Xcode Overview

Tools > Xcode

2009-01-06

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction

Xcode is a suite of development tools used to create Mac and iPhone applications.

This document provides an overview of Xcode and its components.

You should read this document to learn about Xcode's capabilities and its software development workflow.

If you haven't installed Xcode on your computer, see *Xcode Installation Guide* for instructions.

Organization of This Document

This document contains the following chapters.

- "The Xcode Environment" (page 7) describes the major areas of the Xcode environment.
- "The Xcode Development Process" (page 11) provides the major steps of the Xcode software development process.
- "The Xcode Tools" (page 13) describes the major tools that make up the suite.

See Also

In addition to this document, these documents provide configuration and usage information about Xcode:

- *Xcode Workspace Guide* provides an overview of the Xcode workspace, and shows how to use its components and features.
- *Xcode Project Management Guide* provides practical descriptions of the major development tasks developers perform with Xcode.

Introduction

See Also

The Xcode Environment

Xcode is a suite of software development tools used to create applications that run on Mac OS X and iPhone OS. This suite includes applications, command-line tools, frameworks, and libraries you use to develop software products. The centerpiece of the suite is the Xcode application, which provides an elegant and powerful user interface for creating and managing software development projects.

In addition to the Xcode application, several other applications help you in your software development tasks, including:

- User-interface design
- Performance measurement and analysis
- Product distribution
- Team-based development with source control systems, such as Subversion, CVS, and Perforce

This chapter describes the major areas of the Xcode environment.

iPhone Simulator

iPhone Simulator facilitates the development of iPhone applications on your Mac. With the simulator you can develop and test your application in an integrated environment, using your screen, keyboard, and mouse. Once you've ironed out design problems, you can test your application on devices to make sure the user interface works as expected and that your code performs well on actual hardware.

To learn more about the iPhone simulation environment, see Using iPhone Simulator.

Class and Data Modeling

Xcode provides powerful and flexible class and data modeling tools. The class modeler helps you explore and understand the classes in your project, whether they're written in Objective-C or C++. It allows you to see class relationships and categories (in Objective-C code). The data modeler lets you manage Core Data entities and relationships between them. **Core Data** lets you manage the relationships and persistence of managed objects, whose backing store is usually a database or a file.

To learn more about the Xcode modelers, see:

- *Xcode Design Tools for Class Modeling*
- *Xcode Tools for Core Data*

Source Code Indexing

Source code indexing (known as Code Sense) allows Xcode to assist you in working with symbols. Source code indexing allows the Xcode text editor to identify element types in your code; for example, class names, constants, instance variables, and so on. Xcode creates an index of the symbols your project uses, which is used by several components, such as the text editor and the Research Assistant.

See "Code Sense" in *Xcode Project Management Guide* to learn more.

Source Code Editing

The Xcode text editor allows you to edit your source code as well as any text file, including XML and HTML files. The text editor uses source code indexing to provide several features, including code completion.

Xcode helps you edit several files at once through refactoring. **Refactoring** allows you to make structural changes to your code that don't affect its functionality and behavior, such as renaming a class.

For more information see:

- The Text Editor
- *Xcode Refactoring Guide*

User-Interface Design

One of the most important steps in developing your application is designing its user interface. Xcode includes a graphical user-interface design tool that allows you to lay out your application's controls in the way they appear to your application's user. You also connect controls to other controls or to controller objects, following the Model-View-Controller paradigm that drives Mac and iPhone application development. Your user-interface designs are saved as nib files that your application loads at runtime.

To learn more about user-interface design, see:

- "The Model-View-Controller Design Pattern"
- *Interface Builder User Guide*

Build System

Xcode includes a powerful build system used to create a wide variety of software products, including applications, frameworks, and plug-ins. Xcode lets you customize your build process through build phases (or build steps) and build configurations (or build flavors, such as Debug and Release).

When building large projects, you may want to leverage the power of multiple Macs, which may include your coworkers' workstations or your organization's build-server farms. With distributed builds, you can harness the power of a few iMac computers or a group of Xserve servers.

To learn more about the Xcode build system, see:

- Building Products
- *Xcode Build System Guide*

Performance Analysis

You carry out performance analysis and tuning on your application to ensure that it uses your customer's computer or device as efficiently as possible. Part of this analysis and tuning is finding and eliminating memory leaks, which may cause a computer or device to operate slowly and the system to shut down your application.

Performance analysis can be software centric or hardware centric. Xcode supports both kinds of performance analysis. For details, see:

- *Instruments User Guide*
- *Shark User Guide*

Source Management

Xcode provides two ways to safeguard your code: *source control* and *snapshots*. **Source control** lets you share your work with other developers, while keeping the code safe in an administered server. **Snapshots** allow you to revert your code to a previous state at a particular point in your development process.

To learn about source control and snapshots, see *Xcode Source Management Guide*.

The Xcode Development Process

As described in "The Xcode Environment" (page 7), Xcode is a suite of software development tools used to create Mac and iPhone applications. The centerpiece of this suite is the Xcode application, which you use to organize and edit your source files, view documentation, build your product, debug your code, and optimize your product's performance.

The Xcode application development process is made up of several tasks:

1. Configure your development environment.

   Xcode lets you customize your development environment in many ways, including the layout of the project window, the workings of the text editor used to edit source code, the content that appears in the Documentation window, the keyboard shortcuts for menu commands, and many more.

   To learn about customizing your working environment, see *Xcode Workspace Guide*.

2. Create your project.

   Xcode provides several project templates that get you started. You choose the template that implements the type of application (or other type of product) you want to develop.

   To learn more about creating Mac applications, see Creating Projects. "Creating a Project" in *iPhone Development Guide* shows how to create an iPhone application.

3. Design the user interface.

   Interface Builder lets you design your application's user interface graphically and save those designs as resource files that your program loads at runtime.

   To learn more about creating user interfaces with Interface Builder, see *Interface Builder User Guide*.

4. Write code.

   Xcode provides several features that help you write code fast, including class and data modeling, code completion, direct access to documentation, source-code refactoring, and more. See *Xcode Workspace Guide* to learn more about these features.

5. Build and run your application.

   When developing Mac applications, you generally build and run them on your development computer (see *Xcode Project Management Guide* for details). iPhone applications, on the other hand, run in iPhone Simulator application or on an iPhone OS–based device.

iPhone Simulator:

iPhone Simulator implements the iPhone OS API in a Mac application, providing a runtime environment that mimics the environment devices provide. By allowing you to run your applications in Mac OS X, the simulator provides a way to quickly test your application's functionality without the need for an actual device. However, running applications on the simulator is not the same as running them on devices.

The simulator uses Mac OS X versions of the low-level iPhone OS frameworks instead of the versions that run on devices. But, in general, the simulator is a great tool for performing initial testing of your application. Keep in mind that, because the simulator does not emulate device functionality, you must always perform final testing and performance analysis of your application on actual devices.

To learn more about building and running iPhone applications, see "Creating a Project" in *iPhone Development Guide*.

To compile and debug your code, Xcode relies on the open-source tools GCC and GDB.

6. Unit-test your code.

   Performing unit tests on your code is important to ensure that no errors or bugs are introduced as you make changes and improvements to the implementation of your application's functionality. Xcode provides an easy-to-use unit-test environment with which you can test Objective-C and C++ code.

   Note: The Xcode unit-testing environment does not support iPhone applications.

   To learn more about unit testing in Xcode, see *Xcode Unit Testing Guide*.

7. Measure and tune your application's performance.

   There are several tools available for you to gather performance data about your application and to improve your application's performance.

   Instruments is a dynamic performance analysis tool that lets you peer into your code as it's running and gather important metrics about what it's doing. You can view and analyze the data Instruments collects in real time, or you can save that data and analyze it later. You can collect data about your application's use of the CPU, memory, file system, and the network, among other resources. See *Instruments User Guide* for details.

   Shark is another tool that helps to find performance bottlenecks in your code. It produces profiles of hardware and software performance events and shows how your code works as a whole and its interaction with the operating system. For more information, see *Shark User Guide*.

The Xcode Tools

The Xcode suite is made up of several applications and command-line tools whose purpose is to provide an easy-to-use, flexible, and powerful development environment. This chapter describes the major tools that make up the suite.

The Xcode Application

You use Xcode to organize and edit your source files, view documentation, build your product, debug your code, and optimize your product's performance. Xcode is a highly customizable integrated development environment (IDE) with many features that let you create a pleasant and efficient working environment.

The center of your software development efforts is the *project*. An **Xcode project** groups the source files, libraries, media, and other resources needed to build your product. The most visible type of software product you can create with Xcode is the application. However, you can also develop Automator actions, command-line tools, frameworks, plug-ins, and kernel extensions.

To learn more about the Xcode application, see *Xcode Workspace Guide*.

Interface Builder

The Interface Builder application helps you design the user interfaces of your applications. With this application, you create your user interface by picking controls from a library of configurable elements and arranging them with the help of lay out guides. You also connect these elements with each other and to "action" methods to form a graph of user interface objects that define your application's user experience. The files that store these object graphs are called **nib files**. At runtime, your application loads nib files to reconstitute the user interface you designed. This makes it easy to design and troubleshoot user interfaces because the difficult task of positioning each control and connecting it to the appropriate methods in your source code is done through an elegant and effective graphical user interface.

To learn more about Interface Builder, see *Interface Builder User Guide*.

Instruments

The Instruments application lets you trace and profile your application as it runs. Instruments also traces most system components in Mac OS X. In this way, Instruments helps you understand the behavior of both user programs and the operating system.

Note: On iPhone OS you can profile and trace only your applications, not the operating system.

To learn more about Instruments, see *Instruments User Guide*.

Shark

The Shark application lets you carry out in-depth hardware-based performance analysis. Shark profiles the system while your code runs to find out where time is being spent. It can also produce profiles of hardware and software performance events, such as cache misses, virtual memory activity, memory allocations, function calls or instruction dependency stalls. This information is an invaluable first step in your performance tuning task, so you can see which parts of your code or the operating system are the bottlenecks.

In addition to showing you where time is being spent, Shark can give you advice on how to improve your code. Shark is capable of identifying many common performance pitfalls and visually present the costs of these problems to you.

To learn more about Shark, see *Shark User Guide*.

PackageMaker

The PackageMaker application allows you to create an **installation package**, which is used to deliver your product to your customer. Your customer opens your installation package with the Installer application, which places the package's payload in the appropriate locations in your customer's file system.

PackageMaker lets you specify installation requirements to ensure that the product is installed only on systems that meet those requirements, such as a specific amount of available random-access memory (RAM) or free disk space, a particular Mac OS X release, the presence of particular components or applications, and so forth.

Note: To deliver your iPhone applications to customers, you don't use PackageMaker. Instead, you upload your application to the iPhone Dev Portal, where it's made available to customers through the App Store. Go to the iPhone Dev Portal for details.

To learn more about PackageMaker, see *PackageMaker User Guide*.

GCC and GDB

GCC and GDB are open-source tools Xcode uses to compile and debug your code. However, instead of a command-line interface to them, Xcode provides an elegant graphical interface that makes their use more intuitive.

To learn more about GCC and GDB, see the Reference Library > Tools > Compiling & Debugging category.

Glossary

Core Data A technology that lets you manage the relationship and persistence of managed objects, whose backing store is usually a database or a file.

installation package A file used to deliver a product to its user. A user opens an installation package with the Installer application, which places the package's payload in the appropriate locations on her or his file system.

nib file A file that stores a graph of user interface objects that define a product's user interface.

project A collection of plans and resources for building one or more software products. A project groups source files, libraries, media, and other resources used to build one or more products.

refactoring The process of modifying source code for the purpose of improving its readability and maintainability while retaining the program's functionality and behavior.

snapshot A record that stores the state of a directory tree at the time it was taken.

source control A set of tools and procedures used to safeguard and manage files and changes made to them over time. Also known as Source Control Management (SCM) or version control.

Source Control Management (SCM) See **source control**.

Document Revision History

This table describes the changes to *Xcode Overview*.

Date	Notes
2009-01-06	Made minor changes and added glossary.
2008-10-15	Made minor content changes.
2008-05-27	New document that describes the Xcode IDE and its major components, and provides and overview of the development process.

Document Revision History