Beginners Guide for iOS Program Development with XCode on OS X

Part II: Objective-C language CLI example

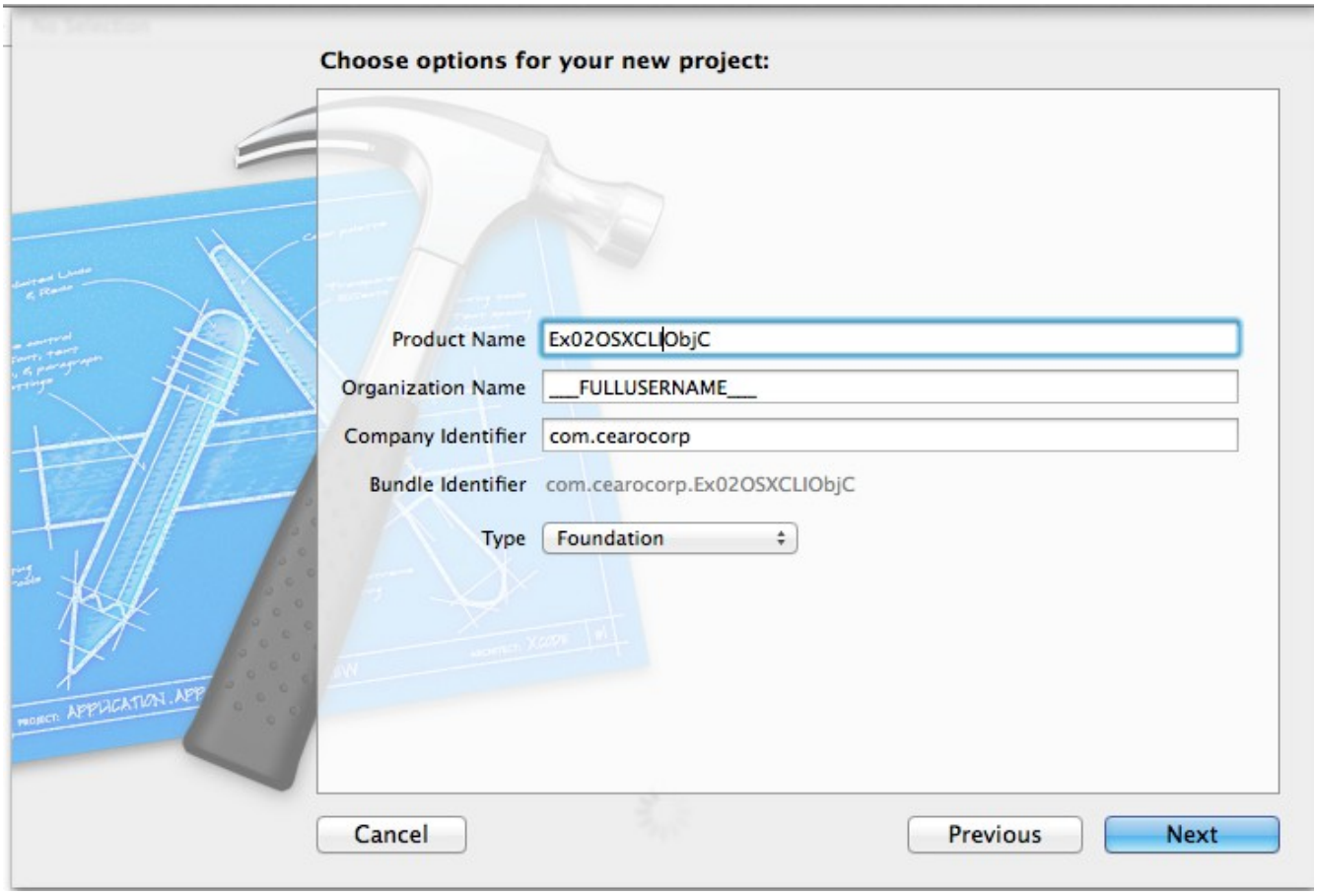
Christiaan Rogiers

Second Example – Objective-C in terminal

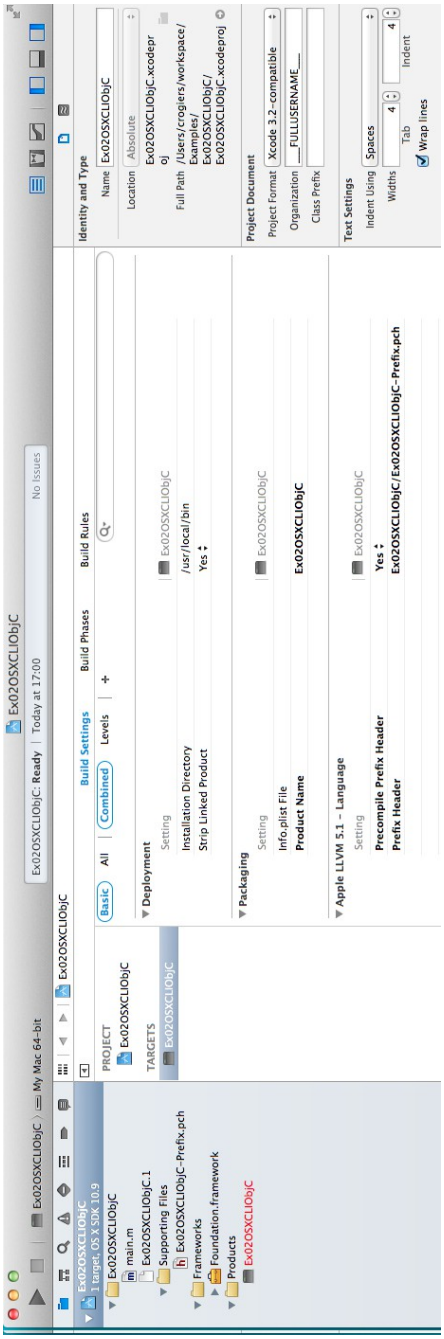
In our first example in Part I, we showed how we quickly could get a C language program running in a terminal and showing a line of text. The tools used to do this were all part of the Apple XCode IDE (Integrated Development Environment).

In this second part, we are going to do the same but in Objective-C. Most of it will be the same as in the first example, so we might skip a few screen shots.

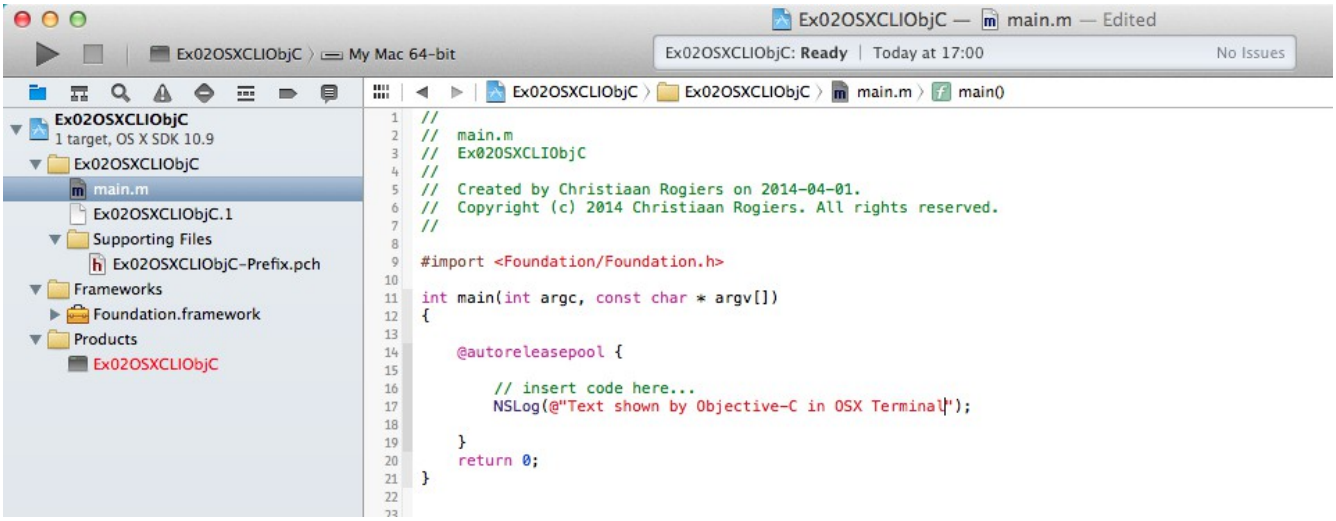
After starting with the same 'New – Project' sequence, we get the same 'Choose options for your new project' screen:



But this time we choose 'Foundation' as 'Type', and NOT 'C'.



This does look very similar to what we got in the first example. But there are differences: instead of 'main.c', we got a 'main.m' (the .m being typical for Objective-C source code files) and under 'Frameworks' you see 'Foundation.framework'. Let's open the 'main.m' file now:



In the 'main.m' file we see the '#include' was replaced by '#import' and the '<stdio.h>' was replaced by '<Foundation/Foundation.h>'. So we are using the Objective-C Foundation library and header file, rather than the C language stdio library and header file.

Again we see a main function, returning an integer and possibly taking some arguments (which we do not use).

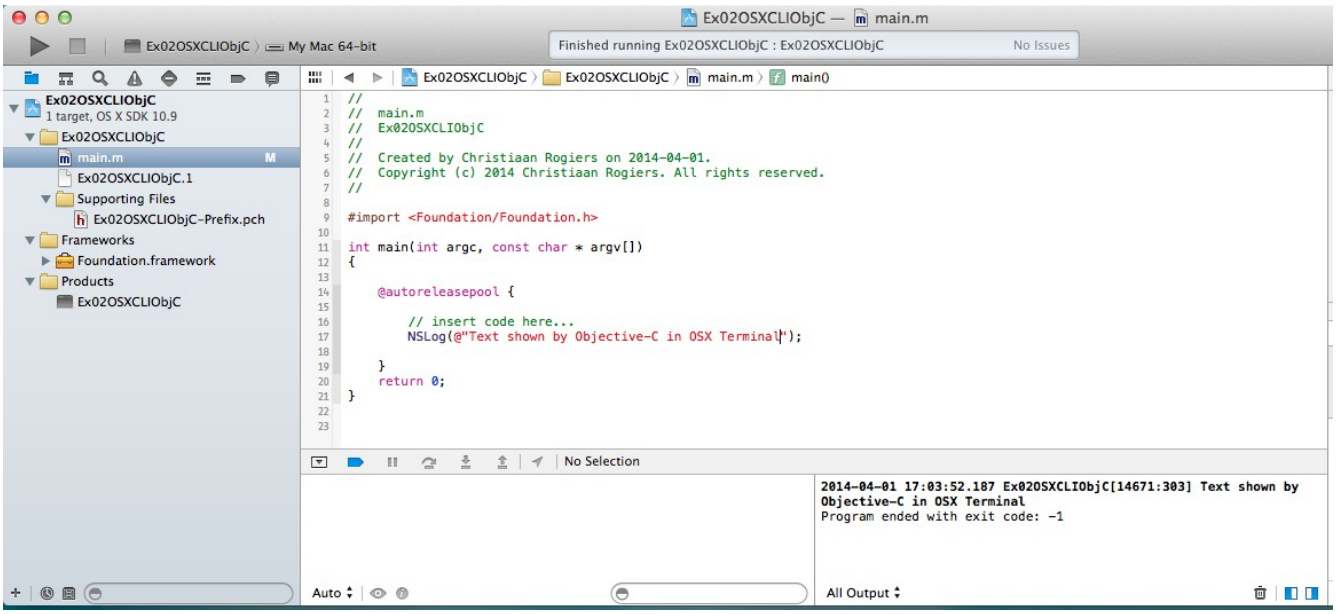
But the code inside the main function before the 'return' statement is different.

First of all it is enclosed in a '@autoreleasepool' function which surrounds it with curly brackets. This function is provided by Apple for dealing with memory management. Explaining that in detail would go beyond the scope of this guide. Just know that 'garbage collection' when dealing with memory allocation and release is an important, but not easy aspect, of programming in most languages.

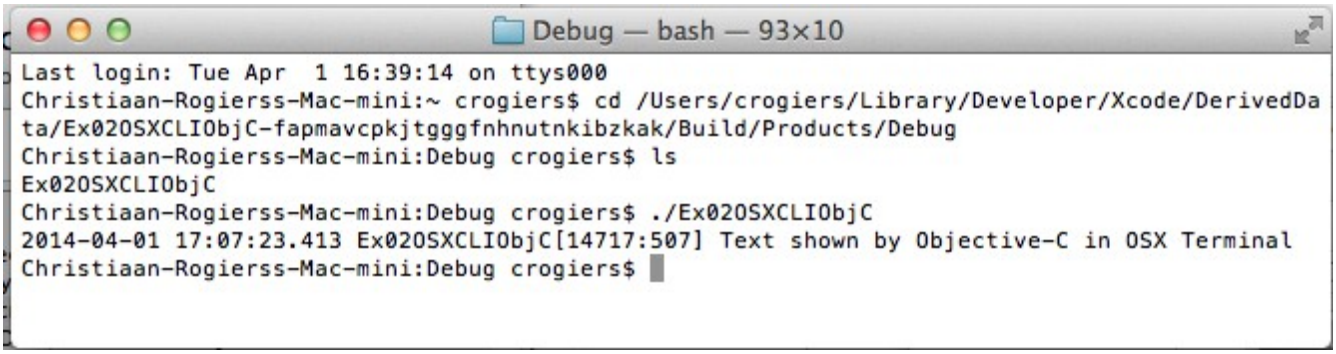
The 'printf' function from stdio is replaced with the NSLog function from the Foundation library. Again we replaced the 'Hello, World!' by our own text. But now it is preceded by a '@' symbol. Apparently this is needed for the compiler to know that this is an Objective-C type string and not a C++ string.

Ready to run.

This is the result within XCode showing in the lower right corner.



And this is the result when you run it in a terminal, which you could do exactly the same way we did in the first example.



And this concludes the second example.