chapter one

Getting Started with iOS 6

WELCOME! IT'S GREAT to welcome new developers to the world of iOS 6, and I'm happy to help you get started. For most developers with experience on other platforms, iOS is unlike the development environments they are used to. For starters, it lets you build apps for some of the most exciting products today (and, indeed, for many, many days). When you build an app for iPhone, iPod touch, or iPad you become part of the exciting ecosystem centered on Apple's extraordinary technologies and designs. You can find many books, articles, and media stories about Apple, its products, and their designs. There is analysis and prognostication; there are books and training materials for users. And there are books and training materials for that special cadre of people who extend Apple's handiwork: the developers.

This chapter gets you started as quickly as possible. You'll see how to register as a developer. After that, you'll be able to download tools and documentation from `developer.apple.com`. In this chapter you learn the basics of the Objective-C programming language and the highlights of the history of iOS 6—how we got here. Then you'll find a high-level overview of the Xcode integrated development environment (IDE). Before you know it, you'll be following the steps at the end of the chapter to build your first iOS 6 app.

Doing Your Homework

How did you decide to start developing for iOS? Some people use iOS devices and just want to find out more about what makes them tick. Other people have an idea for a great app and would like to build it themselves. Still others are IT professionals who want to expand their skills to this new platform. And others are IT professionals who have been asked to find out how to port an existing or planned project to iOS devices.

Getting Yourself Ready

No matter which category you find yourself in, you probably need to do a bit of homework before you start. You should have some background in programming. It can be long ago or recent, and it can be in advanced languages derived from C or in scripting languages such as PHP. (As noted in the Introduction, some experience with object-oriented programming can definitely help.)

You should be familiar with iOS from a user's perspective. If you plan to develop for only one of the devices (iPhone or iPad, perhaps), you can just explore that device and its features. However, to fully understand the iOS ecosystem, it's good to have both devices and to share data between them using iCloud.

Apple has fairly aggressively pushed out new versions of its devices on roughly an annual basis. It has followed a pattern of dramatically lowering the prices on the previous version of each device as a new one becomes available. You may be able to find a model that is several years old (you may even know someone who can give you one) that you can use for testing. As long as you can install iOS 6 on it, you'll have a test device and not have to worry about mixing up your actual data with test data.

Adopting a Developer's Point of View

When people use computers, they usually focus on a task that they need to accomplish. As a developer, you need to learn a secondary focus: watch *how* people do things rather than *what* they do. Develop this skill and use it to observe how people behave with iOS devices. You have a perfect test subject: yourself.

When something goes wrong or doesn't work the way you expect it to, don't just push on to try it another way. Take a moment to think back not so much about *what* you did wrong but *why* you did it. Did you mistake one icon for another? (Perhaps the icon's meaning wasn't clear.) Did you assume that an action would be carried out differently than it actually was? It doesn't matter if you made a mistake or if the app has a bug in it; in either case, something broke the chain of logic in the user interface and the app. Get used to spotting and analyzing these little glitches. Each one is a learning experience if you just pay attention to it before moving on with the task at hand.

Exploring the App Store

As of this writing, Apple's App Store has surpassed three quarters of a million apps. There are all kinds of apps for all kinds of purposes. Explore the App Store to see what people are writing. If you have an idea for an app, look to see how other people are approaching the topic.

Even if you have an idea for your own app, continue browsing in the App Store in other genres. Many apps are free, so download and install any that seem interesting in any way. If you are planning to build an app for people to use for keeping track of livestock breeding, you may spot an interface element in a game or other app that would be useful in your own app. You can't see every app in the App Store, but keep yourself up to date.

Reading Reviews

Read reviews on the App Store as well as reviews in the media, including blogs. Remember at this point that you're looking for points that reviewers pick up on, both good and bad. Listen to friends as they point out what they like and dislike about the apps they use.

Understanding the App World—Past, Present, and Future

For most people, the app world began in the summer of 2008. On July 10, the App Store opened (it's part of iTunes which received an update). The next day, July 11, the iPhone 3G went on sale. It ran iOS 2.0.1. The phrase "there's an app for that" was a key part of the marketing of the new iPhone 3G. Before long, people around the world understood the basics of apps that could be downloaded from iTunes directly onto an iPhone.

The app world is just a few years old. Every day, new people join it as they get their first iOS device or, as in your case, they decide to start developing apps. As you explore this world, keep a few critical milestones in mind to help you to make sense of information that you find in your studies:

- As noted, the first release of iOS to developers was iOS 2 in July of 2008.
- iOS 3 in June of 2009 added new features such as copy-and-paste. (Yes, in case you didn't know or have forgotten, you didn't have them at the start.)
- iOS 4 was released in June of 2010. iOS 4.2.1 in November supported the iPad.
- iOS 5, released in October of 2011 was the first unified release for iPhone, iPad, and iPod touch.
- iOS 6 (the subject of this book) was released in September 2012.

Along with new versions of the iOS operating system, the engineers at Apple were updating OS X as well as Xcode, the tool for developers of both operating systems as well as third-party apps. (Xcode is discussed later in this chapter and in Chapter 2, "Getting Up to Speed with Xcode.")

Xcode changes less frequently than the operating systems, but there have been very significant changes accompanying the unification of iOS for all the iOS devices, as well as major changes to the structure of Xcode itself.

Looking at the Master-Detail Application Template

All of this background information matters because, as you browse the web and discussion groups, it's important that you know what version you're looking at. Here's an example of how the evolution has taken shape. Xcode contains a number of templates that you can use as the basis for your apps. One of the commonly used templates today is the Master-Detail Application template for iOS (discussed more later in this chapter). It often serves as the basis of apps, and it will serve as the foundation of the app that you will build throughout this book. When you build the app, you can run it on the iPad simulator. Figure 1-1 shows it running in the simulator in landscape mode.

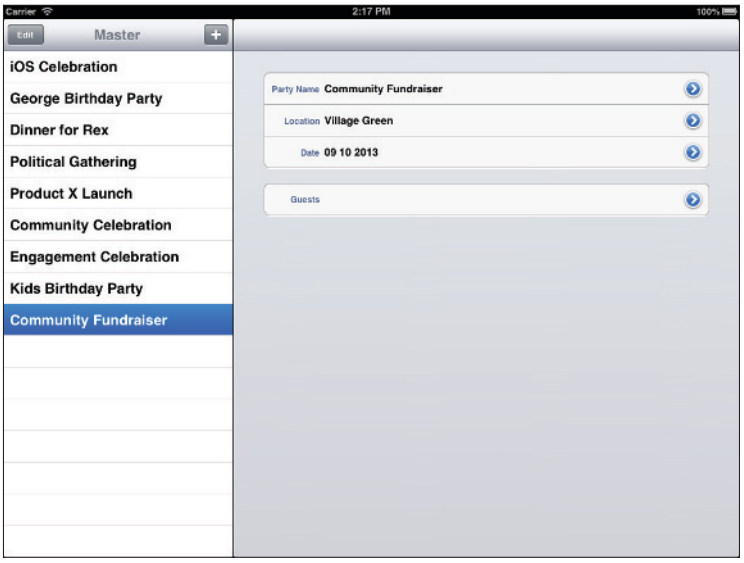

FIGURE 1-1 Master-Detail Application template running on the iPad simulator in landscape mode.

What you see in Figure 1-1 is an iPad feature called a *split view*. It combines two views in one. (Settings uses this architecture so you may recognize the bare bones of the design.) On the

left, a *master* view lets you look at an overview; on the right, a *detail* view shows details for the selected item in the master view. When you tap the + at the top of the master view, a new item is added to the list. By default, the template simply inserts placeholder data (a timestamp).

When you rotate the simulator (you'll learn how to do that in Chapter 2), you'll see that the master view disappears (because there isn't room for it). Figure 1-2 shows the simulator in portrait mode.

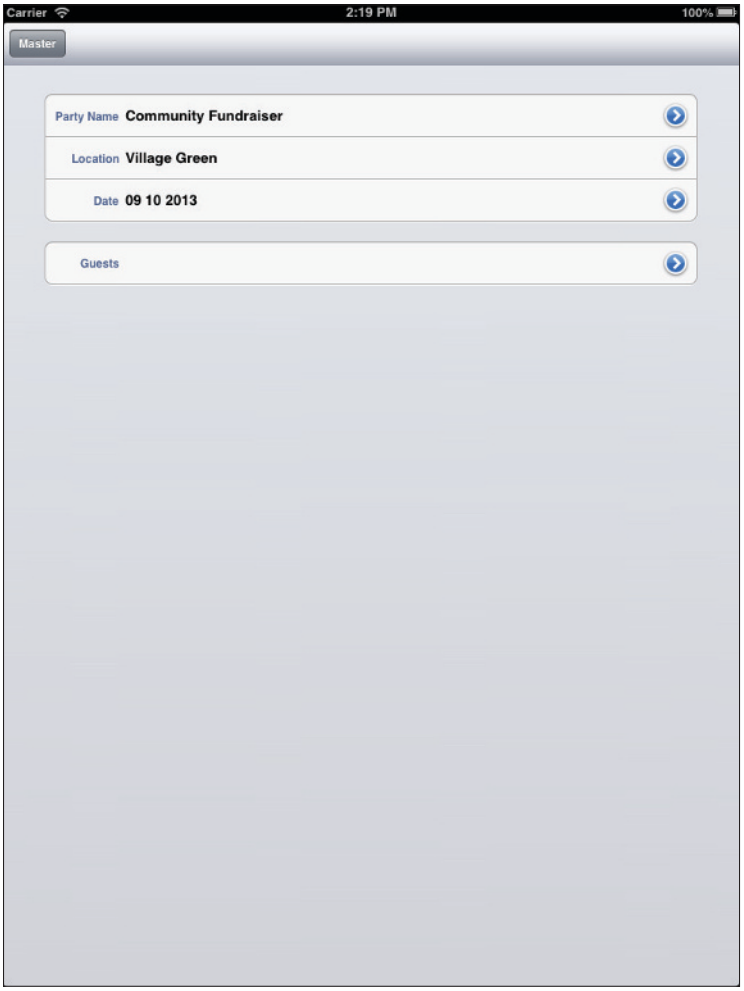

FIGURE 1-2 Master-Detail running on the iPad simulator in portrait mode.

However, there's a Master button in the top bar that will let you open a popover with the data. Tap the button, and the master view slides in from the left as you see in Figure 1-3. You can push the master pane back to see whatever it hides.

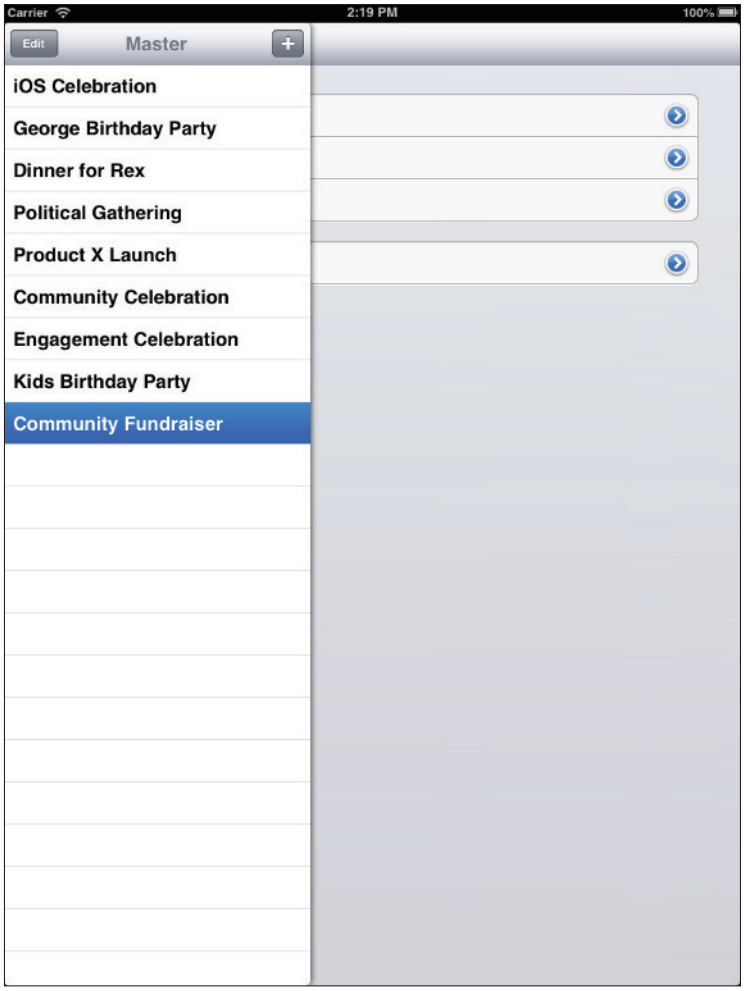

FIGURE 1-3 Tap the button to show the master view in portrait mode.

One of the architectural changes to both Xcode and iOS is the result of the introduction of *storyboards* and the concept of *universal apps*. You can use that combination to write a single app that behaves and looks right on both iPhone and iPad. Furthermore, with the Auto Layout features introduced in 2012, apps can also adjust to changing screen sizes, as happened with iPhone 5 and with iPad mini. Auto Layout lets you handle both changes in screen size and changes in aspect ratio (the latter being reflected in the difference between iPhone 5 and previous models).

Because the template has universal code in it, you can run the same code on the iPhone simulator. However, because the screen size in iPhone is smaller than in iPad, you can see only one view at a time. Figure 1-4 shows the master view.

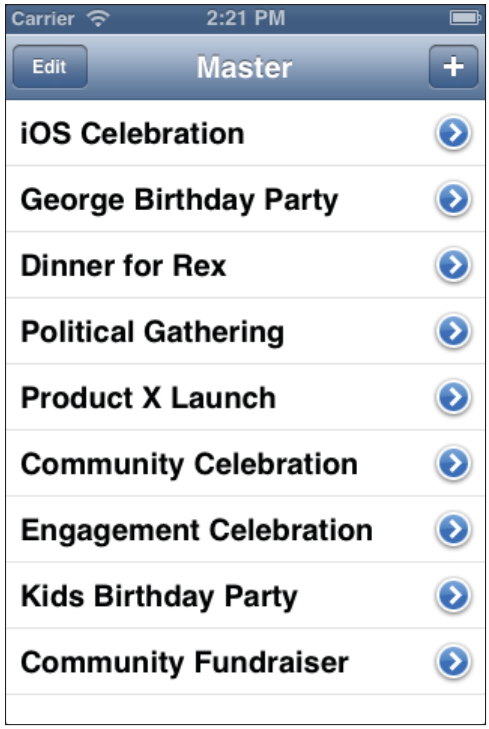

Figure 1-4 Master-Detail running on the iPhone simulator in portrait mode, in master view.

From the master view, tap on the item you want to switch to the detail view, as shown in Figure 1-5. You can click the Master button to toggle back.

Also, as part of the template, the views adjust to device orientation as you see in Figure 1-6.

All of this is part of the template: you don't have to write anything.

The reason for showing you this sneak preview of the tools you have available is to point out that much of what you're seeing is very new. In some cases, the best advice from two years ago is outdated today.

FIGURE 1-5 You can toggle between the master and detail screens on iPhone.

FIGURE 1-6 The views rotate properly.

Registering as a Developer

Have you registered as a developer at `developer.apple.com`, the website of Developer Technical Services (DTS)? It's a simple process, and it formalizes your status in the community of developers. Actually, there are several statuses for you to choose among. Most people register as an individual; at the time of this writing, it costs $99 for a year. That entitles you to online access to Apple's documentation, the critical development tools that are centered on Xcode, developer previews of new technologies including the operating systems, and, occasionally, invitations (free or priced) to certain Apple events. Most important to many developers, you gain access to the App Store so Apple can distribute your apps whether they are free or priced. (And you get 30 percent off the price of the app as well as 30 percent of the price of in-app purchases.)

In addition, you get two technical support incidents that you can use to ask Apple engineers for advice. You can send them your code and ask them why something is happening (or why something isn't happening). You can purchase more incidents for $50 each. There are separate programs for iOS, OS X, and Safari (Safari is free).

When you look at `developer.apple.com`, you'll see that you have other choices than an individual program. You can register as a business (if you are legally constituted as a business). You can register through an educational institution that pays the fee for all of its students, and you may find other options on `developer.apple.com`. Take the time now at least to register on `developer.apple.com`. You can come back later to choose which program(s) you want to join. The basic free registration gives you access to the basic resources (but not the App Store). Free registration consists of providing or creating an Apple ID. At that time or later, you can associate that developer account with a specific program.

When you become an iOS developer, you're not alone. There are online communities and discussion boards where you can meet other developers. In many places, there are local organizations of developers. You can find a local group by looking on sites such as `meetup.com` or by asking in an Apple Store or a third-party store that sells Apple products. If you can't find a local group but are planning to travel to a larger city nearby, take the time to inquire in that city's Apple Store; you may get a suggestion of a group near you or the email address of someone else who is interested.

While on the topic of traveling, as a registered developer you may get an invitation to Apple's Worldwide Developers Conference (WWDC). It is usually held toward the beginning of June in San Francisco. As of this writing, it's limited to 5,000 attendees, and in 2012 it sold out in less than two hours. To date, the limit on attendees has depended both on the size of the venue and on the fact that scores of Apple engineers attend both to present sessions during the week and to be available for consultations with developers. There are drop-in labs for just about every aspect of the operating systems, where you can ask your questions and ask for advice. It's an in-person version of DTS.

If San Francisco seems far away (or tickets are not available), have no worries—videos of the sessions are posted usually within two weeks of the conference. The entire conference is

covered by non-disclosure agreements, so the only legal way to access these videos is by being a registered developer. (An exception is the keynote opening session; reporters are invited to it and are invited to write about it. You can usually sit comfortably at home and get the highlights of the keynote on your TV news.)

WWDC affects every Apple developer because it serves as the annual conference to bring developers up to date. Having so many developers together either on-site or through video means that Apple has a chance to preview new features and to explain existing ones. A public release of one of the operating systems (often OS X) within a month or so after the conference is common. In 2012, OS X 10.8 Mountain Lion was released a few weeks after WWDC, and iOS 6 was released three months later. Apple's hardware announcements have recently come in the fall and spring.

That's the crux of what you need to know as a new Apple developer. And here's a tip for you: When you meet other developers at any kind of event such as a Meetup, an Apple Store, or an Apple event, you will always be asked the same question. Don't be surprised. Now that you're part of the community, you'll be asked if you're working on an app. The answer is "Yes." You're reading this book and starting on the road to your first app.

Introducing Basic Programming Concepts

Many people begin learning the basics of programming by writing a short program—often one that displays simple text, such as "Hello, World." Depending on the language, you can do this in a single line of code or a few (see the article "Hello world program" on Wikipedia to find out much more). This is the basic code:

```
main()
 {
        printf("hello, world");
 }
```

It's been more than a quarter century since the first days of Hello World, but many people still learn with this first step in programming. Unfortunately, programming today isn't the linear step-by-step process that Hello World suggests. Technology has moved beyond the linear process of early programming languages into a world of objects and non-linear control.

In its developer documentation, Apple has a 20-page document that shows you how to write a Hello World program. In part, the difference between 20 pages and three lines of code reflects the development environments. In order to write a Hello World program, a few lines of code is sufficient; on the other hand, to use the panoply of developer tools in Xcode and iOS to do that is overkill. However, what developers have learned over the last decades is that the line-after-line model of writing code doesn't scale well. If you want to write an app in the style of Hello World, you'll be at it for quite a while.

In order to build the powerful, complex, and attractive apps that people want today, you need more complex tools than a keyboard and an empty file. In this section, you visit some of the concepts behind the tools. The details are covered in the remaining chapters of this book.

Object-Oriented Programming in Objective-C

If you have experience in programming languages (and you should know at least one to get the most out of this book), you may be put off when you first see the language of iOS, Objective-C. What jumps out at people the first time is the brackets. Here's a line of Objective-C code:

```
self.detailViewController =
  (DetailViewController *)
    [[self.splitViewController.viewControllers lastObject]
      topViewController];
```

Don't panic. Before long you'll understand the brackets and be able to parse that line of code.

Objects in Objective-C

Object-oriented programming is the predominant programming style today. In it, *objects* are created that combine data and logic. A code object often corresponds to a real-world or on-screen object. In Figure 1-1, you see a split view. At the left, you see a master view, at the right, you see a detail view, so you have a total of three views. These are objects on the screen as well as objects in the code.

Objects can refer to other objects, and they do not have to be visible. The three views shown in Figure 1-1 are each contained within another object—a *view controller* object that's not visible itself. People usually talk about the view controllers rather than the views they contain. Thus, it is appropriate to say that the split view controller in Figure 1-1 contains both a master view controller and detail view controller. Each of those three view controllers contains a view, and those views are what the user sees.

In Objective-C, an object is defined as a *class*. You write code for the class. The code defines the logic of the class, which is embodied in *methods*. (These are somewhat analogous to functions in C++ and similar languages, but they differ in a critical point, which is explored in the following section, "Messaging in Objective-C.") A class may also have *properties*. These define data elements (more specifically, they provide accessors to the class's data elements).

When the code is executed, a class can be *instantiated*. This means that there is a memory location set aside for the code and properties of the class. It is real. An instance of a class can store data in its properties, and it can execute the code in its methods. It is common to have

multiple instances of a class at runtime, but in some cases there is only one (and in many cases, you write code for classes that are instantiated only under specific circumstances).

As in any object-oriented language, objects can be based on other objects. In Objective-C, a built-in class such as a view controller embodies the basic functionality required for all view controllers. In your own app, you may *subclass* the built-in `UIViewController` class that is part of Cocoa Touch with your own class. In fact, the Master-Detail Application template does it for you: you have a `MasterViewController` and a `DetailViewController` class. They are subclasses of `UIViewController`, and they inherit the methods and properties of `UIViewController`. You can see these files at the left of the Xcode window shown in Figure 1-7. (You learn more about Xcode in Chapter 2.)

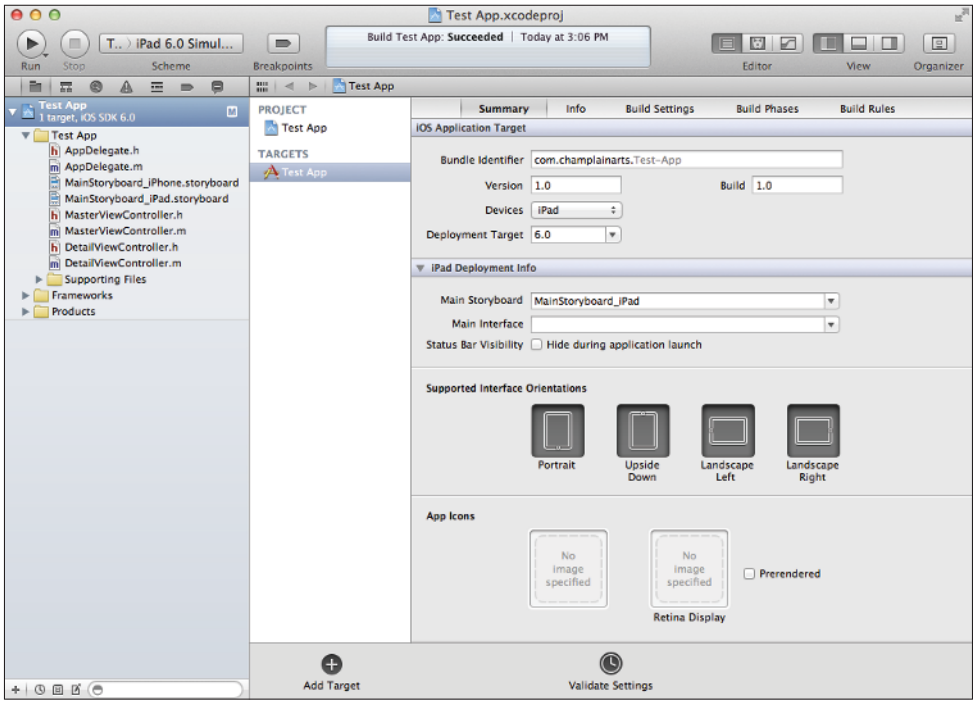

FIGURE 1-7 The project's files are shown at the left of the Xcode window.

You may notice that there are pairs of files for the classes. A file with the .h extension contains the headers—the declarations of the class's methods and properties. A file with the `.m` extension contains the definitions of the properties and methods—the code that implements them.

This is a very high-level overview of Objective-C. As you read on, you'll find out more about these basic principles.

Messaging in Objective-C

The most important point to understand is that Objective-C is a dynamic, messaging language. In traditional programming languages such as C, each line of code is executed, one after another. Control statements let you alter that line-by-line execution. You can go directly to another line of code (a technique now frowned on) or you can execute code conditionally or in loops.

You an also write functions or subroutines. They are executed line-by-line, just as your main program is. However, they can be called from your main program. Thus, in your main program, you execute the code line by line, but, if you call a function or subroutine, control passes to that code and then returns to the next line in your main program.

The `printf` function in the Hello World program shown earlier is a built-in function of C. Control is transferred to `printf` and, when it's completed, it returns to the main program.

In other object-oriented languages such as C++, you can instantiate an object just as you can in Objective-C. Once you have an instance of an object, you can call a function within it just as you call the `printf` function. When your code is compiled, these links are set up.

Objective-C uses a *messaging* model rather than a calling model. At runtime, you create an instance of an object just as you would in another language. However, rather than calling a function, you send a message to the object. That message causes a method in the object to execute. It is very similar to calling a function, but there is a critical distinction. When you call a function in another language, the function you are calling must be defined, and your main code must identify the function to be called. In Objective-C, you send a message to an object, and, it is quite possible that what that object is will not be defined until runtime. Thus, some of the error checking that occurs in the compiler for other languages is done at runtime. This allows for a great degree of flexibility.

For now, just remember that you are sending messages rather than calling functions. As you start to develop code, the distinctions will start to make more sense.

Frameworks

When you're writing an app, you rarely start from a blank piece of paper or an empty file. Xcode has built-in templates that are functional, so your job is to enhance and customize them. As a developer, you have access to a great deal of sample code on `developer.apple.com`. There is also more code on the web (but remember to be careful to use only current code).

Within iOS, you will find a number of *frameworks*. These are collections of classes that can work together to provide functionality. You can also develop your own frameworks, but in this book, the emphasis is on the provided frameworks. As you start to get a sense for the major frameworks, you'll see what is already built into iOS.

iOS is the operating system for iPhone, iPad, and iPod touch. You implement your apps using the Cocoa Touch application programming interface (API). Cocoa Touch—the API—is the language used by developers. iOS is used by developers, marketers, and users.

Graphical Coding

There's another difference between developing with Xcode and iOS and writing Hello World—some of your coding doesn't involve typing code. When you get around to developing your interface, you draw it with Interface Builder, which is part of Xcode. When you want to link objects in the interface such as buttons to the code that runs them, you simply drag from the button to the code in your file.

You also use graphical coding to set up data relationships; you use graphical elements such as checkboxes to manage your project's settings. There is a lot of code to type, but there is also a lot of non-typed coding to do.

Model-View-Controller

The last major concept to think about is *model-view-controller* (MVC) architecture. It was developed in the 1970s at Xerox PARC, and was first used in the Smalltalk language. When Objective-C was designed in the 1980s it adopted MVC, and it remains a linchpin of the architecture.

MVC organizes the objects in an object-oriented system. This organization creates triplets of objects that implement a model, a view, and a controller. Simply put, a *model is data*. A *view is a representation of the data* along with the controls to work with it. The model knows nothing about the view, and the view knows nothing about the model. This makes for highly portable and maintainable code. It also reflects the fact that with both models and views, you, the designer, can exercise a great deal of logical control. In addition, as you will see with Xcode, graphical user interfaces to design your model and your view are available for you.

The complexity lies with the controller object. The controller knows about both the model and the view. Most of what seems like "real" coding is done in the controller.

You have already seen views in Figure 1-1. Although there is no visual representation of a view controller itself, you have learned that each view has a view controller. As for the model, when you build a project from an Xcode template, you often have a choice of using Core Data for the model or of using another technique.

In the next section, you will see how to set up your data model. It's not terribly complicated: just a checkbox. So it's on to building your first project. It will be the Master-Detail Application shown in Figures 1-1 to 1-7.

Installing and Using Xcode

Until Xcode 4, installing Xcode was a bit complicated. Now, Xcode is installed just as any other app from the Mac App Store. Go to the Mac App Store, and search for Xcode. Then "buy" it, and it will be downloaded and installed automatically. You're ready to go.

> Xcode is free, but the operation of installing software through either app store (Mac or iOS) is called a purchase even if there is no charge. It is also important to point out that from time to time, developers have access to pre-release versions of Xcode. They are available for download from `developer.apple.com`.

The following steps walk you through an overview of the Xcode process that will enable you to build the Master-Detail Application and run it as you have seen in this chapter.

1. Buy and install Xcode.
2. Launch Xcode. You will see the screen shown in Figure 1-8.

Figure 1-8 The Xcode Welcome screen.

3. From the menu, select Create a new Xcode project.
4. As shown in Figure 1-9, you can select from the built-in templates for iOS and OS X. Select the Master-Detail Application in iOS, and click Next at the lower right.

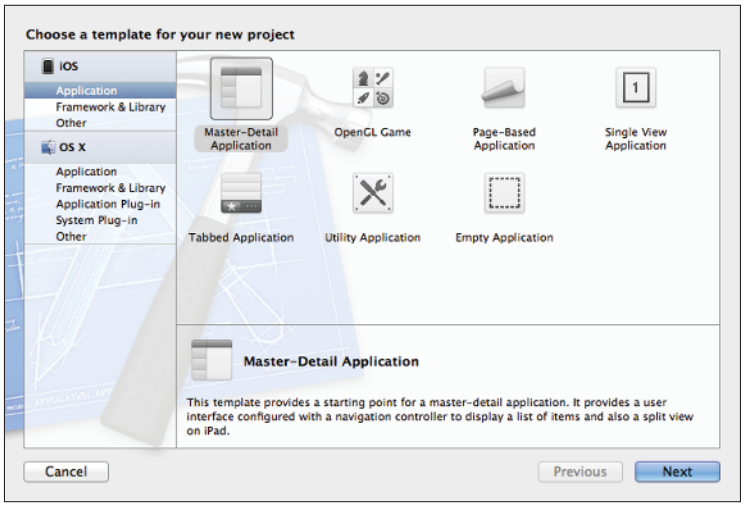

FIGURE 1-9 Select the Master-Detail Application template.

5. On the next screen, fill in the information requested, as shown in Figure 1-10.
   a. The name of the product and your organization name are up to you.
   b. By convention, the company identifier is a reverse domain name, which is guaranteed to be unique.
   c. You can omit the class prefix.
   d. For devices, choose universal to create both iPad and iPhone versions.
   e. Mark the checkboxes at the bottom to use storyboards and use automatic reference counting. If you want Xcode to flesh out your model with Core Data, check that checkbox. (It is not used in the example files you can download for this book.)
6. Click Next to continue.
7. On the next screen, choose the location for the project's files. Click Next. Xcode creates the files for you and opens the project, as you saw in Figure 1-7.

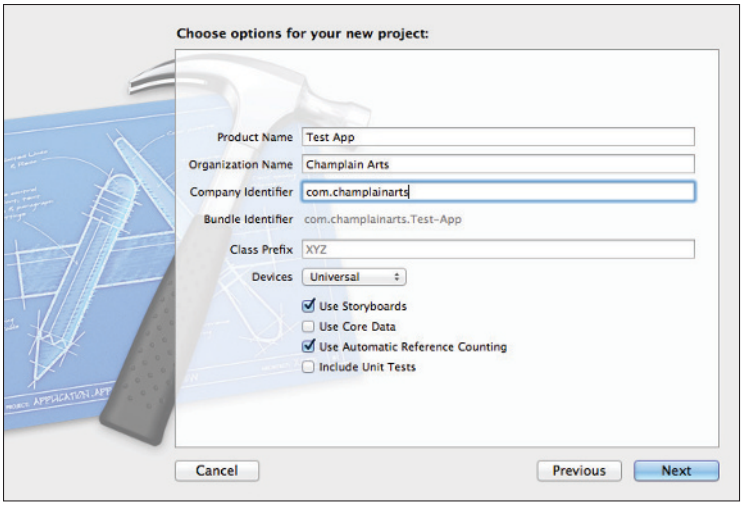

FIGURE 1-10 Fill in the project's options as indicated.

8. Click the Run button at the top left of the window shown in Figure 1-7 to build and run the project. Use the pop-up menu to the right of the Run button to choose whether to run the project on the iPad simulator or the iPhone simulator.
9. Run your project. You'll see the precursors of the images you see in Figures 1-1 to 1-6.

Summary

This chapter showed you how to prepare to be an iOS developer. You should practice looking at apps with a new eye—look at *how* they do things in addition to *what* they do. You have some familiarity with the basic principles and concepts of iOS; you'll learn more about the specifics later in the book. You also should have installed Xcode as described in this chapter. You should follow the steps to build your first project from the built-in Master-Detail Application template. It's important to do this now so that if, by some chance, there is an error in your Xcode installation, you catch it before moving on.

In Chapter 2, you explore Xcode itself. It has powerful tools to help you build your app. (It is actually the same tool that the engineers at Apple use to build iOS itself.)