Beginners Guide for iOS Program Development with XCode on OS X

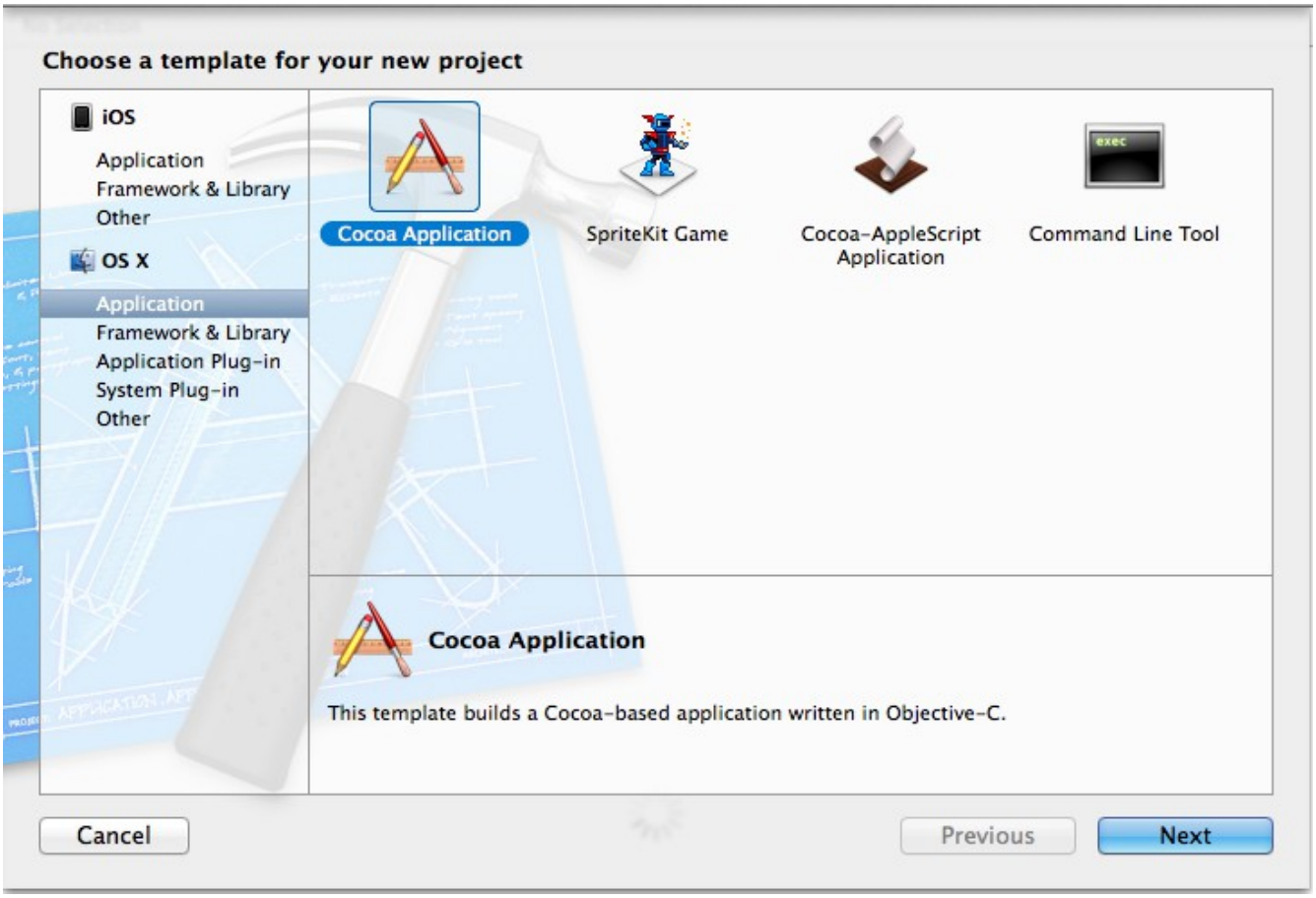
Part III: Objective-C language OS X GUI example

Christiaan Rogiers

Third Example – Objective-C in OS X GUI

After two examples showing text in a terminal it is time to look at an example showing text in a graphical user interface (GUI).

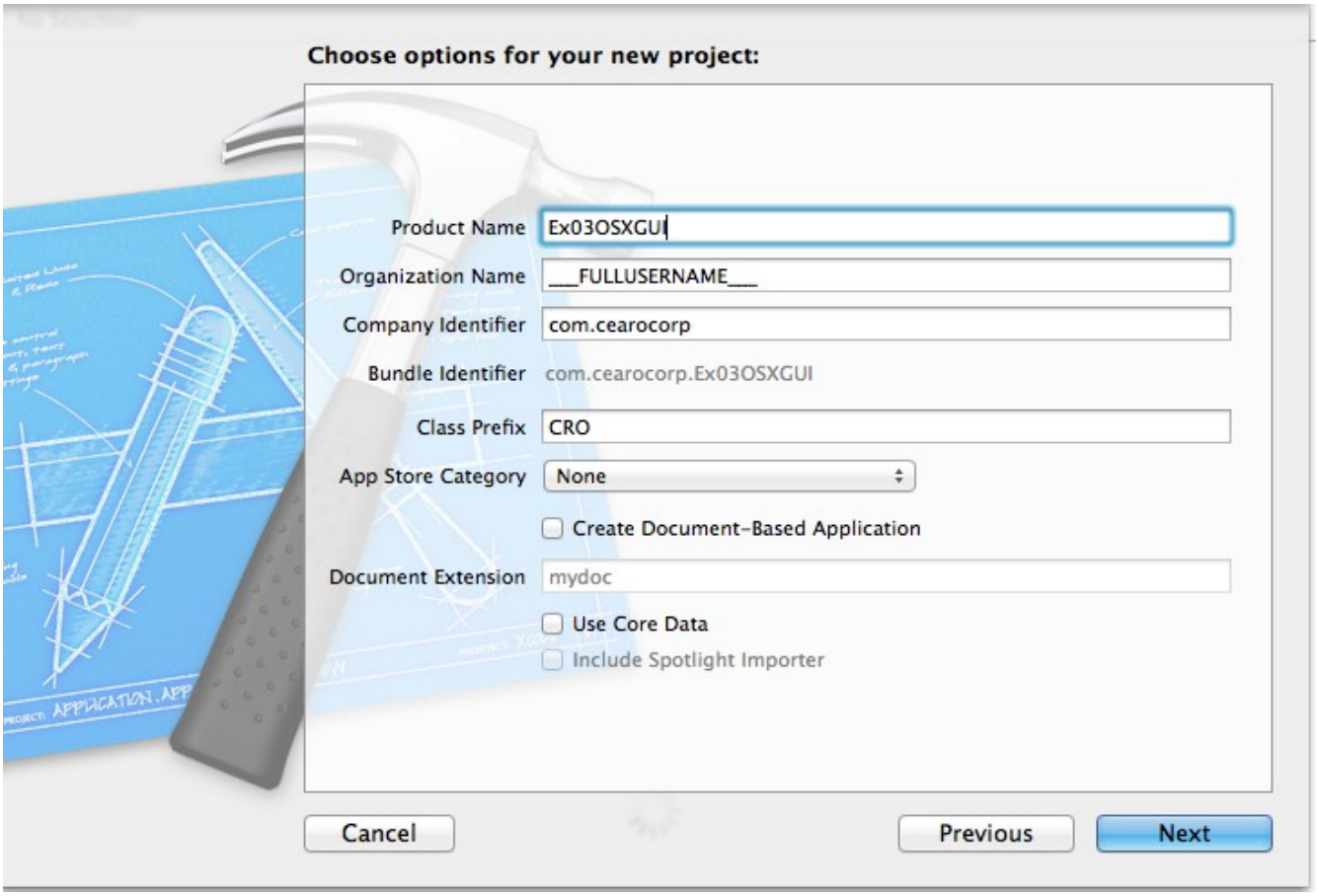
After we have gone through the usual 'File – New – Project' sequence, we get our 'Choose a template for your new project' screen.



So we choose 'Application' under 'OS X' and 'Cocoa Application'. And as the screen says, this will build a Cocoa-based application written in Objective-C. Cocoa being the GUI frameworks Apple derived from the NextStep platform.

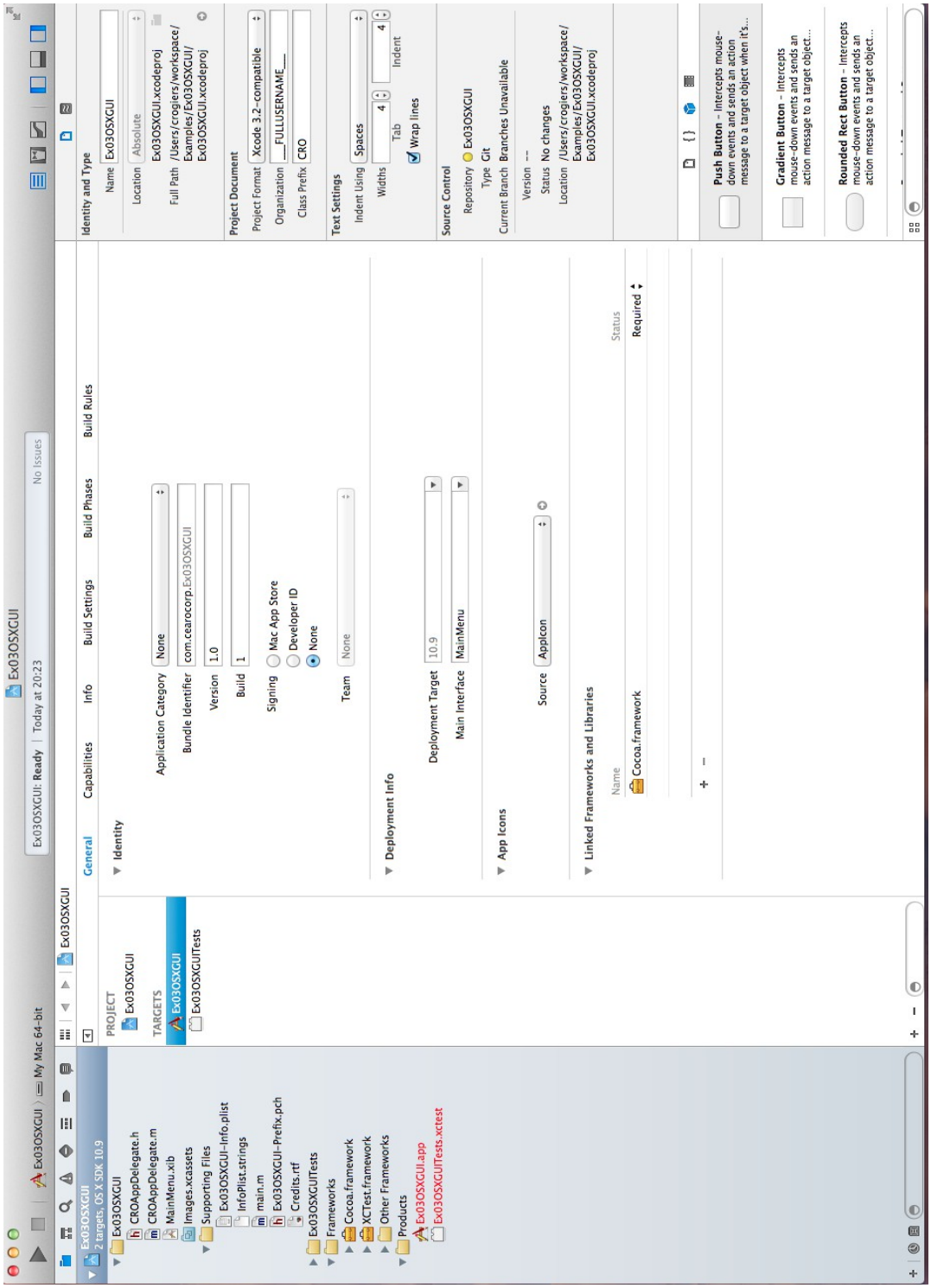
'Next'

We then get the 'Choose options for your new project' screen. So again we type in a 'Product Name'. 'Organization Name' we can leave as is. For 'Company Identifier' we can come up with something ourselves. The 'com.' with your company name behind it is a suggestion. It is the kind of things which may further along make it easier to distinguish between your own code and adopted code from elsewhere. Along the same lines, the class prefix is something of your own choice. Once we look at the code, you will see examples of your files and code which use it. This is a way of avoiding that you try to overwrite existing classes or functions by using existing names. The rest we leave as is, and 'Create'.



'Next' takes you to the screen which lets you decide where to save your project.

And on that screen 'Create' takes you to the overview screen, as it did in previous examples.


Ex03OSXGUI
Ex03OSXGUI Ready | Today at 20:23
No Issues
Ex03OSXGUI
My Mac 64-bit
Ex03OSXGUI
2 targets, OS X SDK 10.9
Ex03OSXGUI
CROAppDelegate.h
CROAppDelegate.m
MainMenu.xib
Images.xcassets
Supporting Files
Ex03OSXGUI-Info.plist
InfoPlist.strings
main.m
Ex03OSXGUI-Prefix.pch
Credits.rtf
Ex03OSXGUITests
Frameworks
Cocoa.framework
XCTest.framework
Other Frameworks
Products
Ex03OSXGUI.app
Ex03OSXGUITests.xctest
PROJECT
Ex03OSXGUI
TARGETS
Ex03OSXGUI
Ex03OSXGUITests
General
Capabilities
Info
Build Settings
Build Phases
Build Rules
Identity
Application Category
None
Bundle Identifier
com.cearocorp.Ex03OSXGUI
Version
1.0
Build
1
Signing
Mac App Store
Developer ID
None
Team
None
Deployment Info
Deployment Target
10.9
Main Interface
MainMenu
App Icons
Source
AppIcon
Linked Frameworks and Libraries
Name
Status
Cocoa.framework
Required
Identity and Type
Name
Ex03OSXGUI
Location
Absolute
Ex03OSXGUI.xcodeproj
Full Path
/Users/crogiers/workspace/Examples/Ex03OSXGUI/Ex03OSXGUI.xcodeproj
Project Document
Project Format
Xcode 3.2-compatible
Organization
__FULLUSERNAME__
Class Prefix
CRO
Text Settings
Indent Using
Spaces
Widths
4
4
Tab
Indent
Wrap lines
Source Control
Repository
Ex03OSXGUI
Type
Git
Current Branch
Branches Unavailable
Version
--
Status
No changes
Location
/Users/crogiers/workspace/Examples/Ex03OSXGUI/Ex03OSXGUI.xcodeproj
Push Button – Intercepts mouse-down events and sends an action message to a target object when it's...
Gradient Button – Intercepts mouse-down events and sends an action message to a target object...
Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...

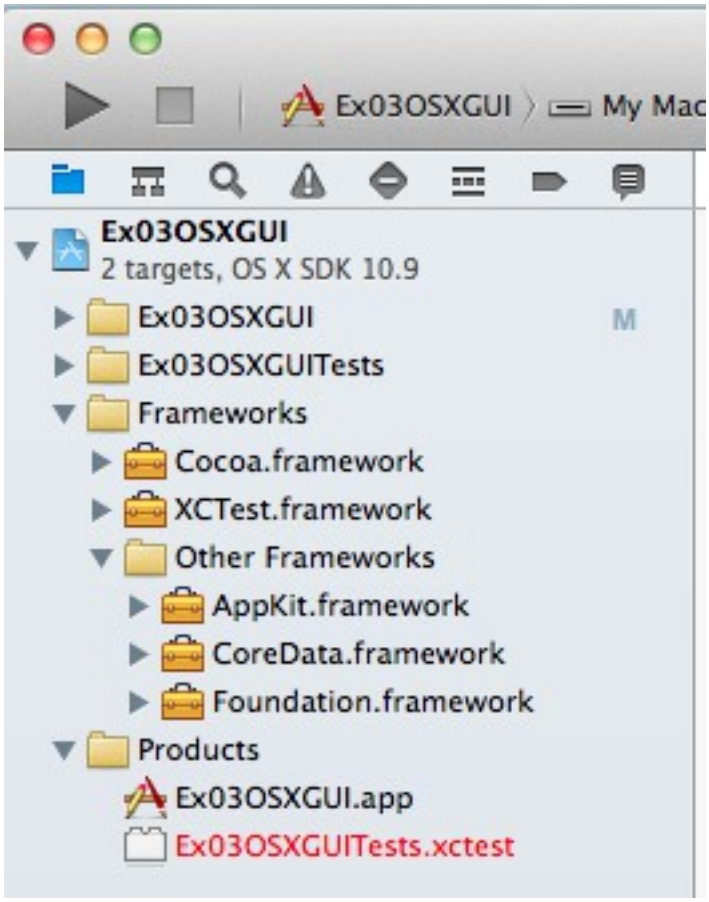
And this is where it becomes very obvious that a GUI does require a lot more code. If you expand the folder with the name of your project in the left navigation pane, you see two examples of files where the prefix we have set, is used.

In our example that is the 'CROAppDelegate.h' header file and its corresponding 'CROAppDelegate.m' source code file. As mentioned in the previous example, the '.m' clearly shows this is Objective-C.

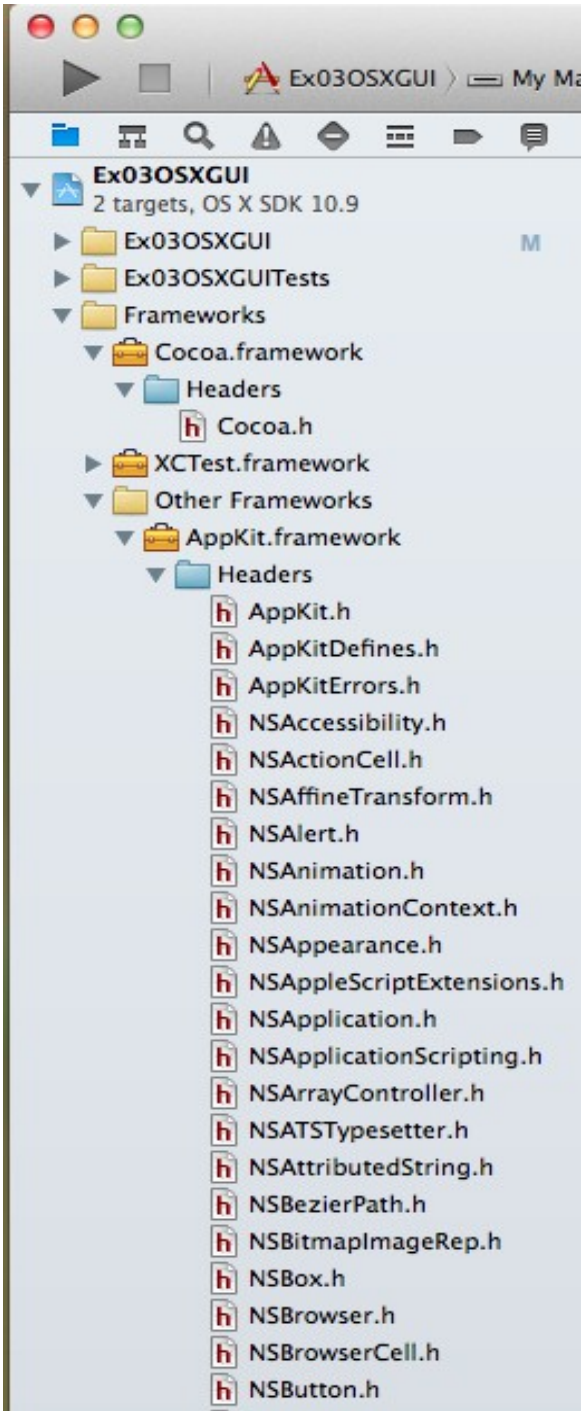
We also see a 'MainMenu.xib' file, which we will use later. It contains the visual elements which make this a GUI application, and to which we will later add visual elements of our own making, such as the text we want to show.

Under 'Supporting Files' we see the 'main.m' we also saw in the previous example.

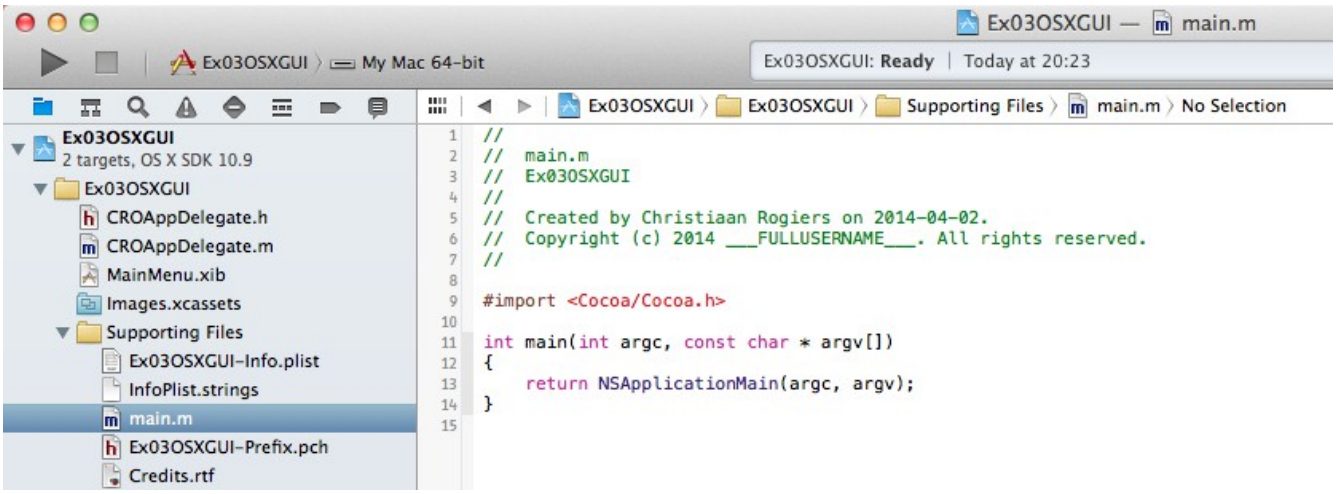
It is under 'Frameworks' that we see clearly how much more is involved in a GUI application. If you open the 'Other Frameworks' you see another three frameworks.



You want another example of how much there is available in frameworks? Just expand the AppKit.framework, followed by its 'Headers'. By the way the 'NS' in front of these header files is another example of a prefix used to distinguish between these and similar files but from other frameworks or our own code. 'NS' shows again where Apple got some of this from: it stands for NextStep.



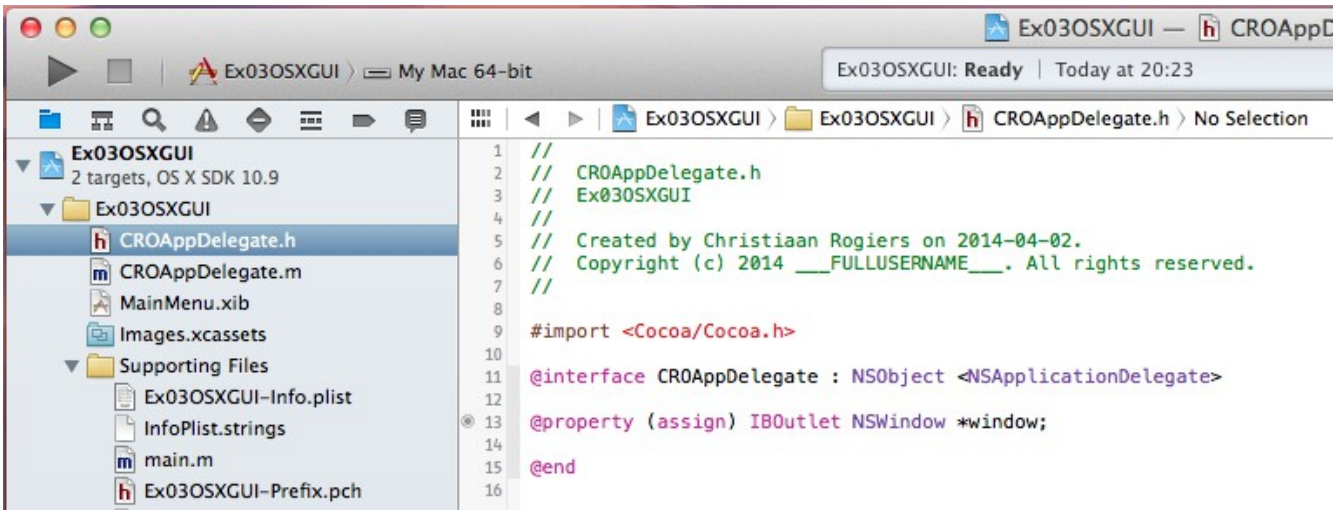
Let's start with a look at the main.m file:



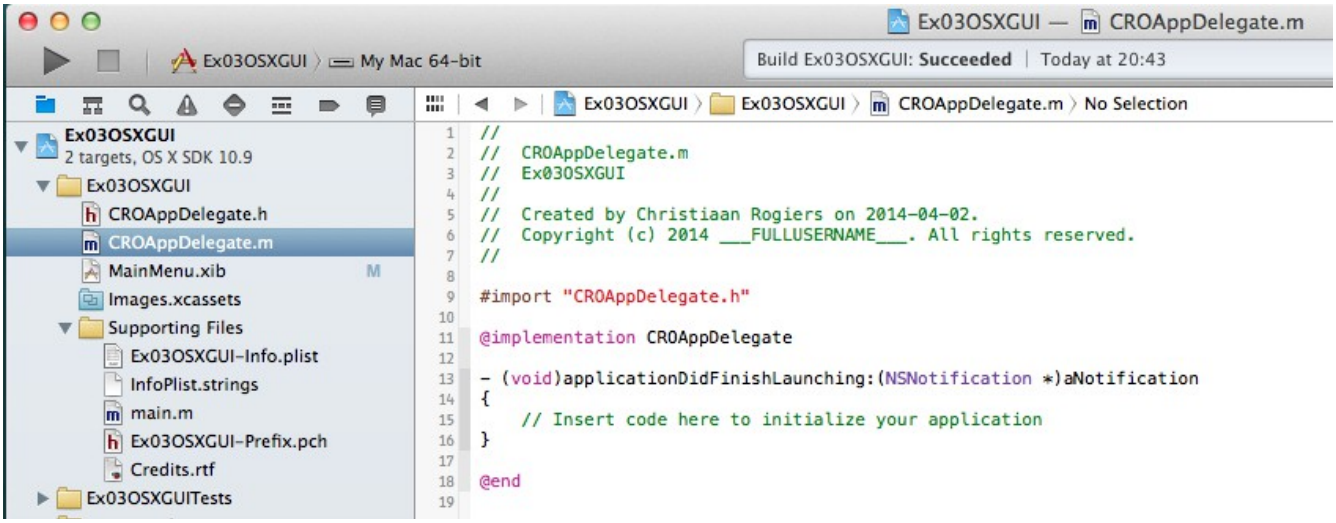
This time the '#import' goes with '<Cocoa/Cocoa.h>'.

The main function is there as usual, but rather than return a 0 or 1, it returns NSApplicationMain(argc, argv).

The CROAppDelegate.h header file looks like this:



And the CROAppDelegate.m source code file that goes with it looks like this:



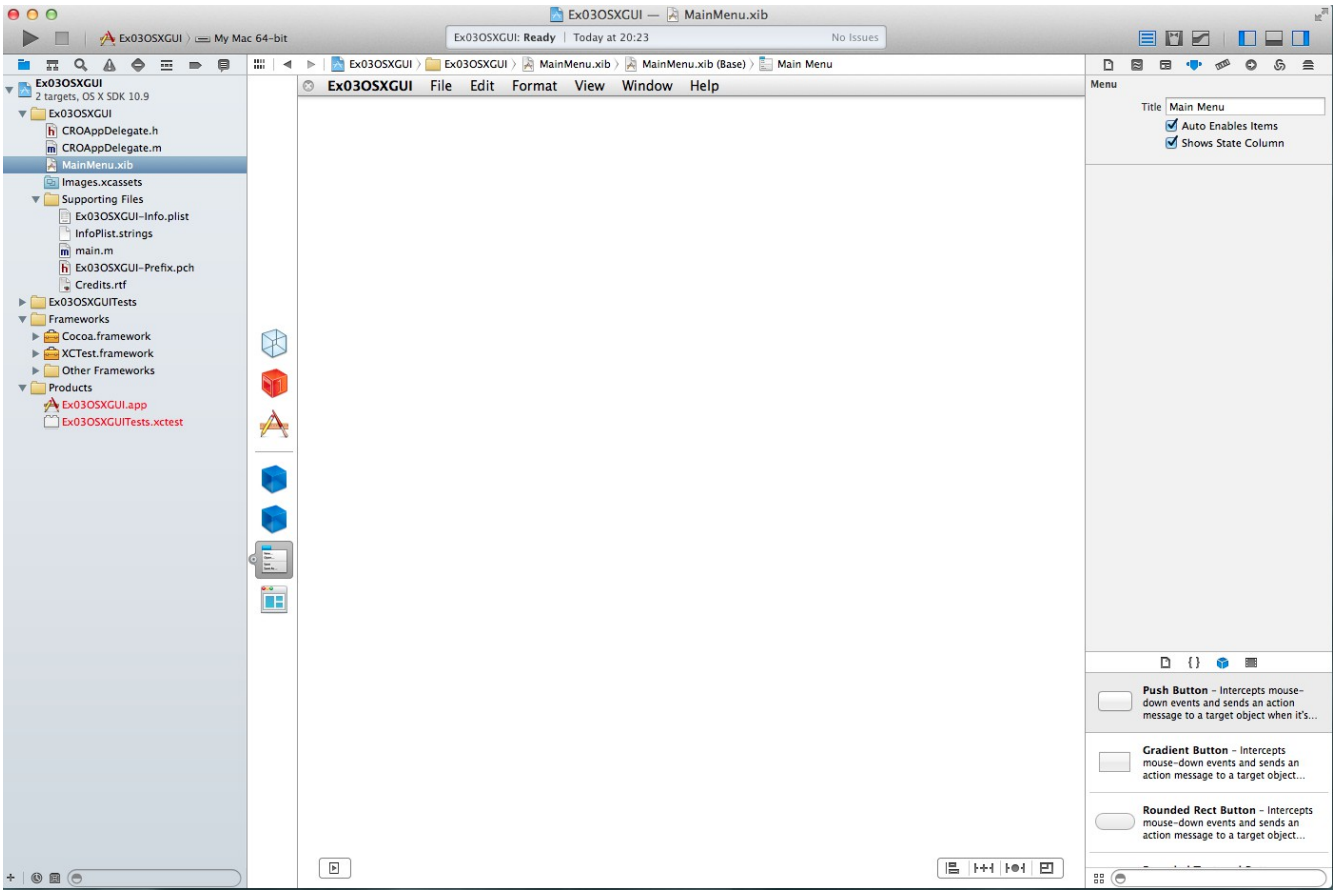
What you see here is typical for all Objective-C programs and their OOP (object oriented programming) way of doing things.

In the '.h' header file we see '@interface', where the CROAppDelegate class is derived from the NSObject class, and we see '@property'. The property in this case is an NSWindow which goes by the name of 'window'. The '*' in '*window' mains that we are giving the address (or pointer) to 'window' rather than the window object itself. Note we only have properties here, but no methods. The interface ends with an '@end'.

In the '.m' source code file, we import the corresponding header file. We see '@implementation' which gives us (in this case) one method that goes with our CROAppDelegate. The implementation ends with an '@end'.

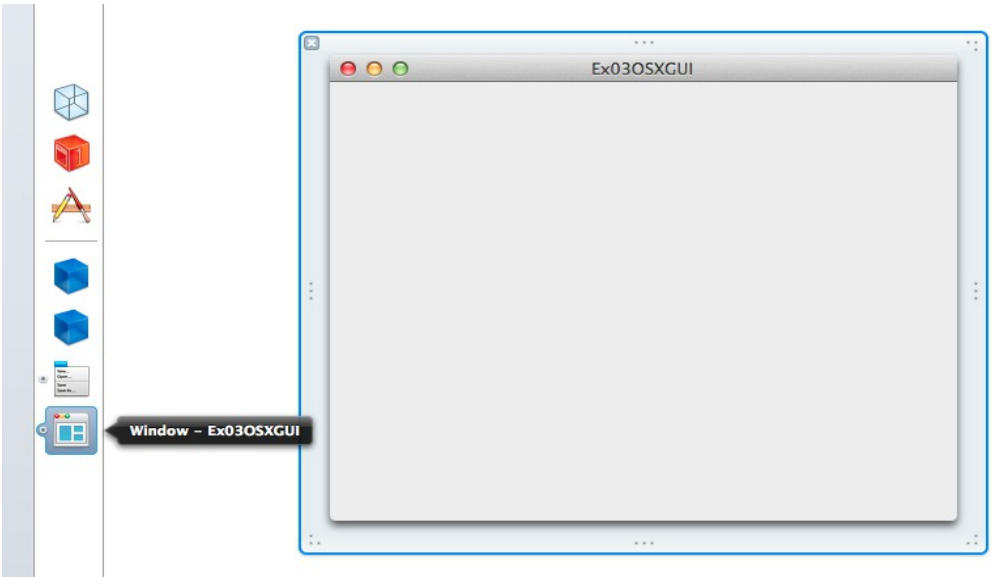
What all this means is that we define our classes in header files and say there what properties they have. It is what we call the interface. The nitty gritty of providing the source code to do something (methods) with a class sits in the '.m' files which we refer to as the implementation.

And now it is time for some real GUI magic. Therefore we open the 'MainMenu.xib' file:

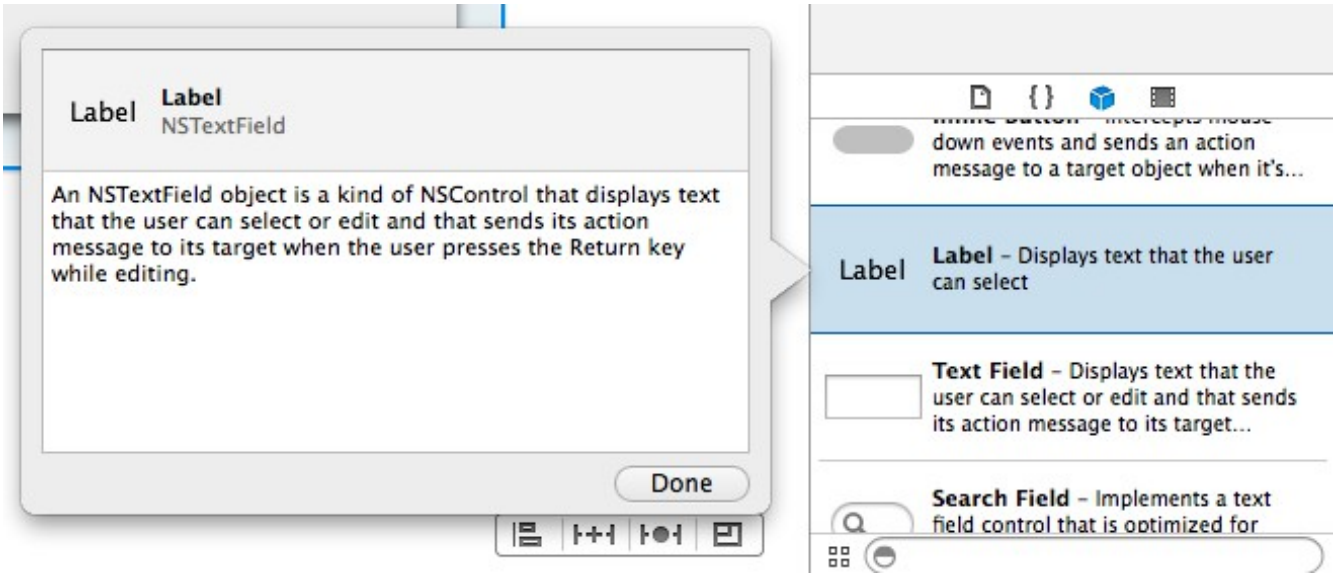


Here we have our drag-and-drop graphical editor, which allows us create a scene.

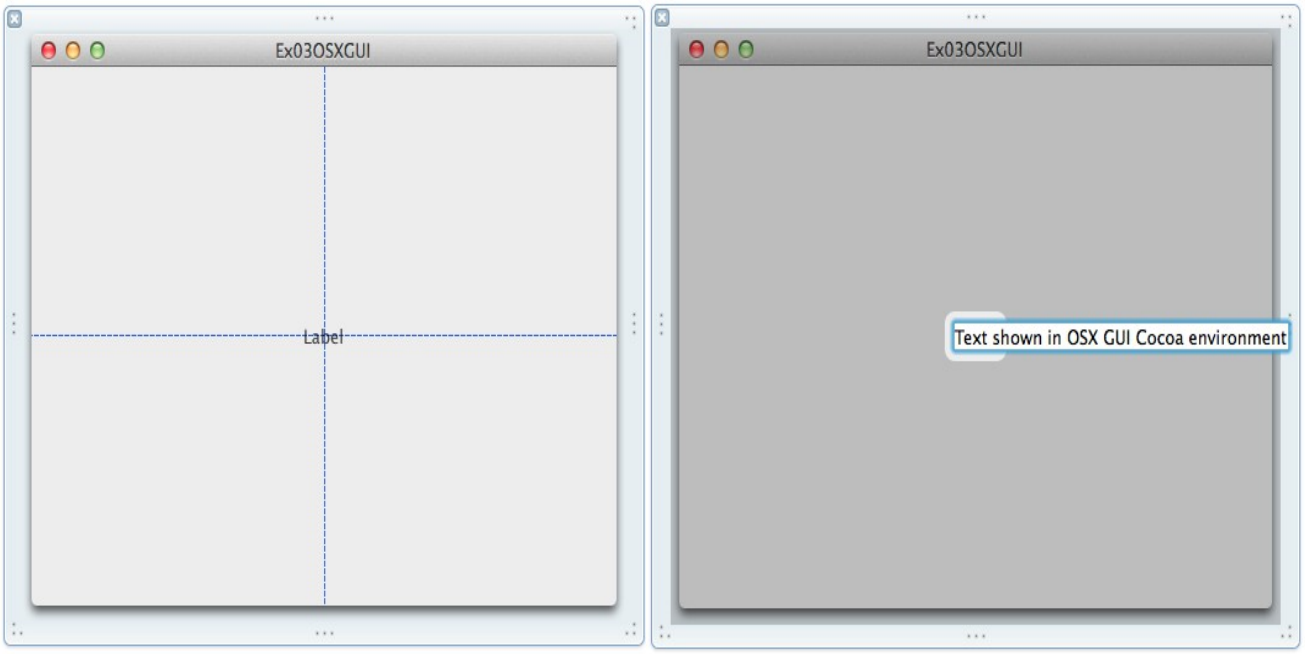
From the symbols on the left of this middle editor pane, we drag the Window symbol onto our scene. That way we get our program window showing.



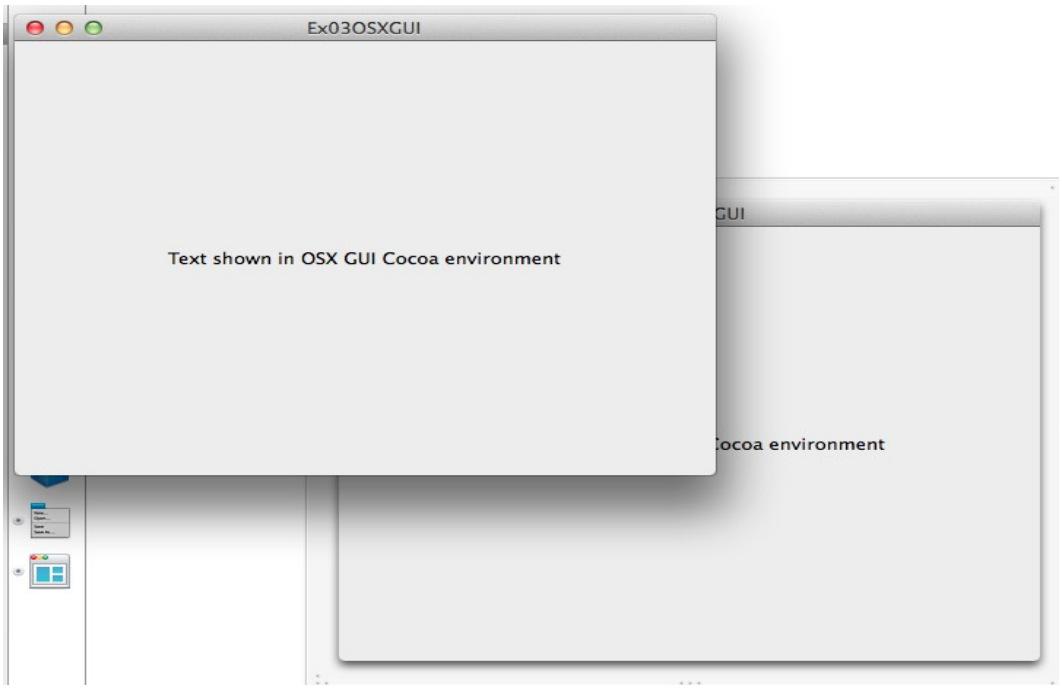
Then we go to the list of possible elements on the bottom right and drag a label on top of our scene window.



And we can then change the label text to the text we want to show:



Now we can use the little triangular play button, to let XCode do its magic. It will compile and link all necessary code and run our program. This program being an OS X GUI application, it opens its own window and shows our text in it.



That's all folks, for this example.