What's New in Xcode

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

Figures

Xcode 3.0 Feature Overview 13

Introduction

This document provides an overview of the new and improved features in Xcode releases.

This document is intended for people interested in the changes Xcode releases bring about. You can use this document to learn about features that enhance your workflow, improve your efficiency, and increase your productivity.

Organization of This Document

This document includes the following articles:

Xcode 3.1 Feature Overview

Xcode 3.1 is a major revision to the Xcode IDE. The main goal of this release is to support iPhone application development. However, this release also includes new features to enhance your workflow.

This article describes the new features and improved functionality in Xcode 3.1.

iPhone SDK

Using Xcode 3.1 and the iPhone SDK you can develop applications for iPhone OS. The iPhone SDK includes iPhone OS frameworks and developer tools with which you can build iPhone applications. It also includes iPhone Simulator, an environment that allows you to run iPhone applications on your Mac to perform initial testing. You can also run your application on actual devices, after becoming a member of the iPhone Dev Center.

For more information on the iPhone SDK, see *Xcode Overview*.

Project Package Format

Xcode 3.1 introduces the 3.1 project-package format.

Xcode 3.1 reads and builds project packages v2.1–3.0 and automatically upgrades project packages v1.5–2.0 to v3.1.

In general, project packages v3.1 can be opened and built in Xcode 3.0 and 2.5. Xcode tells you when a project package uses a feature it doesn't support.

Toolbar Items

Xcode 3.1 introduces two toolbar items:

- **Active SDK.** Lets you specify the SDK to use to build the active target. You can choose between iPhone OS SDKs and Mac OS X SDKs.
- **Overview.** Groups the "active" project settings into one, easily accessible control. With this toolbar item you can set a project's active target, SDK, build configuration, architecture and executable.

See "Setting Build Factors" in *Xcode Project Management Guide* for more information.

Text Editor

In Xcode 3.1 the text editor gains two new features:

- **Edit All in Scope.** The Edit All in Scope command allows you to edit several symbolname occurrences at the same time in a more convenient way than using Find & Replace. For example, you can change the name of a method argument, and Xcode replicates the change within the method's body at the same time. See "Editing Symbol Names" in *Xcode Workspace Guide* for details.
- **Symbol assist menu.** The symbol assist menu is an automatic shortcut menu that appears when you select a symbolname in a source code file. This menu allows you to perform one of several symbol-related commands on the selected symbols, including Edit All in Scope and Jump to Definition.

> **Note:** The symbol assist menu appears only when Edit All in Scope is active. See "Code Sense Preferences" in *Xcode Workspace Guide* for details.

Property List Editor

The user interface of the Property List Editor application has been revamped to make it easier to edit property list files. Among the improvements are:

- Reorder/move elements
- Cut/Copy/Paste element
- Property list types

  You can create files based on particular property-list schemas (such as the iPhone Settings schema used in iPhone applications)

Xcode now provides property list editor that uses the same user interface the Property List Editor application uses.

Framework and Library Use

Xcode 3.1 provides a new, straightforward way for adding frameworks and libraries to a target. See "Managing Files and Folders in a Project" in *Xcode Project Management Guide* for details.

Build System

The Xcode 3.1 build system received several improvements:

- Conditional build settings

In Xcode 3.0 you can define build setting specifications for particular architectures and build variants. Xcode 3.1 adds another condition: The SDK to use to build the active target.

See "Conditional Build Settings" in *Xcode Build System Guide* for details.

- Support for weak/required frameworks (new)

  Similar to symbol weak-linking, framework weak-linking allows you to build a product that is able to use a framework that may not be present at runtime. See "Managing Libraries and Frameworks" in *Xcode Project Management Guide* for details.

- Compiler Version build setting group (new)

  The C/C++ Compiler Version build setting is more visible in the settings editor; it's now in the Compiler Version build setting group.

- New `xcodebuild` options

  The xcodebuild tool supports these new options:

  - `-sdk <sdk_name>` Specifies the SDK to use.
  - `-fine <tool_name>` Provides the filepath to `tool_name`.

  For details, see the man page for `xcodebuild`.

- Strings files output encoding option (new)

  The new `STRINGS_FILE_OUTPUT_ENCODING` build setting lets you specify the output encoding of strings files.

New Project and New Target Assistants

Xcode 3.1 updates the New Project and New Target assistants with new project and target types. For details, see "Creating Projects" and "Creating Targets" in *Xcode Build System Guide*.

Open Quickly

Xcode 3.1 simplifies the Open Quickly dialog and makes it more effective. See "Opening Files by Filename or Symbolname" in *Xcode Workspace Guide* for details.

FileMerge Application

Xcode 3.1 improves the FileMerge application by:

- Performing faster comparisons
- Handling files with different line encodings properly
- Ignoring whitespace on lines

SCM Workflow

In Xcode 3.1 you can use drag-and-drop to import and check-out directories. And, when you check out a project directory, Xcode configures SCM in the project packages the directory contains so that you can perform SCM operations without having to configure SCM first (as in previous Xcode releases). For details, see "Browsing and Modifying Repositories" in *Xcode Source Management Guide*.

Compilers

Xcode 3.1 adds support for GCC 4.2 and LLVM-GCC 4.2 (see *LLVM-GCC Release Notes*).

Xcode Persistent Cache

To improve the security of Xcode caches, Xcode 3.1 places them in a more secure location. If you don't use earlier versions of Xcode, you should delete `/Library/Caches/com.apple.Xcode` and `/Library/Caches/Xcode`.

In Xcode 3.1 you can also delete Xcode caches using the **Empty Caches** command. For more information, see "Xcode Persistent Cache" in *Xcode Project Management Guide*.

Xcode 3.0 Feature Overview

Xcode 3.0 is a major revision to the Xcode application. The main goal of this version of Xcode is to streamline your development workflow. The new or improved Xcode features that enhance your workflow are grouped in the following areas:

- Text editing
- Program debugging
- Software architecture
- Data modeling
- Documentation access
- SCM repository management
- Project integrity

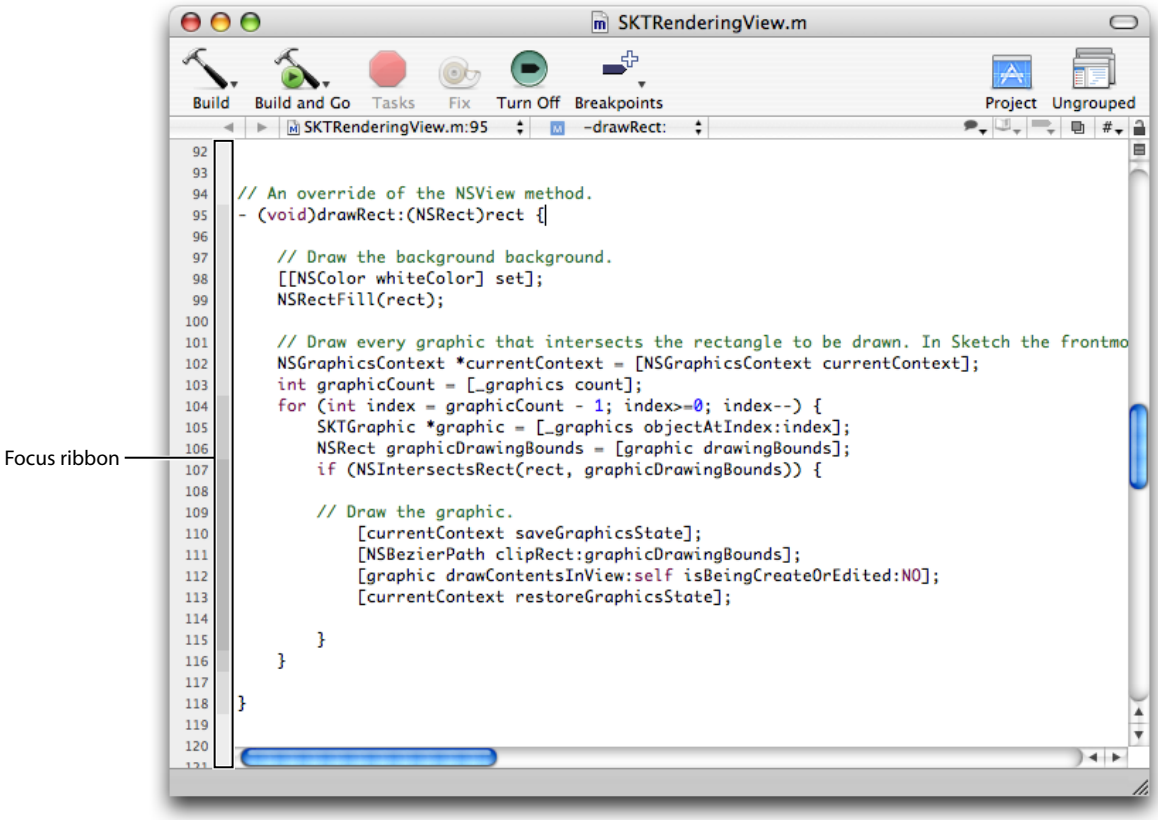
The following sections describe new or improved functionality in Xcode 3.0.

Text Editor

The text editor is substantially improved in Xcode 3.0 to improve its performance and streamline your development workflow.

- **Focus ribbon:** The editor now includes a focus ribbon. This ribbon shows the scope depth of the corresponding code in the editor pane using a gray scale: a white ribbon identifies code at the highest scope; increasingly darker shades of gray identify code within lower scopes. Figure 1 identifies the focus ribbon.

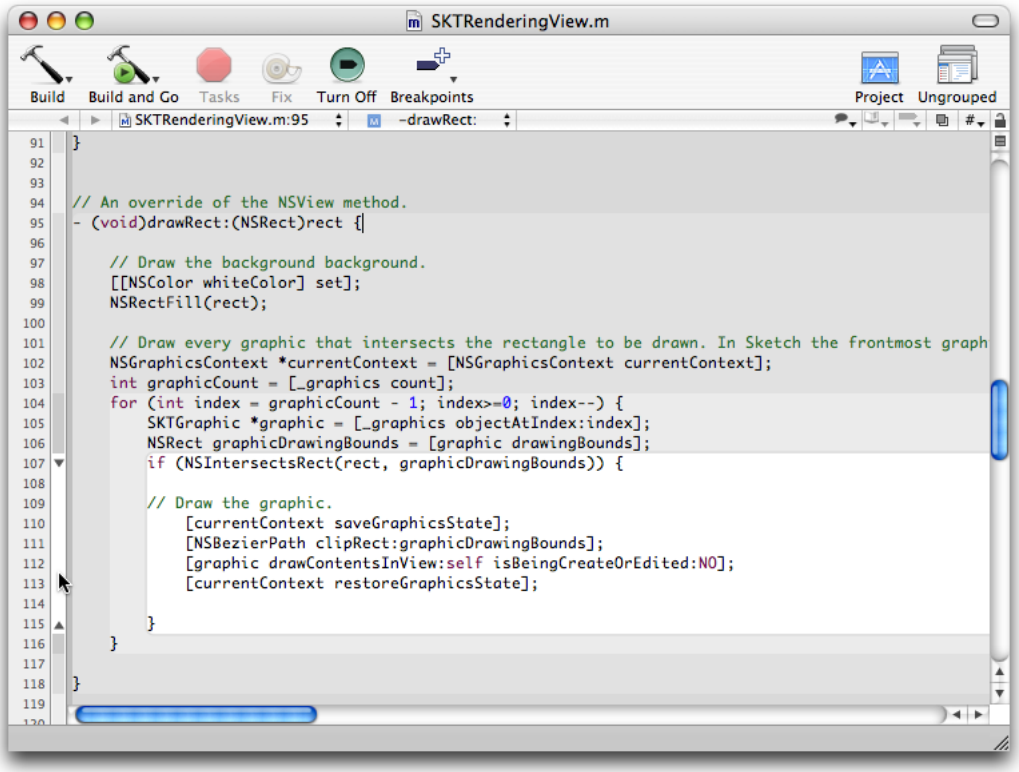
Figure 1 The editor focus ribbon




```
92
93
94  // An override of the NSView method.
95  - (void)drawRect:(NSRect)rect {
96
97      // Draw the background background.
98      [[NSColor whiteColor] set];
99      NSRectFill(rect);
100
101     // Draw every graphic that intersects the rectangle to be drawn. In Sketch the frontmo
102     NSGraphicsContext *currentContext = [NSGraphicsContext currentContext];
103     int graphicCount = [_graphics count];
104     for (int index = graphicCount - 1; index>=0; index--) {
105         SKTGraphic *graphic = [_graphics objectAtIndex:index];
106         NSRect graphicDrawingBounds = [graphic drawingBounds];
107         if (NSIntersectsRect(rect, graphicDrawingBounds)) {
108
109         // Draw the graphic.
110             [currentContext saveGraphicsState];
111             [NSBezierPath clipRect:graphicDrawingBounds];
112             [graphic drawContentsInView:self isBeingCreateOrEdited:NO];
113             [currentContext restoreGraphicsState];
114
115         }
116     }
117
118 }
119
120
```

- **Code focus:** You can use the focus ribbon to survey the scope levels of a source file. Moving the pointer along the focus ribbon, you can highlight the code at particular focus levels, as shown in Figure 2.

Figure 2 Highlighting code using the focus ribbon



- **Code folding:** To allow you to concentrate on specific areas of a source file, editor windows now allow you to collapse scoped code or comments so that they are not shown in the content area. To fold or unfold code you can use the triangles that appear in the focus ribbon or the commands in the View > Code Folding menu or the editor shortcut menu. Folded text is represented by a small graphic in the editor pane until it is unfolded.
- **Inline code completion:** The editor now suggests the most likely completion inline (reducing the need to choose a completion from the completion pop-up menu). To accept the suggested completion, press Tab. To display the completion pop-up menu with all the suggested completions, press Option-Escape.

  Figure 3 shows that the editor suggests the `tableCopy` completion to the `mu` text. The text `table` is highlighted to indicate that all the suggested completions to `mu` start with `table`. Pressing Tab takes the `table` subcompletion.

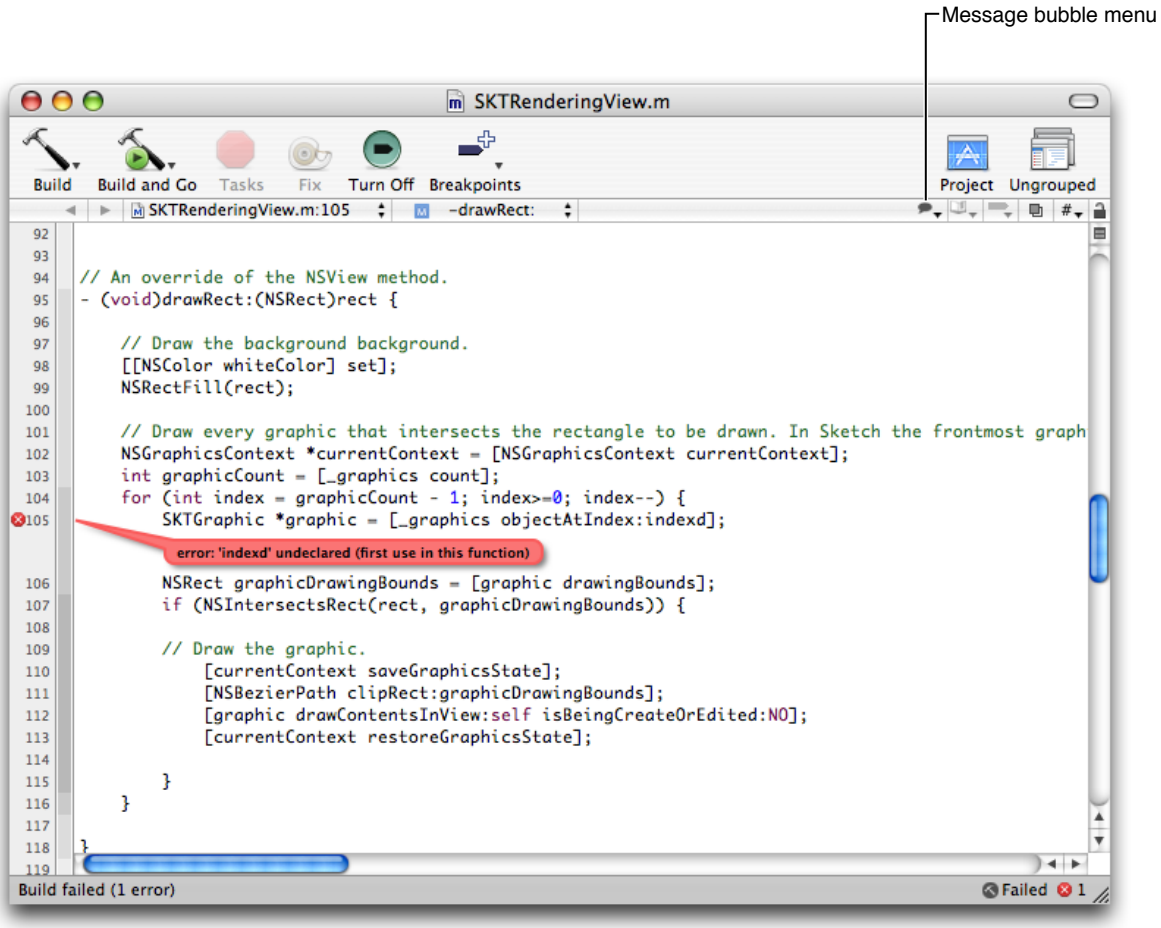
Figure 3 Inline code completion

```
[currentContext saveGraphicsState];
[NSBezierPath clipRect:graphicDrawingBounds];
[graphic drawContentsInView:self isBeingCreateOrEdited:NO];
[graphic mutableCopy
[currentContext restoreGraphicsState];
```

Code completion works with C-based source code, filenames in `#include` and `#import` statements, and build setting names in build configuration files.

- **Message bubbles:** Xcode now displays information about your project in source editor windows using message bubbles, as shown in Figure 4. This feature lets you, for example, determine the cause of a build error without having to examine up the Errors & Warnings group in the Groups & Files list. As you fix errors, you can jump to the next one by choosing Build > Next Build Warning or Error.

Figure 4 Message bubble displaying a build error



The bubble menu in a source editor window lets you specify the kind of message bubbles you're interested in: error bubbles, warning bubbles, or breakpoint bubbles.

- **Function menu:** In addition to C-based languages, the function menu now works with Java, Perl, Ruby, and other languages.

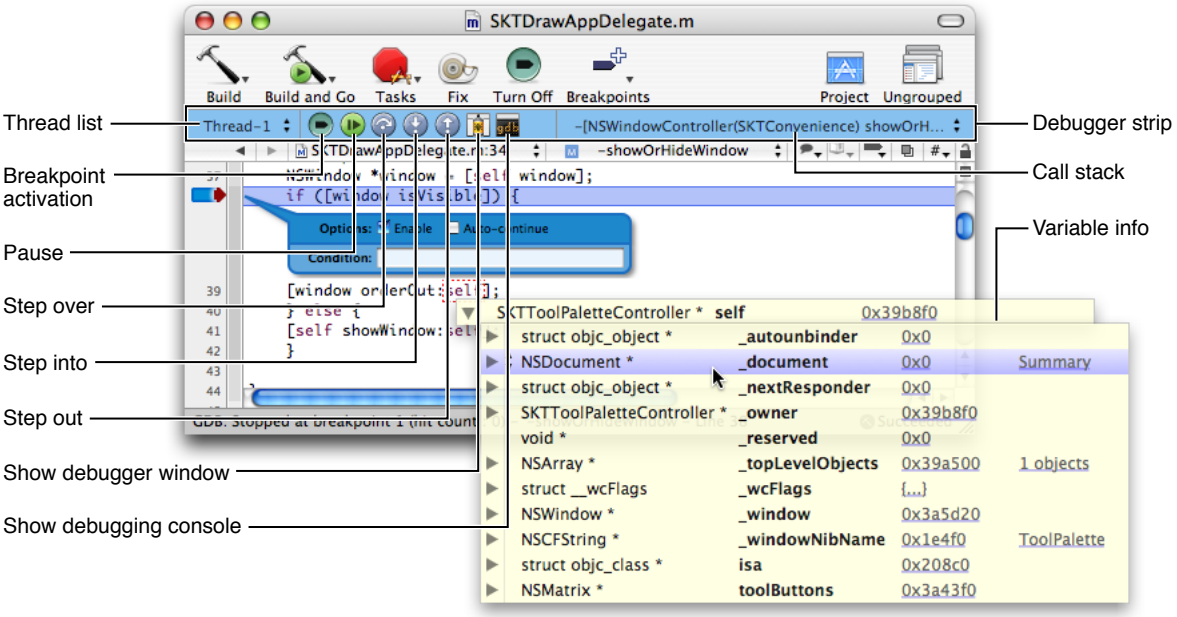
Streamlined Debugging

Xcode 3.0 improves your debugging workflow and allows you to debug applications in their natural environment. The new features that improve your debugging experience are:

- **The debugger strip** provides editor-based debugging. The debugger strip is a small control strip that appears above the editor pane in source editor windows. With the debugger strip you can perform a number of debugging tasks without moving away from your code. That is, you don't have to open the Debugger window to pause your program or to step through it.
- **The debugger HUD (heads-up display)** lets you debug your program without activating the Xcode application. The debugger HUD is a window tied to a particular program that floats about other windows in your desktop. This application-centered debugging lets you debug you program in its natural environment.
- **The debugger datatip** allows you to view the values of variables at runtime in a source editor window. You don't have to switch to the Debugger window to view a variable's values.

Figure 5 shows an editor window in which a program is being debugged.

Figure 5 Debugging code through an editor window

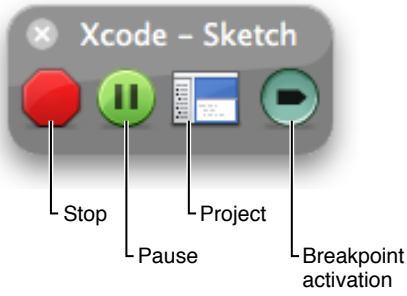


The debugger datatip appears when you place the pointer over a variable. This datatip lets you view details about the variable, such as its type and value, while debugging a program without leaving the editor.

Note: To turn on datatips for variables while debugging, choose the feature from the Debug menu.

In addition to being able to debug code from an editor window, you can debug your program without making Xcode the active application using the debugger HUD (Figure 6). This feature provides you the benefits of a GUI-based debugger without introducing interactions that would not happen during normal execution, such as another application becoming active.

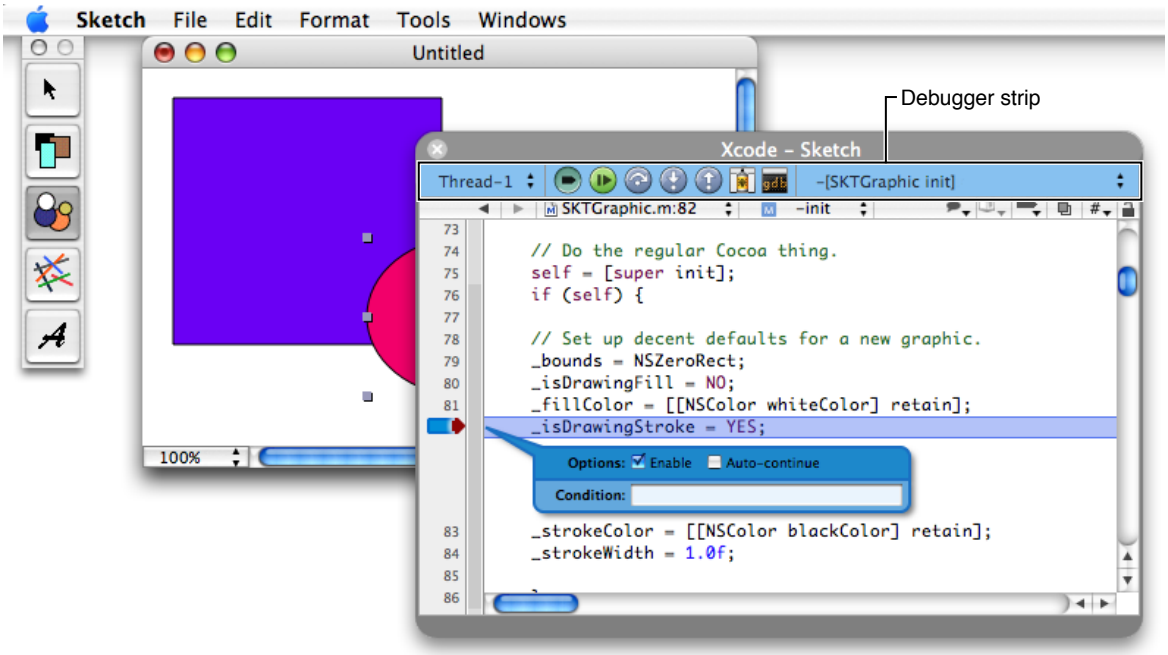
Figure 6 The debugger HUD in a running program



To open the debugger HUD for a running program, choose HUD from the Debug menu in the corresponding Xcode project. Alternatively, in the Debug preferences pane, you can specify that Xcode open the HUD automatically when you run your application.

When you pause a program through the HUD or it reaches a breakpoint (and breakpoints are active), the HUD displays the source file containing the breakpoint reached in a source editor, as shown in Figure 7. The HUD source editor provides the same functionality that source editor windows provide in Xcode. That is, you can perform the same debugging operations in the HUD that you can perform within a source editor in Xcode.

Figure 7 The debugger HUD in a paused program



Note: The Console window now combines the functionality of the Run Log and Standard IO windows.

Refactoring

Xcode now provides the ability to refactor your code. **Refactoring** allows you to improve the readability and maintainability of a program's source code while retaining its functionality and behavior. The refactoring operations that modify source code are called **transformations**. The transformations Xcode performs are tailored to the Xcode IDE. Therefore, in addition to source files, Xcode can process nib files and key-value-coding (KVC) methods.

Xcode 3.0 implements five refactoring transformations:

- **Rename:** Transformation that renames a declaration. Xcode renames the symbol in the appropriate source files and nib files in the project.
- **Create superclass:** Transformation that creates a superclass for an Objective-C class. Xcode inserts a new class in the inheritance hierarchy of the class being operated on. Skeleton interface and implementation code for the new superclass is placed in the interface and implementation files of the class being transformed.
- **Move up:** Transformation that moves instance variables and methods from an Objective-C class to the parent class.
- **Encapsulate:** Transformation that creates accessors for an instance variable or C `struct` field.
- **Modernize for loop:** Transformation that converts `for` and `while` loops to use the less verbose and more efficient Objective-C 2 `for` loop. This transformation is performed only on loops that iterate over all members of an array or set.

Note: Project indexing must be completed before Xcode can perform refactoring transformations. In addition, files that don't compile successfully may not be processed by the refactoring engine.

To perform a refactoring operation:

1. Select the symbol to refactor in a text editor window.

   If it's possible to perform a transformation on the symbol, the Refactor command in the Edit menu is available.

2. Choose Edit > Refactor.

   The Refactoring window appears.

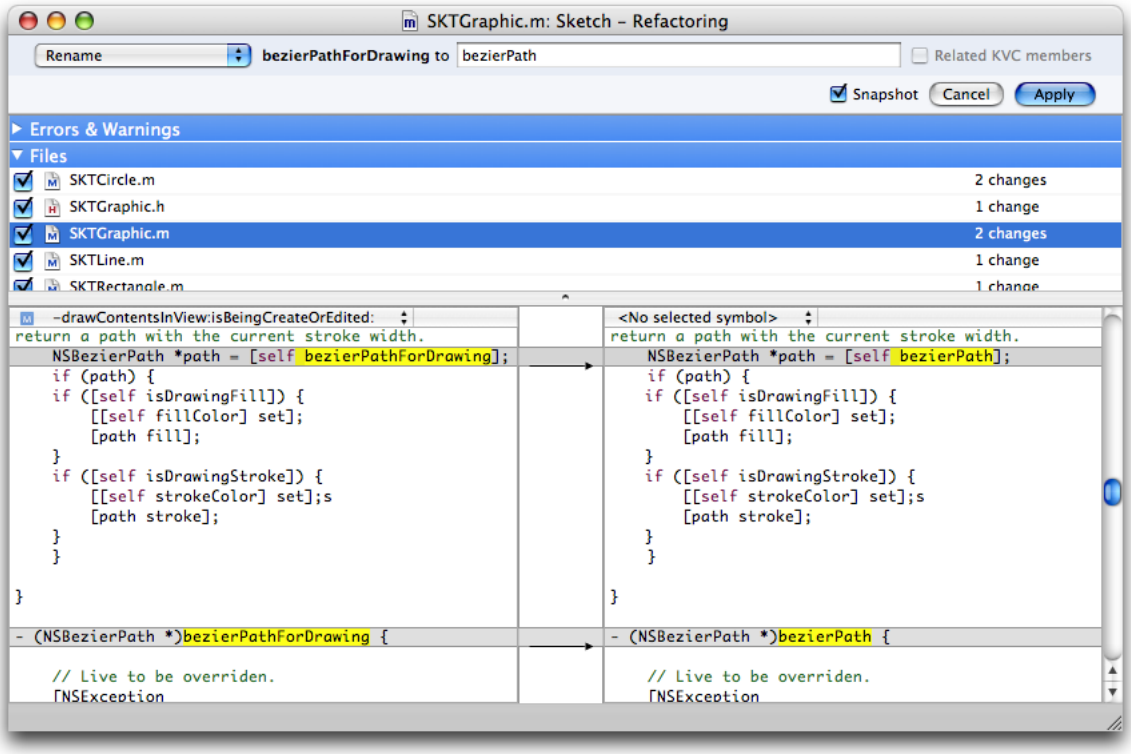
3. Choose the refactoring operation to perform.

   The refactoring-operation pop-up menu allows you to choose an operation Xcode can perform on the selected symbol.

4. Preview the refactoring operation.

Click Preview. The Refactoring window (Figure 8) displays the files that the refactoring operation would modify.

Figure 8 The Refactoring window



Highlighting a file displays a comparison pane pinpointing the changes Xcode would perform if you execute the operation. The left side of the comparison pane shows the unmodified file. The right side shows how the file would look after applying the changes.

5. Apply the proposed changes.

   Click Apply. Xcode makes the changes you previewed on the selected files in the file list.

   If Snapshot is selected in the Refactoring window, Xcode creates a snapshot of your project before making the changes. For more on snapshots, see "Project Snapshots" (page 21).

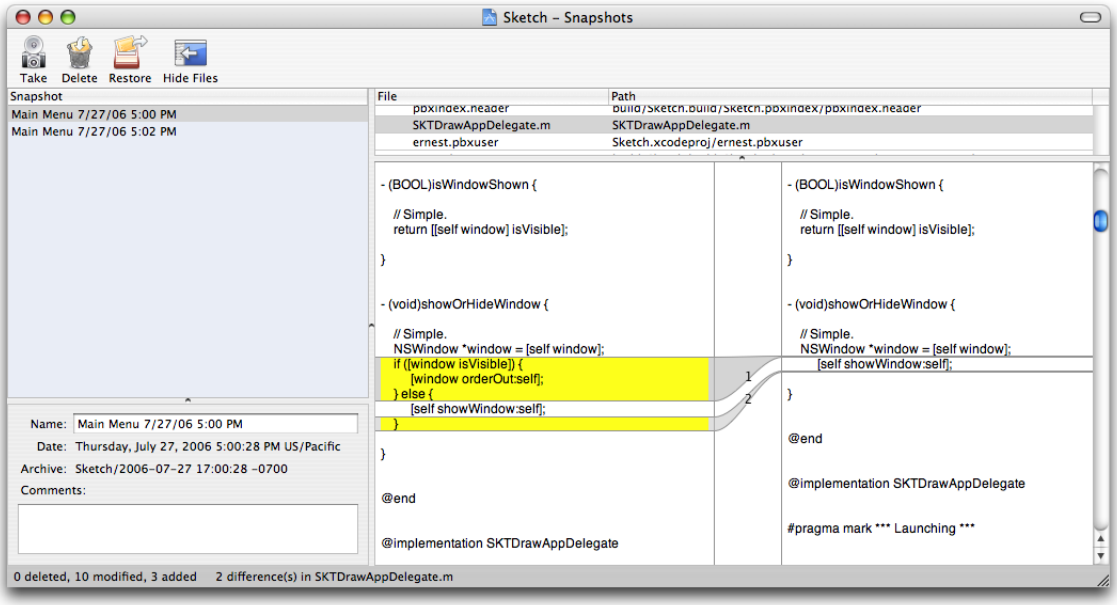
Important: Refactoring works only in pure C or Objective-C source files, that is, headers files and implementation files with no C++ code. In addition, the refactoring engine doesn't process Objective-C++ files (those with the `.mm` extension. If you have `.mm` files that do not contain Objective-C++ code, you must change their extension to `.m` so that they are processed by the refactoring engine.

Project Snapshots

To allow you to more easily revert changes you made to your project, Xcode now lets you create snapshots of your entire project directory. With snapshots you can revert to a previously saved state when you need to discard changes you made to several files in a project. This feature is particularly useful when you need to make exhaustive or risky modifications across several files and you want to ensure that you can safely discard those changes.

The Snapshots window (Figure 9) lets you view the files that make up particular snapshots as well as restore or delete snapshots (to conserve disk space).

Figure 9 The snapshot window



In the Snapshots window you can also rename snapshots to identify specific snapshots with suitable names.

Xcode stores snapshots in your home directory at `~/Library/Application\ Support/Apple/Developer Tools/Snapshot Repository`. To change this location, use the `XCSnapshotRepositoryPath` Xcode expert setting. For example, the following command sets the snapshot directory to `/Volumes/Backup/Xcode Snapshots`:

```
> defaults write com.apple.Xcode XCSnapshotRepositoryPath /Volumes/Backup/Xcode\
 Snapshots/
```

To learn more about snapshots, see "Project Snapshots" (page 21).

Build Settings Editor

The build settings editor contains organizational improvements to make build settings easier to find and functional changes that let you view the buildtime values of all build settings.

The build setting list now shows how Xcode organizes build settings by listing them under the build setting category they belong to. You can reveal and hide the build settings under each category to display only the build settings you are interested in. When the Research Assistant is open, it displays the description of the build setting selected in the list. See "The Research Assistant" (page 24) for details.

Using the build settings shortcut menu, you can now configure the build setting list to display build setting titles (for example, Build Products Path, Build Variants, and so on) or the corresponding build setting names (for example, `SYMROOT`, `BUILD_VARIANTS`, and so forth). You can also configure the list to display build setting definitions (also known as build setting specifications) or the values the build settings at the project or target level (which is what the build setting editor in earlier versions of Xcode displays).

The build settings editor now displays build setting values using text, checkboxes, or pop-up menus, depending on the value type of the build setting. To edit a build setting, you can double-click it; select it, wait, and click it again (see "Editing List Items" (page 29) for details); or select it and choose Edit Definition at This Level from the Action pop-up menu.

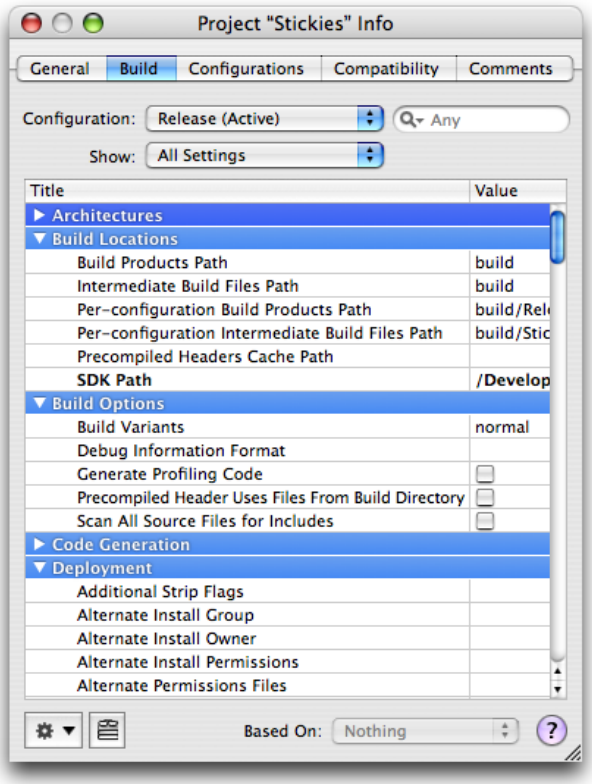
The Action pop-up menu also lets you add architecture-specific build settings. See "Architecture-Specific Build Settings" (page 23) for more information.

To delete a build setting definition, select the build setting from the list and choose Delete Definition at This Level from the Action pop-up menu.

Using the search field you can now filter the build setting list using one of the following criteria: name, title, definition, value, category, or description.

Figure 10 shows the improved build settings editor.

Figure 10 The build settings editor



Architecture-Specific Build Settings

The build settings editor now lets you define architecture-specific build settings. **Architecture-specific build settings** allow you to specify build options for specific architectures, such as PowerPC or Intel. This feature is particularly useful when building universal binaries of your applications, which can be used in PowerPC-based Macs and Intel-based Macs.

To add an architecture-specific build setting:

1. Select the build setting to which you want to an architecture-specific version.
2. Choose Add Per-Architecture Setting from the Action pop-up menu.

Note: Architecture-specific build settings override their corresponding architecture-independent build setting. To use the architecture-independent value in the definition of an architecture-specific build setting, add `$inherited` to the definition.

Important: Architecture-specific build settings are supported only in Xcode 3.0. See "Project File Compatibility Checking" (page 29) for details about maintaining compatibility with earlier versions of Xcode.

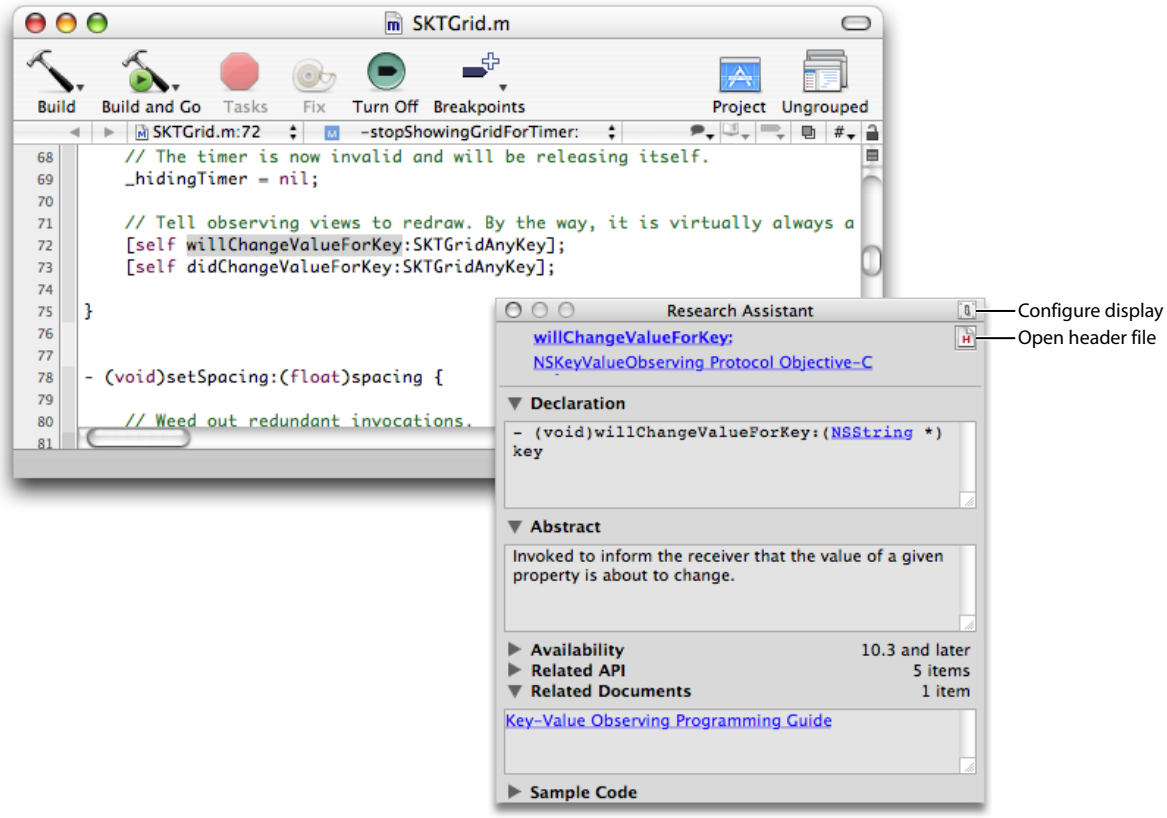
The Research Assistant

In earlier versions of the Xcode application, you can Option–double-click a symbol in the text editor to view its reference documentation in the documentation window. However, this practice takes your focus away from the code you're working on. It may also provide you with more documentation than you need. In most cases, you may be looking for a symbol's declaration or availability information. The documentation window shows the entire reference documentation for the symbol together with documentation for other symbols, which you must scan to find the information you need. Xcode 3.0 provides a mechanism to view reference documentation that addresses these issues: the Research Assistant.

The Research Assistant window (Help > Research Assistant) offers a concise view of essential reference documentation for a specific symbol. The information you can view in the Research Assistant includes a symbol's declaration, description, and availability information. You can also access sample code and related documents through hyperlinks. In addition, you can configure the order in which these items are displayed and whether they are shown in the window. The Research Assistant also provides reference documentation on build settings when you configure them in the build settings editor or in configuration files.

When you edit source code or a build setting, the Research Assistant window, shown in Figure 11, unobtrusively displays the corresponding reference documentation in the way you prefer to view it.

Figure 11 The Research Assistant window



After analyzing the information the Research Assistant provides for a symbol, you can access the symbol's entire reference documentation or open the symbol's header file from within the Research Assistant.

Data Model Versioning and Mapping

Xcode 3.0 adds the ability to have versioned data models. With this feature you can have multiple versions of a data model in a project. A versioned data model appears as a group containing data model files (each for each version of the data model) in the Groups & Files list.

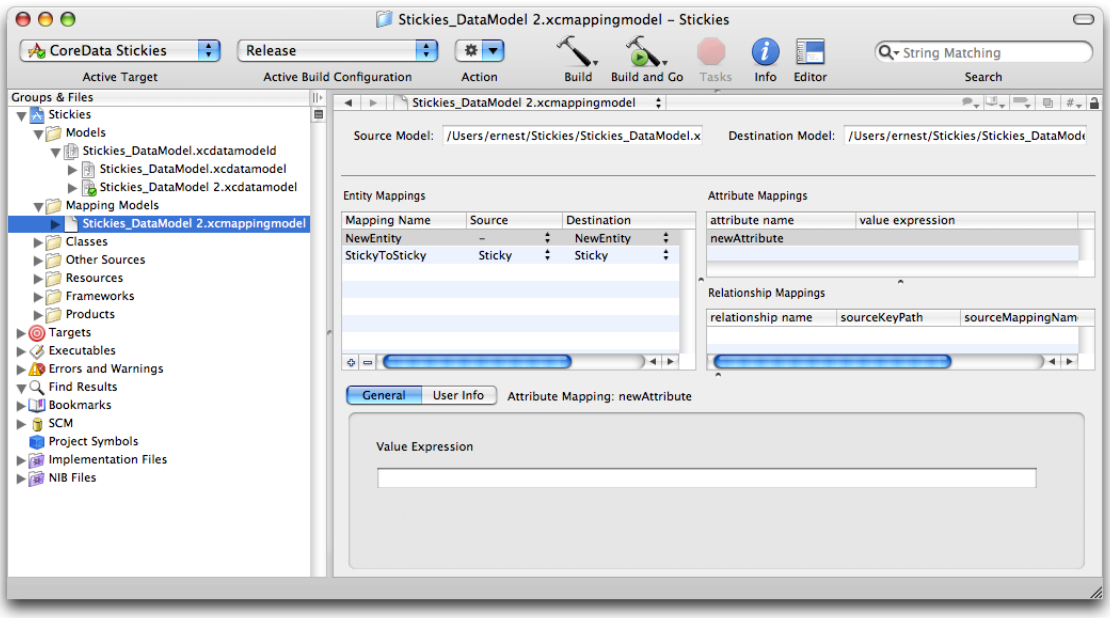
After creating a version of a data model, you may need to migrate your Core Data stores. To perform this task at runtime, you use new Core Data API and a mapping model file, which describes how to move data from one model to another.

To create a new version of a data model:

1. Select the data model in the Groups & Files list.
2. Choose Design > Data Model > Add Model Version.

To create a mapping model file for a new version of a data model use the Design > Mapping Model new-file template. Then use the mapping model editor to define the mappings between the source and target models. Figure 12 shows two versions of a data model and the mapping model editor.

Figure 12 Versioned data models and the mapping model editor



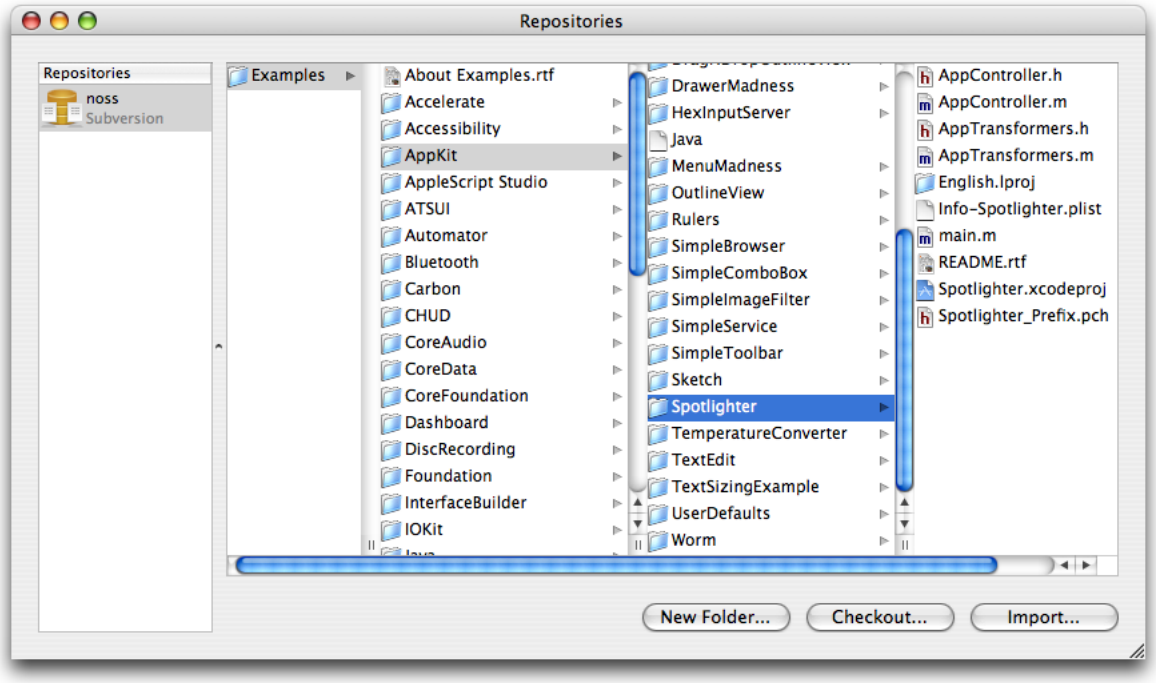
For detailed information on versioned data models and the data model migration process, see *Core Data Model Versioning and Data Migration Programming Guide*.

Important: Versioned data models are supported only in Xcode 3.0. See "Project File Compatibility Checking" (page 29) for details about maintaining compatibility with earlier versions of Xcode.

SCM Repository Management

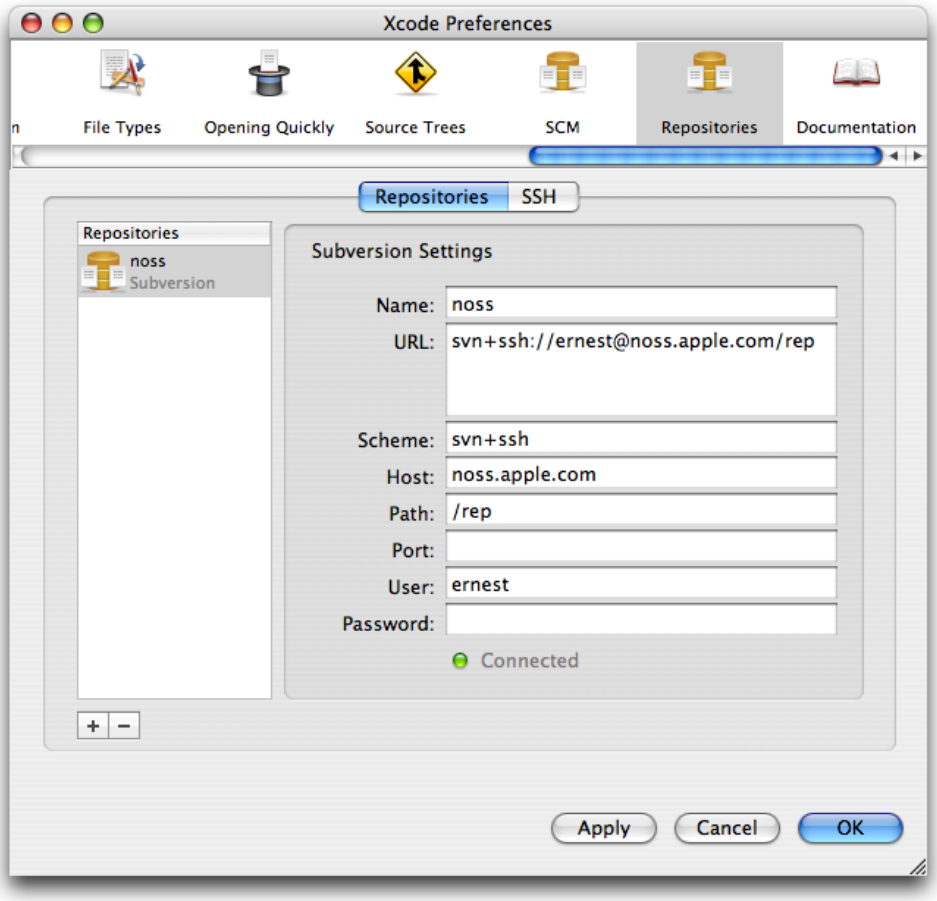
Xcode now provides a GUI that allows you to navigate SCM repositories. The Repositories window (Figure 13) lets you import an unmanaged project into a repository and check out projects from a repository to your computer.

Figure 13 The Repositories window



You configure access to repositories in Xcode Preferences > Repositories. The Repositories preferences pane, shown in Figure 14, allows you to enter repository access information for the SCM systems Xcode supports: Subversion, CVS, and Perforce (the settings shown are the ones used for Subversion repositories). It also lets you store passphrases for the SSH keys in your home directory (`~/.ssh`), which Xcode uses in SSH-based connections.

Figure 14 The Repositories preferences pane



Debugging Information Format

In new Xcode 3.0 projects the format for debugging information is DWARF (Debugging With Attributed Record Formats). The Stabs format is still available, however. DWARF's extensibility attributes as well as its ability to describe complex execution environments provide a strong foundation upon which future debugging facilities in Xcode and Mac OS X can be built.

For more information on DWARF, see http://dwarf.freestandards.org.

Xcode also supports storing debug information in dSYM files. A **dSYM file** stores an executable's debug information to minimize the size of the executable file without compromising the program's debugging experience. For example, if you or your customer need to debug an executable on the customer's computer, you don't have to build a debug version of the program and deliver it to the customer. All you would need to do is make the executable's dSYM file available to the user. You or the use can then debug the program just as you would debug it during development.

Editing List Items

The procedure to edit text items in lists—such as the Groups & Files list or the build setting list in the build setting editor—has changed to resemble how filenames are changed in Finder windows. To change a list item:

1. Select the item by clicking it once.
2. Click the item again and wait about half a second.

   The item's text can now be edited.

Building for Multiple Architectures

The Architectures build setting editor—which appears when you double-click the Architectures (`ARCH`) build setting—now offers 32-bit and 64-bit as its options:

- Selecting only 32-bit produces a 32-bit universal binary (`ARCH=ppc i386`).
- Selecting only 64-bit produces a 64-bit universal binary (`ARCH=ppc64 x86_64`)
- Selecting both 32-bit and 64-bit produces a universal binary for 32-bit and 64-bit architectures (`ARCH=ppc ppc64 i386 x86_64`).

In Debug configurations of Xcode 3.0 projects, Architectures is defined as `$NATIVE_ARCH`, referring to the architecture of the computer Xcode is running on. Selecting or deselecting the 32-bit or 64-bit options in the Architectures build setting editor when the build setting is defined as `$NATIVE_ARCH` produces a binary appropriate for debugging on the host computer:

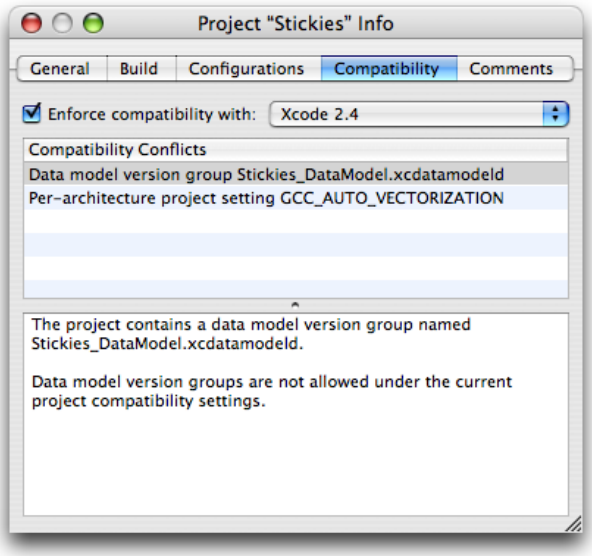
- Selecting only 32-bit produces a 32-bit binary for the host architecture (`ARCH=$NATIVE_ARCH_32_BIT`).
- Selecting only 64-bit produces a 64-bit binary for the host architecture (`ARCH=$NATIVE_ARCH_64_BIT`).
- Selecting both 32-bit and 64-bit produces a 32-bit binary on a 32-bit computer and a 64-bit binary on a 64-bit computer (`ARCH=$NATIVE_ARCH_ACTUAL`).

When you need to define the Architectures build setting to a more granular degree, use the textual editor (see "Editing List Items" (page 29)). For example, to generate a binary for the host architecture containing both 32-bit and 64-bit object code, define Architectures as `$NATIVE_ARCH_32_BIT $NATIVE_ARCH_64_BIT`.

Project File Compatibility Checking

Some of the features Xcode 3.0 introduces are not compatible with the project file format used in Xcode 2.4. In a team environment you may need to ensure a project file can be used in either Xcode 2.4 or Xcode 3.0. The Compatibility pane in the project editor (Figure 15) lets you specify the Xcode version with which a project file must remain compatible.

Figure 15 The Compatibility pane in the project editor



Important: Xcode performs some compatibility checking while you make changes to your project. However, to make sure your project remains compatible with a particular version of Xcode, you should confirm that the the Compatibility pane doesn't list any conflicts before sharing the project with other members of your team.

The Documentation Window

The content shown in the Xcode documentation window is now made up of document sets, which can be searched individually. Each document set can be installed individually. With this feature you can select a document set, such as Core Reference Library or Java Reference Library, to confine a search to a smaller set of documents. Confining searches to particular document sets produces smaller and more relevant search results. In addition, a new type of search, called Title Search, allows you to base Reference Library searches on document titles.

Note: Document sets other than the Core Reference Library set are considered additions to the core set. Therefore, when you install individual document sets, you must always install the Core Reference Library set, as well.

Coming soon: Xcode's support for multiple documentation sets will be open to third parties to add their own documentation to Xcode. At that time, Apple will provide a specification and tools for integrating documentation into Xcode.

Document Revision History

This table describes the changes to *What's New in Xcode*.

Date	Notes
2009-01-06	Made minor corrections.
2008-05-22	Updated for Xcode 3.1.
2006-08-01	New document that provides an overview of new and improved features in the Xcode application.