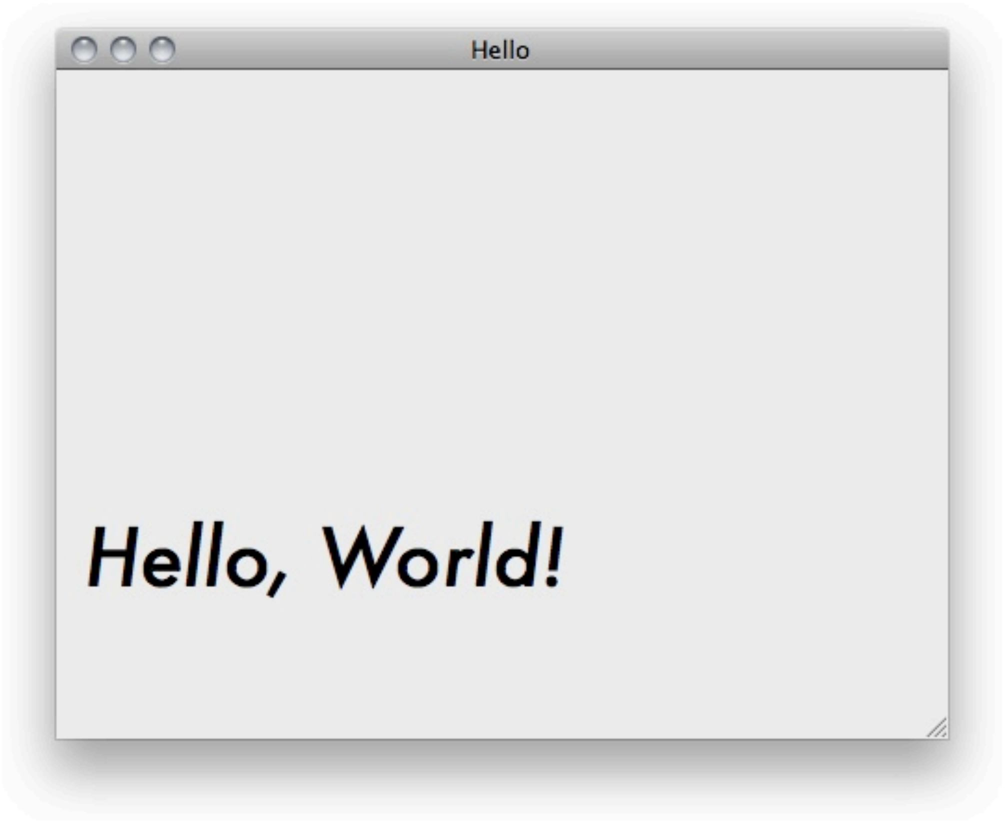
Tutorial: Using Xcode to Write “Hello, World!” for Mac OS X

This short tutorial shows how to create a project for a Mac application that prints text in a window. Working through this tutorial, you get acquainted with the software-creation workflow in Xcode: creating the project, designing the user interface, writing code, and running the application. You also learn how to fix code errors Xcode detects as you write the code, and you get an introduction to the Xcode debugging facilities.

Hello is a simple application. When the user launches it, a window appears, displaying the text “Hello, World!,” similar to the window shown in Figure 1-1.

Figure 1-1 The Hello application in action



Under the hood, the user interface consists of a window that contains a view. Views know how to display data. These objects have a built-in method through which Cocoa manages drawing into the view. You need to provide the code that draws the “Hello, World!” message.

In this tutorial you use Objective-C and the Cocoa framework to create a view and implement the drawing routine. You don’t need to be familiar with Cocoa or Objective-C to complete this tutorial, but you should be familiar with programming in some language, preferably a C-based or object-oriented language. To perform the tutorial you must have Xcode installed on your Mac. Visit developer.apple.com to download Xcode.

In this tutorial you:

- Create the Hello project
- Add a source file to the project
- Lay out the user interface of the Hello window
- Write the code that displays the message on the Hello window
- Build and run the Hello application
- View the messages produced by building the Hello application

Create the Cocoa Application Project

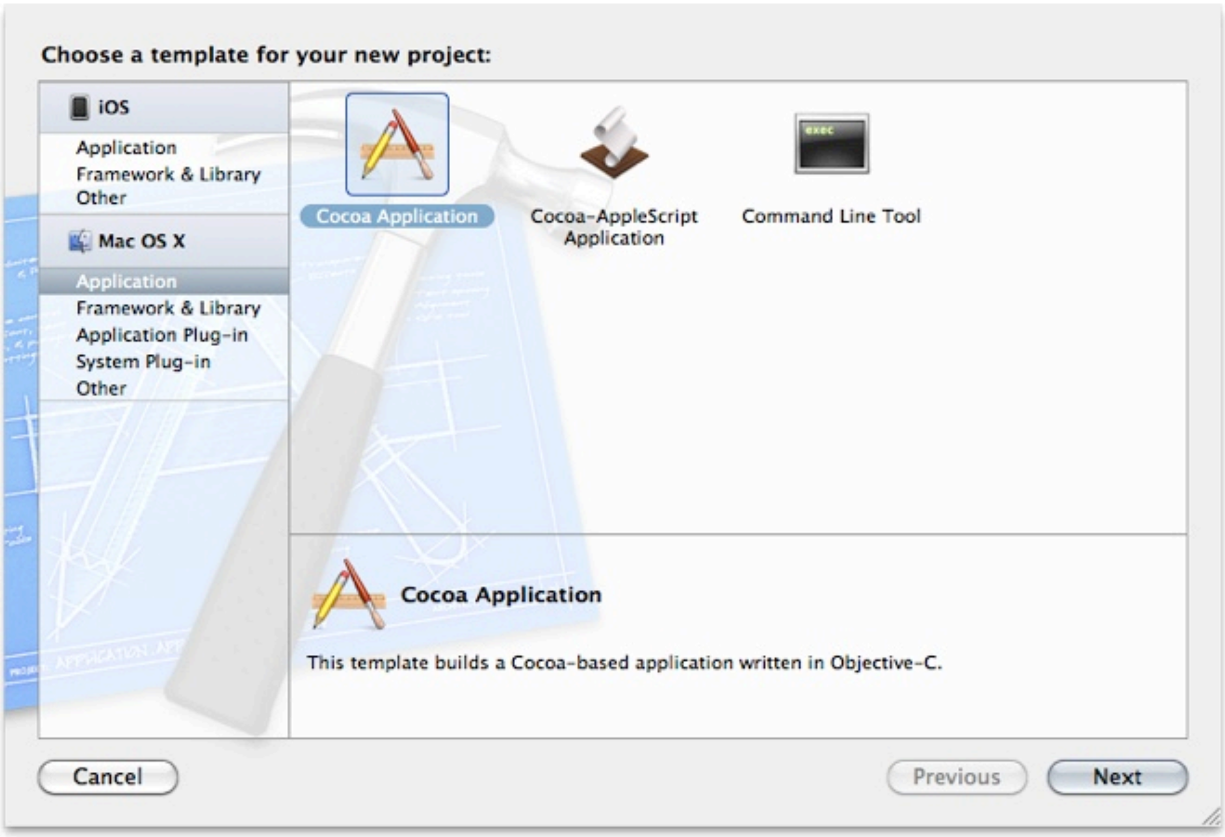
Xcode provides project templates that generate several types of products, including applications, frameworks, plug-ins, and static libraries.

To create the Cocoa application project on which the Hello application is based:

1. Launch Xcode, located in the Applications folder of your Xcode installation.

   Ensure that there are no Xcode windows open.

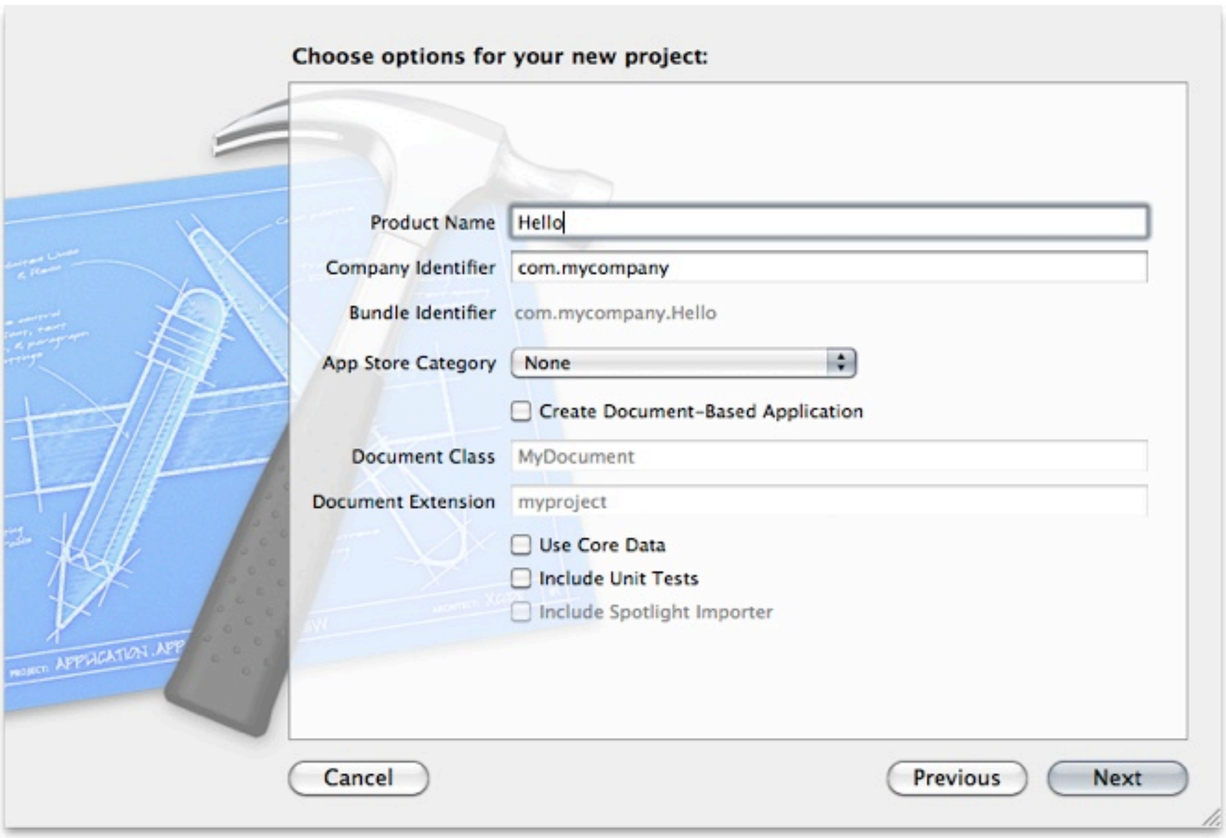
2. Choose File > New > New Project.
3. In the Mac OS X group, select Application, then select Cocoa Application template, and click Next.



Cocoa Application

This template builds a Cocoa-based application written in Objective-C.

4. Specify the options for the project, and click Next:
   - **Product Name:** `Hello`.
   - **Company Identifier:** `com.mycompany`.
   - **App Store Category:** None.
   - **Create Document-Based Application:** not selected, which disallows entry into the Document Class and Document Extension fields.
   - **Use Core Data:** not selected.
   - **Include Unit Tests:** not selected.


Choose options for your new project:

Product Name: Hello

Company Identifier: com.mycompany

Bundle Identifier: com.mycompany.Hello

App Store Category: None

☐ Create Document-Based Application

Document Class: MyDocument

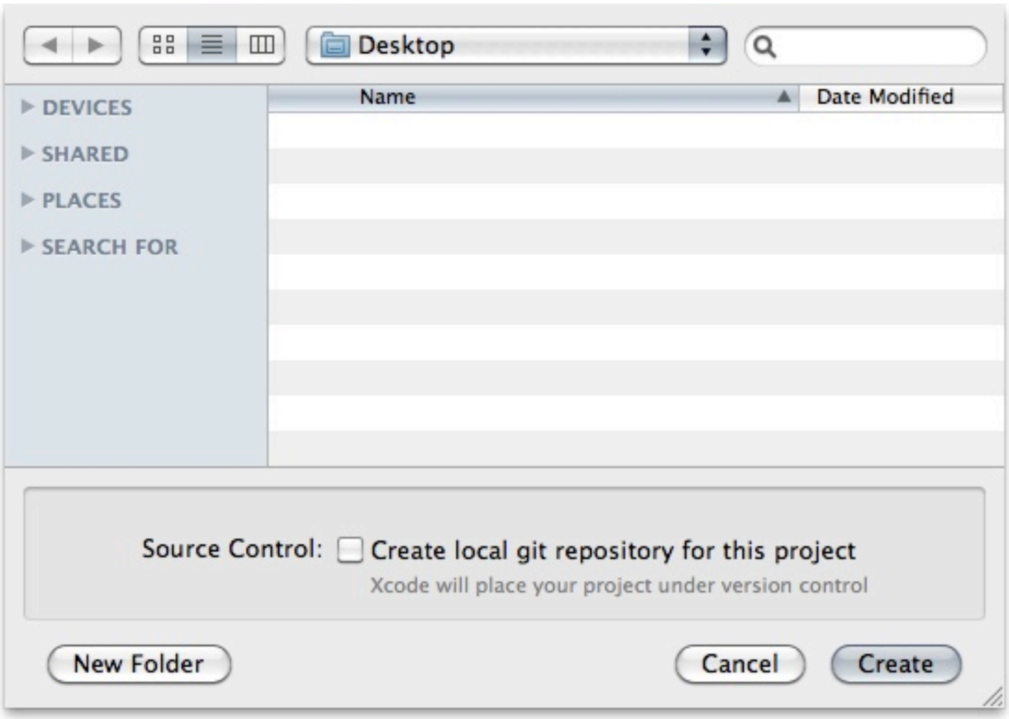
Document Extension: myproject

☐ Use Core Data

☐ Include Unit Tests

☐ Include Spotlight Importer

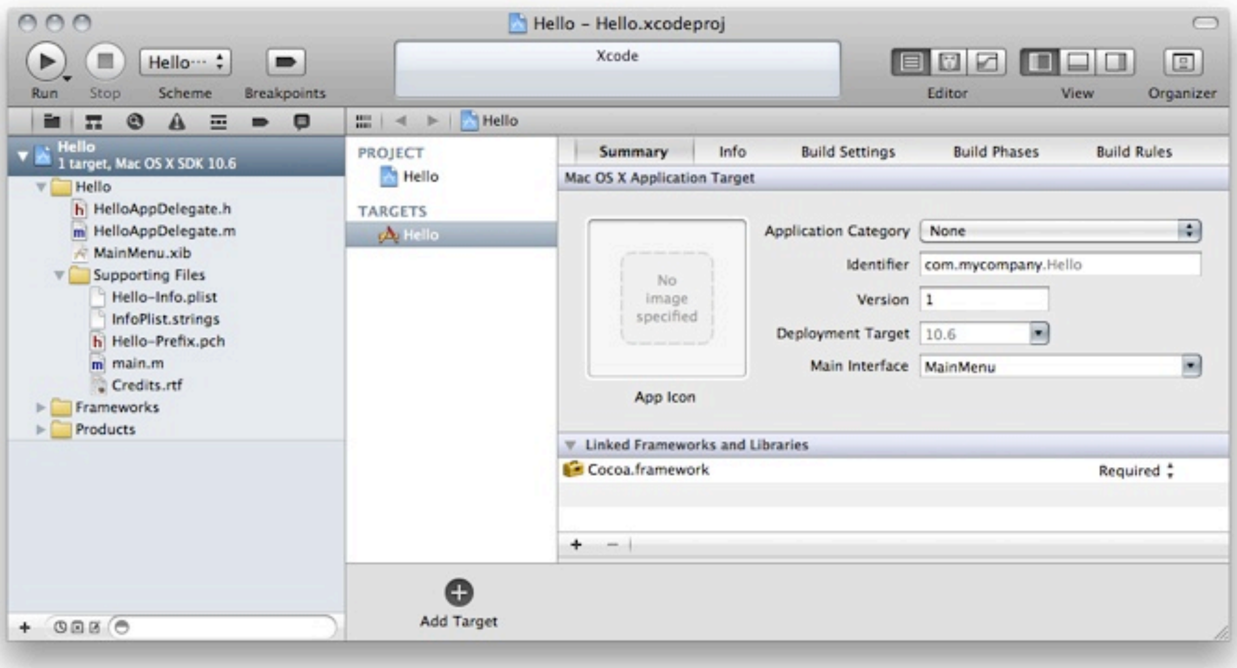
5. In the dialog that appears, navigate to the file-system location where you want to place the project directory (for example, the Desktop), ensure the “Create local git repository for this project” option is not selected, and click Create.



Source Control: ☐ Create local git repository for this project

Xcode will place your project under version control

After creating the project directory in your file system, Xcode opens it in a workspace window.



Application Category: None
Identifier: com.mycompany.Hello
Version: 1
Deployment Target: 10.6
Main Interface: MainMenu

Cocoa.framework Required

The project contents appear in a pane (known as the *project navigator*) on the left side of the window. The first item in the group represents the project, and it's named after the product name you specified when you created the project. Xcode groups the components of the Hello project in three groups:

- **Hello:** Contains the files that make up the project. These files include source code files and a user-interface file. This group also contains a subgroup, named Supporting Files, that contains files used in supporting tasks. You don't modify these files in this tutorial.
- **Frameworks:** Identifies frameworks or libraries your code relies on for its functionality; for example, the Cocoa framework.
- **Products:** Contains the products your project produces, such as an application.

Create the NSView Subclass

Now you add a class to the project through which the Hello application displays its message.

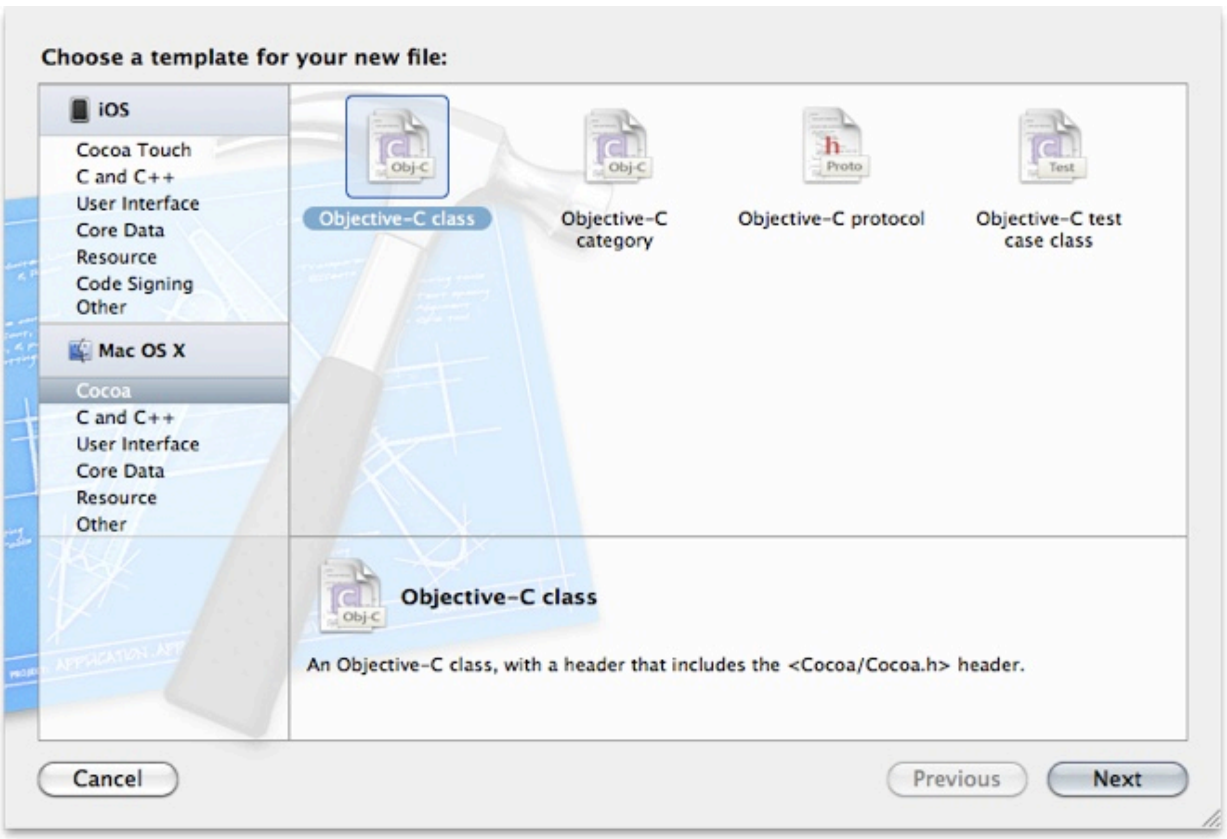
Cocoa draws in objects known as *views*. The essential functionality of a view is implemented by the `NSView` class, which defines the basic drawing, event handling, and printing architecture of an application. You typically don't interact with the `NSView` class directly. Instead you create an `NSView` subclass, and override the methods whose behavior you need to customize. Cocoa automatically invokes these methods.

To create the `NSView` subclass in the Hello project:

1. Choose File > New > New File.

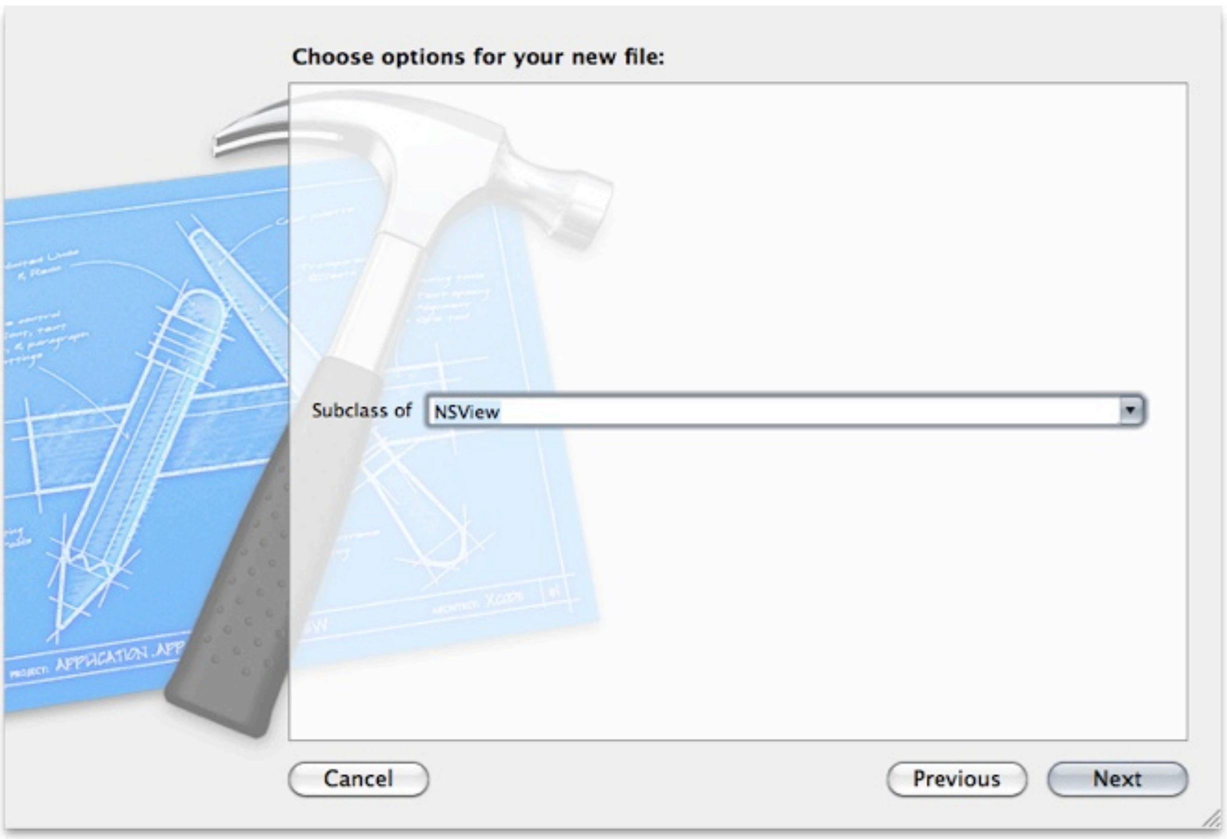
   Note: If the New File command is dimmed in the New menu, ensure that the Hello workspace window has the focus (that it is the frontmost window) by clicking inside it. The commands Xcode makes available in the menu bar depend on the window that has the focus. Many project-related commands are not available when a window other than a workspace window has the focus.

2. In the Mac OS X group, select Cocoa, then select the Objective-C class template, and click Next.



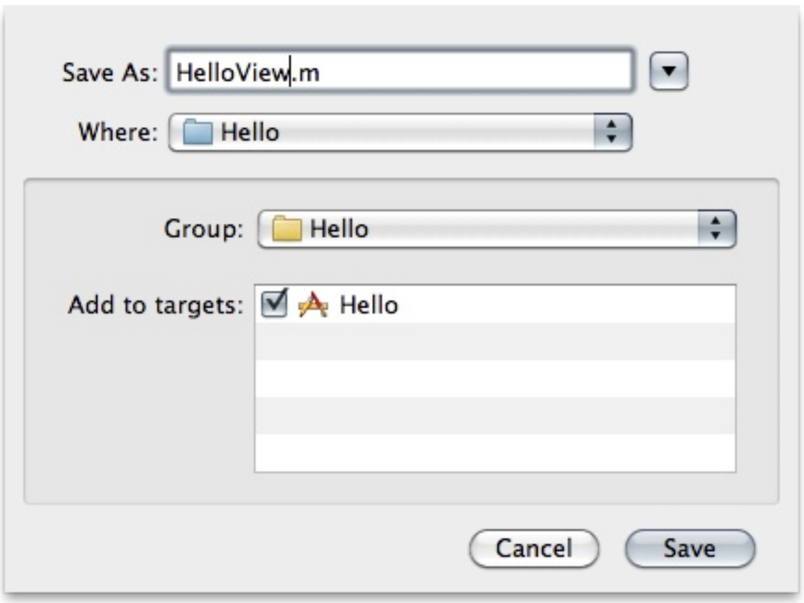
An Objective-C class, with a header that includes the <Cocoa/Cocoa.h> header.

3. Specify that the new file is a subclass of `NSView`, and click Next:



Subclass of NSView

4. In the dialog that appears, enter `HelloView.m` as the filename, choose the Hello group (which uses a yellow folder icon) from the Group pop-up menu, and click Save.

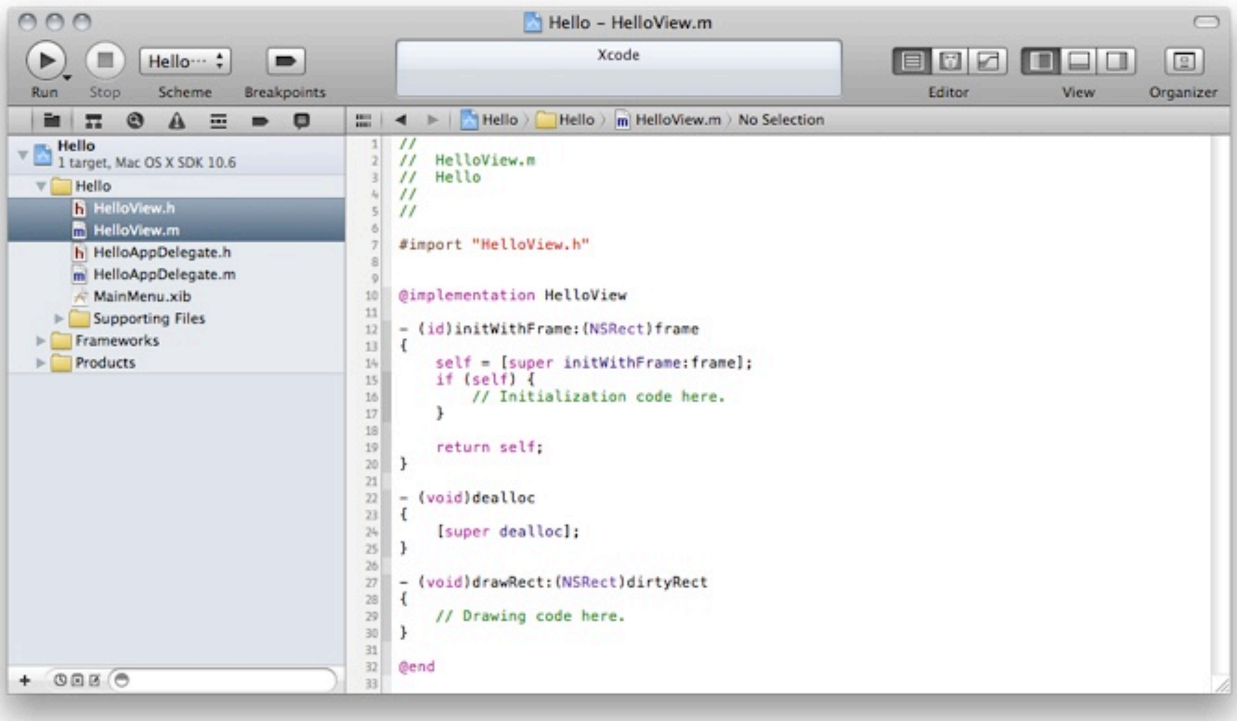

Save As: HelloView.m

Where: Hello

Group: Hello

Add to targets: [✓] Hello

Xcode adds the header and implementation files for the `HelloView` class to the project. They are listed in the project navigator.



```
//
//  HelloView.m
//  Hello
//
//

#import "HelloView.h"


@implementation HelloView

- (id)initWithFrame:(NSRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.
    }

    return self;
}

- (void)dealloc
{
    [super dealloc];
}

- (void)drawRect:(NSRect)dirtyRect
{
    // Drawing code here.
}

@end
```

Design the User Interface

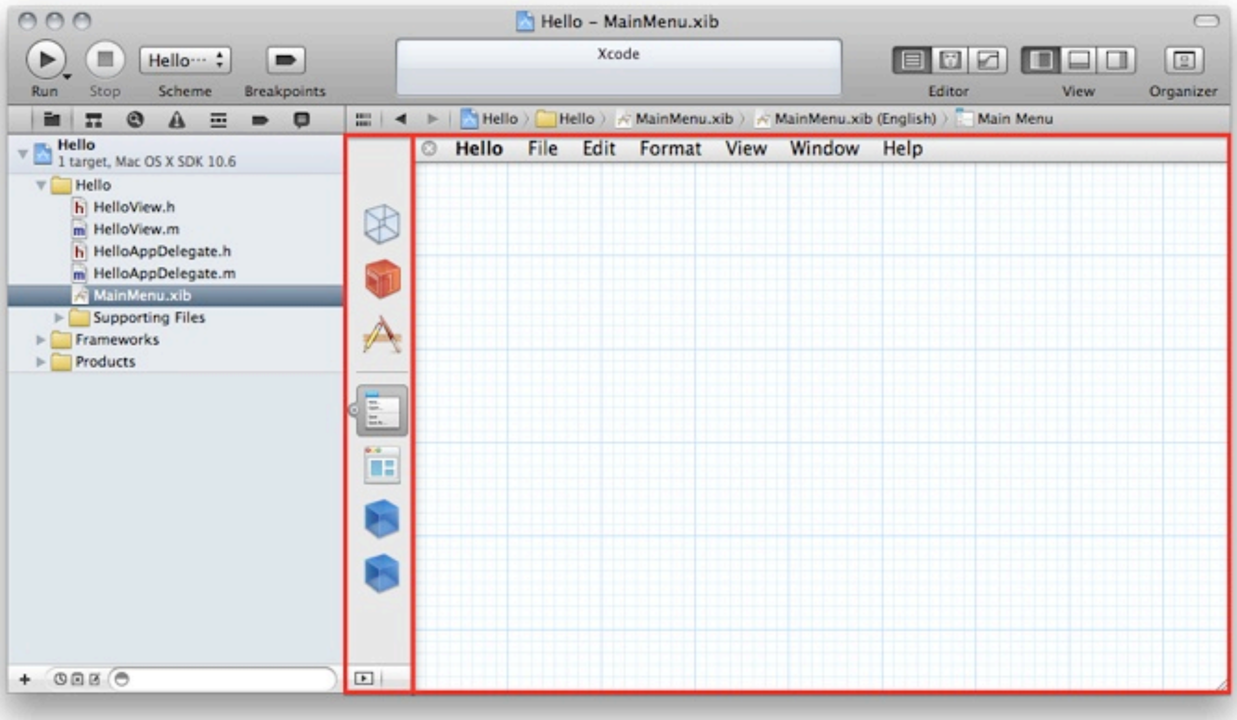
Interface Builder is the graphical user-interface editor used to edit the documents, called *nib files*, that define an application's user interface. You directly lay out and manipulate user-interface objects (known as *controls*) to construct your user interfaces.

Add an instance of the `HelloView` class to the Hello application window:

1. In the project navigator, select the `MainMenu.xib` file. Xcode opens the file in Interface Builder, the Xcode user-interface editor.

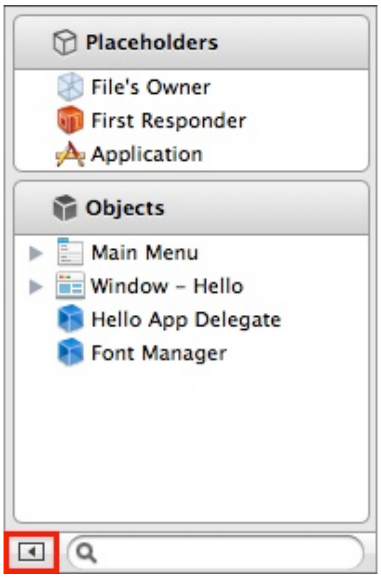
This editor has two major areas, highlighted in Figure 1-2: the dock (on the left) and the canvas (on the right). The dock displays the objects in the nib file. The canvas is where you lay out your application's user interface using the objects in the nib file.

Figure 1-2 The Interface Builder dock and canvas

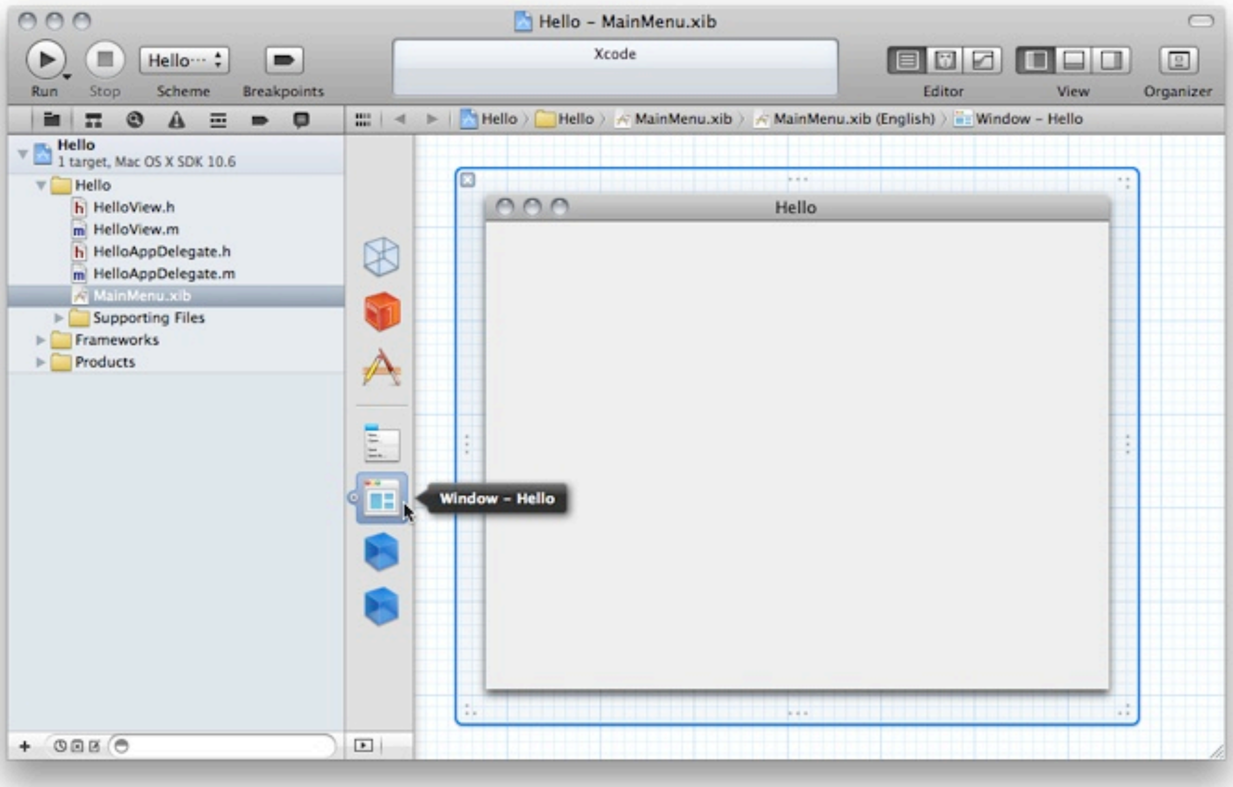


2. If the canvas shows the Hello menu bar object, as the previous screenshot does, click its close box (x) to remove it from the canvas.
3. If the dock appears in outline view (Figure 1-3) instead of icon view (Figure 1-2), click the highlighted button to change it to icon view.

Figure 1-3 Interface Builder dock in list view



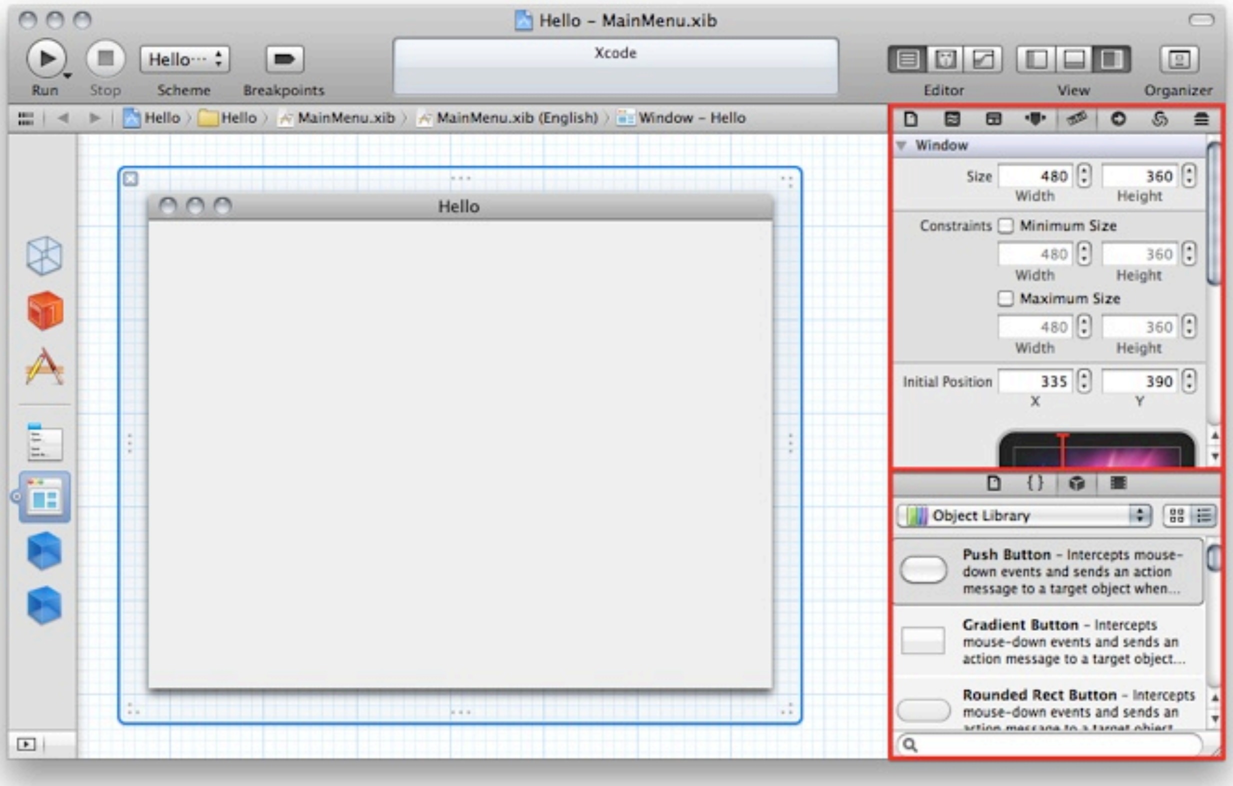
4. Click the "Window – Hello" item in the dock to display the Hello window in the canvas.



5. Choose View > Navigators > Hide Navigator to narrow the focus of the workspace (you won't be navigating the project in the next few steps).
6. Choose View > Utilities > Show Utilities.

   The utility area (Figure 1-4) contains two panes: the inspector pane (top) and the library pane (bottom). The library pane contains libraries for file templates, code snippets, objects, and media.

Figure 1-4 The utility area in the workspace window



Size: Width 480, Height 360
Constraints: Minimum Size — Width 480, Height 360
Maximum Size — Width 480, Height 360
Initial Position: X 335, Y 390

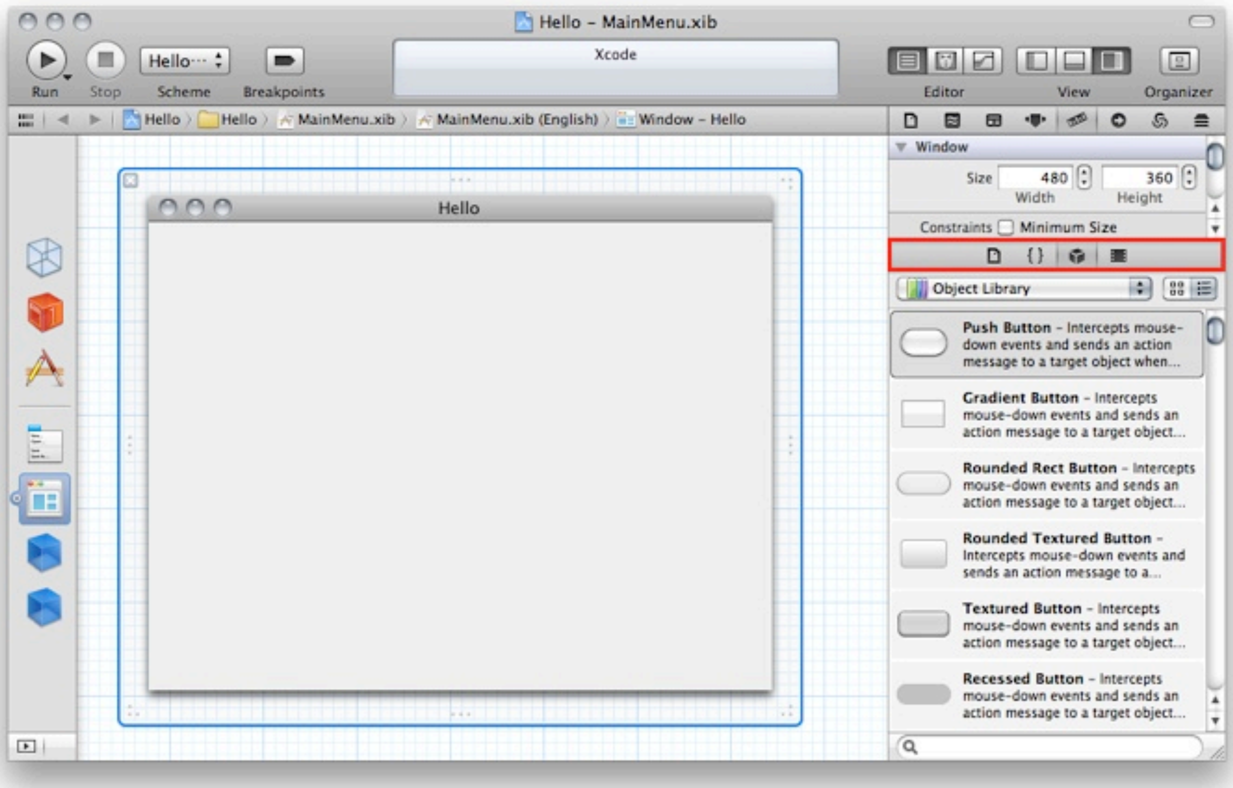
Push Button – Intercepts mouse-down events and sends an action message to a target object when...

Gradient Button – Intercepts mouse-down events and sends an action message to a target object...

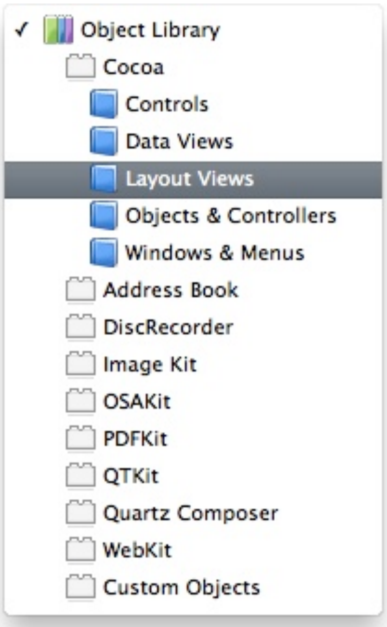
Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object

7. Display the Object library by choosing View > Utilities > Object Library.

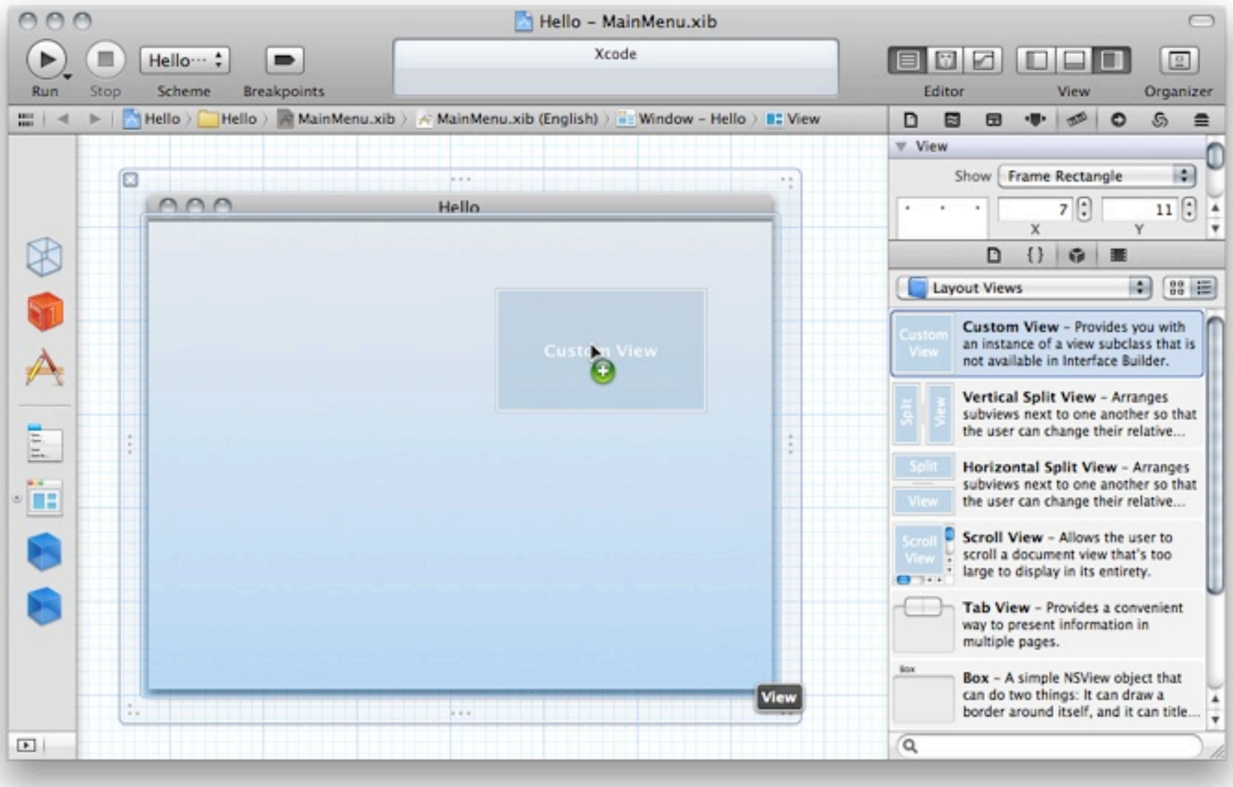
   Make the library pane taller by dragging its selector bar up. (This also makes the inspector pane shorter.)


Hello – MainMenu.xib
Run
Stop
Scheme
Breakpoints
Xcode
Editor
View
Organizer
Hello
MainMenu.xib
MainMenu.xib (English)
Window – Hello
Window
Size 480 360
Width Height
Constraints Minimum Size
Object Library
Push Button – Intercepts mouse-down events and sends an action message to a target object when...
Gradient Button – Intercepts mouse-down events and sends an action message to a target object...
Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object...
Rounded Textured Button – Intercepts mouse-down events and sends an action message to a...
Textured Button – Intercepts mouse-down events and sends an action message to a target object...
Recessed Button – Intercepts mouse-down events and sends an action message to a target object...

8. From the Object Library pop-up menu, choose Cocoa > Layout Views.



9. Drag the Custom View object from the library to the Hello window in the canvas.



Custom View – Provides you with an instance of a view subclass that is not available in Interface Builder.

Vertical Split View – Arranges subviews next to one another so that the user can change their relative…

Horizontal Split View – Arranges subviews next to one another so that the user can change their relative…

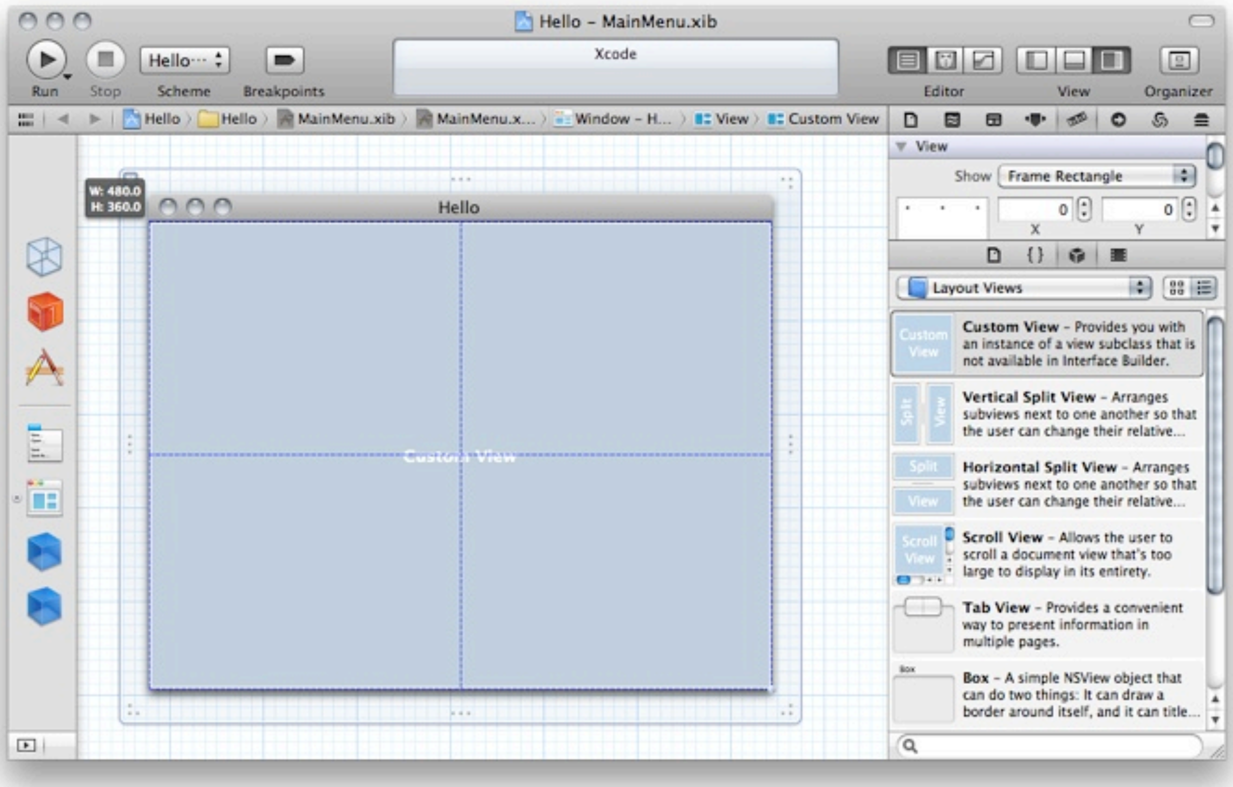
Scroll View – Allows the user to scroll a document view that's too large to display in its entirety.

Tab View – Provides a convenient way to present information in multiple pages.

Box – A simple NSView object that can do two things: It can draw a border around itself, and it can title…

You've created an instance of the `NSView` class and added it to the window.

10. Resize the newly added view by dragging its sides to the Hello window's borders, so that it occupies the entire content area of the Hello window.



Custom View – Provides you with an instance of a view subclass that is not available in Interface Builder.

Vertical Split View – Arranges subviews next to one another so that the user can change their relative…

Horizontal Split View – Arranges subviews next to one another so that the user can change their relative…

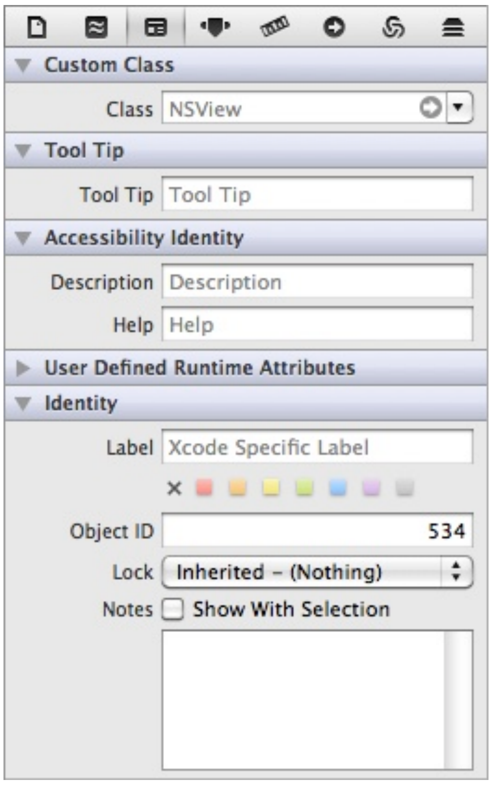
Scroll View – Allows the user to scroll a document view that's too large to display in its entirety.

Tab View – Provides a convenient way to present information in multiple pages.

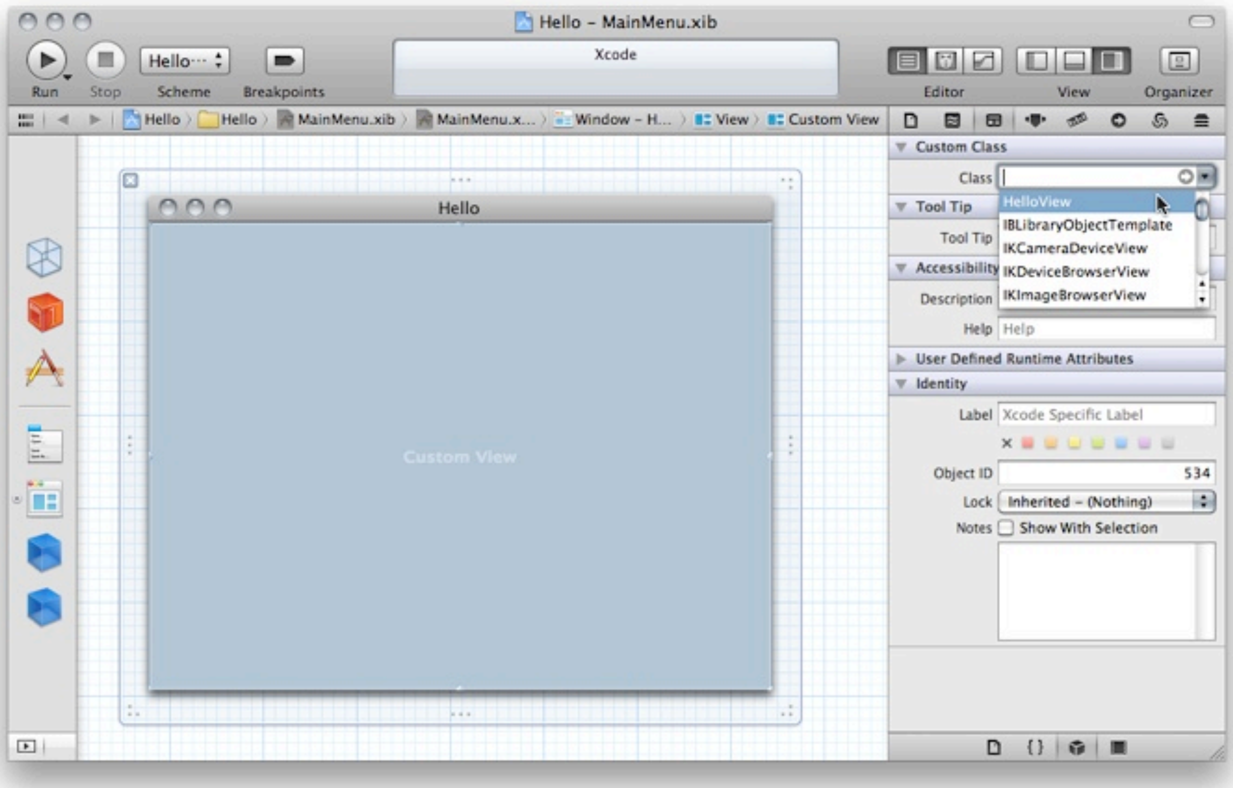
Box – A simple NSView object that can do two things: It can draw a border around itself, and it can title…

11. Choose View > Utilities > Identity Inspector.

    The Identity inspector lets you specify details about user-interface elements that identify them to users of your application (tooltips, also called *help tags*) and to the system (class, runtime attributes, object ID, and so on).

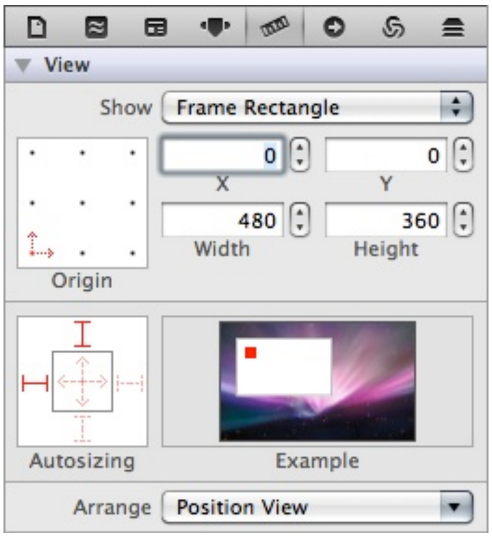


12. In the Identity inspector, choose `HelloView` from the Class pop-up menu in the Custom Class section.



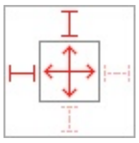
Notice that the label for the view changes from "Custom View" to "HelloView."

13. Choose View > Utilities > Size Inspector.



In the Size inspector you can enter precise values for positioning and sizing controls. The Autosizing area lets you specify how (and whether) controls change size and position as the enclosing window changes size. (You can also change the layout by moving and resizing controls on the Interface Builder canvas.)

14. In the Autosizing area, click the vertical and horizontal dotted lines in the inner square.



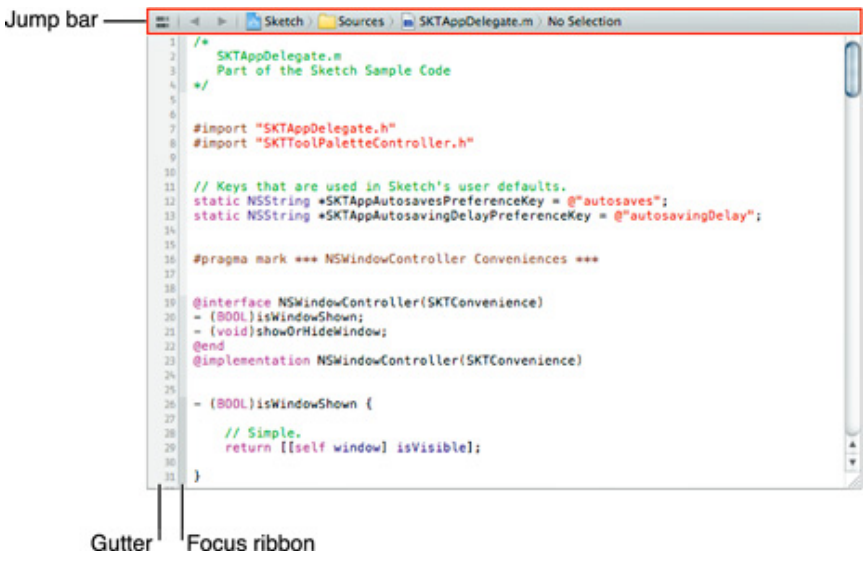
Notice that the dotted lines change to solid ones. Solid lines in the inner square indicate the directions the view resizes automatically. In this case, the view resizes vertically and horizontally when the user changes the window size. The example animation to the right of the Autosizing area provides a simulation of the new sizing behavior.

There is much more to Interface Builder than you've seen here. When you develop more advanced applications, you use the inspectors to set connections to associate the code you write to interact with user-interface objects.

Write the Code

You can view and edit a source file in the workspace window by selecting the file in the project navigator, which opens it in the source editor, shown in Figure 1-5.

Figure 1-5 The source editor



```
/*
    SKTAppDelegate.m
    Part of the Sketch Sample Code
*/


#import "SKTAppDelegate.h"
#import "SKTToolPaletteController.h"


// Keys that are used in Sketch's user defaults.
static NSString *SKTAppAutosavesPreferenceKey = @"autosaves";
static NSString *SKTAppAutosavingDelayPreferenceKey = @"autosavingDelay";


#pragma mark *** NSWindowController Conveniences ***


@interface NSWindowController(SKTConvenience)
- (BOOL)isWindowShown;
- (void)showOrHideWindow;
@end
@implementation NSWindowController(SKTConvenience)


- (BOOL)isWindowShown {

    // Simple.
    return [[self window] isVisible];

}
```

The gutter displays line numbers (when the “Line numbers” option in Text Editing preferences is selected; see *Text Editing Preferences Help*) and the location of breakpoints, errors, and warnings in the file.

The focus ribbon helps you to concentrate your attention on your code by:

- Identifying the scope of a block of code
- Allowing you to hide blocks of code

The Jump bar allows you to:

- View related files
- Move backward and forward through the set of project files you’ve viewed
- Jump to another location within the current file or to another file in the project

To enter the source code for the Hello application:

1. Choose View > Navigators > Project.
2. Choose View > Utilities > Hide Utilities.
3. In the project navigator, select `HelloView.m` to open it in the source editor. Listing 1-1 shows the initial implementation of the `HelloView` class.

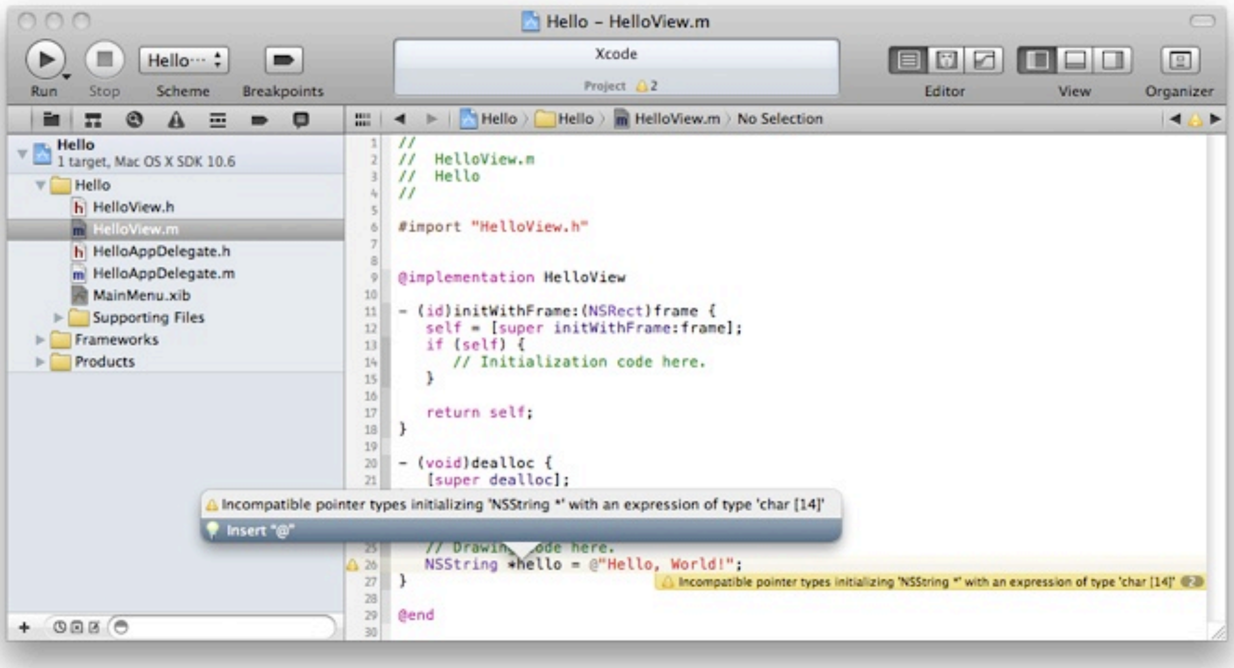
   Listing 1-1 Initial implementation of the `HelloView` class

   ```
   #import "HelloView.h"
   @implementation HelloView

   - (id)initWithFrame:(NSRect)frame {
      if ((self = [super initWithFrame:frame])) {
      // Initialization code here.
      }
      return self;
   }

   - (void)dealloc {
      // Clean-up code here.
      [super dealloc];
   }

   - (void)drawRect:(NSRect)dirtyRect {
      // Drawing code here.
   }
   @end
   ```

4. Insert this code line in the body of the `drawRect:` method:

   ```
   NSString *hello = "Hello, World!";
   ```

   Notice that the gutter shows a warning icon. This means that Xcode has found a problem in the code you just typed. To get information about the problem, click the warning icon. Xcode describes the problem and offers a solution.



```
//
//  HelloView.m
//  Hello
//

#import "HelloView.h"


@implementation HelloView

- (id)initWithFrame:(NSRect)frame {
   self = [super initWithFrame:frame];
   if (self) {
      // Initialization code here.
   }

   return self;
}

- (void)dealloc {
   [super dealloc];
```

Incompatible pointer types initializing 'NSString *' with an expression of type 'char [14]'

Insert "@"

```
   // Drawin  ode here.
   NSString *hello = @"Hello, World!";
}

@end
```

Incompatible pointer types initializing 'NSString *' with an expression of type 'char [14]'

Double-click “Insert "@"” (or press Return) to convert the C string into an Objective-C string object. You’ve just taken advantage of **Live Issues** (in-line issue detection and diagnosis) and **Fix-it** (automatic issue correction).

The fixed code line should look like this:

```
NSString *hello = @"Hello, World!";
```

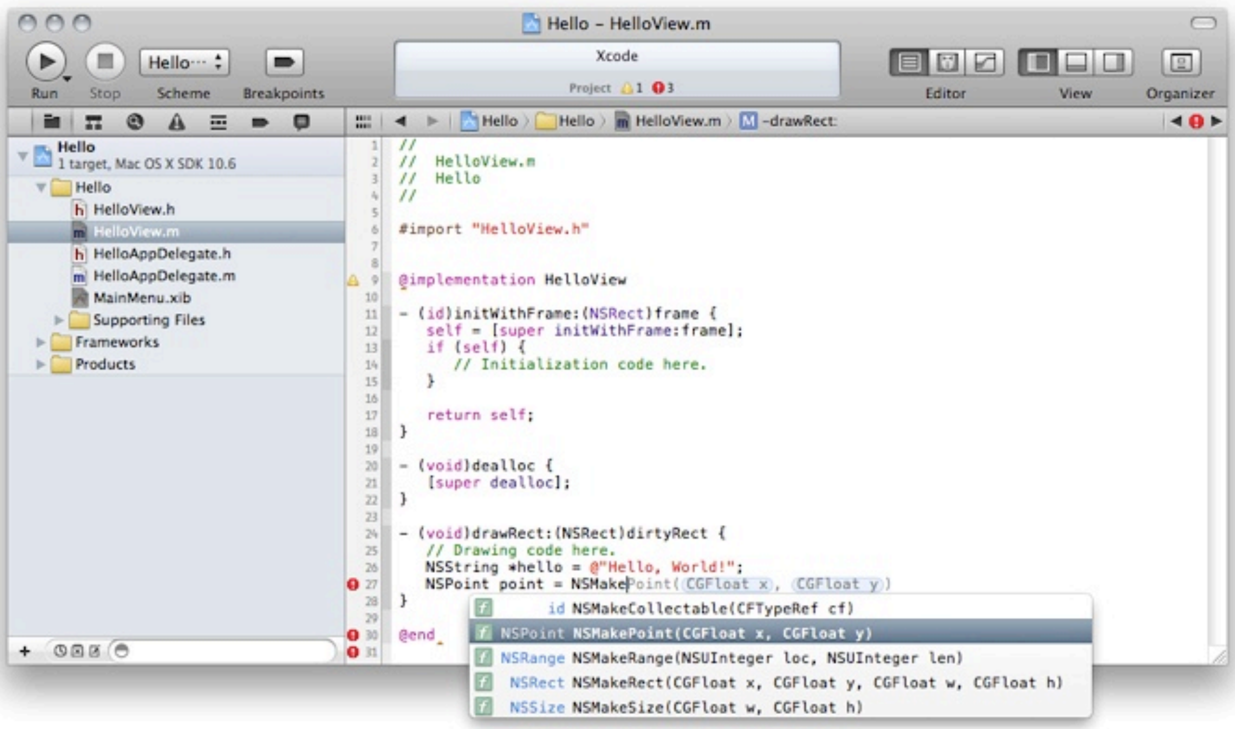
This code line creates the string that the Hello application draws into the view.

Fix-it detects another problem: The `hello` variable is unused in the `drawRect:` method. That’s why there’s still a warning icon in the gutter. You’ll fix that problem shortly.

5. Type this text below the code line you added in the previous step:

```
NSPoint point = NSMake
```

As you type the name of a symbol, Xcode suggests completions to what you’re typing.



```
//
//  HelloView.m
//  Hello
//

#import "HelloView.h"


@implementation HelloView

- (id)initWithFrame:(NSRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.
    }

    return self;
}

- (void)dealloc {
    [super dealloc];
}

- (void)drawRect:(NSRect)dirtyRect {
    // Drawing code here.
    NSString *hello = @"Hello, World!";
    NSPoint point = NSMakePoint(CGFloat x, CGFloat y)
}

@end
```

```
      id NSMakeCollectable(CFTypeRef cf)
 NSPoint NSMakePoint(CGFloat x, CGFloat y)
 NSRange NSMakeRange(NSUInteger loc, NSUInteger len)
  NSRect NSMakeRect(CGFloat x, CGFloat y, CGFloat w, CGFloat h)
  NSSize NSMakeSize(CGFloat w, CGFloat h)
```

This is code completion. You specify whether Xcode provides completions as you type in Text Editing preferences.

Because Xcode sees that you're assigning the function's return value to a variable of type `NSPoint`, Xcode selects the `NSMakePoint` completion in the completion list. Press Return to choose that completion.

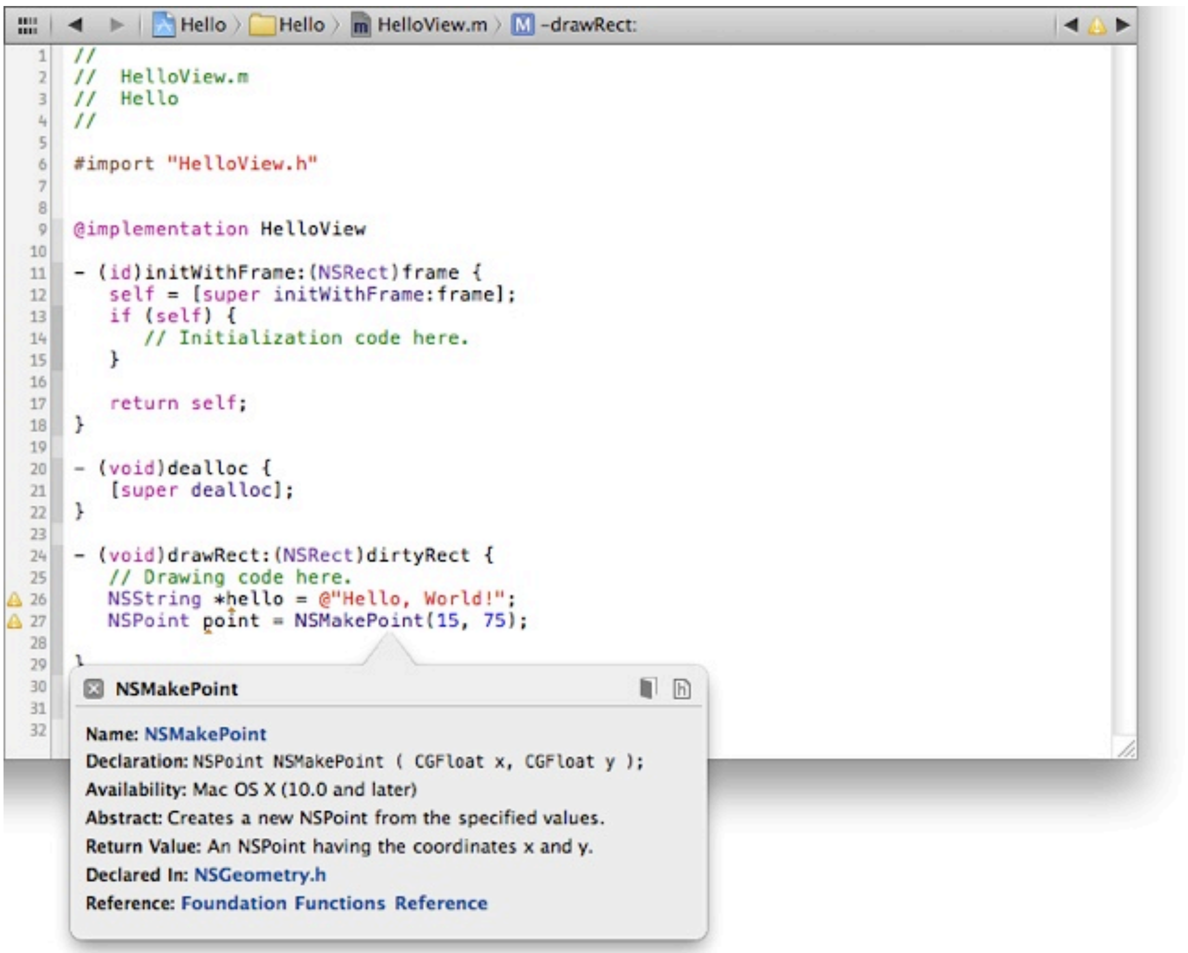
Xcode highlights the first parameter in the completion.

6. Type `15`, press Tab, and type `75`. Add a semicolon to the end of the line. The code line should look like this:

```
NSPoint point = NSMakePoint(15, 75);
```

This call creates a point at the specified coordinates. This point is the origin for drawing the message.

7. Place the cursor on the `NSMakePoint` function name, and choose Help > Quick Help for Selected Item.



```
//
//  HelloView.m
//  Hello
//

#import "HelloView.h"


@implementation HelloView

- (id)initWithFrame:(NSRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.
    }

    return self;
}

- (void)dealloc {
    [super dealloc];
}

- (void)drawRect:(NSRect)dirtyRect {
    // Drawing code here.
    NSString *hello = @"Hello, World!";
    NSPoint point = NSMakePoint(15, 75);
}
```

NSMakePoint

Name: NSMakePoint
Declaration: NSPoint NSMakePoint (CGFloat x, CGFloat y);
Availability: Mac OS X (10.0 and later)
Abstract: Creates a new NSPoint from the specified values.
Return Value: An NSPoint having the coordinates x and y.
Declared In: NSGeometry.h
Reference: Foundation Functions Reference

Quick Help provides a summary of the API reference for the selected symbol. From the Quick Help window you can access the rest of the developer library to get in-depth information about the symbol.

> **Tip:** You can also hold down Option while moving the pointer over your code. When you put the pointer over a symbol, Xcode underlines the symbol. Click the symbol while holding down Option to display the Quick Help window.

8. Complete the implementation of the `drawRect:` method so that it looks like Listing 1-2.

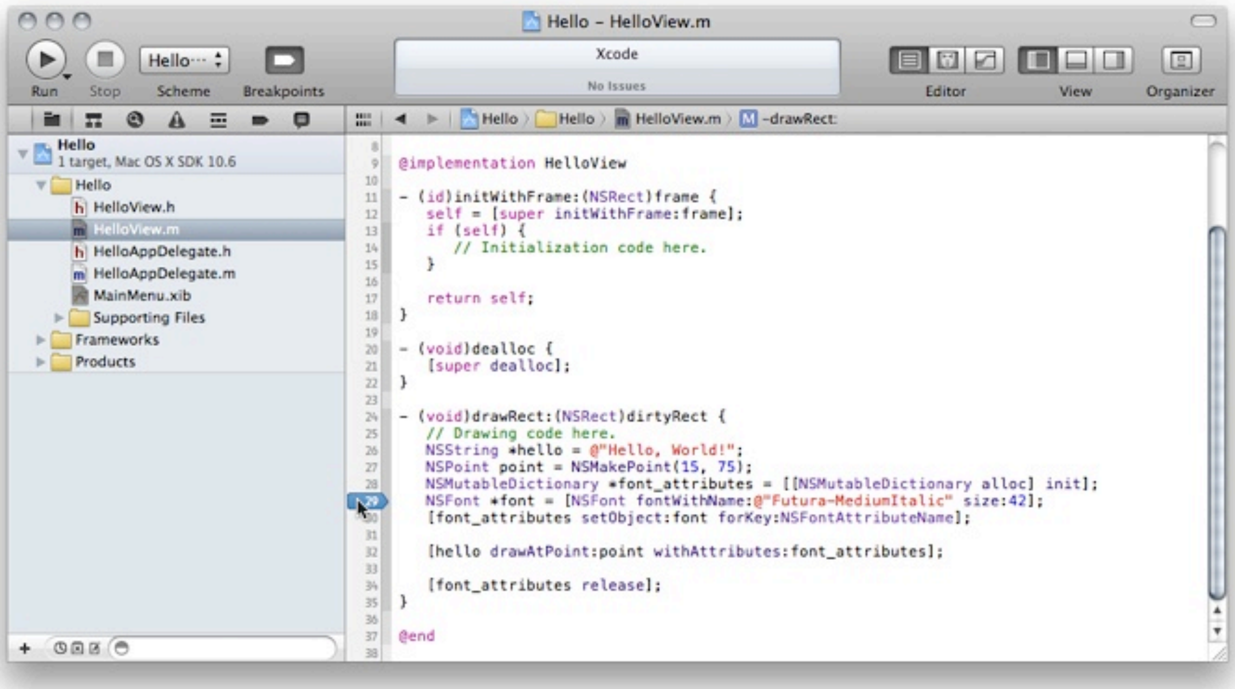
   Listing 1-2 Implementation of the `drawRect:` method

```
- (void)drawRect:(NSRect)dirtyRect {
   NSString *hello = @"Hello, World!";
   NSPoint point = NSMakePoint(15, 75);
   NSMutableDictionary *font_attributes = [[NSMutableDictionary alloc] init];
   NSFont *font = [NSFont fontWithName:@"Futura-MediumItalic" size:42];
   [font_attributes setObject:font forKey:NSFontAttributeName];

   [hello drawAtPoint:point withAttributes:font_attributes];

   [font_attributes release];
}
```

   After typing the code, correct transcribing errors found by Fix-it (the code provided has no problems).

9. Add a breakpoint to the `drawRect:` method.

   Add the breakpoint by clicking the gutter to the left of the code line with the assignment to the `font` variable. Although the `drawRect:` method has no problems, adding a breakpoint to it allows you to familiarize yourself with the Xcode debugging facilities when you run the Hello application.



```
@implementation HelloView

- (id)initWithFrame:(NSRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.
    }

    return self;
}

- (void)dealloc {
    [super dealloc];
}

- (void)drawRect:(NSRect)dirtyRect {
    // Drawing code here.
    NSString *hello = @"Hello, World!";
    NSPoint point = NSMakePoint(15, 75);
    NSMutableDictionary *font_attributes = [[NSMutableDictionary alloc] init];
    NSFont *font = [NSFont fontWithName:@"Futura-MediumItalic" size:42];
    [font_attributes setObject:font forKey:NSFontAttributeName];

    [hello drawAtPoint:point withAttributes:font_attributes];

    [font_attributes release];
}

@end
```

Notice that adding the breakpoint automatically activates breakpoints for your project—the Breakpoints toolbar button has a pushed-in appearance.

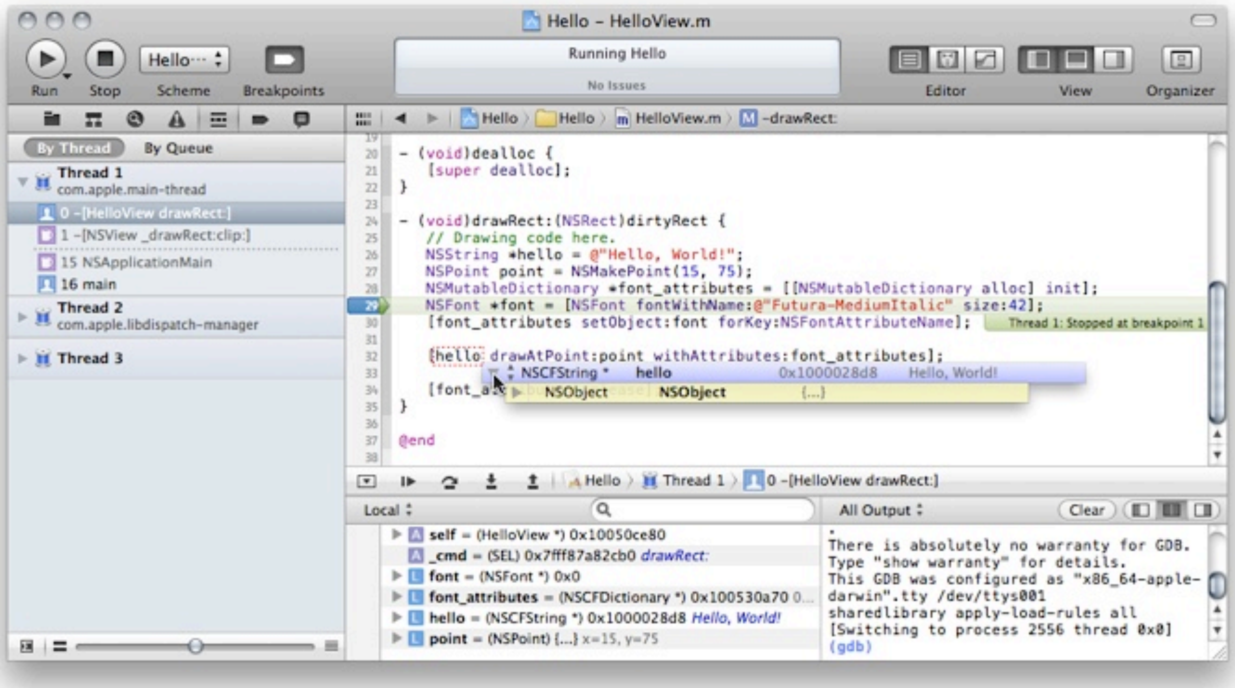
Run the Application

1. Choose Product > Run to run the Hello application.

   The activity viewer (the LCD-like display in the workspace window toolbar) displays information about the tasks Xcode performs in response, which are to build the Hello application and to launch it in an interactive debug session.

   Figure 1-6 shows a debugging session using the debug navigator (on the left), the source editor (on the right), and the debug area (below the source editor) to get information about the running code.

   Figure 1-6 Hello application stopped at a breakpoint



```
- (void)dealloc {
  [super dealloc];
}

- (void)drawRect:(NSRect)dirtyRect {
  // Drawing code here.
  NSString *hello = @"Hello, World!";
  NSPoint point = NSMakePoint(15, 75);
  NSMutableDictionary *font_attributes = [[NSMutableDictionary alloc] init];
  NSFont *font = [NSFont fontWithName:@"Futura-MediumItalic" size:42];
  [font_attributes setObject:font forKey:NSFontAttributeName];

  [hello drawAtPoint:point withAttributes:font_attributes];

  [font_a
}

@end
```

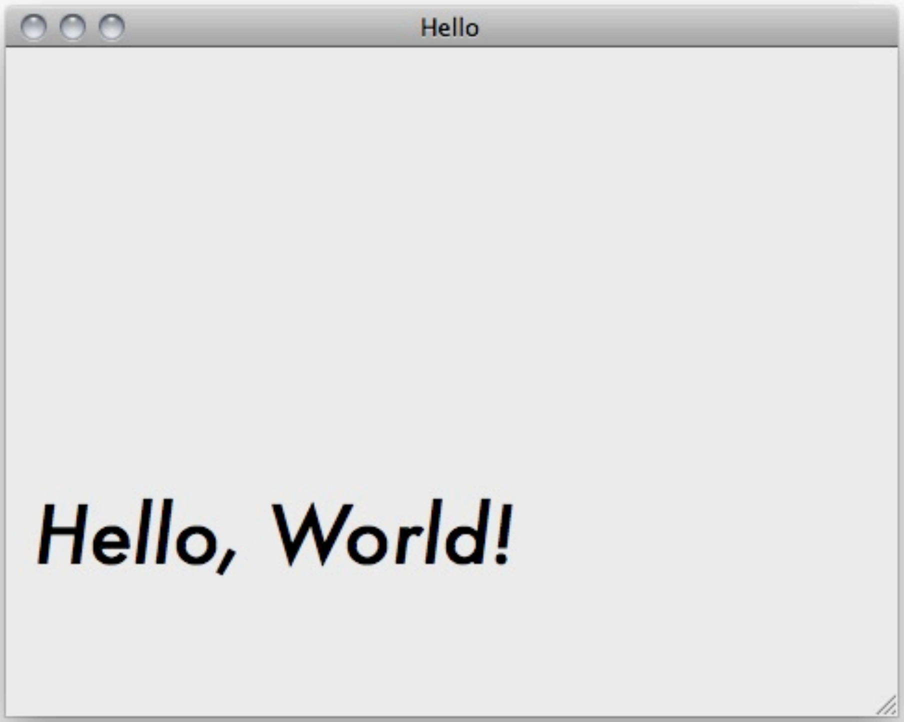
```
self = (HelloView *) 0x10050ce80
_cmd = (SEL) 0x7fff87a82cb0 drawRect:
font = (NSFont *) 0x0
font_attributes = (NSCFDictionary *) 0x100530a70 0...
hello = (NSCFString *) 0x1000028d8 Hello, World!
point = (NSPoint) {...} x=15, y=75
```

```
.
There is absolutely no warranty for GDB.
Type "show warranty" for details.
This GDB was configured as "x86_64-apple-
darwin".tty /dev/ttys001
sharedlibrary apply-load-rules all
[Switching to process 2556 thread 0x0]
(gdb)
```

Notice that the source editor can display the values of variables within a scope. When you place the pointer over the `hello` variable, the source editor displays information about the variable in a datatip. The debug area contains the variables pane and the console pane. The variables pane shows information about the variables of the `drawRect:` method. The console pane shows your program's console output. You can also enter commands directly to the debugger in the console pane.

2. Choose Product > Debug > Continue to continue execution of the Hello application.

   The window of the Hello application appears with the "Hello, World!" message displayed in its bottom-left corner.



3. Choose Hello > Quit Hello or click the Stop toolbar button in the workspace window to stop the Hello application.

View Task and Session Logs

The log viewer (the Xcode session and task log–viewing facility) lets you examine details about tasks Xcode has performed, such as building and running your programs. When things don't go as smoothly as they should, you can use this facility to locate the cause of problems. But even if there are no problems, you can view a log of the activities Xcode performed in response to your execution of the Run command.

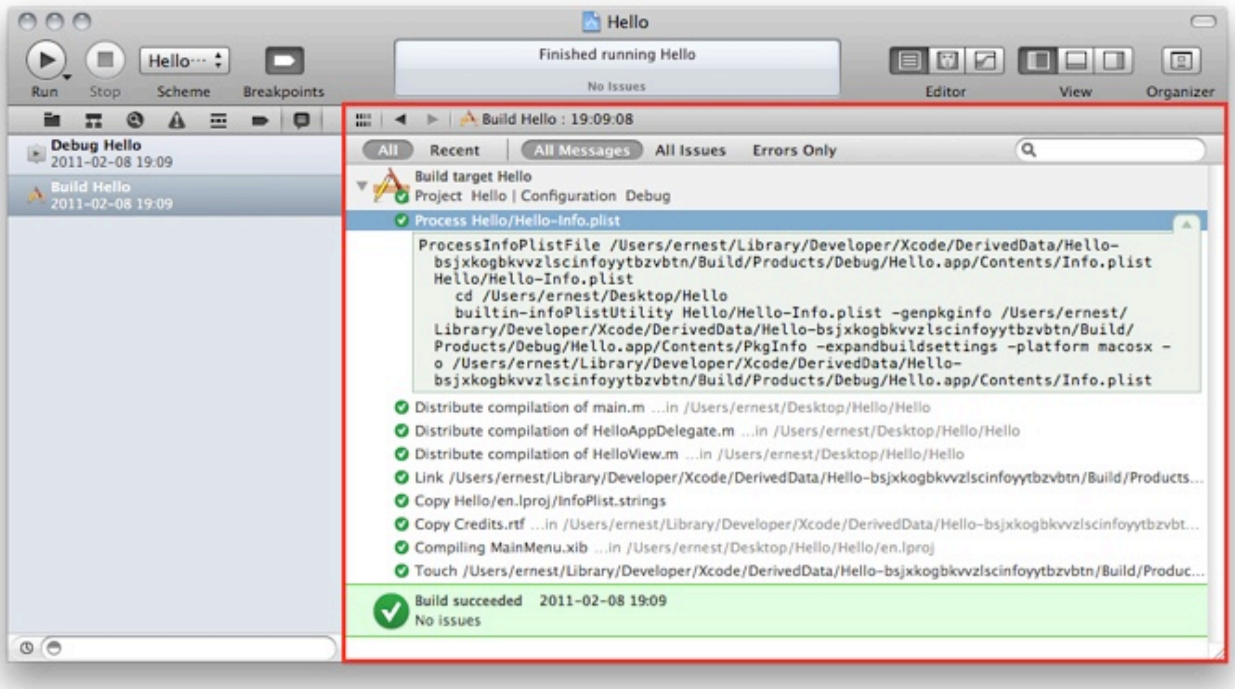
To view details about these activities:

1. Choose View > Navigators > Log.
2. In the log navigator, select the Build Hello task. Then click All and All Messages in the log viewer.

   The log viewer shows the operations it performed while executing the build task on the Hello target, whose product is the Hello application.

   Selecting an operation in the log viewer (Figure 1–7) reveals the transcript button on the right side of the operation. Click the transcript button to display details about the operation.

Figure 1–7 The log viewer in the workspace window



```
Build target Hello
Project  Hello | Configuration  Debug
Process Hello/Hello-Info.plist
ProcessInfoPlistFile /Users/ernest/Library/Developer/Xcode/DerivedData/Hello-
  bsjxkogbkvvzlscinfoyytbzvbtn/Build/Products/Debug/Hello.app/Contents/Info.plist
  Hello/Hello-Info.plist
     cd /Users/ernest/Desktop/Hello
     builtin-infoPlistUtility Hello/Hello-Info.plist -genpkginfo /Users/ernest/
  Library/Developer/Xcode/DerivedData/Hello-bsjxkogbkvvzlscinfoyytbzvbtn/Build/
  Products/Debug/Hello.app/Contents/PkgInfo -expandbuildsettings -platform macosx -
  o /Users/ernest/Library/Developer/Xcode/DerivedData/Hello-
  bsjxkogbkvvzlscinfoyytbzvbtn/Build/Products/Debug/Hello.app/Contents/Info.plist
Distribute compilation of main.m ...in /Users/ernest/Desktop/Hello/Hello
Distribute compilation of HelloAppDelegate.m ...in /Users/ernest/Desktop/Hello/Hello
Distribute compilation of HelloView.m ...in /Users/ernest/Desktop/Hello/Hello
Link /Users/ernest/Library/Developer/Xcode/DerivedData/Hello-bsjxkogbkvvzlscinfoyytbzvbtn/Build/Products...
Copy Hello/en.lproj/InfoPlist.strings
Copy Credits.rtf ...in /Users/ernest/Library/Developer/Xcode/DerivedData/Hello-bsjxkogbkvvzlscinfoyytbzvbt...
Compiling MainMenu.xib ...in /Users/ernest/Desktop/Hello/Hello/en.lproj
Touch /Users/ernest/Library/Developer/Xcode/DerivedData/Hello-bsjxkogbkvvzlscinfoyytbzvbtn/Build/Produc...
Build succeeded    2011-02-08 19:09
No issues
```