘⇧**D cmd-shift-D**,
- or call the menu **Xcode > Services > Doxygenise**.



The helper generates a template for the comment lines.

```
///
/// @brief      Description
/// @param      pin pin description
/// @param      times times description
/// @param      ms ms description
///
void blink(uint8_t pin, uint8_t times, uint16_t ms);
```

- Use the **tab** key to replace the light-blue fields with the comments.

In this example, the comment includes the `@brief` description of the function, list all the `@parameters` as well as the `@returned` value.

Here's the sub-lists of the keywords I use the most:

- For the main page with details about the author, copyright, license, references and links.

```
///
/// @mainpage   embed1
/// @details    details
/// @n
/// @n
/// @n @a       Developed with [embedXcode](http://embedXcode.weebly.com)
///
/// @author     Rei VILO
/// @author     http://embeddedcomputing.weebly.com
/// @date       17/07/12 20:37
/// @version    version
///
/// @copyright  © Rei VILO, 2012
/// @copyright  CC = BY NC SA
///
/// @see        ReadMe.txt for references
///
```

Note the `@mainpage` keyword.

- For a file with details about the author, copyright, license, references and links.

```
///
/// @file       embed1.pde
/// @brief      Main sketch
/// @details    details
/// @n @a       Developed with [embedXcode](http://embedXcode.weebly.com)
///
/// @author     Rei VILO
/// @author     http://embeddedcomputing.weebly.com
/// @date       17/07/12 20:37
/// @version    version
///
/// @copyright  © Rei VILO, 2012
/// @copyright  CC = BY NC SA
///
/// @see        ReadMe.txt for references
/// @n
///
```

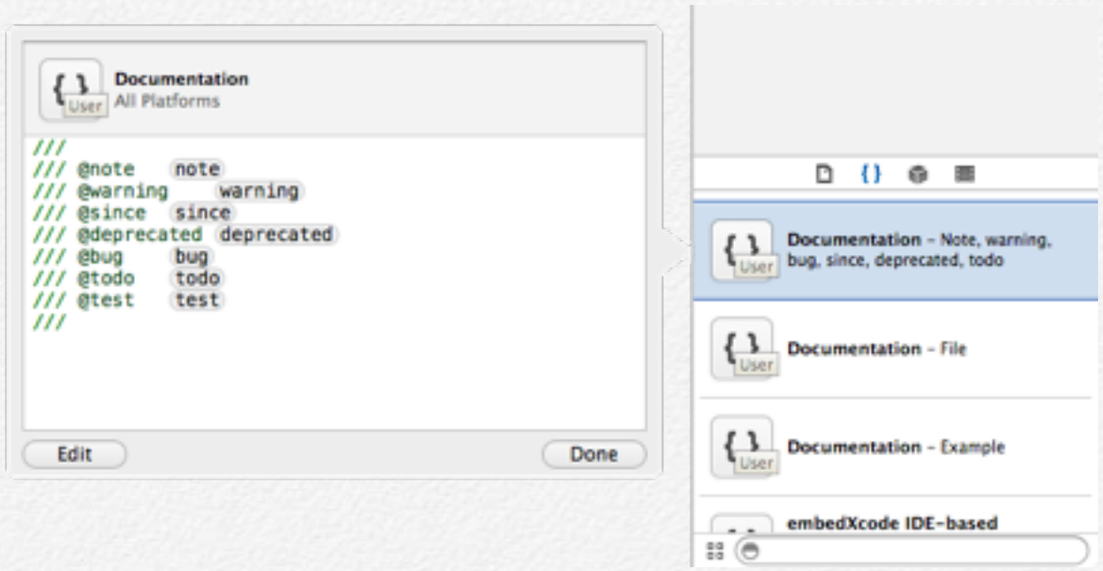
By default, the projects and files templates include self-documenting headers.

- For a function with details for parameters.

```
///
/// @brief      Description
/// @param      a a description
/// @return     return value description
///
uint8_t twice(uint8_t a) {
  return 2*a;
}
```

A result is documented with the keyword `@return`.

- The snippets for the documentation are under the **User** list.



The snippet for different details provides the keywords for note, warning, bug, to-do, test, ...

```
///
/// @note        note
/// @warning     warning
/// @since       since
/// @deprecated  deprecated
/// @bug         bug
/// @todo        todo
/// @test        test
///
```

The snippet for code allows to include an example of code.

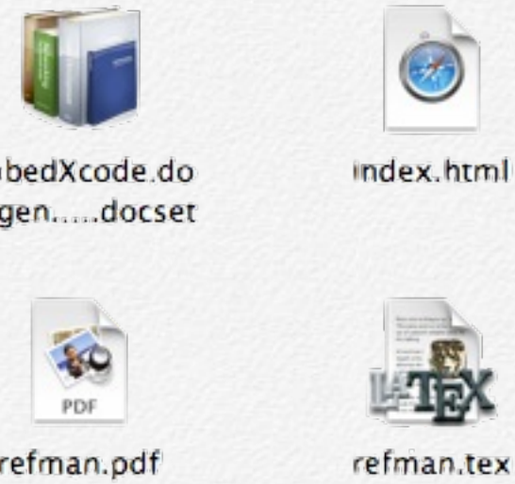
```
///
/// @code {.cpp}
///             code
/// @endcode
///
```

Doxygen includes many more options. Please refer to its documentation ⓦ.

Select the Output Formats

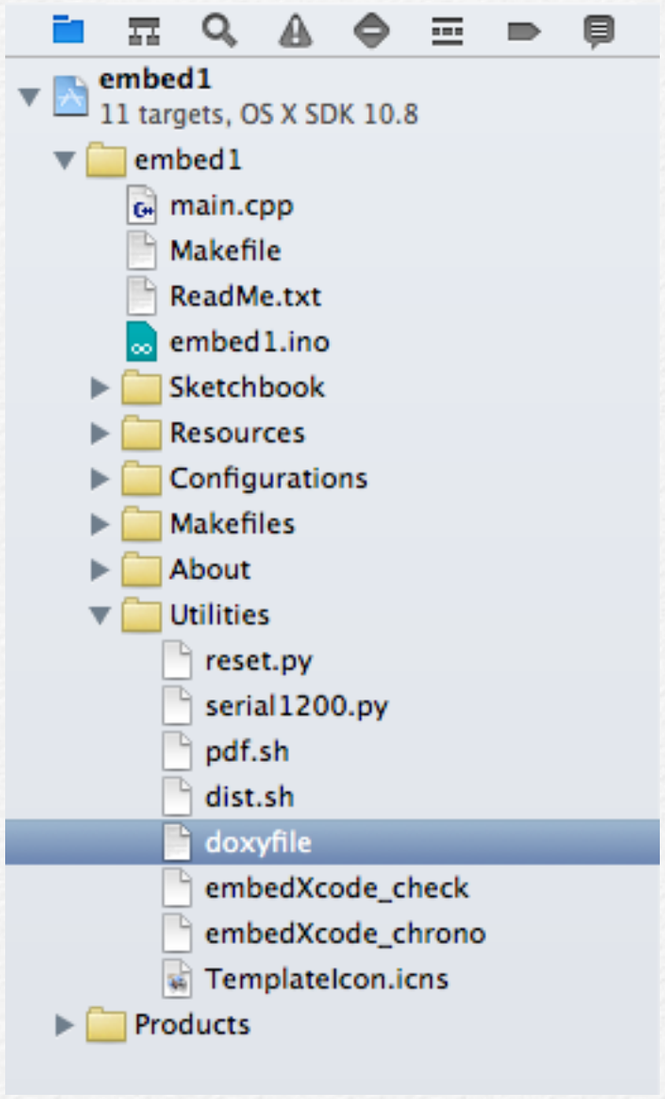
⊞ This section requires embedXcode+.

By default, embedXcode generates the documentation under four different formats:



- A LateX `.tex` document;
- A PDF document `.pdf` file, based on the LatexX document. The LateX and PDF options are linked;
- An HTML-formatted documentation, with `index.html` as entry;
- A help file for Xcode called documentation set or `.docset`. The documentation set is incorporated in the Xcode help library. This option requires the HTML-formatted documentation.

The parameters for the documentation outputs are defined in the `doxyfile` file, located under **Utilities**.



Here are the main settings for selecting the outputs:

```
GENERATE_LATEX          = YES
GENERATE_HTML           = YES
GENERATE_DOCSET         = YES
```

- `GENERATE_LATEX` prepares the LateX `.tex` and PDF document `.pdf` files;
- `GENERATE_HTML` prepares an HTML-formatted documentation;
- `GENERATE_DOCSET` creates the documentation set `.docset` file and incorporates it in the Xcode help library. This option requires `GENERATE_HTML`.

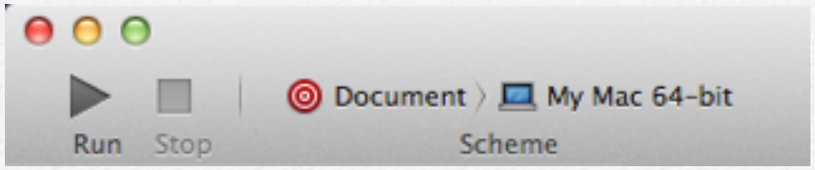
There are many more options available.

For more information about the Doxygen options, please refer to the Doxygen Manual ⓦ.

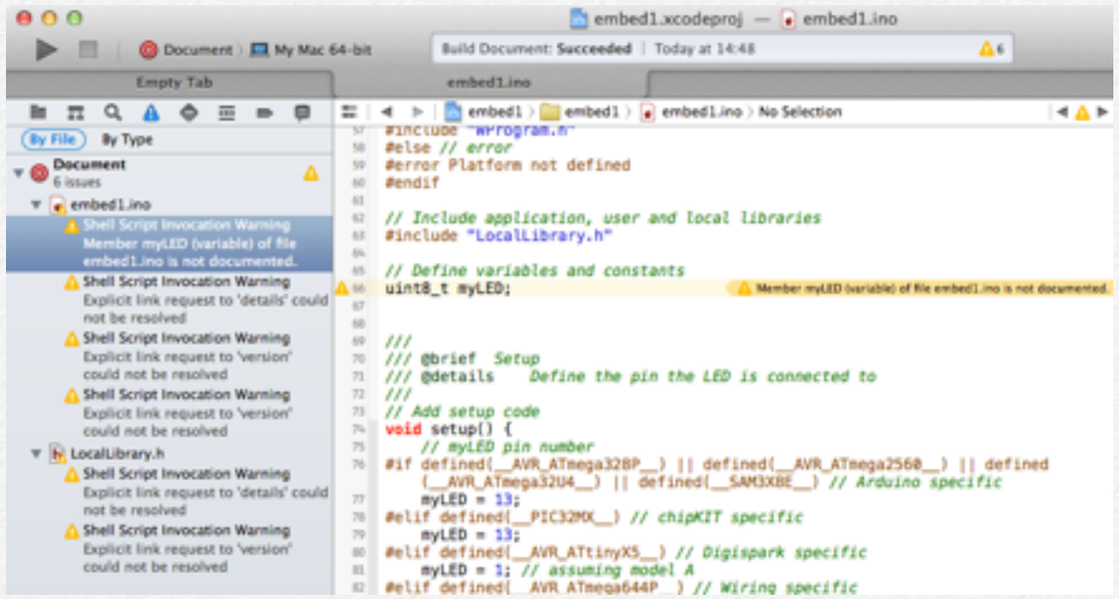
Build the Documentation

⊞ This section requires embedXcode+.

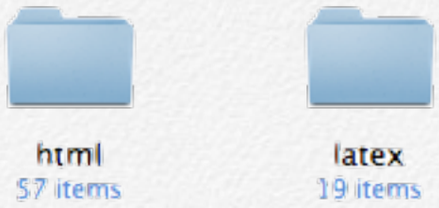
To build the documentation, just select the **Document** target and press **Run**.



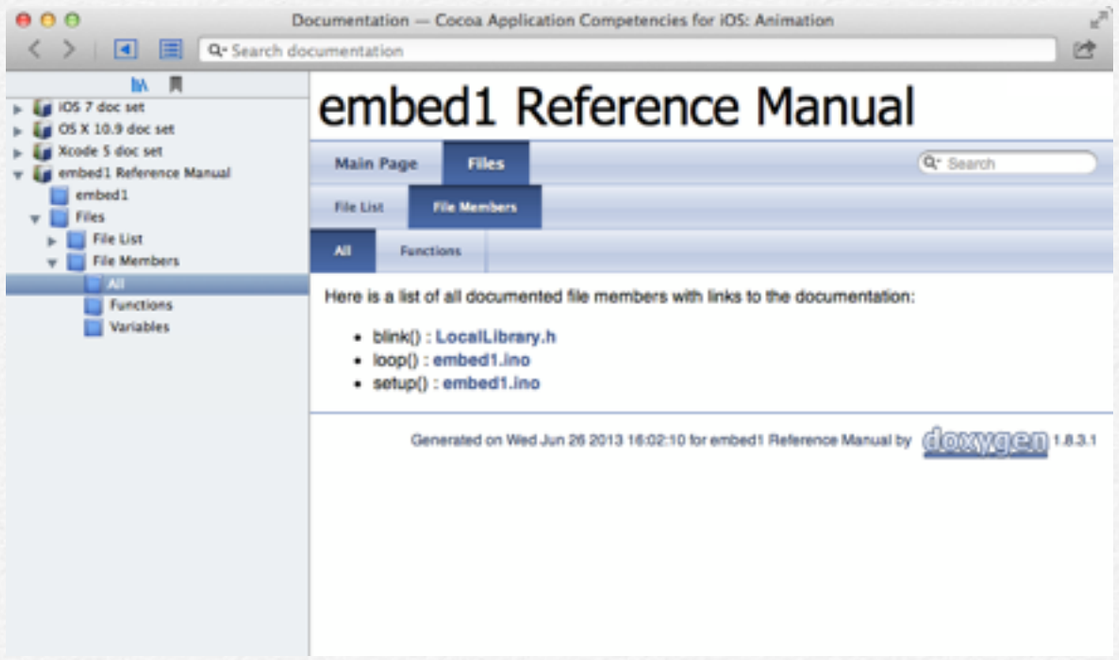
Doxygen builds the documentation and issues warnings for undocumented portion of your code.



The HMTL and `.docset` files are located under the `Utilities/html` folder, while the LateX and PDF files are under the `Utilities/latex` folder.

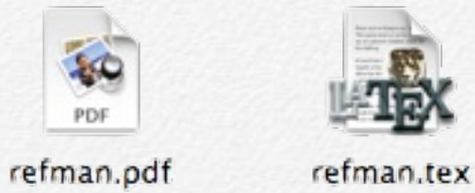


The documentation is packed in a specific file called `.docset` for documentation set, and added into Xcode documentation and API reference.



A PDF file is also automatically generated. It is located under the `Utilities/latex` folder.

To convert the TEX file into a PDF file, you need a LateX to PDF converter. I've chosen TeXShop because it's easy to use.



To create the PDF file manually, double-click on a `.tex` file to launch TexShop and prepare the `.pdf` file.

Use the Documentation

⊞ This section requires embedXcode+.

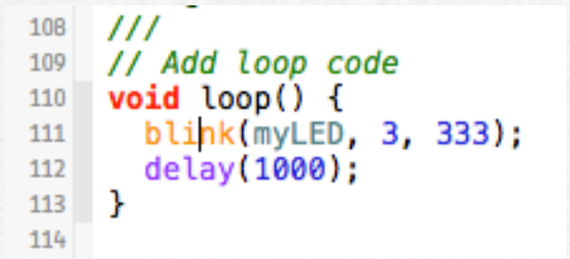
Once generated and if the `.docset` option has been selected, the documentation is now fully integrated into Xcode and available through multiple ways:

- directly from the code thanks to **Quick help**.
- or using the **Documentation and API Reference** window.

Quick Help

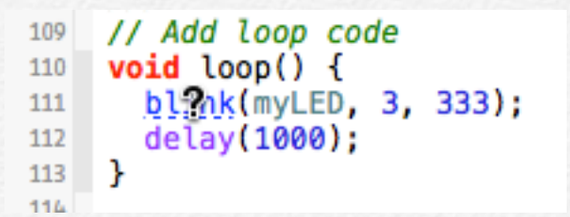
Documentation is available directly from the code thanks to Quick help.

- Select a function or place the cursor on a function, here `blink`.



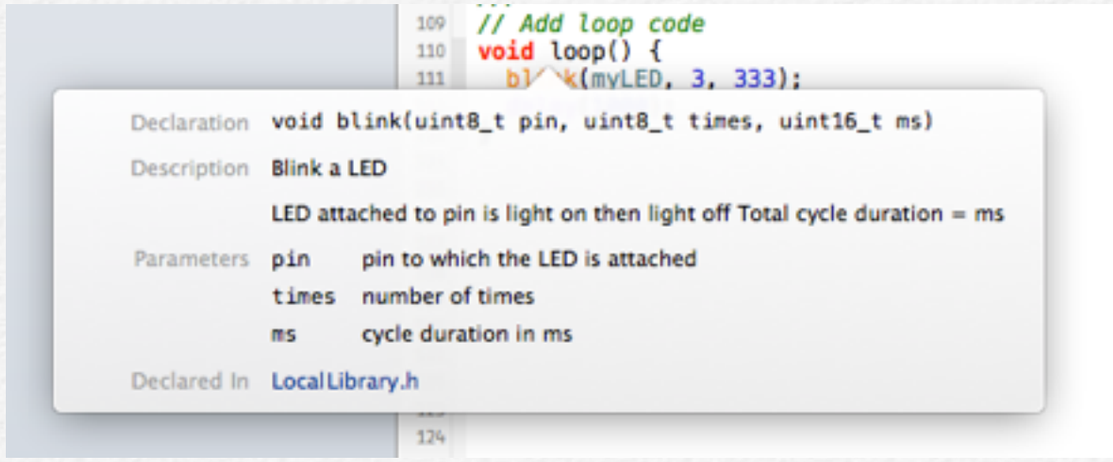
- Press ⌥ **alt** while the cursor is hovering the function.

  The cursor turns into an interrogation mark:

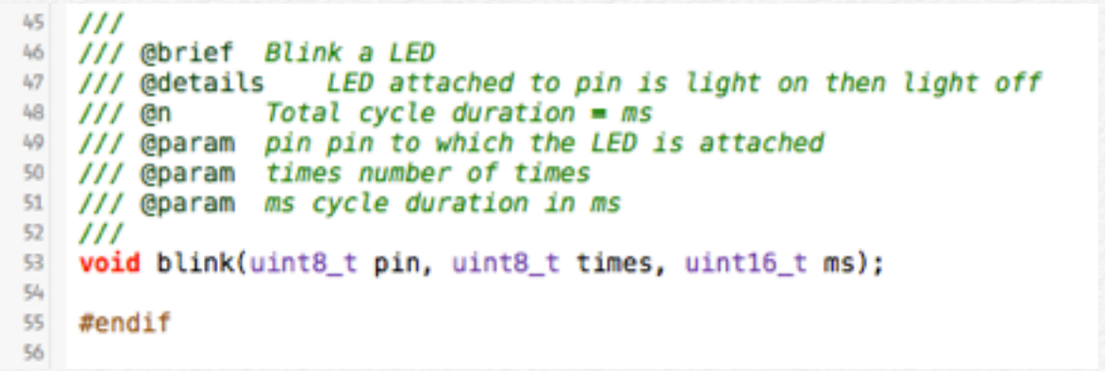


- While pressing ⌥ **alt**, click on the name of the function `blink`.

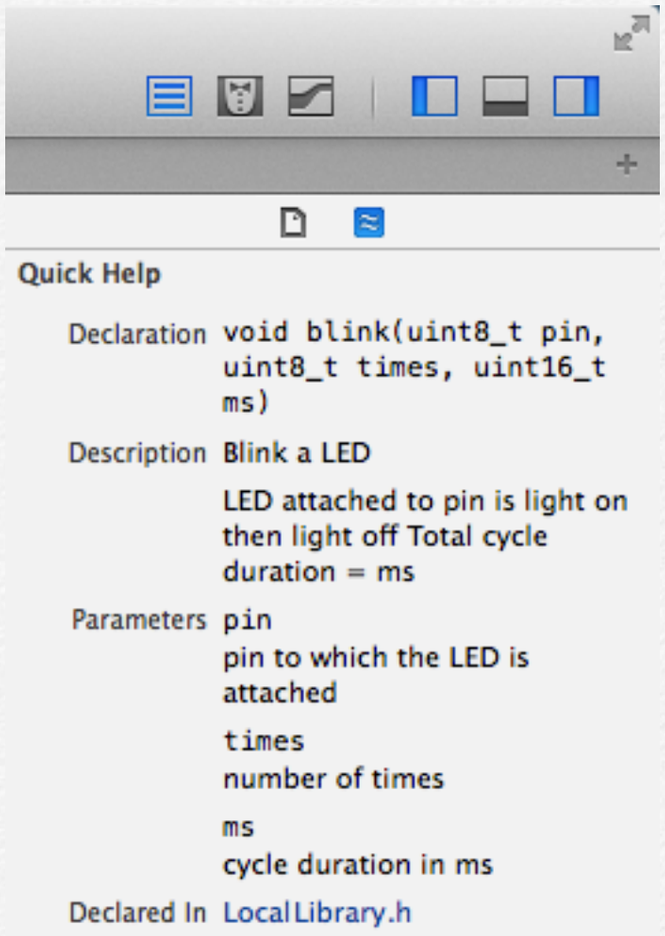
A contextual help pops-up.



If you click on the blue link **LocalLibrary.h** in the help balloon, Xcode opens the library at the definition of the function:



When the cursor selects a function, the **Quick Help** pane on the right provides all the information available for the function, here `blink`.



If you click on the blue link **LocalLibrary.h** in the quick help pane, Xcode opens the library at the definition of the function:

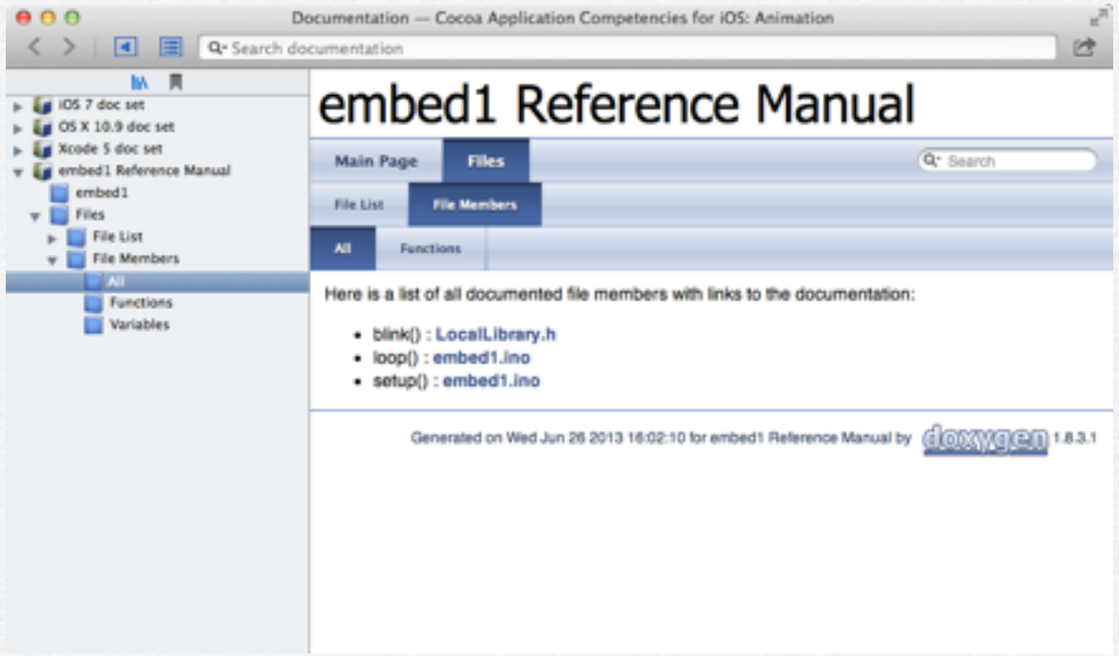
```
45  ///
46  /// @brief  Blink a LED
47  /// @details    LED attached to pin is light on then light off
48  /// @n      Total cycle duration = ms
49  /// @param  pin pin to which the LED is attached
50  /// @param  times number of times
51  /// @param  ms cycle duration in ms
52  ///
53  void blink(uint8_t pin, uint8_t times, uint16_t ms);
54
55  #endif
56
```

Documentation and API Reference

The documentation is also available on the **Documentation and API Reference**.

- Call the menu **Help > Documentation and API Reference**.
- The **Documentation** window opens.

On the column on the left, select the documentation, here **embed1 Reference Manual** and navigates using the sections and the hyperlinks.

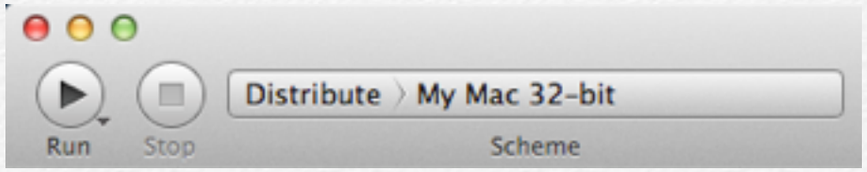


Distribute the Project

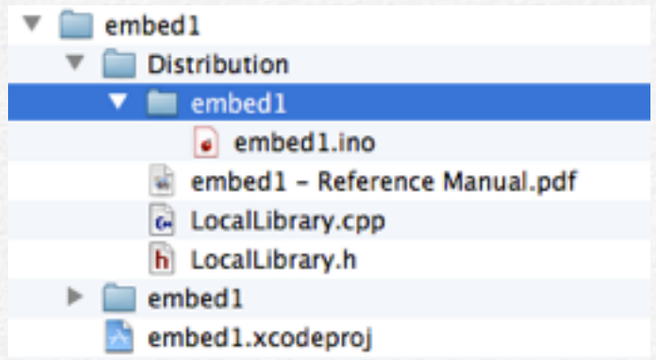
⊞ This section requires embedXcode+.

Once the documentation has been generated, the project can be distributed.

To distribute the project, just select the **Distribute** target and press **Run**.

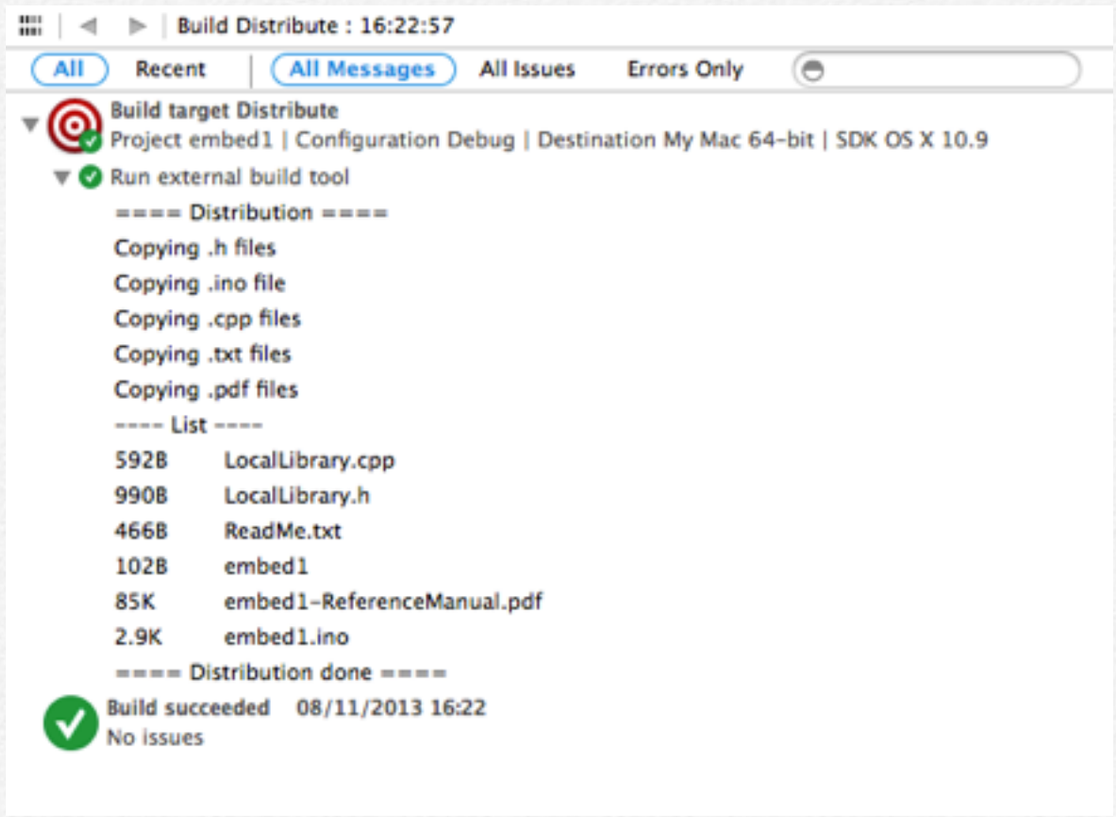


A new folder `Distribution` is created and contains:



- all the headers and code files,
- documentation if a PDF file if available,
- and main sketch —`.pde` or `.ino`— in a specific folder with the same name.

The **Messages** window provides the list of the files copied to the `Distribution` folder.



Find Solutions to Issues

7

Learn more about embedXcode compatibility with Arduino, use embedXcode specific pre-processing variable and find solutions to most common issues.

Compatibility

Projects developed by embedXcode are highly compatible with the Arduino IDE and alike.

However, unlike Arduino, code with embedXcode is true C++.

The main consequence is the need for declaring prototypes of the functions in the main sketch.

In the example provided below, the prototype for `functionB()` is required, as `functionB()` is called by `functionA()` but defined after.

```
// Prototypes
void functionA();
void functionB();

// Functions
void functionA() {
    Serial.println("functionA");
    functionB();
```

```
void functionB() {
    Serial.println("functionB");
}
```

Without prototyping `functionB()`, compilation would raise an error.

Prototypes aren't required for libraries as they are already included in the header file.

As for Arduino, every single header file for a library shall mention an `#include` statement to the core library.

```
// Core library - IDE-based
#if defined(WIRING) // Wiring specific
#include "Wiring.h"
#elif defined(MAPLE_IDE) // Maple specific
#include "WProgram.h"
#elif defined(DIGISPARK) // Digispark specific
#include "Arduino.h"
#elif defined(MPIDE) // chipKIT specific
#include "WProgram.h"
#elif defined(ENERGIA) // LaunchPad specific
#include "Energia.h"
#elif defined(ARDUINO) && (ARDUINO >= 100) //
Arduino 1.0 specific
#include "Arduino.h"
#elif defined(CORE_TEENSY) // Teensy specific
#include "Arduino.h"
#elif defined(ARDUINO) && (ARDUINO < 100) //
Arduino 23 specific
#include "WProgram.h"
#else // error
#error Platform not defined
#endif
```

An easy solution is to Insert #include Statements From Code Snippet, especially when developing code for multiple MCUs.

For more on library development, please refer to Writing a Library for Arduino ⓦ.

Arduino requires naming all the libraries in the main sketch, while embedXcode allows naming only the libraries which are directly called by the main sketch.

Naming all the libraries in the main sketch doesn't affect embedXcode.

embedXcode Pre-Processing Variable

A project using embedXcode declares the pre-processing variable `EMBEDXCODE`, with the release number as value.

```
EMBEDXCODE = 105
```

The variable and the value are passed on to the compiler and the linker as a `-D` variable.

```
-DEMBEDXCODE=105
```

This allows you to manage conditional pre-processing statements as `#define` and `#include` based on the IDE you use, either Xcode or one of the large Processing-based Wiring-derived Arduino-like IDEs family.

The `EMBEDXCODE` variable is already used in the `main.cpp` code file so it is only considered when compiled by Xcode, and ignored by the other IDEs.

Solutions to Most Common Issues

I've listed the most common causes of errors and the most frequently asked questions.

Before reporting an issue, there are basic checks to be performed.

- Saying that compilation fails is not enough: more information is needed.
- If the sketch doesn't compile successfully with the standard IDE, the error isn't generated by Xcode and embedXcode alone.

Please check this list before reporting the issue.

Symptom	Project preparation fails.
Solution	Was the first build launched while indexing was running?
Reference	See Automatic Procedure and Launch the Procedure

Symptom	Project preparation fails.
Solution	Are the IDEs of the boards installed under the **/Applications** folder?
Reference	See Install the IDEs of the Boards

Symptom	Compilation fails.
Solution	By default, no library is compiled. if the project uses libraries, mention them in the main makefile after the `APP_LIBS_LIST` and `USER_LIBS_LIST` variables.
Reference	See Manage the Libraries for Compilation

Symptom	Compilation fails.
Solution	Some user libraries may be incompatible. Exclude the conflicting libraries.
Reference	See Manage the Libraries for Compilation

Symptom	Compilation fails.
Solution	More information is needed.
Reference	See Display the Log Navigator

Symptom	Compilation fails.
Solution	Does the sketch compiles normally on the standard IDE?
Reference	If the sketch doesn't compile on the standard IDE, then the error isn't generated by Xcode or embedXcode alone.

Symptom	Compilation fails.
Solution	What is the selected target? The targets **All**, **Upload** and ⊞ **Fast** requires a board to be connected.
Reference	See Select a Target

Symptom	Compilation fails.
Solution	What are the messages in the Log navigator?
Reference	See Error Messages

Symptom	Compilation fails.
Solution	Compilation of an embedXcode project fails with a Processing-based Wiring-derived Arduino-like IDE. Use the `EMBEDXCODE` pre-processing variable.
Reference	See embedXcode Pre-Processing Variable

Symptom	Code-sense doesn't work.
Solution	Re-index the keywords for code-sense.
Reference	See Re-Index the Keywords for Code-Sense

Symptom	Code-sense doesn't include all the libraries.
Solution	Re-index the keywords for code-sense.
Reference	See Declare Sources for Code-Sense

Symptom	Xcode can't find the libraries.
Solution	Is the sketchbook directory defined?
Reference	See Install the IDEs of the Boards

Symptom	Xcode can't find the libraries.
Solution	Is the library folder created inside the sketchbook folder?
Reference	See Add User's Libraries

Symptom	**Xcode can’t find the libraries.**
Solution	Is the library mentioned in the main makefile? Or is the library excluded, as `Wifi` and `ArduinoRobot`?
Reference	See Manage the Libraries for Compilation

Symptom	**The main sketch with extension** .pde **or** .ino **isn’t considered as C++ code.**
Solution	Declare .pde or .ino main sketch as C++ file.
Reference	See Declare Sketch .pde or .ino File as C++ File

Symptom	**Upload fails.**
Solution	Check the USB connection and the drivers.

Symptom	**Upload fails.**
Solution	The Digispark and Teensy boards have specific upload procedures. Some LaunchPads MSP430F5529 may require a firmware update.
Reference	See Digispark Uploading Specific Procedure, Teensy 3.0 Uploading Specific Procedure or LaunchPad MSP430F5529 Upload Specific Procedure.

Symptom	**Upload fails after having used a programmer.**
Solution	The programmer has erased the boot-loader. You need to burn the boot-loader again.
Reference	See Using Standard USB Upload After a Programmer

Symptom	**Click-to-error doesn’t point to the right line of code.**
Solution	Define the directories for the targets.
Reference	See Define the Directories for the Targets

Symptom	**I want the same code for different platforms.**
Solution	Use the conditional statements based on MCU or on IDE.
Reference	See Write Specific Code for Multiple Platforms

Symptom	**My board isn't listed.**
Solution	Create a specific ⊞ Board Configuration Settings File for your board or contact me.
Reference	See ⊞ Add a Configuration File for a New Board

Symptom	**Compilation fails with Maple boards with the message** `'uint8_t' was not declared in this scope.`
Solution	Maple ignores the C99 standard `[u]int{8|16|32|64}_t` types. Refer to the procedure to add them.
Reference	See Maple Boards

Symptom	**The documentation elaborated by Doxygen isn't correct.**
Solution	Refer to the Doxygen Manual on the Doxygen website.
Reference	See ⊞ Comment the Code

Symptom	**I need more information about Xcode.**
Solution	Buy a book about Xcode.
Reference	See Install Xcode

Symptom	**I need more information about Arduino.**
Solution	Buy a book about Arduino.
Reference	See Install the IDEs of the Boards

Symptom	**My issue is not listed.**
Solution	Post on a forum or contact me.
Reference	See Forums, Links and Official Websites

Appendixes

This section includes additional information about embedXcode as well as copyright and license.

Editions

embedXcode comes in two different editions:

-  **embedXcode** with core features, and
-  **embedXcode+** with extended functionalities.

Features specific to embedXcode are denoted by ⊞.

		embedXcode	embedXcode+
Supported Platforms	Arduino	■	■
	chipKIT	■	■
	Digispark	■	■
	Energia	■	■
	Maple	■	■
	Microduino	■	■
	Teensy	■	■
	Wiring	■	■

		embedXcode	embedXcode+
Boards	Configured boards	29	34 ⊞
	Select among two connected boards		⊞
	Template for new board		⊞
	WiFi, USBtinyISP and USB ASP programmers		3 ⊞

		embedXcode	embedXcode+
IDE	Auto-completion	■	■
	Code sense	■	■

		embedXcode	embedXcode+
Project	Single platform	■	■
	Multiple platforms		⊞
	Comment field	■	■
	License field		⊞
	Code snippets		⊞

		embedXcode	embedXcode+
Documentation	User Manual	■	⊞
	On-line support		⊞

		embedXcode	embedXcode+
Targets	All, Build	■	■
	Serial, Upload	■	■
	Fast, Make		⊞
	Document, Distribute		⊞
	External debugger		⊞

		embedXcode	embedXcode+
References	…to core libraries	■	■
	…to application libraries	■	■
	…to user libraries	■	■
	Unlimited number of references		⊞
	List of libraries in main makefile		⊞

		embedXcode	embedXcode+
Project Self-Documentation	Comment snippets		⊞
	PDF		⊞
	HTML		⊞
	Xcode help		⊞

As per release 127

Release History

Date	Release	Comment
Jan 22, 2012	a	Initial release for Xcode project
Feb 02, 2012	b	chipKIT operational
Feb 04, 2012	c	Code-sense operational
Feb 06, 2012	d	User libraries
Feb 06, 2012	e	Code checking while typing with Index as target
Feb 08, 2012	f	Code-sense, click-to-error with standard targets
Feb 14, 2012	g	Initial release for Xcode template
Feb 14, 2012	1	Multi-application, check-as-you-type, template
Feb 18, 2012	2	Improvements
Feb 26, 2012	3	Modular makefiles
Mar 06, 2012	4	Arduino 1.0 implementation
Mar 12, 2012	5	Bugs fixed
Mar 15, 2012	6	All Arduino/chipKIT/Wiring/Energia and user libraries included by default
Apr 05, 2012	7	Wiring 1.0 and Energia 1.0 implementations
Apr 08, 2012	8	Bugs fixed
Apr 16, 2012	9	Code-sense reference defined by selected board
May 23, 2012	10	Leaflabs Maple implementation
Jun 03, 2012	11	MCU- or IDE-based platform identification for #include library
Jun 05, 2012	12	IDE-based identification for all

Roadmap

Features Roadmap

Apart from maintenance, the following features are under consideration:

- Integrate the `.html` help files from the installed boards IDEs, as Arduino, chipKIT MPIDE, Energia, Maple, Teensy and Wiring.
- Follow the new specifications for the Arduino 1.5 release.
- Integrate the serial console and the debug session inside Xcode and not on a separate `Terminal` windows.
- Instead of a makefile-based project and template, use the tool-chains directly, as Xcode 5 includes a `/Application/Xcode.app/Contents/Developer/Toolchain` folder.

To be implemented, most of those features require help from Xcode experts and documentation from Apple.

Already delivered:

- Enrich the automatic project configuration with more features.
- Release the documentation as an electronic book.
- Provide some sort of debugging and make link with `gdb`.

Obsolescence Notice

Because the tools evolve, some features are becoming obsolete.

- Support for Arduino 0023 is discontinued.

Arduino 1.0 is mainstream now and new version 1.5 is under development.

- Support for chipKIT, Maple and Wiring boards is put on hold.

Those three platforms haven't been updated for a long time.

Maple still lacks very basic but critical libraries such as `stream` and `strings`.

Wiring is working on a new release, called Wiring++, but with no specific release date.

Despite new boards from chipKIT, MPIDE still relies on the old Arduino 0023 framework. The MPIDE team is working on a new release based on the Arduino 1.0 or 1.5 framework.

Support for those boards will resume when updated IDEs are available.

- embedXcode for Xcode 4 on OS X 10.7 *Lion* or OS X 10.8 *Mountain Lion* is no longer maintained.

OS X 10.9 *Mavericks* is a free update from OS X 10.7 *Lion* and OS X 10.8 *Mountain Lion*. A specific edition of embedXcode Legacy is still available for Xcode 4 on OS X 10.7 *Lion* or OS X 10.8 *Mountain Lion*.

Links, Forums and Official Websites

Links

Here are the links for embedXcode.

embedXcode	
Website	http://embedXcode.weebly.com ⓦ
Tutorial	http://embedxcode.weebly.com/tutorial ⓦ
Download	http://embedxcode.weebly.com/download ⓦ
Contact	http://embedxcode.weebly.com/contact ⓦ
RSS feed	feed://embedxcode.weebly.com/1/feed ⓦ

Forums

A dedicated forum for embedXcode is available for each platform.

Platform or IDE	Forum
Arduino	http://forum.arduino.cc/index.php/topic,49956.html ⓦ
chipKIT MPIDE	http://www.chipkit.org/forum/viewtopic.php?f=6&t=796 ⓦ
Digispark	http://digistump.com/board/index.php/topic,539.0.html ⓦ
Leaflabs Maple	http://forums.leaflabs.com/topic.php?id=1593 ⓦ
Microduino	
Energia for MSP430	http://forum.43oh.com/topic/2042-embedxcode ⓦ
Energia for Stellaris	http://forum.stellarisiti.com/topic/314-embedxcode-stellaris-on-xcode ⓦ
Teensyduino	http://forum.pjrc.com/threads/169-embedXcode-Teensy-3-0-on-Xcode ⓦ
Wiring	http://forum.wiring.co/index.php?topic=82.0 ⓦ

Official Websites

For each IDE or platform, the name of the board and the links for the website, the download, the wiki and the forum, are provided.

	Arduino
IDE	Arduino
Website	http://www.arduino.cc ⓦ
Download	http://arduino.cc/en/Main/Software ⓦ
Forum	http://arduino.cc/forum/ ⓦ

	Adafruit
Website	http://www.adafruit.com ⓦ
Wiki	http://ladyada.net/products/atmega32u4breakout/ ⓦ
Forum	http://forums.adafruit.com ⓦ

	chipKIT
IDE	MPIDE
Website	http://chipkit.net ⓦ
Download	http://chipkit.net/started/ ⓦ
Wiki	http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,892&Cat=18 ⓦ
Forum	http://www.chipkit.org/forum/ ⓦ

	DFRobot
IDE	Arduino
Bluetooth library	http://www.dfrobot.com.cn/download/PlainProtocol.zip ⓦ
Website	http://www.dfrobot.com ⓦ
Wiki	http://www.dfrobot.com/wiki/index.php/Bluno_SKU:DFR0267 ⓦ
Forum	http://www.dfrobot.com/forum ⓦ

Digispark

IDE	Arduino
Website	http://digistump.com ⓦ
Download	http://digistump.com/wiki/digispark/tutorials/connecting ⓦ
Wiki	http://digistump.com/wiki ⓦ
Forum	http://digistump.com/board/ ⓦ

Maple

IDE	**MapleIDE**
Website	http://leaflabs.com/devices/ ⓦ
Download	http://leaflabs.com/docs/download.html ⓦ
Wiki	http://wiki.leaflabs.com/index.php ⓦ
Forum	http://forums.leaflabs.com ⓦ

LaunchPad

IDE	**Energia**
Website	http://www.ti.com/ww/en/launchpad ⓦ
Download	http://github.com/energia/Energia/wiki ⓦ
Wiki	http://energia.nu ⓦ
Forum	http://forum.43oh.com/forum/28-energia/ ⓦ http://forum.stellarisiti.com/forum/63-energia/ ⓦ

Microduino

IDE	Arduino
Website	http://www.microduino.cc ⓦ
Download	http://wiki.microduino.cc/index.php?title=Arduino_IDE_Microduino_Configuration ⓦ
Wiki	http://wiki.microduino.cc ⓦ

Processing

IDE	Processing
Website	http://www.processing.org ⓦ
Download	http://www.processing.org/download/ ⓦ
Wiki	http://wiki.processing.org/w/Main_Page ⓦ
Forum	http://forum.processing.org ⓦ

Protostack

Website	http://www.protostack.com ⓦ

Teensy

IDE	Teensyduino
Website	http://www.pjrc.com/teensy/index.html ⓦ
Download	http://www.pjrc.com/teensy/td_download.html ⓦ
Wiki	http://www.pjrc.com/teensy/index.html ⓦ
Forum	http://forum.pjrc.com/forum.php ⓦ

Wiring

IDE	Wiring
Website	http://wiring.org.co ⓦ
Download	http://wiring.org.co/download/ ⓦ
Wiki	http://wiki.wiring.co/wiki/Main_Page ⓦ
Forum	http://forum.wiring.co ⓦ

References

I've compiled a list of references I've consulted to develop the embedXcode template.

All brand names and trademarks mentioned in this electronic book are the property of their respective owners.

In case a reference is missing, please let me know so I could update the list.

Due to the very nature of internet, some links may be dead.

Boards and Platforms

"MPLAB X project configurations for stk500v2 bootloader", by svofski, July 21, 2011, at http://www.chipkit.cc/forum/viewtopic.php?p=1285#p1285 and http://pastebin.com/31XXwmUV ⓦ.

"MPIDE 0023 mpide-0023-macosx-20111221", by Ricklon, December 21, 2011, at https://github.com/chipKIT32/chipKIT32-MAX/downloads ⓦ.

"Energia: Arduino IDE ported to LaunchPad MSP430", by Robert Wessel, March 18, 2012, at https://github.com/energia/Energia ⓦ.

"Arduino Due upload trace log", by hiduino, November 06, 2012, at http://arduino.cc/forum/index.php/topic,128913.msg984614.html#msg984614 ⓦ.

"Minimal makefile for Teensy 30", by Paul Stoffregen, November 06, 2012, at http://forum.pjrc.com/threads/64-Suggested-Development-Tools-for-Mac?p=263&viewfull=1#post263 ⓦ.

"Mass erase of MSP-EXP430FR5739", by smallbulb, February 07, 2012, at http://www.smallbulb.net/2012/202-mass-erase-of-msp-exp430fr5739 ⓦ.

"Digispark Add-on for Arduino 103", by Erik Kettenburg, January 01, 2013, at http://digistump.com/wiki/digispark/tutorials/connecting ⓦ.

"Arduino IDE 15 3rd party hardware spécification", by Cristian Maglie *et al.*, February-March, 2013, at http://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5---3rd-party-Hardware-specification ⓦ.

"Arduino IDE 15: library specification", by Cristian Maglie *et al.*, February-March, 2013, at http://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5:-Library-specification ⓦ.

"Digispark Add-on for Arduino 104", by Erik Kettenburg, March 20, 2013, at http://digistump.com/wiki/digispark/tutorials/connecting ⓦ.

"MSP430 LaunchPad Energia development on Linux", by Alessandro Pasotti, March 18, 2013, at http://www.itopen.it/2013/03/01/msp430-energia-on-linux ⓦ.

"LaunchPad MSP430", by Alessandro Pasotti, March 18, 2013, at http://github.com/elpaso/energia-makefile ⓦ.

AVR and GNU Tools

"AVRDUDE, A program for download/uploading AVR microcontroller flash and eeprom", by Brian S Dean, October 29, 2007, at ftp://gnumirrorspaircom/savannah/avrdude/avrdude-doc-55pdf ⓦ.

"How to compile AVR-Code with Mac OSX", by Administrator, October 18, 2010, at http://www.definefalsetrue.com/index.php/en/AVR/how-to-compile-avr-code-with-mac-osx.html ⓦ.

"Minimal AVR project template for Xcode", by Jens Willy Johannsen, August 25, 2011, at http://stackoverflow.com/questions/6976500/avr-for-xcode-4 ⓦ.

"Query on -ffunction-section and -fdata-sections options of gcc", by fwhacking, November 04, 2011, at http://stackoverflow.com/questions/4274804/query-on-ffunction-section-fdata-sections-options-of-gcc/11223700 ⓦ.

"Using the GNU Compiler Collection", by gnuorg, collective, November 15, 2011, at http://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc ⓦ.

"Make Documentation", by gnuorg, collective, November 15, 2011, at http://www.gnu.org/software/make/manual/html_node/index.html ⓦ.

"GDB Tutorial - A Walkthrough with Examples", by Department of Computer Science, University of Maryland, March 22, 2009, at http://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf ⓦ.

"Debugging with GDB", by Richard Stallman, Roland Pesch, Stan Shebs, *et al.*, August 30, 2013, at http://sourceware.org/gdb/current/onlinedocs/gdb ⓦ.

"Debugging a Program on the Stellaris Launchpad Board", by Scompo Projects, November 7, 2012, at http://scompoprojects.wordpress.com/2012/11/07/debugging-a-program-on-the-stellaris-launchpad-board ⓦ.

"Open On-Chip Debugger OpenOCD 0.7.0", by Dominic Rath, May 5, 2013, at http://openocd.sourceforge.net ⓦ.

"OpenOCD User's Guide", by The OpenOCD Project, November 17, 2013, at http://openocd.sourceforge.net/doc/pdf/openocd.pdf ⓦ.

Command Line and makefile

"A Makefile for Arduino Sketches", by Martin Oldfield, June 04, 2010, at http://bleaklow.com/2010/06/04/a_makefile_for_arduino_sketches.html ⓦ.

"Advanced Arduino Hacking", by Maik Schmidt, April 01, 2011, at http://pragprog.com/magazines/2011-04/advanced-arduino-hacking ⓦ and https://github.com/maik/pragpub ⓦ.

"Command-line Arduino development", by Akkana Peck, May 30, 2011, at http://shallowsky.com/software/arduino/arduino-cmdline.html ⓦ.

"Arduino from the command line", by Martin Oldfield, June 23, 2011, at http://mjo.tc/atelier/2009/02/arduino-cli.html ⓦ and http://mjo.tc/atelier/2009/02/acli/arduino-mk_0.6.tar.gz ⓦ.

"Arduino makefile", by Álvaro Justen (Turicas), October 11, 2011, at https://github.com/turicas/arduinoMakefile/blob/master/resources.markdown ⓦ.

"A command line toolkit for working with Arduino hardware", by Амперка (amperka), November 01, 2011, at http://arduino.cc/forum/index.php/topic,77458.0.html ⓦ and https://github.com/amperka/ino ⓦ.

Xcode

"Making custom templates for Xcode 4 – March 2011", by Adam (red-glasses), March 21, 2011, at http://blog.red-glasses.com/index.php/tutorials/making-custom-templates-for-xcode-4-march-2011 ⓦ.

"A minimal project template for Xcode 4", by borealkiss, March 11, 2011, at http://blog.boreal-kiss.net/2011/03/11/a-minimal-project-template-for-xcode-4/ and https://github.com/borealkiss/Minimal-Template ⓦ.

"Fixing Xcode 4's Broken Code Completion", by Ben Scheirman, June 08, 2011, at http://benscheirman.com/2011/06/fixing-xcode-4s-broken-code-completion ⓦ.

"Creating Custom Xcode 4 Project Templates", by Me and Mark Publishing, December 05, 2011, at http://meandmark.com/blog/2011/12/creating-custom-xcode-4-project-templates ⓦ.

"Xcode 4 external build system code completion", by Mattias Wadman, January 11, 2012, at http://stackoverflow.com/questions/8726869/xcode-4-external-build-system-code-completion ⓦ.

"Xcode 4 Unleashed", by Fritz F Anderson, May 08, 2012, at http://www.informit.com/store/xcode-4-unleashed-9780672333279 ⓦ.

embedXcode and Similar Projects

"Using Arduino in Xcode", by Robert Atkins, February 28, 2009, at http://robertcarlsen.net/2009/02/28/using-arduino-in-xcode-532 ⓦ.

"Thread Update: Linker Problem, Arduino Uno and Xcode", by Rei Vilo, January 06, 2011, at http://arduino.cc/forum/index.php/topic,49956.0.html ⓦ.

"Arduino makefile for Xcode", by Rei Vilo, July 01, 2011, at http://embedXcode.weebly.com/arduino/20--arduino-makefile-for-xcode ⓦ.

"Programming Arduino with Xcode", by Nick, July 30, 2011, at http://makesomecode.com/2010/07/30/programming-arduino-with-xcode/ ⓦ.

"Visual Micro, Free Arduino Programming IDE for Microsoft Visual Studio", by Visual Micro, October 04, 2011, at http://www.visualmicro.com ⓦ.

"Arduino Eclipse Plug-In", by Jantje, November 06, 2011, at http://www.baeyens.it/eclipse/ ⓦ and https://github.com/jantje/arduino-eclipse-plugin ⓦ.

"New IDE for Mac OS X written in Cocoa", by fabiankr, January 12, 2012, at http://arduino.cc/forum/index.php/topic,86028.0.html ⓦ and https://github.com/fabiankr/Cocoduino ⓦ.

"Arduino with Xcode", by Tim Knapen, November 12, 2011, at https://github.com/timknapen/Arduino-With-XCode ⓦ.

"Initial repository closed", by Rei Vilo, December 04, 2011, at http://github.com/rei-vilo/Xcode-for-MPIDE-Arduino ⓦ.

"Trunk continued by Tim Knapen", by Tim Knapen, December 04, 2011, at https://github.com/timknapen/Arduino-With-XCode ⓦ.

"Master cloned by gnimmel", by Gnimmel, December 04, 2011, at https://github.com/gnimmel/Xcode-for-MPIDE-Arduino ⓦ.

"chipKIT Compatible Arduino-based Makefile", by Christopher Peplin, December 09, 2011, at http://christopherpeplin.com/2011/12/chipkit-arduino-makefile ⓦ and https://github.com/peplin/arduino.mk ⓦ.

"Arduino with Xcode", by Rei Vilo, January 16, 2012, at https://github.com/rei-vilo/Arduino-With-XCode ⓦ.

"mpideXcode — release a : initial release", by Rei Vilo, January 22, 2012, at https://github.com/rei-vilo/mpideXcode ⓦ.

Doxygen

"Bien documenter son code avec Doxygen et Xcode", by mouviciel, February 07, 2009, at http://mouviciel.free.fr/blog/index.php?2009/02/07/46-bien-documenter-son-code-avec-doxygen-et-xcode ⓦ.

"Documenting Objective-C with Doxygen Part I", by Fred McCann, March 18, 2010, at http://www.duckrowing.com/2010/03/18/documenting-objective-c-with-doxygen-part-i/ ⓦ.

"Documenting Objective-C with Doxygen Part II", by Fred McCann, March 18, 2010, at http://www.duckrowing.com/2010/03/18/documenting-objective-c-with-doxygen-part-ii/ ⓦ.

"Using Doxygen to Create Xcode Documentation Sets", by Apple, September 01, 2010, at http://developerapplecom/library/mac/#featuredarticles/DoxygenXcode/index.html ⓦ.

"Doxygen Shortcuts in Xcode 4", by Broken Rules GmbH, March 29, 2011, at http://www.brokenrul.es/blog/?p=761#.T856wD5YtoA ⓦ.

"Using the Doxygen Helper in Xcode 4", by Fred McCann, May 14, 2011, at http://www.duckrowing.com/2011/05/14/using-the-doxygen-helper-in-xcode-4/ ⓦ and http://www.duckrowing.com/wp-content/uploads/2011/05/xcode_doxygen_helper.tgz ⓦ.

"Graphviz 228, Graph Visualization Software", by ATT, April 25, 2012, at http://www.graphviz.org ⓦ.

"Doxygen 181", by Dimitri van Heesch, May 19, 2012, at http://doxygen.org ⓦ.

"TeXShop 311", by Richard Koch, June 06, 2012, at http://www.texshop.org ⓦ.

"ThisService 3", by waffle software, July 25, 2012, at http://wafflesoftware.net/thisservice ⓦ.

"Doxygen 182", by Dimitri van Heesch, August 11, 2012, at http://doxygen.org ⓦ.

"Graphviz: Getting it to work on Mountain Lion", by Anonymous, August 20, 2012, at http://www.graphviz.org/Download_macos.php#comment-1025 ⓦ.

"Graphviz 230, Mountain Lion compatible, Graph Visualization Software", by ATT, March 01, 2013, at http://www.graphviz.org ⓦ.

"MacTeX-2013 Distribution", by the TeX Users Group, May 30, 2013, at http://www.tug.org ⓦ and http://www.tug.org/twg/mactex ⓦ.

Other Tools

"Homebrew", original code by Max Howell, website by Rémi Prévost, May 20, 2009, at http://mxcl.github.com/homebrew/ ⓦ and https://github.com/mxcl/homebrew ⓦ.

"pyserial 26, Python Serial Port Extension", by Chris Liechti, February 11, 2011, at http://pypi.python.org/pypi/pyserial ⓦ.

"pyserial 26: Python Serial Port Extension", by Chris Liechti, November 02, 2011, at http://pypi.python.org/pypi/pyserial ⓦ.

"Use stty instead of Python script with pySerial installed", by David Palmer, September 23, 2012, at http://embedxcodeweeblycom/1/post/2012/09/embedxcode-15-with-leonardo-supporthtml#comments ⓦ.

Other References

"Publishing with iBooks Author — An Introduction to Creating Ebooks for the iPad", by Nellie McKesson and Adam Witwer, O'Reilly Media / Tools of Change, February 10, 2012, at http://shop.oreilly.com/product/0636920025597 ⓦ.

Contributions from Users

"Support for Arduino Due — Thanks to Mike Roberts", by Mike Roberts, January 15, 2013.

"Configuration file for chipKIT uC32 — Thanks to John James", by John James, April 21, 2013.

A warm 'Thank you!' to the users who tested new boards and reported them!

Tests, Variables and Sizes

This section provides the versions of the IDEs and plug-ins, the list of the tests, the variables used by embedXcode and the sizes of the executables compared to those from the IDEs.

IDEs and Plug-Ins Versions

The test protocol includes build, link and upload with a functioning sketch on the boards using those versions of the IDEs and plug-ins.

Platform	IDE	Plug-In	Date	Comment
Arduino	Arduino 0023		Nov 09, 2011	No longer supported
	Arduino 1.0.5		May 05, 2013	Recommended release
	Arduino 1.5.5 beta		Nov 28, 2013	Required for Arduino Due and Yún boards, but unstable and prone to crash
chipKIT	MPIDE 0023		July 15, 2013	
Digispark	DigisparkArduino 1.0.4 IDE		May 19, 2013	
LaunchPad	Energia 0101E0011		Dec 17, 2103	
Maple	Maple IDE 0.0.12		Sep 13, 2011	Support on hold
Microduino	Arduino 1.0.5	Microduino	May 27, 2013	
Teensy	Arduino 1.0.5	Teensyduino 1.1.7	Dec 09, 2103	Required for Teensy 3.1
Wiring	Wiring 1.0 (0100)		Oct 18, 2011	Support on hold

What Has Been Tested

The test protocol includes build, link and upload with a functioning sketch on the boards. Obviously, I can only perform the tests on the boards and programmers I own.

Users have reported successful use of embedXcode on Arduino Due, chipKIT uC32 and Max32.

If you're successful with another board, please report it to me so I can update the list. Thank you!

Platform	What I Have Tested	What Users Have Tested	What Has Not Been Tested
Arduino	Arduino Uno, Duemilanove, Mega2560, Mini Pro, Leonardo, Yún USB, Yún WiFi, Yún Ethernet Sparkfun Uno	Arduino Due	
Adafruit	Adafruit Atmega32u4 Breakout Board with Adafruit USBtinyISP programmer		
chipKIT	chipKIT Uno32	chipKIT uC32, Max32	
Digispark	Digispark		
LaunchPad	LaunchPad with MSP430G2331, MSP430G2452 or MSP430G2553, LaunchPad with MSP430F5529 Experimeter Board with MSP430FR5739 LaunchPad Stellaris with LM4F120H5QR or Tiva C Series with TM4C123GH6PM		
Maple	Maple revision 5		
Microduino	Microduino Core and Core+		
Protostack	Protostack 28-pin AVR board with Adafruit USBtinyISP and Protostack USBASP programmers		
Teensy	Teensy 3.0, Teensy 3.1		Teensy 2.0
Wiring	Wiring S		

IDE and MCU Variables

Here are the variables for each IDE and corresponding MCUs.
Note some IDEs define two variables.

IDE	IDE Variable	MCU Variable
Arduino 23	`ARDUINO=23`	`__AVR_ATmega328P__ __AVR_ATmega2560__ __AVR_ATmega1280__`
Arduino 1.0.x	`ARDUINO=102`	`__AVR_ATmega328P__ __AVR_ATmega2560__ __AVR_ATmega1280__ __AVR_ATmega32U4__`
Arduino 1.5.x	`ARDUINO=150`	`__AVR_ATmega328P__ __AVR_ATmega2560__ __AVR_ATmega1280__ __AVR_ATmega32U4__ __SAM3X8E__`
chipKIT MPIDE	`MPIDE=23` and `ARDUINO=23`	`__32MX320F128H__ __32MX340F512H__ __32MX795F512L__`
Digispark	`DIGISPARK` and `ARDUINO=102`	`__AVR_ATtinyX5__`
Energia	`ENERGIA=9` and `ARDUINO=101`	`__MSP430G2231__ __MSP430G2452__ __MSP430G2553__ __MSP430F5529__ __MSP430FR5739__ __LM4F120H5QR__ __TM4C123GH6PM__`
Maple	`MAPLE_IDE`	`MCU_STM32F103RB MCU_STM32F103ZE MCU_STM32F103CB MCU_STM32F103RE`
Microduino	`MICRODUINO` and `ARDUINO=101`	`__AVR_ATmega328P__ __AVR_ATmega644P__`
Teensy	`TEENSYDUINO`	`__AVR_ATmega32U4__ __MK20DX128__ __MK20DX256__`
Wiring	`WIRING=100`	`__AVR_ATmega644P__`
Xcode	`EMBEDXCODE=111`	

Unlike all other IDE variables, `MICRODUINO` and `DIGISPARK` are not recognised by their respective IDEs.

HEX and BIN Files Sizes

For the compiled HEX and BIN files, embedXcode provides sizes very close to and even better than those obtained with the corresponding IDEs. All measures are in bytes.

IDE	Board	Maximum	Official IDE	embedXcode	Difference
Arduino 23	Arduino Uno	32 256	1 094	1 100	-6
	Arduino Mega 2560	258 048	1 664	1 670	-6
Arduino 1.0	Arduino Uno	32 256	1 152	1 142	10
	Arduino Mega 2560	258 048	1 696	1 718	-22
Arduino 1.5	Arduino Uno	32 256	1 152	1 170	-18
	Arduino Mega 2560	258 048	1 696	1 746	-50
chipKIT MPIDE	chipKIT Uno32	126 976	5 928	5 928	0
Energia	LaunchPad with MSP430G2452	8 192	630	634	-4
	LaunchPad with MSP430G2553	16 384	638	642	-4
	Experimeter Board with MSP430FR5739	15 872	756	764	-8
	LacunhPad Stellaris with LM4F120H5QR	262 144	1 896	1 896	0
Digispark	Digispark (Tiny core)	6 012	812	816	-4
Maple IDE	Leaflabs Maple rev 5	108 000	12 896	12 016	880
Teensy	Teensy 3.0	131 072	8 028	7 824	204
Wiring	Wiring S	63 488	2 232	2 238	-6

License

embedXcode comes in one single personal license, for personal use.

embedXcode+ comes in two licenses:

- A personal license if you are an individual, you donate on your own funds and you use embedXcode+ for your personal use.
- A commercial license if you are a legal entity, as a company or a self-employed developer, or for commercial development.

The personal license is upgradable to the commercial license.

The user might be invited to confirm the terms of the personal license.

The violation of this agreement may terminate the license.

Disclaimer

THE EMBEDXCODE SOFTWARE IS PROVIDED TO YOU "AS IS," AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEORY UNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY OR LIKELIHOOD OF SUCH DAMAGES.

Architecture

The MCUs are grouped in families or architectures.

As example, one characteristic of the architectures is the number of bits used to process data: it could be 8-bit for the ATmega, 16-bit for the MSP430 or 32-bit for the ARM.

Other characteristics include the structure of the hardware and the instruction set.

Each architecture requires its dedicated toolchain.

Related Glossary Terms

Arduino, Boards, MCU, Platform, Toolchain

Arduino

The Arduino name includes hardware, a framework and an IDE.

The boards are based on two major architectures:

- 8-bit ATmega and
- 32-bit ARM SAM.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Architecture, Boards, DFRobot, Framework, IDE, Microduino

Boards

A board is hardware defined by its MCU.

For example, the LaunchPad Stellaris is based on the ARM Stellaris LM4F120H5QR MCU.

All the embedded computing boards use a Processing-based Wiring-derived Arduino-like IDE.

They share the same Processing-based IDE, use the same Wiring-derived or Arduino framework and bring the same Arduino-like programming with sketches written in C++.

The IDE includes the toolchain specific to the boards.

Related Glossary Terms

Architecture, Arduino, C++ Language, chipKIT, DFRobot, Digispark, Framework, IDE, LaunchPad, Maple, MCU, Microduino, Platform, Teensy, Wiring

---

Index Rechercher un terme

C++ Language

The C++ is a programming language based on the C language.

It features object-oriented features like classes.

It is used for programming the embedded computing boards.

Termes connexes du glossaire

Boards, File .c++, File .h, File .ino, File .pde, Framework, IDE, Sketch, Toolchain

---

Index Rechercher un terme

Chapter 7 - Compatibility

chipKIT

chipKIT uses the PIC32 MCUs. The IDE is MPIDE.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards, MPIDE

Index Rechercher un terme

Debugger

Debugging allows to execute a program one line at a time, check and change the values of the variables, trace the calls of functions, ... It consists on a combination of hardware and software.

There are two ways of debugging:

- Software debugging, most of the time performed by injecting code like the basic `printf("i=%d\n", i)`.
- Hardware debugging, with a specific chip called a debugger or emulator along the MCU being debugged, and a dedicated application.

The LaunchPad line of boards by Texas Instruments is the only one to include a hardware debugger. It requires the external application GDB, or GNU debugger included in the GCC toolchain.

Termes connexes du glossaire

Energia, IDE, Toolchain

DFRobot

DFRobot BLuno board integrates Bluetooth 4.0 or BLE into a standard Arduino Uno board.

The BLuno board is considered as a standard Arduino Uno board and thus, requires no specific plug-in.

See Appendixes — Links, Forums and Websites for more information.

Termes connexes du glossaire

Arduino, Boards

Index Rechercher un terme

Digispark

Digispark uses the ATtiny85 MCU and requires a specific upload procedure.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards

Index Rechercher un terme

embedXcode

embedXcode is a template for Xcode 4. It eases development for the most popular embedded computing boards.

A second version called embedXcode+ includes additional extended functionalities. The features specific to each version are listed at Appendixes - Versions.

embedXcode relies on the IDEs of the boards for the frameworks and toolchains.

Related Glossary Terms

embedXcode+, Framework, IDE, Xcode

---

embedXcode+

embedXcode+ is a template for Xcode 4. It eases development for the most popular embedded computing boards.

embedXcode+ adds extended functionalities to embedXcode.

As embedXcode, embedXcode+ relies on the IDEs of the boards for the frameworks and toolchains.

The features specific to embedXcode+ are mentioned by ⊞ and listed at Appendixes - Editions.

Related Glossary Terms

embedXcode, Framework, IDE, Xcode

Energia

Energia is the Arduino 1.0-based IDE for the LaunchPad boards.

Although all the LaunchPad line of boards managed by Energia include a hardware debugger, the IDE doesn't feature the corresponding application.

So an external application is required: GDB, or GNU debugger, already included in the GCC toolchain.

Related Glossary Terms

Debugger, IDE, LaunchPad

Index Rechercher un terme

Chapter 1 - Install the IDEs of the Boards

Appendixes - Links, Forums and Official Websites

File .c++

File extension for **C++ code** file.

The C++ code file defines the classes and functions declared in a header file.

The files with `.pde` or `.ino` extensions are actually C++ code.

Related Glossary Terms

C++ Language, File .h, File .ino, File .pde, Sketch

Index Rechercher un terme

File .docset

File extension for **documentation set**, Apple's proprietary format for Xcode help.

Related Glossary Terms

File .tex

Index

Rechercher un terme

File .h

File extension for **header file**.

A header file contains the list of public constants, variables, classes and functions defined in a C++ code file.

Related Glossary Terms

C++ Language, File .c++

Index Rechercher un terme

File .ino

File extension for the **sketch**, the main part of a program.

The `.ino` extension is used by Arduino 1.0 and 1.5, Digispark, Energia and Teensy.

It replaces the `.pde` extension.

The `.pde` and `.ino` files aren't recognised as C++ code by Xcode. During the first compilation, the project is prepared by embedXcode: the files are recognised as C++ code to allow code-sense.

Related Glossary Terms

C++ Language, File .c++, File .pde, Sketch

Index Rechercher un terme

File .pde

File extension for the **sketch**, the main part of a program.

The `.pde` extension is used by Arduino 0023, chipKIT MPIDE, Maple and Wiring.

It has been superseded by the `.ino` extension.

The `.pde` and `.ino` files aren't recognised as C++ code by Xcode. During the first compilation, the project is prepared by embedXcode: the files are recognised as C++ code to allow code-sense.

Related Glossary Terms

C++ Language, File .c++, File .ino, Sketch

Index Rechercher un terme

File .tex

File extension for **LateX** file, a language for documents with high quality formatting.

The LateX files are generated by Doxygen and converted into PDF documents.

Related Glossary Terms

File .docset

Index Rechercher un terme

Chapter 6 - Select the Output Formats

Framework

The framework includes a set of libraries and provides an hardware abstraction layer.

The libraries are invoked by the `#include` statement.

The same code can virtually run on any boards with an IDE based on that framework.

The framework for the boards is mostly written in C and C++.

The two references are Wiring and Arduino.

Related Glossary Terms

Arduino, Boards, C++ Language, embedXcode, embedXcode+, IDE, MCU, Platform, Toolchain, Wiring

---

Index Rechercher un terme

IDE

IDE stands for **integrated development environment** and is an application used for developing programs.

The IDEs used for the boards are based on the Processing IDE, making them very similar.

They feature a text editor and runs on Windows, Linux or Mac OS X. They use on the Wiring and Arduino frameworks, use the C++ language and rely on different toolchains.

They are used to develop programs to be run on different boards. Each board has its own version of the IDE, differentiated by the colours of the interface.

Another example of IDE is Xcode.

Related Glossary Terms

Arduino, Boards, C++ Language, Debugger, embedXcode, embedXcode+, Energia, Framework, MapleIDE, Microduino, MPIDE, Platform, Processing, Teensyduino, Toolchain,

---

Index Rechercher un terme

LaunchPad

The LaunchPads are built by Texas Instruments and feature different architectures, two of them being supported by Energia:

- 16-bit MSP430 LaunchPad and MSP430FR3759 Experimenter Board
- 32-bit ARM Stellaris or Tiva C Series LaunchPad

The IDE is Energia.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards, Energia

Maple

Maple boards are based on 32-bit ARM MCUs.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards, MapleIDE

MapleIDE

MapleIDE is the Wiring-based IDE for the Maple boards.

The MapleIDE requires special drivers.

Please refer to Chapter 1 - Install the IDEs of the Boards.

Related Glossary Terms

IDE, Maple

Index Rechercher un terme

MCU

MCU stands for **micro-controller unit** and includes all the parts of a computer: processing unit, read only memory or EEPROM, flash memory to store the programs and random-access memory or SRAM to store data. The MCUs are grouped in families or architectures.

It also includes general purpose inputs outputs or GPIOs. Some of them can be used for specialised buses like, among the most popular:

- UART or universal asynchronous receiver/transmitter, most commonly serial port,
- SPI or serial peripheral interface,
- I²C or inter-integrated circuit,
- TWI or two wire interface,
- I²S or inter inter-chip sound,
- CAN or controller area network.

Other extras may include a real-time clock, a FPU or floating-point unit, ...

Related Glossary Terms

Architecture, Boards, Framework, Platform, Toolchain

Index Rechercher un terme

Microduino

Microduino uses the same MCUs as Arduino but on a compact form-factor.

Microduino requires a plug-in for the Arduino IDE.

See Appendixes — Links, Forums and Websites for more information.

Termes connexes du glossaire

Arduino, Boards, IDE

Index Rechercher un terme

MPIDE

MPIDE is the Arduino 0023-based IDE for the chipKIT boards.

Related Glossary Terms

chipKIT, IDE

Platform

A platform is a mix of IDE, frameworks, boards, architectures and toolchains.

As an example, the Arduino platform includes

- An IDE called Arduino;
- Two frameworks, Arduino 1.0 and Arduino 1.5, the later still in beta;
- Many different boards, which can be grouped in two architectures:
  - the 8-bit ATmega-based boards, as Arduino Uno or Arduino Mega2560,
  - and the 32-bit SAM-based Arduino Due board;
- Two toolchains, one for each architecture.

Related Glossary Terms

Architecture, Boards, Framework, IDE, MCU, Toolchain

Processing

All the boards use the Processing IDE, adapted for C++.

Processing doesn’t feature any board. Instead, it runs on the main computer and provides an easy interface for displaying graphs based on data acquired from the board.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

IDE

Sketch

A sketch is basically a program written in C++ and based on the Wiring and Arduino framework.

The valid file extensions for a sketch are `.pde` or `.ino`.

Related Glossary Terms

C++ Language, File .c++, File .ino, File .pde

Index Rechercher un terme

Teensy

Teensy includes two major architectures:

- 8-bit ATmega for Teensy 2 and
- 32-bit ARM for Teensy 3.

The IDE is Teensyduino.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards, Teensyduino

Teensyduino

Teensyduino is the Arduino 1.0 IDE-compatible plug-in for the Teensy boards.

Please refer to Install the IDEs of the Boards — Teensy Boards to avoid possible conflicts with the Arduino 1.0 IDE.

Related Glossary Terms

IDE, Teensy

Index Rechercher un terme

Toolchain

A toolchain is a set of tools used to build, link, upload and debug a sketch to a board.

The tools from the toolchain are called by the IDE.

The MCUs are grouped by architectures and each architecture has its specific toolchain.

Most of the toolchains used with the embedded computing boards are based on GCC or GNU Compiler Collection.

Related Glossary Terms

Architecture, C++ Language, Debugger, Framework, IDE, MCU, Platform

---

Wiring

Wiring focuses on defining the framework.

Some boards are especially designed for Wiring, as the ATmega64-based Wiring S.

See Appendixes — Links, Forums and Websites for more information.

Related Glossary Terms

Boards, Framework, IDE

Index Rechercher un terme

Xcode

Xcode is the official IDE from Apple.

Xcode version 4.6.3 is recommended for embedXcode.

Related Glossary Terms

embedXcode, embedXcode+, IDE

Index Rechercher un terme