A Quick Tour of the Foundation Kit

You’ve already seen that Objective-C is a pretty nifty language, and we haven’t even finished exploring all the features it has to offer. For now, we’re going to take a quick side trip and have a look at Cocoa’s Foundation framework. Although strictly part of Cocoa and not built in to Objective-C, the Foundation framework is so important that we thought it worth exploring in this book.

As you saw in Chapter 2, Cocoa is actually composed of many different frameworks. Of these, the ones most used for desktop (OS X) applications are Foundation and Application Kit. The Application Kit has all the user interface objects and high-level classes. You’ll get a taste of the AppKit (as the cool kids call it) in Chapter 16.

If you are going to be developing applications for iOS, then you will be working with the User Interface Kit (UIKit). We will give a 10,000–foot overview of UIKit in Chapter 15. UIKit is considered parallel to AppKit for iOS. It has all the interface objects for iOS applications.

Solid Foundation

Foundation, as its name implies, is the foundation for both the UI frameworks, and because it has no UI objects, you can use the same objects for iOS and OS X applications.

Foundation framework has a bunch of useful low-level, data-oriented classes and types. We’ll be visiting a number of these, such as NSString, NSArray, NSEnumerator, and NSNumber. Foundation has more than a hundred classes, all of which you can explore by looking at the documentation installed with Xcode. You can read the documentation in Xcode’s Organizer window by selecting the “Documentation” item in the toolbar.

Foundation framework is built on top of another framework called CoreFoundation. CoreFoundation is written purely in C, and you can use it if you desire, but we will not be discussing it in this book. Don’t be confused when talking about two different frameworks that
might have similar names. If you come across function names or variable names that start with “CF,” they are part of CoreFoundation. Most of them have equivalents in Foundation framework, and some of them can be easily converted from one to the other.

**Using the Project Boilerplate Code**

Before we continue, here’s a note about the projects for this chapter and for the rest of this book. We’ll still be making Foundation tool projects, but we’ll leave in the boilerplate code, which follows:

```c
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool
    {
        // insert code here...
        NSLog(@"Hello, World!");
    } return 0;
}
```

Take a look through this code. `main()` starts with the keyword `@autoreleasepool`, and all the Cocoa code is written within the brackets that appear between the keyword and the return statement. This is a sneak preview of Cocoa memory management, which we’ll discuss in the next chapter. For now, please just nod, smile, and leave the `@autoreleasepool` stuff in there. If you take it out, you won’t hurt yourself, but you’ll get some very strange messages when you run your programs.

**Some Useful Types**

Before digging into real live Foundation classes, let’s take a look at some structs that Cocoa provides for our benefit.

**Home on the Range**

The first structure is `NSRange`:

```c
typedef struct _NSRange
{
    unsigned int location;
    unsigned int length;
} NSRange;
```

This structure is used to represent a range of things, usually a range of characters in a string or a range of items in an array. The `location` field holds the starting position of the range, and `length` is the number of elements in the range. For the string “Objective-C is a cool language”, the word “cool” can be described by the range that starts at location 17 and has length 4. `location` can have the value `NSNotFound` to indicate that the range doesn’t refer to anything, probably because it’s uninitialized.