Xcode Notes for EECS 381

Revised 1/23/09

David Kieras, University of Michigan

Mac OS X is based on Unix, and if you download and install the free developer tools package, you get a complete Unix GNU toolset for programming - open a Terminal window, and start using makefiles and gcc! You can also use the Xcode IDE, with its excellent project management support, great editing environment, and valuable graphical debugger.

However, like MS's Visual Studio, Xcode is intended to support complex application development, making it at least as confusing as Visual Studio for the relatively simple projects in this course. Like all such tools, there is a learning curve to scramble up, with many details to learn.

These notes focus only on things which are specific to setting up Xcode for use in this course; it is not a tutorial on using Xcode.

Setting up an Xcode 3.x Standard C Project

First, create a project of the appropriate type:

1. Start XCode.
2. Select File/New Project ...
3. In the Assistant left-hand window, look for (may need to scroll down to) and select “Command Line Utility” and then select "Standard Tool” for a C project, then Choose.
4. Give the project a name, like "P2", and select a directory for the project files to reside in, then Finish. Note that the executable will be named the same as the project name.

Second, set the project settings to tell the gcc compiler/linker to behave close to Standard, improve debugging and performance, and eliminate some irrelevant stuff (this step is highly recommended):

1. Select Project/Edit Project Settings.
2. In the Project Info window, select tab: Build.
3. Select Configurations: All Configurations.
4. Select Show: All Settings.
4. From the list under Architectures, I suggest setting Architectures to "Native Architecture of Build Machine (use the selecting list under the tiny buttons on the right).
5. Under Build Options, I suggest setting Debug Information Format to DWARF (only).
6. Scan down to Code Generation and set the following:
Accelerated Objective-C Dispatch - Off
Generate Debug Symbols - On
Optimization Level - None (best for debugging)

7. Scan further to Language and do the following settings:
Allow 'asm' etc - OFF

C Language Dialect - use the pop-up to select C89
Codewarrior-style inline etc - OFF
Compile Sources as - use the pop-up to select C.
Enable C++ Exceptions - OFF
Enable C++ Runtime Types - OFF
Enable Objective-C Exceptions - OFF
Recognize Pascal Strings - OFF

8. Scroll down to Warnings, and set the following:
Incomplete Objective-C Protocols - OFF
Pedantic Warnings - ON - this help result in a "more Standard" C compile
You turn on other warnings as you wish: I suggest the following:
Check Switch Statements, Mismatch Return Type, Missing Braces, Missing Newline, Sign Comparison, Unused Variables.

9. Tweak the optimization. Go back to the Configuration: Selector at the top of the info window and select Release. Find the Optimization Level under Code Generation, and set it to something else, such as Fastest, or Fastest, Smallest. Normally you develop in Debug, but then set to Release to produce a "production" version of the program.

10. Close the info window.
11. Do a Clean All, then rebuild.

Setting up an Xcode 3.x Standard C++ Project

There is a problem which often surfaces in Project 2. It results from a combination of how Mac OS's legacy file system treats file names in a case-preserving, but case-insensitive way, together with an overly-complex scheme in Xcode for reducing compile times by trying to be "smart" about header files with "headermaps". The result is that Xcode behaves differently than almost every other C/C++ programming environment with how header files are searched for and found - this is a serious problem for cross-platform programming.

For purposes of this course, the problem shows up because with the default settings, Xcode will get seriously confused if you #include a file named "String.h" that differs only in case from another file, named "string.h", even though they differ drastically in where they are - one can be a programmer-created file and the other can be part of the "system" - the Standard library - it makes no difference, they still get confused. The symptom in this case is a trillion errors having to do with functions in <string.h> not being recognized.

Fortunately, there is a workaround, but Apple has shamefully poorly documented it and has not yet fixed the problem. There is very little information about why the workaround works, because the "headermaps" are supposed to be invisible to the user (yeah, right!), but it appears to work for Xcode version 3.x and earlier 2.x. If you don't anticipate this problem (because your header names won't conflict with library header names) you don't need to fix the "headermaps" problem - you can skip that step in the recipe below.

First, create a project of the appropriate type:

1. Start XCode.
2. Select File/New Project ...
3. In the Assistant left-hand window, look for (may need to scroll down to) and select “Command Line Utility” and then select "C++ Tool” for a C++ project, then Choose.
4. Give the project a name, like "P2", and select a directory for the project files to reside in, then Finish. Note that the executable will be named the same as the project name.

Second, fix the headermaps problem (this step is optional).

1. In the main project window, select the target (like "P2") under the Targets icon.
2. Do get info (cmd-I).
3. In the resulting Target Info window, select the tab: Build.
4. Select Configuration: All Configurations.
5. Select Show: User-defined Settings.
6. Hit the button with the gear-wheel "Action" icon and select add a new setting.
7. Enter the name of the setting:
USE_SEPARATE_HEADERMAPS
8. Enter the value of the setting:
YES
(You can change a settings value by selecting the setting line and hitting Edit.)
9. Close the Info Window
10. IMPORTANT: CLOSE THE PROJECT!!! (File/Close Project).
11. Reopen the project.
12. Under Build, select Clean All (I'm not sure if this is needed).

Third, set the project settings to tell the gcc compiler/linker to behave close to Standard, improve debugging and performance, and eliminate some irrelevant stuff (this step is highly recommended):

1. Select Project/Edit Project Settings.
2. In the Project Info window, select tab: Build.
3. Select Configurations: All Configurations.
4. Select Show: All Settings.
4. From the list under Architectures, I suggest setting Architectures to "Native Architecture of Build Machine (use the selecting list under the tiny buttons on the right).
5. Under Build Options, I suggest setting Debug Information Format to DWARF (only).
6. Scan down to Code Generation and set the following:
Accelerated Objective-C Dispatch - Off
Generate Debug Symbols - On
Optimization Level - None (best for debugging)

7. Scan further to Language and do the following settings:
Allow 'asm' etc - OFF
C Language Dialect - use the pop-up to select ANSI-C even for C++ - yes, poor labels

Codewarrior-style inline etc - OFF
Compile Sources as - use the pop-up to select C++.
Enable Objective-C Exceptions - OFF
Set Other C++ Flags to the value: -fno-elide-constructors
This will turn off current optimizations in order to get a more "textbook" sequence of calls to copy constructors - this is recommended only for learning purposes in Project 2.
Recognize Pascal Strings - OFF

8. Scroll down to Warnings, and set the following:
Incomplete Objective-C Protocols - OFF
Pedantic Warnings - ON - this help result in a "more Standard" C++ compile
You turn on other warnings as you wish: I suggest the following:
Check Switch Statements, Mismatch Return Type, Missing Braces, Missing Newline, Sign Comparison, Unused Variables.

9. Tweak the optimization. Go back to the Configuration: Selector at the top of the info window and select Release. Find the Optimization Level under Code Generation, and set it to something else, such as Fastest, or Fastest, Smallest. Normally you develop in Debug, but then set to Release to produce a "production" version of the program.

10. Close the info window.
11. Do a Clean All, then rebuild.

Making Your Own Project Templates

If you don't want to go through the above exercise every time, you can create your own version of the project templates. As long as you work starting from a copy of the originals, you can't mess up the original templates.

1. Locate the current templates, and make a copy of one to modify.
For Xcode 3.1, the templates are in either:
/Users/<yourname>/Library/Application\ Support/Developer/Shared/Xcode/Project\ Templates
(preferred)
or
/Developer/Library/Xcode/Project\ Templates
(machine wide)

You may want to create a directory like "Custom Projects" to hold your own project templates - take care that they are preserved if you do a reinstall or update of Xcode. Xcode 3.1 separates them out into User templates for you.

2. You'll change the name later. Open the copied template with Xcode (double click on it).

3. Modify the project settings as needed, but don't modify anything with placeholders of the form <<something>> like "<<PROJECTNAME>>.
If you want a different set of starter files, create them and add them to the project.

4. Test build the project, double-check that you like the settings and the starter files, then clean all targets.

5. Close the project.

6. Open the project directory and Trash the build folder.

7. Control-click on the .xcodeproj file and chose Show Package Contents.
Trash the files with names like username.mode1 and username.pbxuser.

8. Open TemplateInfo.plist and modify FilesToMacroExpand to include the filenames for the "starter" files that you want to be in the project - only if you changed the starter set.

9. Change the template (package/folder) name to the name you want for it this is how it will look in the project template display.

10. Place the template file where you want it - see #1 above for locations.

11. Quit Xcode, restart it, then create a project using the template to test it.