Code Organization in Xcode Projects

Adrian Kosmaczewski

2009-07-28

Xcode does not impose any structure to your source code tree. This is both cool and useful to quickly throw a couple of lines for a prototype, but in my experience, this approach does not scale. More often than not, without any hygiene, your project can become a mess. Just using Xcode defaults, after a while your resources will sit beside your .xcodeproj file, all the project classes will be thrown together in the Classes folder, and if you have a relatively large project, this approach makes finding individual files painful.

Of course, Xcode provides "Groups" to organize your source code, but the idea is to be able to quickly identify the different kind of files that make up your Xcode project, either for Mac or for the iPhone, without having to open the Xcode project file. This means having both a folder structure, and an internal source code file structure. All of this will help you maintain your project in the future, which means cheaper costs, and less time spent looking for bugs.

All of this is also particularly useful when browsing projects via Google Code, Github or any other kind of file view of source code repositories. If your code is organized in a nice folder structure, it is easier to explore than if all the files sit in the same folder.

In this post I will enumerate some best practices that I use in all of my projects.

So let's say that you start a new Xcode project. Here's the Xcode window that is presented to you (seen in "Condensed" mode, which is the one I prefer):

This is the project layout as seen in the Finder:

As you can see (and as you might have experienced), in the default layout used by Xcode, all new source code files will be thrown into the "Classes" folder, while all the new Resources will be just stored in the project root. In the long term, this layout can be really painful to deal with. So let's just start rearranging things a bit:

Organize source files in folders and mirror them in Xcode

When I start a new Xcode project, I usually do the following:

1. I remove the “Classes” group (just deleting references, not moving the items to the trash, as shown in the image below)
2. Then I add the following subfolders to the “Classes” folder:
   - AppDelegate
   - Controllers
   - Models
   - Helpers
3. Finally, drag the enhanced “Classes” folder from the Finder to the Xcode project window, asking to “recursively create groups for every subfolder”:

Create separate folders for different Resource elements

The next step is to actually create a resource folder in Finder, and add a subfolder for every kind of resource that my project will use: sounds, images, SQLite databases, NIBs, etc. Then I do the following:

1. Remove the Resource group from the Xcode project (just deleting references)
2. Then I drag the newly created “Resources” folder from the Finder to the Xcode project window, asking to “recursively create groups for every subfolder”, like we did for the “Classes” folder.

Doing this has an interesting side effect: when you localize your application in other languages, each folder will contain a subfolder with the localized resources inside (for example, an “en.lproj” for English, “es.lproj” for Spanish, and so on).

Organize your code consistently

Each @implementation *.m file should always present methods in this order:

1. init and dealloc
2. public methods
3. public @dynamic properties
4. delegate methods (for each supported protocol)
5. private methods

Use #pragma statements to separate the regions shown above

Each logic group of methods should be separated from each other using the following lines (just type “#p” and hit the TAB key in Xcode!)

```
// ...
    return cell;
}

#pragma mark -
#pragma mark UIAlertViewDelegate methods
```

```
- (void)alertView:(UIAlertView *)alertView
        clickedButtonAtIndex:(NSInteger)buttonIndex
{
// ...
```

The advantage of this approach is that later, you can use those #pragma marks to generate an automatic layout in the symbols pop-up of Xcode:

You can get this pop-up window clicking on this sector of your Xcode window:

Only leave public methods in the header files

This means putting private methods definitions in a (Private) category on top of the *.m file. This will remove all compiler warnings (about "this class might not respond to this selector") and will cleanly separate what's public from what's not:

```
#import "UntitledViewController.h"

@interface UntitledViewController (Private)
- (id)returnPrivateObject;
- (void)changeInternalState:(NSString *)param;
@end

@implementation UntitledViewController

- (id)init
{
```

Use consistent coding conventions

You can use my own, if you wish.

Treat warnings as errors

I've said a lot about that in a previous post, but here's a quick reminder:

Create "Distribution" configurations for the new project

Duplicate the "Release" configuration and create two new ones: "Distribution Ad Hoc" and "Distribution App Store". Each one will have to be configured with their corresponding provisioning profiles.

Add an "Entiitlements.plist" file to the project

Remember to uncheck (disable, turn off) the "get-task-allow" value (I still don't understand why every Xcode project does not create this file automatically).

Then add the “Entitlements.plist” value to the corresponding key in the “distribution” configurations created in the previous step.

Use source control

As soon as your project source tree is ready, commit it to your repository, whichever this is.

Conclusion

This is how the final project might look like:

Of course, you might as well enforce the above best practices using your own default project templates; depending on your requirements, this might be a useful thing to do. You must store those new Xcode templates in the following locations:

- **iPhone:** /Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Project Templates
- **Mac:** /Developer/Library/Xcode/Project Templates

Hope this helps! As usual, feel free to add your comments, best practices, rants and other reactions in the comments section below.