About Objective-C

Important: This documentation contains preliminary information about an API or technology in development. This information is subject to change, and software implemented according to this documentation should be tested with final operating system software.

Objective-C is the primary programming language you use when writing software for OS X and iOS. It's a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime.

At a Glance

This document introduces the Objective-C language and offers extensive examples of its use. You'll learn how to create your own classes describing custom objects and see how to work with some of the framework classes provided by Cocoa and Cocoa Touch. Although the framework classes are separate from the language, their use is tightly wound into coding with Objective-C and many language-level features rely on behavior offered by these classes.

An App Is Built from a Network of Objects

When building apps for OS X or iOS, you'll spend most of your time working with objects. Those objects are instances of Objective-C classes, some of which are provided for you by Cocoa or Cocoa Touch and some of which you'll write yourself.

If you're writing your own class, start by providing a description of the class that details the intended public interface to instances of the class. This interface includes the public properties to encapsulate relevant data, along with a list of methods. Method declarations indicate the messages that an object can receive, and include information about the parameters required whenever the method is called. You'll also provide a class implementation, which includes the executable code for each method declared in the interface.

Relevant Chapters: Defining Classes, Working with Objects, Encapsulating Data

Categories Extend Existing Classes

Rather than creating an entirely new class to provide minor additional capabilities over an existing class, it's possible to define a category to add custom behavior to an existing class. You can use a category to add methods to any class, including classes for which you don't have the original implementation source code, such as framework classes like `NSString`.

If you do have the original source code for a class, you can use a class extension to add new properties, or modify the attributes of existing properties. Class extensions are commonly used to hide private behavior for use either within a single source code file, or within the private implementation of a custom framework.

Relevant Chapters: Customizing Existing Classes

Protocols Define Messaging Contracts

The majority of work in an Objective-C app occurs as a result of objects sending messages to each other. Often, these messages are defined by the methods declared explicitly in a class interface. Sometimes, however, it is useful to be able to define a set of related methods that aren't tied directly to a specific class.

Objective-C uses protocols to define a group of related methods, such as the methods an object might call on its delegate, which are either optional or required. Any class can indicate that it adopts a protocol, which means that it must also provide implementations for all of the required methods in the protocol.

Relevant Chapters: Working with Protocols

Values and Collections Are Often Represented as Objective-C Objects

It's common in Objective-C to use Cocoa or Cocoa Touch classes to represent values. The `NSString` class is used for strings of characters, the `NSNumber` class for different types of numbers such as integer or floating point, and the `NSValue` class for other values such as C structures. You can also use any of the primitive types defined by the C language, such as `int`, `float` or `char`.

Collections are usually represented as instances of one of the collection classes, such as `NSArray`, `NSSet`, or `NSDictionary`, which are each used to collect other Objective-C objects.

Relevant Chapters: Values and Collections

Blocks Simplify Common Tasks

Blocks are a language feature introduced to C, Objective-C and C++ to represent a unit of work; they encapsulate a block of code along with captured state, which makes them similar to closures in other programming languages. Blocks are often used to simplify common tasks such as collection enumeration, sorting and testing. They also make it easy to schedule tasks for concurrent or asynchronous execution using technologies like Grand Central Dispatch (GCD).

Relevant Chapters: Working with Blocks

Error Objects Are Used for Runtime Problems

Although Objective-C includes syntax for exception handling, Cocoa and Cocoa Touch use exceptions only for programming errors (such as out of bounds array access), which should be fixed before an app is shipped.

All other errors—including runtime problems such as running out of disk space or not being able to access a web service—are represented by instances of the `NSError` class. Your app should plan for errors and decide how best to handle them in order to present the best possible user experience when something goes wrong.

Relevant Chapters: Dealing with Errors

Objective-C Code Follows Established Conventions

When writing Objective-C code, you should keep in mind a number of established coding conventions. Method names, for example, start with a lowercase letter and use camel case for multiple words; for example, `doSomething` or `doSomethingElse`. It's not just the capitalization that's important, though; you should also make sure that your code is as readable as possible, which means that method names should be expressive, but not too verbose.

In addition, there are a few conventions that are required if you wish to take advantage of language or framework features. Property accessor methods, for example, must follow strict naming conventions in order to work with technologies like Key-Value Coding (KVC) or Key-Value Observing (KVO).

Relevant Chapters: Conventions

Prerequisites

If you are new to OS X or iOS development, you should read through *Start Developing iOS Apps Today* or *Start Developing Mac Apps Today* before reading this document, to get a general overview of the application development process for iOS and OS X. Additionally, you should become familiar with Xcode before trying to follow the exercises at the end of most chapters in this document. Xcode is the IDE used to build apps for iOS and OS X; you'll use it to write your code, design your app's user interface, test your application, and debug any problems.

Although it's preferable to have some familiarity with C or one of the C-based languages such as Java or C#, this document does include inline examples of basic C language features such as flow control statements. If you have knowledge of another higher-level programming language, such as Ruby or Python, you should be able to follow the content.

Reasonable coverage is given to general object-oriented programming principles, particularly as they apply in the context of Objective-C, but it is assumed that you have at least a minimal familiarity with basic object-oriented concepts. If you're not familiar with these concepts, you should read the relevant chapters in *Concepts in Objective-C Programming*.

See Also

The content in this document applies to Xcode 4.4 or later and assumes you are targeting either OS X v10.7 or later, or iOS 5 or later. For more information about Xcode, see *Xcode Overview*. For information on language feature availability, see *Objective-C Feature Availability Index*.

Objective-C apps use reference counting to determine the lifetime of objects. For the most part, the Automatic Reference Counting (ARC) feature of the compiler takes care of this for you. If you are unable to take advantage of ARC, or need to convert or maintain legacy code that manages an object's memory manually, you should read *Advanced Memory Management Programming Guide*.

In addition to the compiler, the Objective-C language uses a runtime system to enable its dynamic and object-oriented features. Although you don't usually need to worry about how Objective-C "works," it's possible to interact directly with this runtime system, as described by *Objective-C Runtime Programming Guide* and *Objective-C Runtime Reference*.