OsiriX Plug-ins Developer Toolkit

Version 1.0

Antoine Rosset, UCLA, California, 2004
rossetantoine@bluewin.ch

TABLE OF CONTENT

How to develop on MacOS X?

Developing software on MacOS X is really easy, great and it's totally free! The complete development environment is available with any Macs!

The **language** you will use to develop an OsiriX plug-in is Objective-C. Objective-C is a dynamic and object-oriented language, like Java. It is very similar to Java and C++. It uses the same syntax as Java and C++. It is much less complex than C++ and much faster than Java. If you already know Java or C++, you could probably learn Objective-C in less than 2 days...

The **framework** you will use is Cocoa. Cocoa is a powerful and object-oriented framework that will allow you to create any kind of plug-in with complex graphic user interface.

I personally learned Objective-C and Cocoa in a single book: "Cocoa Programming for Mac OS X – 2nd Edition" by Aaron Hillegass. OsiriX was my first Objective-C and Cocoa software that I wrote with the help of this book!

Other useful resources to learn Cocoa and Objective-C:

- http://cocoadevcentral.com/
- http://www.cocoadev.com/
- http://www.oreillynet.com/pub/ct/37
- http://developer.apple.com/macosx/
- http://cocoa.mamasam.com/
- And the Help menu of Xcode "Show Documentation Window"

The **development environment** is Xcode. Xcode is a really powerfull development environment, based on the open-source GCC compiler. OsiriX has been completely written in this environment in less than 6 months!

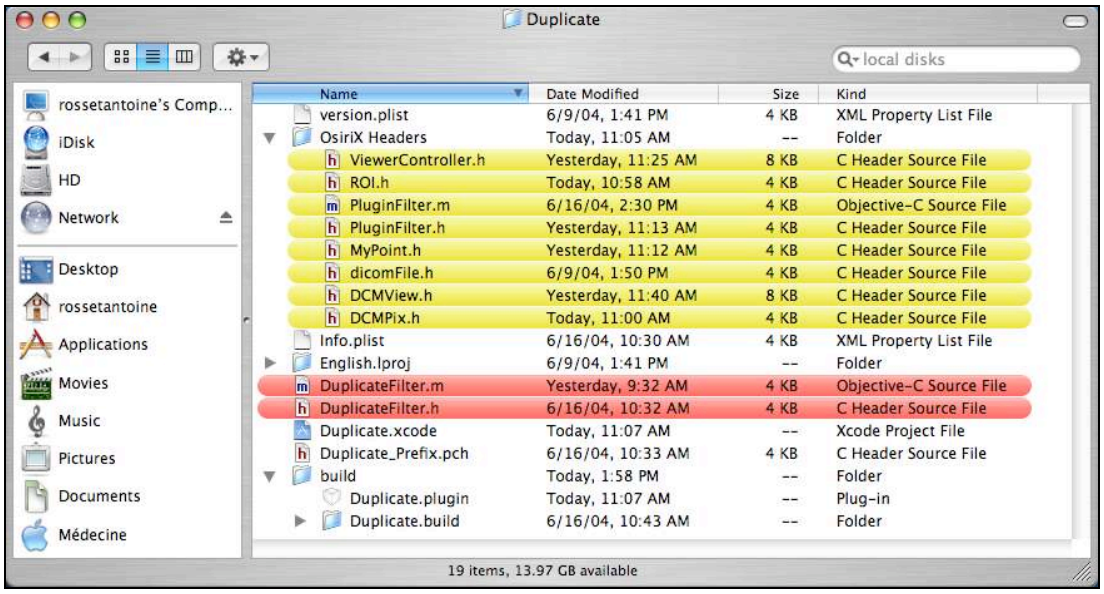
Writing a plug-in for OsiriX is much easier than developing in IDL or in ImageJ...

Ok, ready? Simply insert the "Developer" CD-Rom in your Mac, and double-click on the Installer...

Be part of the revolution, write a plug-in for OsiriX!

How to set up your Xcode environment?

Let's look at a plug-in example of the OsiriX developer toolkit. Open the 'Duplicate' folder.

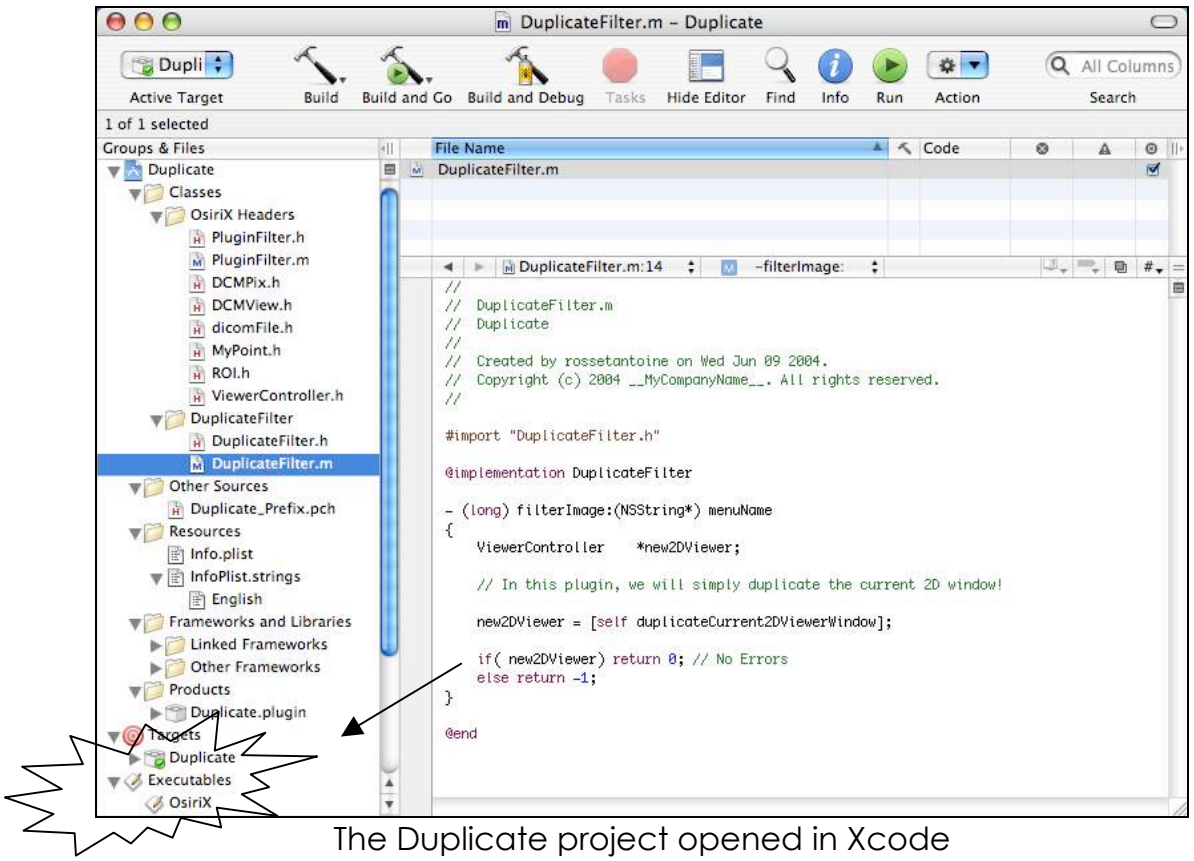


The Duplicate plug-in folder

This folder contains:

- The **version.plist** file and **Info.plist** file, which contain information about the plug-in name, icon, version number, etc. Edit these files directly from Xcode.
- The **OsiriX Headers** folder, which contains the objects you can access from your plug-in. **NEVER** modify these files! They must **ALWAYS** be present in your plug-in!
- The **English.lproj** folder contains localized data if you want to create a multi-language plug-in in Japanese and in English, for example.
- The **DuplicateFilter.m** and **DuplicateFilter.h** files contain the plug-in! It is the right place to write your own algorithm!
- The **Duplicate.xcode** file contains your project: double-click on it to open Xcode
- The **Duplicate_Prefix.pch** file contains a prefix file; you don't need to modify it.
- The folder **build** contains the compiled plug-in

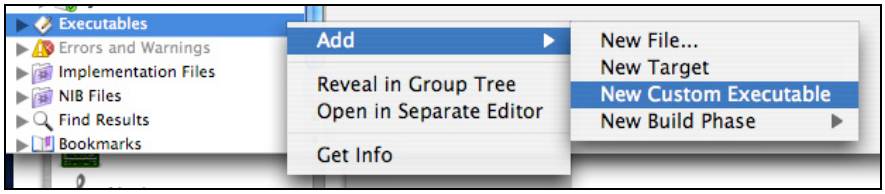
Double-click on the **Duplicate.xcode** file to open Xcode.

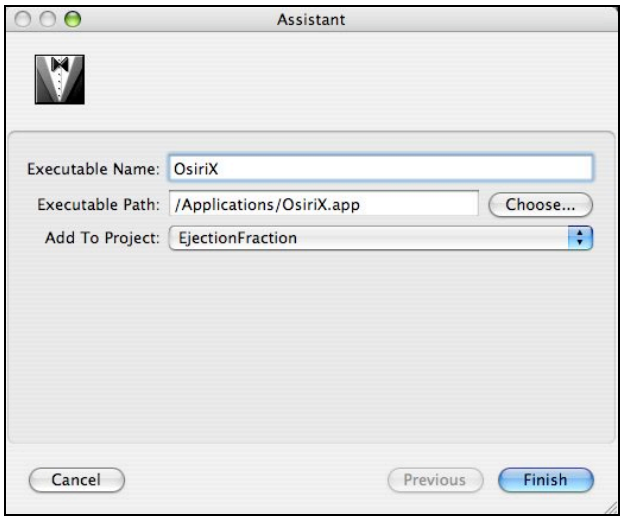


The Duplicate project opened in Xcode

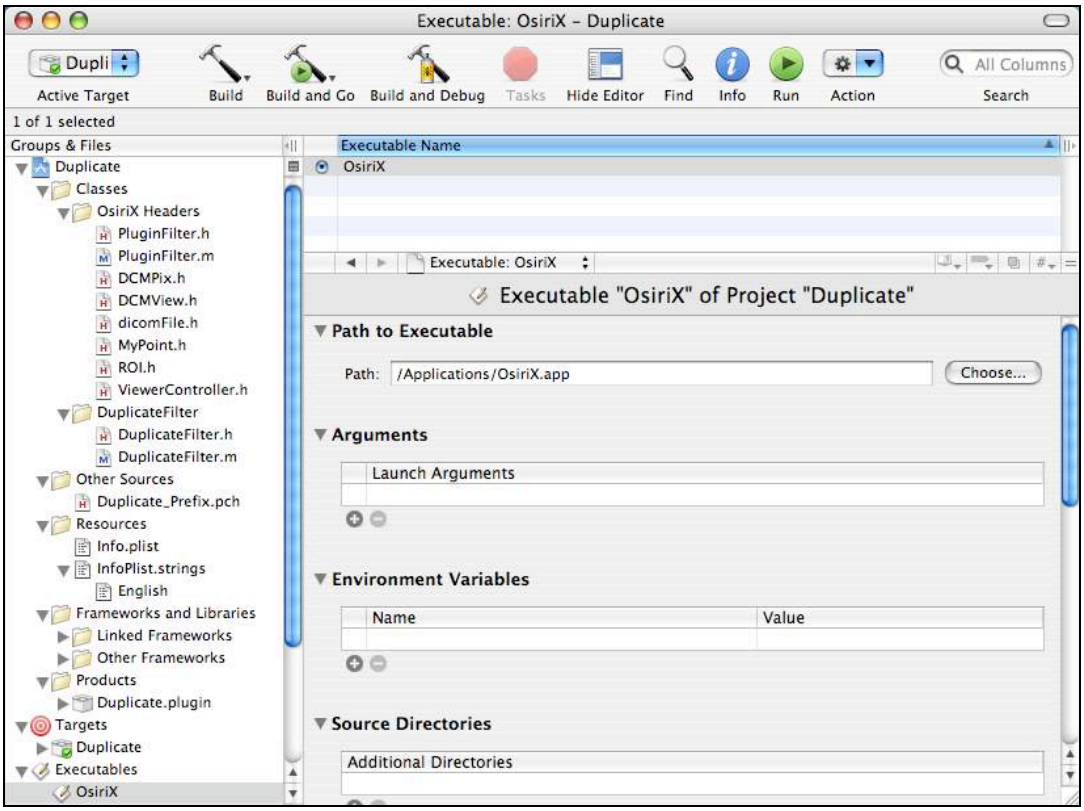
The first thing to do before trying to compile this plug-in is to tell Xcode where your OsiriX application is located. Click on the 'OsiriX' line from the 'Executables' group. Set the 'path to executable' to your OsiriX application, probably located in your 'Applications' folder.

If no executable is displayed, click on 'Executables' while maintaining the 'ctrl' key and select 'Add New Custom Executable':



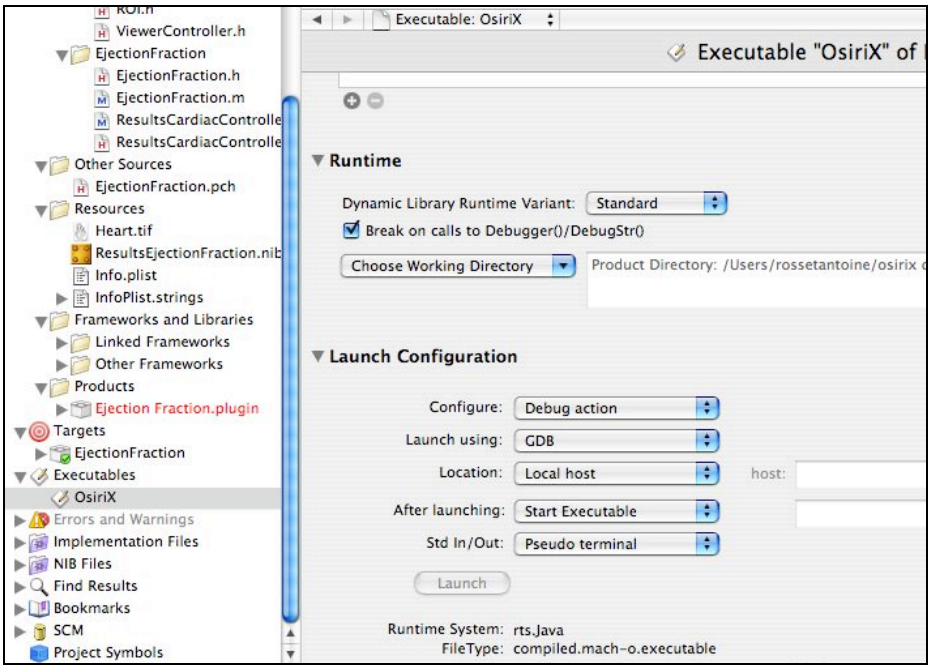


Create a new 'OsiriX' executable



Choose the 'Path to Executable' to the OsiriX application

It is also important to check that the 'Executable' Launch Configuration is correct! The 'Launch using' parameter has to be 'GDB', sometimes it switches wrongly to 'Java Debugger'



Check that 'Launch using' parameter is set to **GDB**, not to **Java Debugger**!

Now, you should be able to compile this example, by pressing on the 'Build' icon from the toolbar.

The compiled plug-in will appear in the 'Build' folder from the 'Duplicate' folder.

In order to debug the plug-in, the plug-in must be located in its folder and at the same time must be loaded by OsiriX! We will create an alias of this plug-in in the 'OsiriX plug-in' folder, located in the 'Application Support' folder of the 'Library' folder. If a copy of the 'Duplicate.plugin' is already there, delete it and replace it by an alias from the 'build' folder.

You can now run and debug this plug-in by pressing on the 'Build and Debug' icon from the toolbar: Xcode will open the 'debugger' window and launch OsiriX. OsiriX will then launch your plug-in.

How does OsiriX communicate with a plug-in?

When OsiriX starts, it will search in the 'plug-in' folder for files with the '.plugin' extension. It will then try to find if the plug-in contains a sub-class of the 'PluginFilter' object and if the object has a 'filterimage' function. OsiriX will then run once the 'initPlugin' function if available.

Your plug-in is a sub-class of the 'PluginFilter'. The 'PluginFilter' object is defined in the 'PluginFilter.m', available in the 'OsiriX header' folder. Never modify this 'PluginFilter.m' file!

The 'filterimage' function ((long) filterImage:(NSString*) menuName) is the main function of a plug-in. OsiriX will always use this function to call your plug-in. The 'menuName' string contains the menu selected by the user: a plug-in can define more than one menu (see How to write the 'Info.plist' file).

In your plug-in you can call virtually any functions of OsiriX.

Here is a list of available objects from OsiriX:

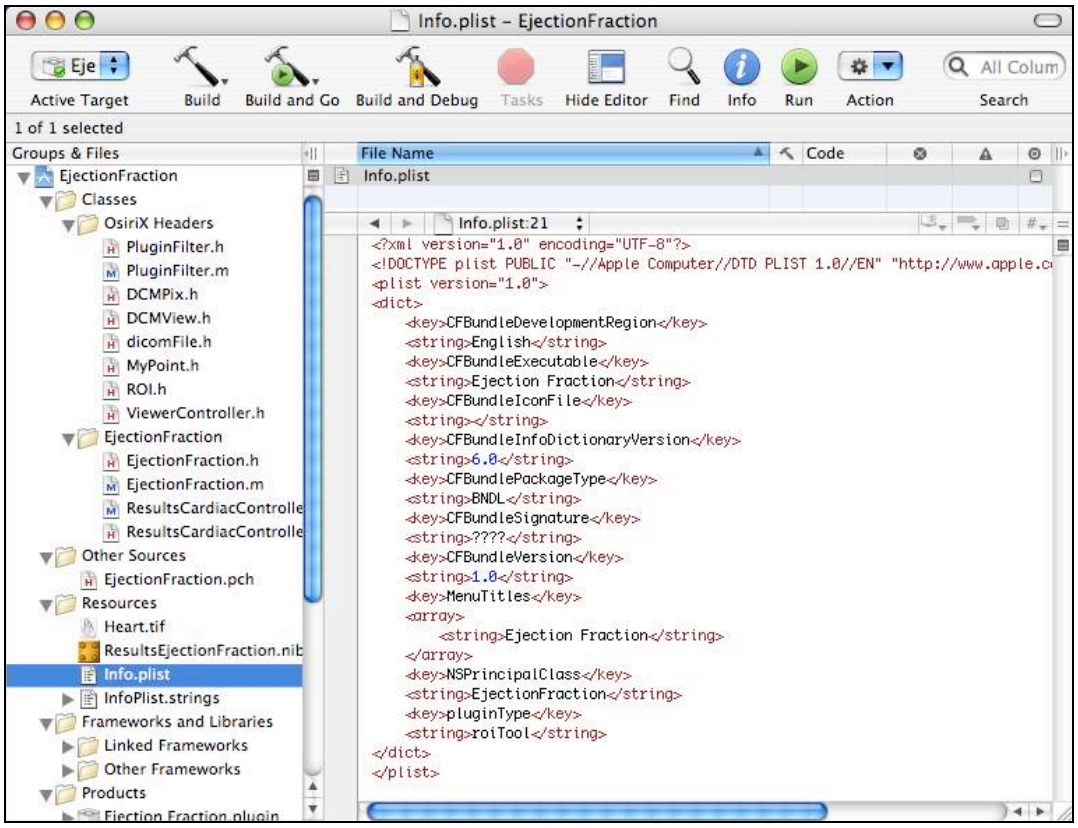
- PluginFilter : your plug-in is a sub-class of this object, it contains some usefull functions
- ViewerController : NSWindowController of a '2D Viewer' window
- DCMView : NSOpenGLView that contains the displayed image
- DCMPix : an object that contains the pixel data of an image
- dicomFile : an object that contains data about the DICOM file
- ROI : an object that contains a ROI
- MyPoint : an object that describes a 2D point

See the example to understand how to use these objects. Look at the headers files of these objects to find which functions are available and how to call them.

In your plug-in, you can define any Cocoa objects and use any 'nib' files you want!

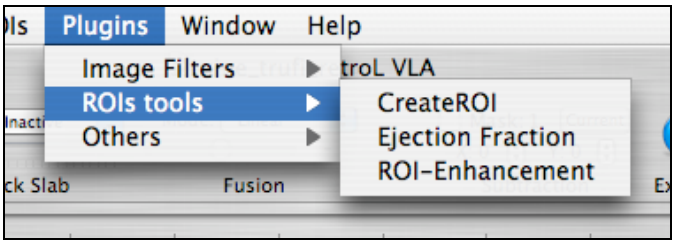
How to write the 'Info.plist' file?

This file contains important information required by OsiriX to understand what your plug-in can do.



The 'Info.plist' file

The field 'MenuTitles' contains the functions list that OsiriX will display in the 'plugins' menu or in the 'fusion window'.

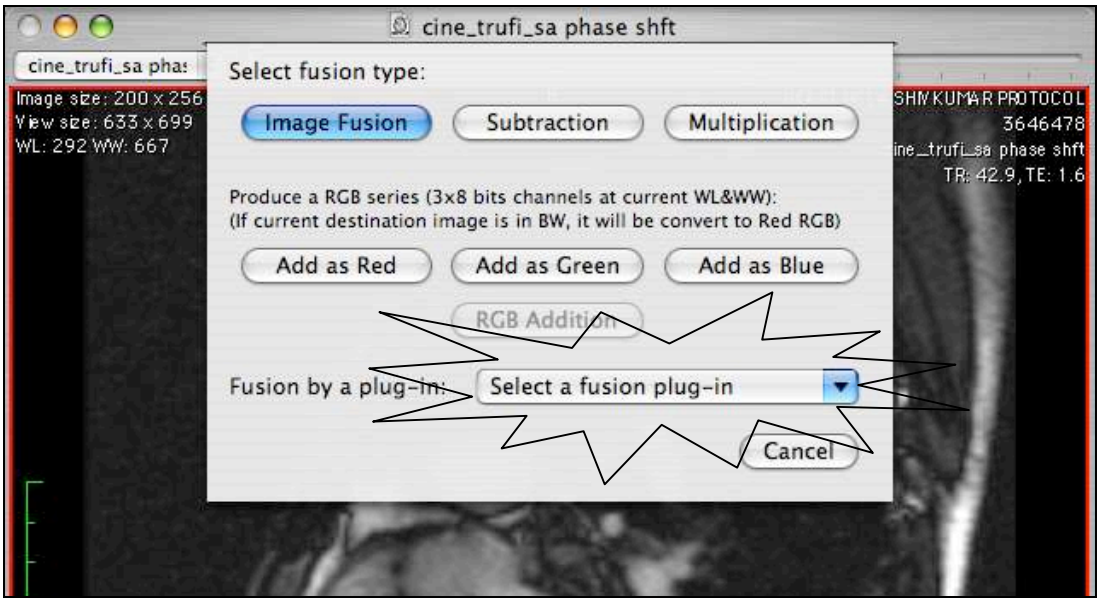


The 'Plugins' menu of OsiriX

If more than one name is defined, OsiriX will create a sub menu with sub items. The selected menu name will be contained in the 'menuName' NSString object of the 'filterImage' function.

The ‘pluginType’ field of the ‘info.plist’ file contains one of the following name:

- **imageFilter**: your plug-in will appear in the ‘Image Filters’ menu
- **roiTool**: your plug-in will appear in the ‘ROIs tools’ menu
- **other**: your plug-in will appear in the ‘Others’ menu
- **fusionFilter**: your plug-in will appear in the ‘Fusion’ window

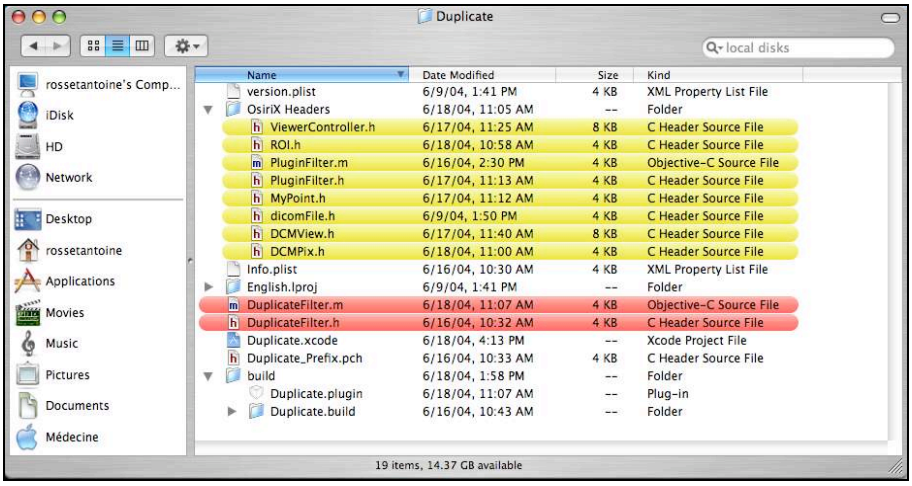


The ‘Fusion’ window appears when the user “drag & drop” a series on another one.

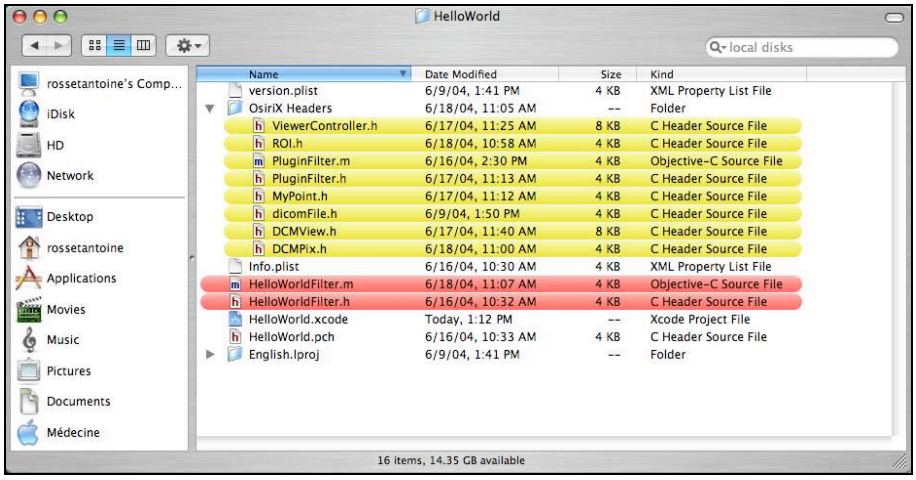
How to create your own plug-in?

In order to create a new plug-in, the easiest way is to duplicate an existing plug-in and rename it:

1. Quit Xcode if it is running
2. Duplicate a plug-in folder: for example the 'Duplicate' folder.
3. In this example, we will create a new plug-in named 'HelloWorld'. First rename the folder and then rename following files:
    a. DuplicateFilter.h to HelloWorldFilter.h
    b. DuplicateFilter.m to HelloWorldFilter.m
    c. Duplicate.xcode to HelloWorld.xcode
    d. Duplicate_Prefix.pch to HelloWorld.pch

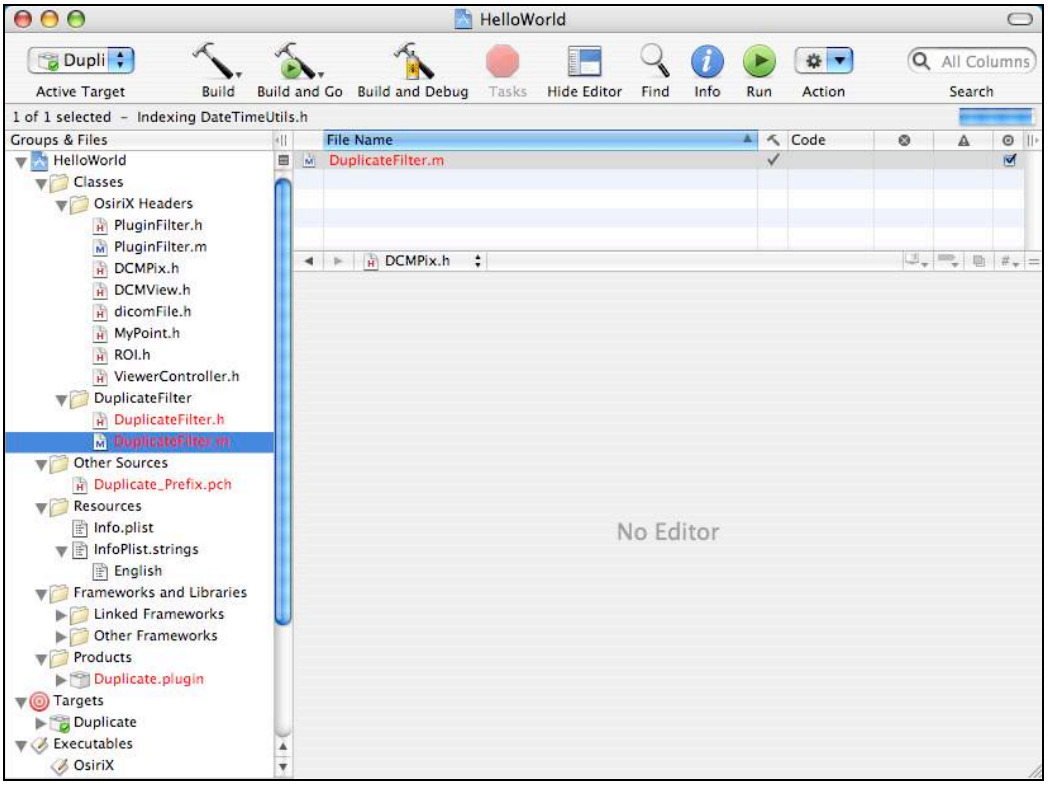


Before renaming



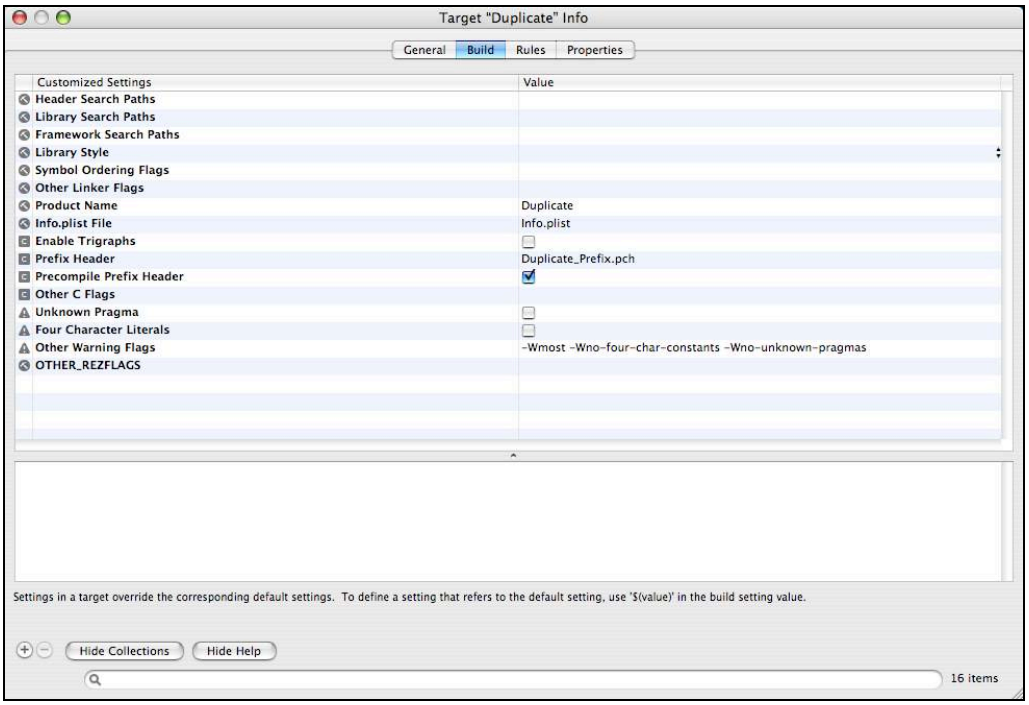
After renaming

4. Delete the “build” folder
5. Start Xcode by double-clicking on the “HelloWorld.xcode” file
6. Replace the “DuplicateFilter.h”,“DuplicateFilter.m”, “Duplicate_Prefix.pch” with your new files.



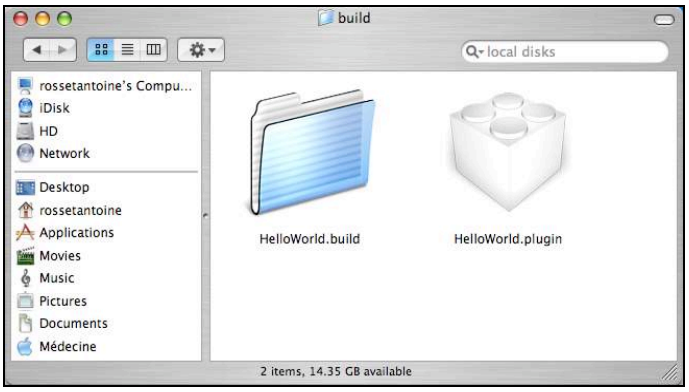
Replace the files in red color

7. Delete the “Duplicate.plugin” object in the “Products” group
8. Select the “Duplicate” target and “Get Info” on it : replace all occurrences of “Duplicate” with “HelloWorld” in the 4 parts of the window: “General”, “Build”, “Rules” and “Properties”.



The "Duplicate" target

9. Edit the "Info.plist" file: Enter one or more "MenuTitles" names. Select type of plug-in:
    - **imageFilter**: your plug-in will appear in the 'Image Filters' menu
    - **roiTool**: your plug-in will appear in the 'ROIs tools' menu
    - **other**: your plug-in will appear in the 'Others' menu
    - **fusionFilter**: your plug-in will appear in the 'Fusion' window
10. Edit the "InfoPlist.strings" file
11. Edit the "HelloWorldFilter.m" and "HelloWorldFilter.h": name your object to "HelloWorldFilter" in the header AND in the source file.
12. Build your filter! And you should now find your new plug-in in the "build" folder.



The new "build" folder

13. Now, we will create an **alias** of this plug-in in the OsiriX "Plugins" folder located in the "Library" folder. This will allow us to debug the plug-in when running OsiriX.
14. Click on the toolbar button "Build and Debug" to see if OsiriX is able to find your plug-in and if Xcode is able to debug it.
15. You are now ready to modify your new object "HelloWorldFilter" defined in the "HelloWorldFilter.m" and "HelloWorldFilter.h" files.

How to avoid the "symbols collision" problem?

Objective-C is dynamic language: contrary to C or C++, the name you gave to your objects, functions or variables are used during the runtime of OsiriX to locate a function/variables. This dynamic link allows a plug-in to access to any functions of OsiriX : a plug-in can really do anything in OsiriX!

However this dynamic link can create problems: the "symbols collision" problem. This problem occurs when two different plug-ins have two objects or two global variables with the same name... For example, if two different plug-ins create a sub-class of 'PluginFilter' object with the name 'MyPluginFilter', OsiriX will not be able to identify which object is needed to be created during the runtime: OsiriX will probably crash...

To avoid this "symbols collision" problem, it is recommended to use specific names for your objects and global variables. Don't use names like "MyPluginFilter" or "ROIPlugin", but instead use names like "CardiacEjectionPlugin", "AutomaticRegistrationROIPlugin", ...

Be specific!

How to create a plug-in with existing source code or libraries?

XCode is based on the powerful GCC compiler and the Java interpreter. That means you can mix in the SAME plug-in classic C, C++, Java, Objective-C, Objective-C++ and Java! Isn't amazing? And it works!

(The only advantage of Objective-C++ is to be able to call and create C++ objects in an Objective-C object)

For example, OsiriX is based on libraries in C++ (VTK, ITK, Offis), C (Papyrus), Java (PixelMed) and Objective-C/C++ (Cocoa).

Files extensions for all these languages:

".c"	C
".cc"	C++
".m"	Objective-C
".mm"	Objective-C++
".java"	Java

Find more information in the XCode documentation and on XCode web sites!

How to distribute your plug-in?

If you write a great plug-in, it would be a good idea to distribute it!

OsiriX is GNU General Public License (GPL) software: free and open-source application. For more information about a GPL software, visit the Free Software Foundation (FSF, http://www.gnu.org/) web site.

Your plug-in doesn't have to be GPL software: that means you can create commercial plug-ins and you don't have to release the source code of your plug-in.

However, it's great to create and distribute GPL plug-in, but you don't have to...

If you create a free plug-in (with or without the source code) you can upload it on the OsiriX web site, to be added to the 'Plug-ins' page: simply write an email to Antoine Rosset, rossetantoine@bluewin.ch.