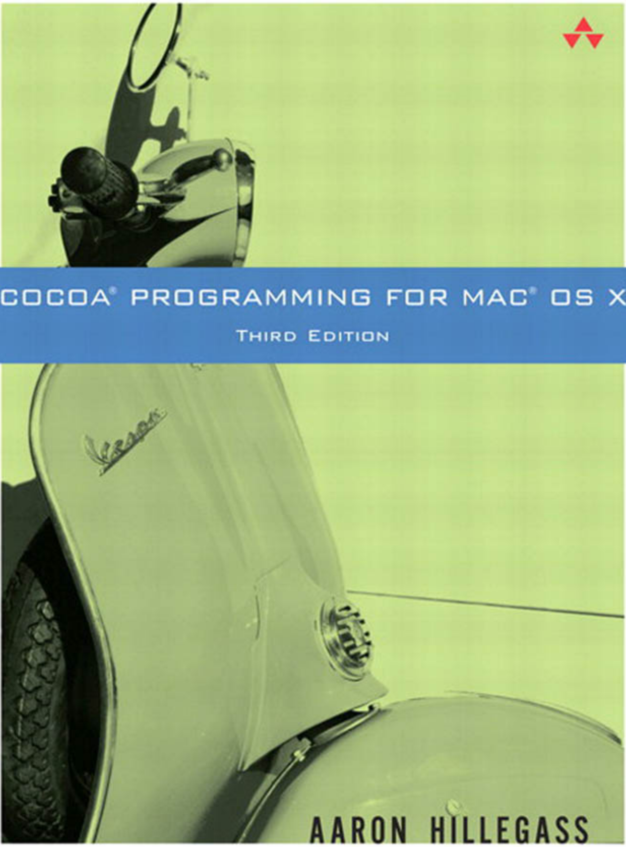


COCOA® PROGRAMMING FOR MAC® OS X

THIRD EDITION

AARON HILLEGASS

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Hillegass, Aaron.
Cocoa programming for Mac OS X / Aaron Hillegass. — 3rd ed.
p. cm.
Includes index.
ISBN 978-0-321-50361-9 (pbk. : alk. paper)
1. Cocoa (Application development environment) 2. Mac OS. 3. Operating systems (Computers) 4. Macintosh (Computer)—Programming. I. Title.

QA76.76.O63H57145 2008
005.26'8—dc22

2008008239

Copyright © 2008 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0321-50361-9
ISBN-10: 0-321-50361-9
Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
Third printing, July 2008

PREFACE

If you are developing applications for the Mac, or are hoping to do so, this book is just the resource you need. Does it cover everything you will ever want to know about programming for the Mac? Of course it doesn't. But it does cover probably 80% of what you need to know. You can find the remaining 20%, the 20% that is unique to you, in Apple's online documentation.

This book, then, acts as a foundation. It covers the Objective-C language and the major design patterns of Cocoa. It will also get you started with the three most commonly used developer tools: Xcode, Interface Builder, and Instruments. After reading this book, you will be able to understand and utilize Apple's online documentation.

There is a lot of code in this book. Through that code, I will introduce you to the idioms of the Cocoa community. My hope is that by presenting exemplary code, I can help you to become not just a Cocoa developer, but a stylish Cocoa developer.

This third edition includes technologies introduced in Mac OS X 10.4 and 10.5. These include Xcode 3, Objective-C 2, Core Data, the garbage collector, and CoreAnimation.

This book is written for programmers who already know some C programming and something about objects. You are not expected to have any experience with Mac programming. It's a hands-on book and assumes that you have access to Mac OS X and the developer tools. The developer tools are free. If you bought a shrink-wrapped copy of Mac OS X, the installer for the developer tools was on the DVD. The tools can also be downloaded from the Apple Developer Connection Web site (http://developer.apple.com/).

I have tried to make this book as useful for you as possible, if not indispensable. That said, I'd love to hear from you if you have any suggestions for improving it.

Aaron Hillegass
book-comments@bignerdranch.com

Chapter 17
CUSTOM VIEWS

All visible objects in an application are either windows or views. In this chapter, you will create a subclass of **NSView**. From time to time, you may need to create a custom view to do custom drawing or event handling. Even if you do not plan to do custom drawing or event handling, you will learn a lot about how Cocoa works by learning how to create a new view class.

Windows are instances of the class **NSWindow**. Each window has a collection of views, each of which is responsible for a rectangle of the window. The view draws inside that rectangle and handles mouse events that occur there. A view may also handle keyboard events. You have worked with several subclasses of **NSView** already: **NSButton**, **NSTextField**, **NSTableView**, and **NSColorWell** are all views. (Note that a window is not a subclass of **NSView**.)

The View Hierarchy

Views are arranged in a hierarchy (Figure 17.1). The window has a content view that completely fills its interior. The content view usually has several subviews, each of which may have subviews of its own. Every view knows its superview, its subviews, and the window it lives on.

Here are the relevant methods from **NSView**:

```
- (NSView *)superview;
- (NSArray *)subviews;
- (NSWindow *)window;
```

Any view can have subviews, but most don't. The following five views commonly have subviews:

1. The content view of a window.
2. **NSBox**. The contents of a box are its subviews.

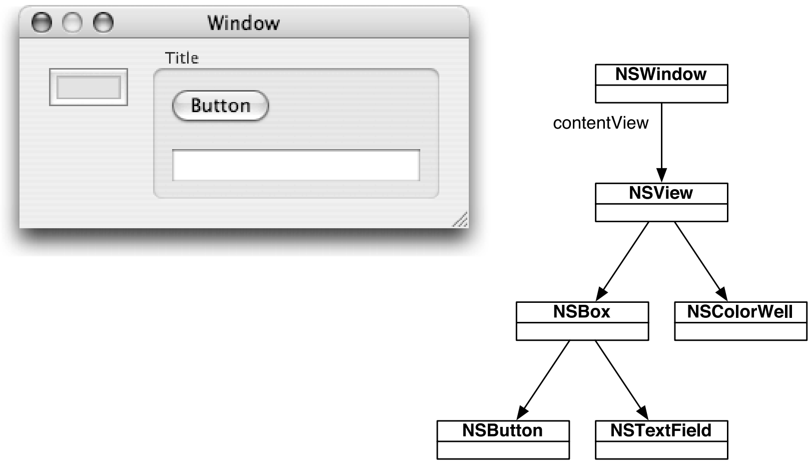
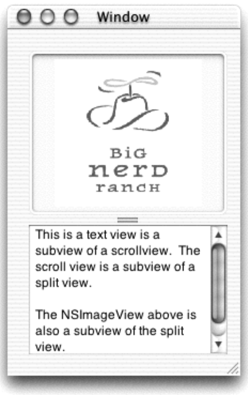


Figure 17.1 Views Hierarchy

3. **`NSScrollView`**. A view that appears in a scroll view is a subview of the scroll view. The scroll bars are also subviews of the scroll view.
4. **`NSSplitView`**. Each view in a split view is a subview (Figure 17.2).



This is a text view is a subview of a scrollview. The scroll view is a subview of a split view.

The NSImageView above is also a subview of the split view.

Figure 17.2 A Scroll View in a Split View

5. **`NSTabView`**. As the user chooses different tabs, different subviews are swapped in and out (Figure 17.3).

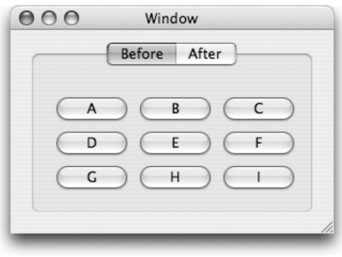


Figure 17.3 A Tab View

Getting a View to Draw Itself

In this section, you will create a very simple view that will appear and paint itself green. It will look like Figure 17.4.

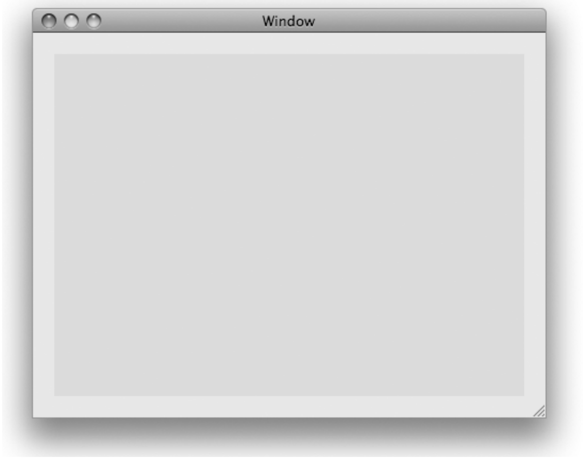


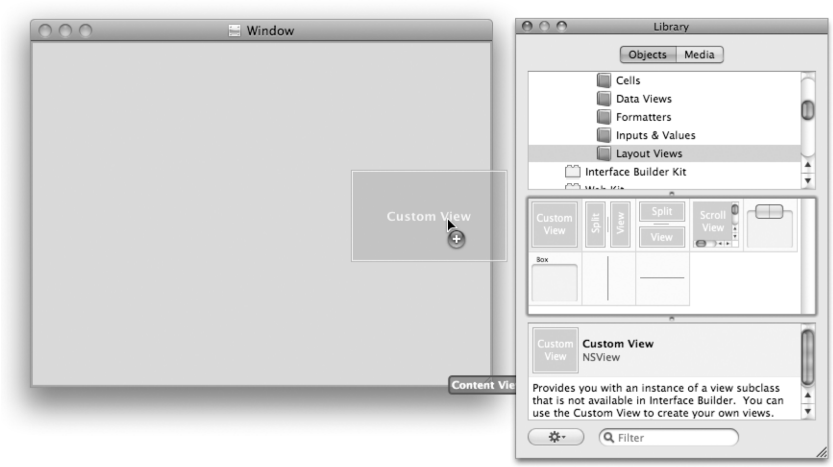
Figure 17.4 Completed Application

Create a new project of type Cocoa Application. Name the project ImageFun.

Using the File->New File menu item, create an Objective-C NSView subclass, and name it **`StretchView`**.

Create an Instance of a View Subclass

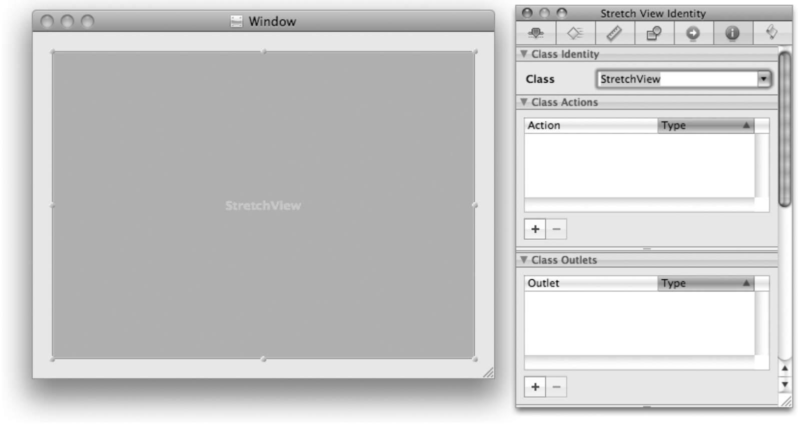
Open MainMenu.nib. Create an instance of your class by dragging out a CustomView placeholder from the Library (under Views & Cells -> Layout View) and dropping it on the window (Figure 17.5).



Custom View
NSView
Provides you with an instance of a view subclass that is not available in Interface Builder. You can use the Custom View to create your own views.

Figure 17.5 Drop a View on the Window

Resize the view to fill most of the window. Open the Info panel, and set the class of the view to be **`StretchView`** (Figure 17.6).

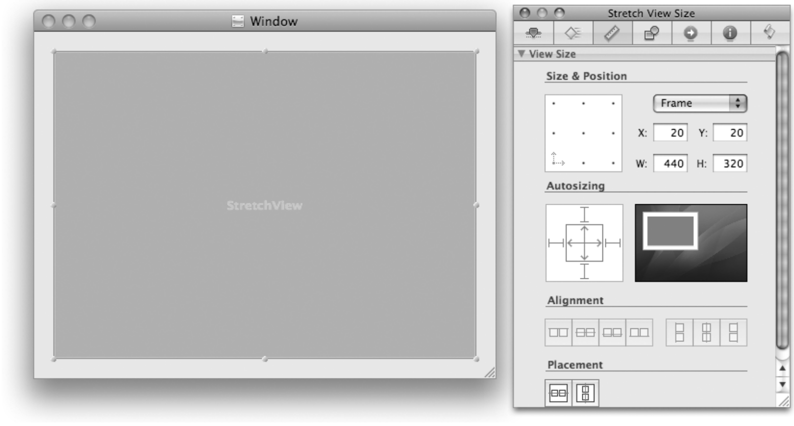


Class: StretchView

Figure 17.6 Set the Class of the View to StretchView

Size Inspector

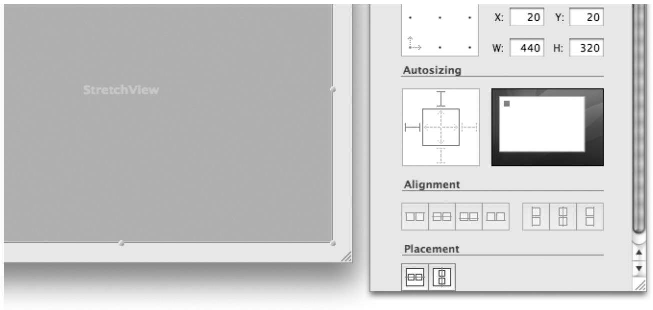
Your **`StretchView`** object is a subview of the window's content view. This point raises an interesting question: What happens to the view when the superview resizes? A page in the Info panel allows you to specify that behavior. Open the Size Info panel, and set it as shown in Figure 17.7. Now it will grow and shrink as necessary to keep the distance from its edges to the edges of its superview constant.



X: 20 Y: 20
W: 440 H: 320

Figure 17.7 Make the View Resize with the Window

If you wanted the view to stay the same height, you could let the distance between the bottom of the view and the bottom of the superview grow and shrink. You could also let the distance between the right edge of the view and the right edge of the window grow and shrink. In this exercise, you do not want this behavior. But if you did want the view to stick to the upper-left corner of the window, the Inspector would look like Figure 17.8.



X: 20 Y: 20
W: 440 H: 320

Figure 17.8 Not This!

Figure 17.9 is a complete diagram of what the Size Inspector means.

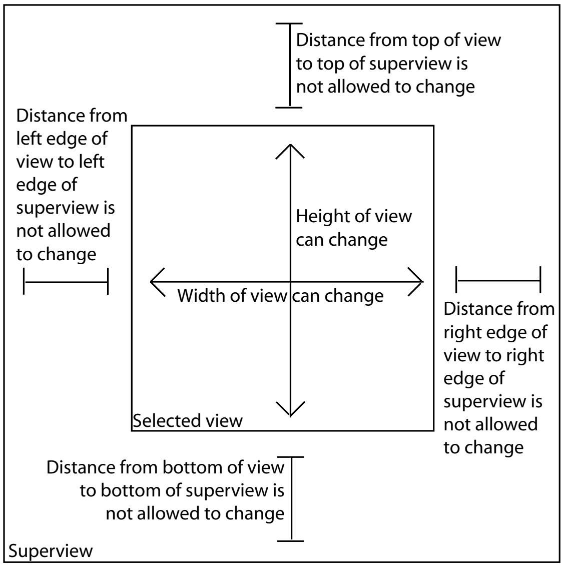

Distance from top of view to top of superview is not allowed to change
Distance from left edge of view to left edge of superview is not allowed to change
Height of view can change
Width of view can change
Distance from right edge of view to right edge of superview is not allowed to change
Selected view
Distance from bottom of view to bottom of superview is not allowed to change
Superview

Figure 17.9 What the Red Lines in the Size Inspector Mean

Save and close the nib file.

drawRect:

When a view needs to draw itself, it is sent the message **`drawRect:`** with the rectangle that needs to be drawn or redrawn. The method is called automatically—you never need to call it directly. Instead, if you know that a view needs redrawing, you send the view the **`setNeedsDisplay:`** message:

```
[myView setNeedsDisplay:YES];
```

This message informs `myView` that it is "dirty." After the event has been handled, the view will be redrawn.

Before calling **`drawRect:`**, the system *locks focus* on the view. Each view has its own graphics context, which includes the view's coordinate system, its current color, its current font, and the clipping rectangle. When the focus is locked on a view, the view's graphics context is active. When the focus is unlocked, the graphics context is no longer active. Whenever you issue drawing commands, they will be executed in the current graphics context.

You can use **`NSBezierPath`** to draw lines, circles, curves, and rectangles. You can use **`NSImage`** to create composite images on the view. In this example, you will fill the entire view with a green rectangle.

Open `StretchView.m` and add the following code to the **`drawRect:`** method:

```
- (void)drawRect:(NSRect)rect
{
    NSRect bounds = [self bounds];
    [[NSColor greenColor] set];
    [NSBezierPath fillRect:bounds];
}
```

As shown in Figure 17.10, `NSRect` is a struct with two members: `origin`, which is an `NSPoint`, and `size`, which is an `NSSize`.

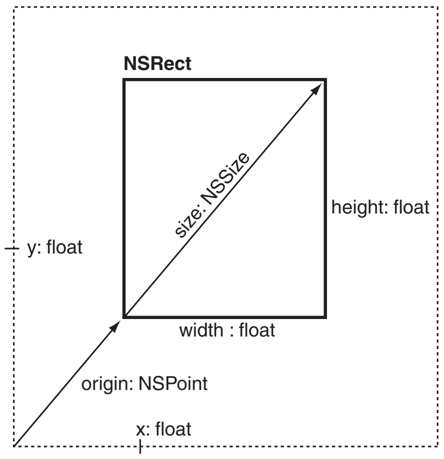


Figure 17.10 NSRect, NSSize, and NSPoint

`NSSize` is a struct with two members: `width` and `height` (both `floats`).

`NSPoint` is a struct with two members: `x` and `y` (both `floats`).

For performance reasons, structs are used in a few places instead of Objective-C classes. For completeness, here is the list of all the Cocoa structs that you are likely to use: `NSSize`, `NSPoint`, `NSRect`, `NSRange`, `NSDecimal`, and `NSAffineTransformStruct`. `NSRange` is used to define subranges. `NSDecimal` describes numbers with very specific precision and rounding behavior. `NSAffineTransformStruct` describes linear transformations of graphics.

Note that your view knows its location as an `NSRect` called `bounds`. In this method, you fetched the `bounds` rectangle, set the current color to green, and filled the entire `bounds` rectangle with the current color.

The `NSRect` that is passed as an argument to the view is the region that is "dirty" and needs redrawing. It may be less than the entire view. If you are doing very time-consuming drawing, redrawing only the dirty rectangle may speed up your application considerably.

Note that **`setNeedsDisplay:`** will trigger the entire visible region of the view to be redrawn. If you wanted to be more precise about which part of the view needs redrawing, you would use **`setNeedsDisplayInRect:`** instead:

```
NSRect dirtyRect;
dirtyRect = NSMakeRect(0, 0, 50, 50);
[myView setNeedsDisplayInRect:dirtyRect];
```

Build and run your app. Try resizing the window.

Drawing with NSBezierPath

If you want to draw lines, ovals, curves, or polygons, you can use **`NSBezierPath`**. In this chapter, you have already used the **`NSBezierPath`**'s **`fillRect:`** class method to color your view. In this section, you will use **`NSBezierPath`** to draw lines connecting random points (Figure 17.11).

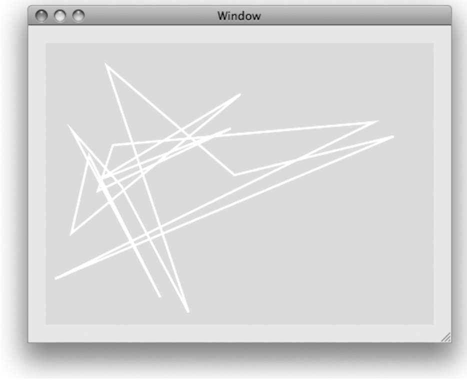


Figure 17.11 Completed Application

The first thing you will need is an instance variable to hold the instance of **NSBezierPath**. You will also create an instance method that returns a random point in the view. Open StretchView.h and make it look like this:

```
#import <Cocoa/Cocoa.h>

@interface StretchView : NSView
{
    NSBezierPath *path;
}
- (NSPoint)randomPoint;

@end
```

In StretchView.m, you will override **initWithFrame:**. As the designated initializer for **NSView**, **initWithFrame:** will be called automatically when an instance of your view is created. In your version of **initWithFrame:**, you will create the path object and fill it with lines to random points. Make StretchView.m look like this:

```
#import "StretchView.h"

@implementation StretchView

- (id)initWithFrame:(NSRect)rect
{
    if (![super initWithFrame:rect])
        return nil;

    // Seed the random number generator
    srandom(time(NULL));

    // Create a path object
    path = [[NSBezierPath alloc] init];
    [path setLineWidth:3.0];
    NSPoint p = [self randomPoint];
    [path moveToPoint:p];
    int i;
    for (i = 0; i < 15; i++) {
        p = [self randomPoint];
        [path lineToPoint:p];
    }
    [path closePath];
    return self;
}
- (void)dealloc
{
    [path release];
    [super dealloc];
}
```

```
// randomPoint returns a random point inside the view
- (NSPoint)randomPoint
{
    NSPoint result;
    NSRect r = [self bounds];
    result.x = r.origin.x + random() % (int)r.size.width;
    result.y = r.origin.y + random() % (int)r.size.height;
    return result;
}

- (void)drawRect:(NSRect)rect
{
    NSRect bounds = [self bounds];

    // Fill the view with green
    [[NSColor greenColor] set];
    [NSBezierPath fillRect: bounds];

    // Draw the path in white
    [[NSColor whiteColor] set];
    [path stroke];
}

@end
```

Build and run your app. Pretty, eh?

Okay, now try replacing `[path stroke]` with `[path fill]`. Build and run it.

NSScrollView

In the art world, a larger work is typically more expensive than a smaller one of equal quality. Your beautiful view is lovely, but it would be more valuable if it were larger. How can it be larger yet still fit inside that tiny window? You are going to put it in a scroll view (Figure 17.12).

A scroll view has three parts: the document view, the content view, and the scroll bars. In this example, your view will become the document view and will be displayed in the content view, which is an instance of **`NSClipView`**.

Although this change looks tricky, it is very simple to make. In fact, it requires no code at all. Open `MainMenu.nib` in Interface Builder. Select the view, and choose Embed Objects in Scroll View from the Layout menu (Figure 17.13).

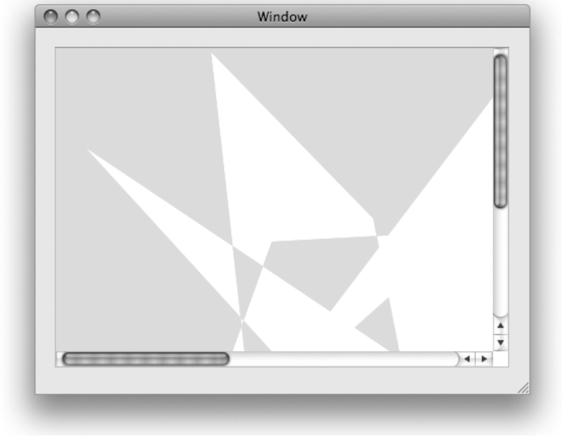


Figure 17.12 Completed Application

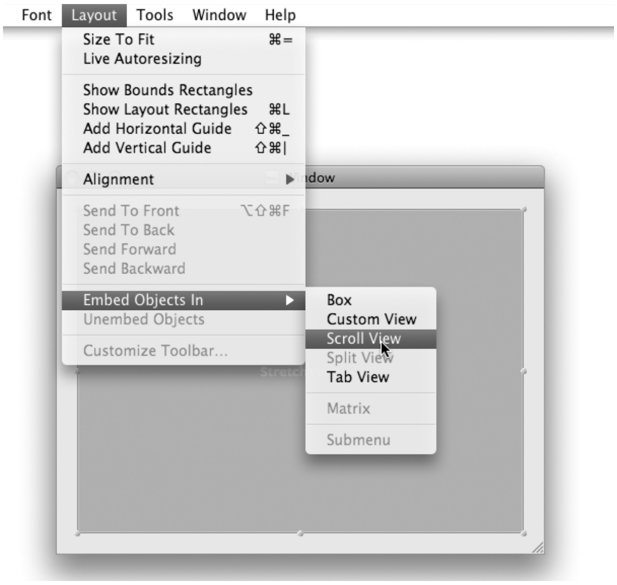
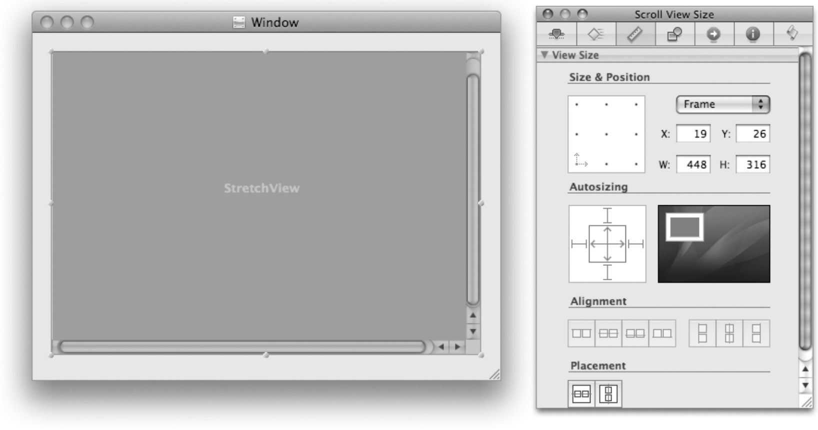


Figure 17.13 Embed the StretchView in a Scroll View

As the window resizes, you want the scroll view to resize, but you do not want your document to resize. Open the Size Inspector, select Scroll View, and set the Size Inspector so that it resizes with the window (Figure 17.14).

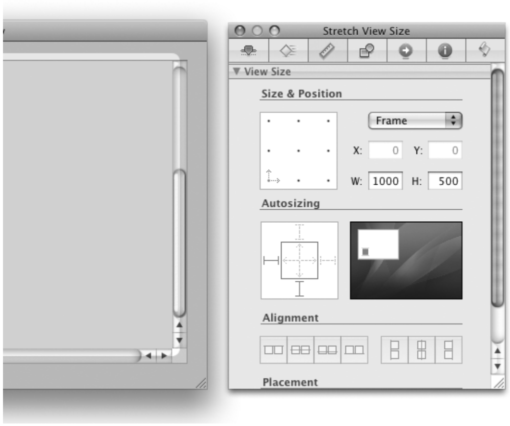


X: 19 Y: 26
W: 448 H: 316

Figure 17.14 Make Scroll View Resize with Window

Note the width and height of the view.

To select the document view, double-click inside the scroll view. You should see the title of the inspector change to Stretch View Size. Make the view about twice as wide and twice as tall as the scroll view. Set the Size Inspector so that the view will stick to the lower-left corner of its superview and not resize (Figure 17.15). Build the application and run it.



X: 0 Y: 0
W: 1000 H: 500

Figure 17.15 Make StretchView Larger and Nonresizing

Creating Views Programmatically

You will instantiate most of your views in Interface Builder. Every once in a while, you will need to create views programmatically. For example, assume that you have a pointer to a window and want to put a button on it. This code would create a button and put it on the window's content view:

```
NSView *superview = [window contentView];
NSRect frame = NSMakeRect(10, 10, 200, 100);
NSButton *button = [[NSButton alloc] initWithFrame:frame];
[button setTitle:@"Click me!"];
[superview addSubview:button];
[button release];
```

For the More Curious: Cells

`NSControl` inherits from **`NSView`**. With its graphics context, **`NSView`** is a relatively large and expensive object to create. When the **`NSButton`** class was created, the first thing someone did was to create a calculator with 10 rows and 10 columns of buttons. The performance was less than it could have been because of the 100 tiny views. Later, someone had the clever idea of moving the brains of the button into another object (not a view) and creating one big view (called an **`NSMatrix`**) that would act as the view for all 100 button brains. The class for the button brains was called **`NSButtonCell`** (Figure 17.16).

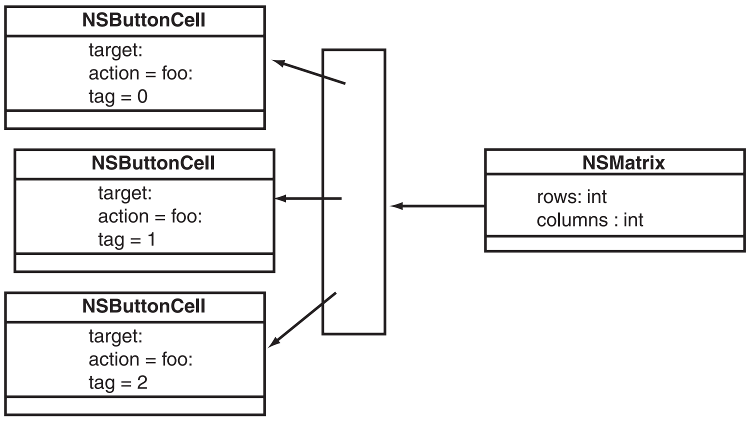


Figure 17.16 NSMatrix

In the end, **`NSButton`** became simply a view that had an **`NSButtonCell`**. The button cell does everything, and **`NSButton`** simply claims a space in the window (Figure 17.17).

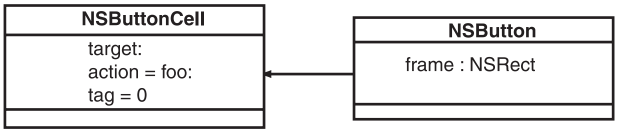
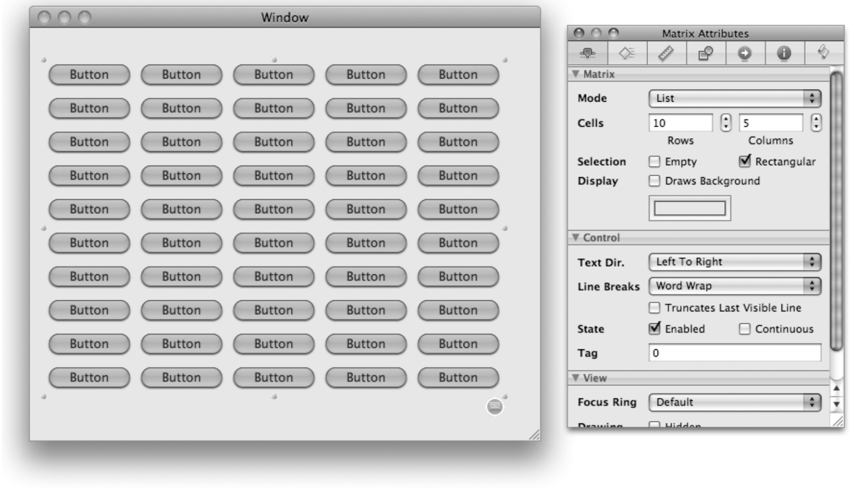


Figure 17.17 NSButton and NSButtonCell

Similarly, **`NSSlider`** is a view with an **`NSSliderCell`**, and **`NSTextField`** is a view with an **`NSTextFieldCell`**. **`NSColorWell`**, by contrast, has no cell.

To create an instance of **`NSMatrix`** in Interface Builder, you drop a control with a cell onto the window, choose Embed Objects In -> Matrix, and then Option-drag as if resizing until the matrix has the correct number of rows and columns (Figure 17.18).



Matrix Attributes

▼ Matrix

Mode: List

Cells: 10 Rows, 5 Columns

Selection: ☐ Empty ☑ Rectangular

Display: ☐ Draws Background

▼ Control

Text Dir.: Left To Right

Line Breaks: Word Wrap

☐ Truncates Last Visible Line

State: ☑ Enabled ☐ Continuous

Tag: 0

▼ View

Focus Ring: Default

Drawing: ☐ Hidden

Figure 17.18 A Matrix of Buttons

An **`NSMatrix`** has a target and an action. A cell may also have a target and an action. If the cell is activated, the cell's target and action are used. If the target and action of the selected cell are not set, the matrix's target and action will be used.

When dealing with matrices, you will often ask which cell was activated. Cells can also be given a tag:

```
- (IBAction)myAction:(id)sender {
    id theCell = [sender selectedCell];
    int theTag = [theCell tag];
    ...
}
```

The cell's tag can be set in Interface Builder.

Cells are used in several other types of objects. The data in an **NSTableView**, for example, is drawn by cells.

For the More Curious: isFlipped

Both PDF and PostScript use the standard Cartesian coordinate system, whereby *y* increases as you move up the page. Quartz follows this model by default. The origin is usually at the lower-left corner of the view.

For some types of drawing, the math becomes easier if the upper-left corner is the origin and *y* increases as you move down the page. We say that such a view is *flipped*.

To flip a view, you override **isFlipped** in your view class to return YES:

```
- (BOOL)isFlipped
{
    return YES;
}
```

While we are discussing the coordinate system, note that *x*- and *y*-coordinates are measured in *points*. A point is typically defined as 72.0 points = 1 inch. In reality, by default 1.0 point = 1 pixel on your screen. You can, however, change the size of a point by changing the coordinate system:

```
// Make everything in the view twice as large
NSSize newScale;
newScale.width = 2.0;
newScale.height = 2.0;
[myView scaleUnitSquareToSize:newScale];
[myView setNeedsDisplay:YES];
```

Challenge

`NSBezierPath` can also draw Bezier curves. Replace the straight lines with randomly curved ones. (*Hint:* Look in the documentation for **`NSBezierPath`**.)

INDEX

Symbols

%@, 38, 41–42, 71–72, 231
$ (in tokens), 231
+ in method names, 52
@ symbol, 24, 38
.???? extension, 163
+ prefix, 203
: (in method name), 34

A

Abstract classes, 123, 154
Accented characters, 228, 284
acceptsFirstResponder, 265, 272–273
Accessor methods, 48, 74–77, 113, 117, 119, 142–143
Action, 22, 80
  See also Target/action
Action methods, 19, 136
Active Build Configuration, 39
add:, 125, 130, 178, 370–372
addEmployeesObject:, 369
addObject:, 34, 44, 63
addObjectsFromArray:, 44
addObserver:, 121, 146, 210–214, 414
Adobe PDF, 2, 262, 283–285, 294, 339
Advanced Mac OS X Programming, 417
Affine transform, 239, 379–380
Alert panels
  confirm delete request, 218–219
  icon, 217
  modal operation, 218
  NSRunAlertPanel(), 217–218
  removeEmployee:, 218–220
  sheet, 218
allKeys, 199, 333
alloc, 33–34, 72
Amazon Web service, 346–349
AND (&), 250
Animation. *See* Core animation
AppController, 85–89, 185–186, 252–256, 412
appendData:, 414
AppKit framework, 6, 79, 107, 283
Apple, Inc.
  Cocoa developers mailing list, 418
  Copland, 3
  Developer Connection, 418
  Developer Technical Support, 418
  GUI guidelines, 16
  history, 1
  Mac OS X, 3
  and NeXT, 3
applicationShouldOpenUntitledFile:, 205
archivedDataWithRootObject:, 161
Archived (objects), 14
Archiving/unarchiving
  decoding, 156–157
  encoding, 154–155
  extension and icon, 163–165
  infinite loops, 165–166
  Info.plist, 157, 163–165, 168
  initWithCoder, 154–156
  NSCoder, 154–157, 166
  NSCoding, 154–157, 167
  NSDocument, 157–160, 168
  NSDocumentController, 157–158
  NSKeyedArchiver, 154, 161, 166
  NSKeyedUnarchiver, 161–163, 166
  NSWindowController, 157–160, 162
  universal type identifiers (UTIs), 168–169
  updateChangeCount:, 168
Arguments and methods, 34, 56, 201

Array controller, 120, 128, 149
Array Controller Attributes, 129, 174, 370
Array Controller Bindings, 129, 174
Arrays, 37
aSelector, 90, 108–109, 210
Assertion checking, 61–62
assign, 119
Atomic setter methods, 119
attributedStringForObjectValue:, 337
Attributed strings, 278–281, 337–338
Attributes, 172–173
Attributes (property), 118–119
Attributes dictionary, 281
Attributes Inspector, 16–17, 174
Autocompletion, 336–337
Autoreleased objects, 52, 72–74
Autorelease Old Value, 76
autorelease pool, 72–73
autoscroll:, 260–261, 316
availableTypeFromArray:, 289
@avg, 120
awakeFromNib, 28–30, 352
AWS ID, 349

B

Background color, 204–206, 213–215, 270
becomeFirstResponder, 265, 272
beginSheetForDirectory:file:types: modalForWindow:modalDelegate: didEndSelector:contextInfo:, 255–256, 283
beginSheetModalForWindow:modalDelegate: didEndSelector:contextInfo:, 220
Berners-Lee, Tim, 3
Bézier path, 239–241, 248
BigLetterView
  adding bold and italic, 286
  drag-and-drop, 300–307
  typing-tutor application, 309
BigLetterView, 266–270, 275, 281–283
  cut:, **copy:**, and **paste:**, 289–290
Big Nerd Ranch, Inc., 3, 6, 417
Binding(s), 117, 120–121, 129–131, 135
  to cells, scroll view, or table view, 128, 178
  Core Data, 177–180
  DepartmentView.nib, 371
  EmployeeView.nib, 372
  key-value coding (KVC), 113–115
  NSArrayController, 135
  to **NSUserDefaultsController**, 207
Bindings Inspector, 114–115, 128, 173
bind:toObject:withKeyPath:options:, 120
Blast button, 389–390
blastem:, 389–390, 393
BMP, 262
BNR prefix, 202
BNR_ prefix, 296–297
bold and italic, 286
BOOL, 23
boolForKey:, 201
bounds, 239–242, 270–272, 283
Box Bindings, 179
Box Size, 362–363
Breakpoints, 58–60
BSD Unix, 1
Bundle, 10, 194–195
Button Attributes, 82, 132, 312, 412
Button Bindings, 131
Button Inspector, 82
Buttons, 15–17, 80–83, 98–99, 101, 116–118

C

CAAnimation, 390–395
CALayer, 387, 390–395
calendarDate, 52
Calendar format string tokens, 54
CAOpenGLLayer, 395
Capitalization, 7–8, 10, 19, 92, 133–134
carArrayController, 181–182
CarLot project, 172
C array, 378, 380
caseInsensitiveCompare:, 133
Categories, 295–297
CATextLayer, 394
Cells, 245–247
CFMakeCollectable(), 377, 379–380
CGColorRef, 378–380, 383
change:, 121, 146, 148
changeBackgroundColor:, 190, 204–205, 212–213
changeKeyPath:, 148
changeNewEmptyDoc:, 188, 191, 204–205
changeParameter:, 398, 400, 402–403
changeViewController:, 356–357, 361–362
char, 23
Characters, 228, 284
characters method, 265

Check Before Change, 76
checkbox, 191
@class, 24, 185, 186
Classes, 2, 5–7
  creating objective-C, 46–58
  existing, 35–46
  online reference, 7
  reusable, 132
  sending messages to, 34
Class methods, 52
clickCount, 250–251
CocoaDev Wiki, 418
Cocoa/OpenGL application
  using **NSOpenGLView**, 397–398
  writing the application, 398–404
ColorFormatter
  ColorFormatter.h, 329
  methods, 332–334
  nib file, 329–331
  NSColorList, 331–332
  search strings for substrings, 332
Color well, 184, 190–193, 205
Color Well Bindings, 330
Command-line tools, 207–208, 405–415
compare: method, 133–134
Compiler, 4, 33, 58
Compositing an image, 256–258
Composition, compared with inheritance, 45–46
Conditional encoding, 166
Conditionally Sets Editable, 179
Console (log), 26–28
containsObject:, 43
Continuous sliders, 83, 114, 253, 399
control:, 334–335
Controller classes, 123–124, 129–130
Coordinate systems, 247, 258–260
copy, 119
copy:, 289–290
Copy and paste operations, 287
Core animation
  CALayer, 387–390
  CALayer and CAAnimation, 390–395
  object diagram, 387
  polynomials, 388–391
Core Data
  bindings, 177–180
  fetch, 173–174, 180–181, 370
  framework, 6
  interface, 173–179
  Level Indicator, 175–176
  NSArrayController, 171, 178, 181
  NSManagedObjectContext, 171–173, 180–181
  NSManagedObjectModel, 171–173
  NSPersistentDocument, 172, 180–181, 356
  Number Formatter Attributes, 175
  object graph, 181
  objects, 180–181
  Views & Cells, 175–177
Core Data relationships
  bindings, 371–372
  custom **NSManagedObject** classes, 366–369
  data model, 365–366
  DepartmentView.nib, 369–371
  EmployeeView.nib, 371–372
  events and nextResponder, 372–373
  interface layout, 369–372
  inverse relationships, 366
Core Foundation (CF) data structures, 377
CoreGraphics framework, 2
count, 25, 43, 199
@count, 120
Counterparts, 26–27
countOfSongs, 143
Cox, Brad, 33
C Primitive, 376–377
C programming languages, 4
createEmployee:, 136–137, 149–151
createNewPolynomial:, 380–381, 389, 391
currentEvent:, 316
currentRect, 259–260
Custom views
  cells, 245–247
  creating views programmatically, 245
  drawRect:, 238–240
  ImageFun project, 235–240
  isFlipped, 247
  NSBezierPath, 239–242, 248
  NSScrollView, 242–244
  NSView, 233–235, 241, 245
  Size Inspector, 237–238
  view hierarchy, 233–235
  view subclass, 236
cut:, 289–290

D

Dalrymple, Mark, 417
Darwin, 1

dataForType:, 289
dataOfType:, 158, 162, 410
dataReady:, 414
dataSource, 99–104, 136, 407–408
dataWithPDFInsideRect:, 283
dateByAddingYears:, 53
Date formatter, 84, 327
dateWithYear:, 52
dayOfCommonEra, 53
dayOfMonth, 53
dayOfWeek, 53
dayOfYear, 53
dealloc, 71–72, 80
Deallocation, 69–72
Debugger, 4, 41, 51, 58–62, 92
declareTypes:, 288–289, 293
decodeBoolForKey:, 156
decodeDoubleForKey:, 156
decodeFloatForKey:, 156
decodeIntForKey:, 156
decodeObjectForKey:, 156
Decoding, 156–157
defaultCenter method, 210–214
Default settings, 200–201
  See also Preferences panel; User defaults
defaults (Terminal tool), 207–208
#define, 202
delegate, 96–99, 105–106, 205
Delegate methods, 297–298
Delegates, 96–99, 107–109, 215
Delete button, 132, 220
deleteRandomPolynomial:, 380–381, 389, 392
deltaX, 250
deltaY, 250
deltaZ, 250
Department entity, 366–369
DepartmentView.nib, 369–371
description method, 41–42, 50–55, 59, 84
Designated initializers, 57
Development tools, 3–4
Dictionary
  attributes, 281, 337
  change:, 121, 146, 148
  key-value pairs, 168, 199
  NSDictionary, 198–199, 409
  NSMutableDictionary, 198–199
  userInfo, 214, 324
  UTIs, 168–169
displayViewController:, 361–363, 372
.doc files, 280
Document architecture, 157–160
Document-based application, 125
Document controller, 157–158
Document object, 125
Document types, 165
Drag-and-drop
  drag destination, 303–307
  dragging-destination methods, 305–306
  draggingSourceOperationMaskForLocal:, 300–302, 307
  draggingUpdated, 307
  drag source, 300–303
  highlighting, 304–305
  operation mask, 307
  and pasteboards, 287
  registerForDraggedTypes, 303–304
  user feedback, 299
drawAtPoint:, 280
Drawing
  with **NSBezierPath**, 240–242
  with OpenGL, 397–404
Drawing text with attributes
  adding bold and italic, 286
  BigLetterView, 281–283
  generating PDF data, 283–285
  NSAttributedString, 278–281
  NSFont, 277–278
  NSFontManager, 286
  NSMakeRange(), 278
  NSMutableAttributedString, 278
  NSShadow, 286
  NSString, 281
drawInRect:, 279–280
drawInRect:fromRect:operation:fraction:, 257
drawInRect:withAttributes:, 281
drawRect:, 238–239, 257, 260, 271–272, 282–283, 304, 341, 389, 398, 403
Dual-mode code, 67, 376

E

Editor view, 12–13
Embed objects in box, 176–177, 311
Embed objects in matrix, 246, 399
Embed objects in scroll view, 242–243

Employee entity, 366–368
EmployeeView.nib, 371–373
encodeBool:forKey:, 155
encodeConditionalObject:forKey, 166
encodeDouble:forKey, 155
encodeFloat:forKey, 155
encodeInt:forKey, 155
encodeObject:forKey, 155
encodeWithCoder:forKey, 154–155, 161, 166
Encoding, 154–155
@end, 24, 29
endSpeedSheet:returnCode:, 324
Entity, 171–174, 180, 365–367
entryDate, 47–51, 61, 71–73
Enumerations, 199
Errors, 161
Event-handling methods, 249, 259
Event object, 249–250
Event queue, 17, 31
Events and nextResponder, 372–373
Exceptions, 57–61, 92
exec(), 405
Extensions, 6, 26, 38, 163–165

F

Fetch (managed objects), 173–174, 180–181, 370
File encoding and localization, 228–229
File's Owner, 17, 189–193
Filesystem defaults, 200–201
File wrapper, 158–159
fileWrapperOfType:, 159
finalize message, 376
FirstLetter category, 295–297
firstResponder, 263–265, 270–272, 291–292
flagsChanged:, 265
Flipping a view, 247
float, 132–133
floatForKey:, 201
floatValue:, 82
Focus, locked and unlocked, 238, 301–302
Focus ring, 275–276
Font methods, 277–278, 286
fontWithName:, 277
fork(), 405
for loop, 37, 50
Formatters, 84, 128–130, 175, 327–334
forwardInvocation:, 139–141
Foundation framework, 6, 127
Foundation Tool, 35–37
Frame load delegate, 353
.framework extension, 6
Frameworks, 2, 6–7, 388
Free Software Foundation, 33, 58, 62
Fuzzy blue box, 275–276

G

Garbage collection
  dual-mode code, 67, 376
  finalize message, 376
  Instruments, 383–385
  malloc(), 375–376
  non-object data types, 376–377
  NSAllocateCollectable(), 376–377, 379–380, 385
  polynomials, 377–383
  reachable objects, 375
  strong references, 376
  weak references, 376, 385
Garbage collector
  assign and retain, 119
  dual-mode code, 67
  and Mac OS versions, 66, 68
  memory management, 66–68
gcc (GNU C compiler), 4, 33, 58
gdb console, 59–60
gdb (GNU debugger), 4, 58
generalPasteboard:, 288, 290
genstrings, 229
getObjectValue:forString:errorDescription:, 328, 334
get prefix, 76
Getter method, 76, 113, 118
GIF, 262
glEndList(), 404
Gliss, 398–404
glLoadIdentity(), 403
Global variables, 202, 206, 212, 217, 279, 288
GLUT.framework, 398

H

.h file, 26–27, 149
handleColorChange:, 214
Header file, 17, 24–25

Helper objects
  connecting, 103–104
  dataSource, 99–104
  and delegates, 96–99, 107–109
hidesOnDeactivate, 184
Highlighting, drag-and-drop, 304–305
hourOfDay, 53
HTML, 280
HTTP, 345

I

IBAction, 24, 26
IBM, 79
IBOutlet, 24
ibtool, 230–231
Icon alert panel, 217
Icons, 163–165
id, 23
Identity Inspector, 19, 86, 182
image, 256
Image file formats, 262
ImageFun, 235–240, 316
Image opacity, 251–254, 257
Image representations, 261
Image View Bindings, 179
Image Well, 175
Immutability, 43, 52
@implementation, 24, 29
Implementation file, 17–19, 24–26
#import, 19, 23–24, 186, 202
#include, 24
incrementFido:, 116–117
Indentation, in Xcode, 24–25
indexOfObject:, 43
Infinite loops, 166
Info.plist, 157, 163–165, 168
Informal protocols, 297
Inheritance, 23, 41, 45–46, 81, 265
Inheritance diagrams, 41, 81
init, 34, 41, 55, 57–58, 80
initialFirstResponder, 269
Initializers, 55–58
initWithCoder:, 154–156, 403
initWithData:encoding:, 410, 414
initWithEntryDate:, 57, 61
initWithFormat:, 45
initWithFrame:, 241, 257, 270, 282, 342
initWithFrame:pixelFormat:, 397
initWithPeople:, 342
inLetterView, 313–314
insertObject:atIndex:, 44
Inspector, 16–17, 21, 128
  See also Bindings Inspector; Identity Inspector; Size Inspector; TextField Inspector
Instances, 2, 5, 33–35
Instance variables, 5, 19
Instruments, 383–385
integerForKey:, 201
@interface, 19, 24
Interface Builder, 4, 8, 13–21, 28, 87–88
interpretKeyEvents:, 273
Invisible objects, 17
Invocations, 139–142
iPing, 411–415
isa pointer, 62–63
isARepeat, 265
isDrawingToScreen, 344
isEqual: method, 42–43
isFlipped, 247
isOpaque, 272
isPartialStringValid:newEditingString:errorDescription:, 335–336
Italicized text, 286
Iterators, 199

J

Java code, 23
Java interface, 154
Jobs, Steve, 1
JPG, 262
Justification (text), 16–17

K

Keyboard events
  BigLetterView, 266–270, 275
  custom view project, 266–274
  firstResponder, 263–265, 270–272
  fuzzy blue box, 275–276
  interpretKeyEvents:, 273
  loop of key views, 267–269
  NSEvent, 265–266
  NSResponder, 265

rollovers, 274–275
TypingTutor project, 266–274
keyCode, 265
keyDown:, 80, 265, 272–273, 373
keyEnumerator, 199
Key paths, 120–121
keyUp:, 265
Key-value coding (KVC)
bindings, 113–115
key paths, 120–121
and nil, 132–133
observers, 115–118
for ordered relationships, 143
project example, 111–113
properties and attributes, 118–119
for unordered relationships, 144
Key-value pairs, 121, 168, 198–199, 226
Key views, 267–269
Key window, 152, 263, 291
keyWindow, 183
knowsPageRange:, 340, 342

L

lastObject, 43
Late Night Cocoa podcast, 418
laterDate:, 54
Lazy copying, 293–294
length, 45, 91, 135, 278, 332, 336
Loading files to apps, 161–163
Localizable.strings, 226–227
Localization
accented characters, 226, 228
explicit ordering of tokens, 231
genstrings, 229
ibtool, 230–231
Keyboard Viewer, 226
languages, 223
Localizable.strings, 226–227
nib file, 224–226
NSBundle, 223–224, 226
NSLocalizedString, 229–231
string tables, 226–230
Unicode UTF-8, 228–229
Xcode Add Localization, 224
location, 278, 296, 332–333, 337, 343
locationInWindow, 250
lockFocus, 238, 301–302
Locks, 119
Logging arrival of notifications, 214
LotteryEntry class, 46–49
lottery.m, 49–50

M

Mac OS X
Developer Tools, 3–4
speech synthesizer, 85–86
window server appearance, 2
main(), 13, 39, 59
Main event loop, 31–32
MainMenu.nib, 13–14, 86, 185–186, 206, 341, 410
mainWindow, 183
malloc(), 375–376
Managed Object Class, 367
managedObjectContext, 171–174, 180–181, 357, 360, 366, 371
ManagingViewController, 357–358, 360–361
Matrix Attributes, 246, 400
Matrix Size, 401
@max, 120
Memory leak, 70, 72, 380, 383, 405
Memory management
accessor methods, 74–77
autoreleased objects, 72–74
dealloc, 71–72
garbage collector, 66–68
retain counts, 68–77
temporary objects, 73–74
messageFontOfSize:, 278
Message forwarding, 139–140
Messages, 4–5, 34, 40, 62–63
Methods
accessor, 48, 74–77, 113, 117, 119, 142–143
action, 19, 136
and arguments, 34
atomic setter, 119
class methods, 52
ColorFormatter, 332–334
defined, 5
event-handling, 249, 259
formatter, 328, 332–334
get prefix, 76
NSString, 91
NSTextField, 84, 91
private, 297
public, 26
Method name, 34

Method naming steps, 215
.m file, 26–27
@min, 120
minuteOfHour, 54
minuteOfYear, 54
modalDelegate, 317
Modal operation, 218, 256
Modal windows, 325
Model classes, 123
modifierFlags, 249–250, 266
Modifier keys, 250, 266, 307
Modularity, and panels, 184, 355
mouseDown:, 80, 249–251, 258–259, 301, 316
mouseDragged:, 249, 251, 259, 261, 316
mouseEntered:, 274–275
Mouse events
  autoscrolling, 260–261
  clickCount, 250–251
  compositing, 256–258
  getting, 251
  and image opacity, 251–254, 257
  NSEvent, 249–250
  NSImage, 252, 256, 261–262
  NSOpenPanel, 251–256
  NSResponder, 249, 372
  #pragma mark, 251, 256
  view coordinate system, 258–260
mouseExited:, 274–275
mouseMoved:, 274
Mouse rollovers, 274–275
mouseUp:, 249, 251, 258–259, 316
Multibyte characters, 39
Mutable array, 43–44
mutableArrayValueForKey, 143
mutableCopy, 43–44
MyDocument, 125–128, 135, 144, 149, 161
MyDocument.h, 127, 135, 145, 149, 218, 341, 356, 407
MyDocument.m, 127, 136, 144, 341, 360–363, 372, 408
MyDocument.nib, 128, 133, 135, 173, 230, 356, 362, 407

N

Name keys, 202
name method, 210–212
NeXT Computer, Inc., 1
nextKeyView, 267–268, 306
nextResponder, 290–291, 372–373
NeXT Software, Inc., 1
NeXTSTEP, 1–3
Nib (NeXT Interface Builder) files, 13, 17, 28–29, 87–88, 183–195
nil, 24, 40, 44, 132–133
Nil-targeted actions
  nextResponder, 290–291
  nib file, 292–293
  and pasteboards, 287–294
  searching the responder chain, 291–292
nonatomic, 119
Non-object data types, 376–377
Notifications
  delegates, 215
  logging arrival, 214
  method naming steps, 215
  NSNotification, 210–212
  NSNotificationCenter, 209–212
  observer, 209–215
  poster, 209–210, 214
  posting, 211–213
  prefixes, 212
  registering for, 211
  userInfo dictionary, 214–215
NSAffineTransformStruct, 239
NSAlert, 107
NSAllocateCollectable(), 376–377, 379–380, 385
NSAnimation, 107
NSAnimationContext, 387, 393–394
NSApplication, 17, 107, 291, 293, 317
NSApplicationMain(), 13
NSArray, 41–43, 407
NSArrayController, 163, 178, 181, 373
  bindings, 135
  key-value coding and nil, 132–133
  model-view-controller, 123
  sorting, 133–135
NSAssert() , 61–62
NSAttributedString, 278–281
NSAutoReleasePool, 36–38, 72
NSBezierPath, 239–242, 248, 262
NSBox, 233
NSBrowser, 107
NSBundle, 63, 194–195, 226
NSButton, 81–82, 118, 246

NSButtonCell, 245–246
NSCalendarDate, 52–55
NSCAssert() , 62
NSClipView, 242
NSCoder, 154–157, 166, 403
NSCoding, 154–157, 167
NSColorList, 331–332
NSControl, 81, 334–335
NSController, 123–124
NSControl subclasses, 81–85
NSData, 142, 158–159, 161–162, 349
NSDate, 52, 54–55
NSDateFormatter, 327
NSDatePicker, 107, 175
NSDecimal, 133, 239
NSDictionary, 198–199, 409
NSDocument, 157–160, 168, 172
NSDocumentController, 157–158, 291
NSDraggingInfo protocol, 305
NSDrawer, 107
NSEvent, 249–250, 265–266
NSException, 57–58
NSFont, 277–278
NSFontManager, 107, 286
NSFormatter
  attributed strings, 337–338
  autocompletion, 336–337
  ColorFormatter, 328–334
  formatter methods, 332–334
  NSControl delegate, 334–335
  partial strings, validating, 335–337
  search strings for substrings, 332
NSGarbageCollector, 68–70
NSGradient, 304
NSImage, 107, 252, 256, 261–262
NSImageView, 175–176
NSInputManager, 284
NSInvocation, 139–141
NSKeyedArchiver, 154, 161, 165, 203–204, 213–214
NSKeyedUnarchiver, 154, 161–163, 165, 204, 206
NSLayoutManager, 107
NSLevelIndicator, 175–176
NSLocalizedString, 229–231
NSLog(), 38
NSMakeRange(), 278
NSManagedObject classes, 366–369
NSManagedObjectContext, 171–173, 180–181, 355, 357
NSMatrix, 107, 245
NSMenu, 107
NSMutableArray, 33–37, 41, 43–44, 68, 134
NSMutableAttributedString, 278
NSMutableDictionary, 198–199
NSNotification, 210–212
NSNotificationCenter, 209–212
NSNull, 44
NSNumber, 35–37
NSNumberFormatter, 327
NSObject, 41–42, 62–63, 80–81, 108, 121
NSObjectController, 123–124
NSOpenGLView, 397–398
NSOpenPanel, 208, 251–256
NSPanel, 183–184, 190
NSPasteboard, 287–289
NSPathControl, 107
NSPersistentDocument, 172, 180–181, 356
NSPipe, 405, 408–409, 412, 414
NSPoint, 239, 241–242, 249, 259, 280–282, 300
NSPrintOperation, 339–342
NSProgressIndicator, 309–314
NSRange, 96, 239, 278, 332, 340
NSRect, 239–242, 245
NSResponder, 80–81, 249, 265, 372
NSRuleEditor, 107
NSRunAlertPanel(), 217–218
NSSavePanel, 107
NSScannedOption, 376–377, 385
NSScrollView, 234, 242–244
NSSecureTextField, 84
NSShadow, 286
NSSize, 239
NSSlider, 81–83, 118, 123, 252–254, 399
NSSortDescriptor, 134
NSSound, 107
NSSpeechRecognizer, 107
NSSpeechSynthesizer, 88–89, 96–99, 101, 107
NSSplitView, 107, 234
NSString, 38–39, 41, 226, 281
  adding a method to, 295–297
  methods, 44–45, 91
NSTableView, 99–105, 107–108, 407
NSTabView, 107, 234–235

NSTask
 asynchronous reads, 410–411
 multithreading *versus* multiprocessing, 405–406
 object diagram, 409
 ping, 411–415
 .tar and .tgz files, 415
 zipinfo, 406–410, 415
NSText, 107
NSTextField, 81, 83–84, 91, 107, 118
NSTextStorage, 107
NSTextView, 79, 107, 151–152
NSTimer
 AppController, 314
 NSProgressIndicator, 309–314
 NSRunLoop, 316
 object diagram, 310
 timer-driven autoscrolling, 316
 typing-tutor project, 309–315
NSTokenField, 107
NSToolbar, 107
NSUndoManager
 adding undo to RaiseMan, 142–145
 editing on insert, 149–151
 key-value coding, 142–145
 key-value observing, 145–146
 NSInvocation, 139
 undo and redo stacks, 140–142
 undoing edits, 146–148
 and windows, 151–152
NSURL, 345
NSURLConnection, 345, 352
NSURLRequest, 345
NSUserDefaults, 200–201
NSUserDefaultsController, 207
NSView, 80–81, 233–235, 241, 245, 283, 300–301, 359
NSViewController, 355, 357, 372
NSWindow, 14, 81, 107, 291
NSWindowController, 157–161, 163, 183–195
 See also View swapping
NSWindow's initialFirstResponder outlet, 88
NSXMLDocument, 345–346, 349–350
NSXMLNode, 345–346, 348, 351–352
NULL, 24
Number formatter, 128–130
Number Formatter Attributes, 130, 175
numberOfRowsInTableView:, 100, 105, 137, 351, 410

O

Object(s)
 accessor methods, 74–77
 archived, 14
 autoreleased, 52, 72–74
 connecting, 20–23
 copies, 37
 Core Data, 180–181
 defined, 5
 delegates, 96–99, 107
 and key paths, 120–121
 po, 41, 59
 pointers, 62
 reachable, 375–376
 retain counts, 68–70
 temporary, 73–74
 unarchived, 14
 See also Archiving/unarchiving
ObjectAllocations instrument, 383
objectAtIndex:, 43
objectEnumerator, 199
objectForKey:, 199, 201
Object graph, 65, 181
Objective-C, 2–5
 classes, creating, 46–58
 code, 23
 debugger, 58–62
 description method, 50–55
 existing classes, 35–46
 extensions, 26, 38
 format string tokens, 38
 functions, calling, 26
 initializers, 55–58
 instances, 33–35
 keywords, 24
 messaging, 62–63
 NSArray, 41–43
 NSMutableArray, 41, 43–44
 NSObject, 41–42
 NSString, 41, 44–45
 protocol, 154–157
 strings, 38–39
 subclassing, 45–46
 types and constants, 23–24
 typographical conventions, 7
 visibility specifiers, 26
The Objective-C Language, 5
object method, 210–211

Object-oriented programming terms, 4–5
Objects & Controllers, 19–20, 128
objectValue, 84
Observable keys, 116–118
Observers, 115–118, 209–215
observeValueForKeyPath:ofObject:change:context:, 121, 146, 148
opacity, 256, 257, 272
Open counterparts in same editor, 26–27
OpenGL, 387, 397
OpenGL application, 397–404
OpenGL.framework, 398
OpenGL View Attributes, 399–401
openItem:, 352
OpenOffice, 280
openPanelDidEnd:returnCode:contextInfo:, 252, 255–256
OpenStep, 2–3
Operation mask, 307
Operators, 120
@optional, 167, 297
Ordered-to-many relationships, 142
otherMouseDown:, 249
otherMouseDragged:, 249
otherMouseUp:, 249
Outlets (instance variables), 19–20
Overriding init, 57–58

P

Pagination, 339–344
Panel Attributes, 192
Panels, 183–194
Partial strings, validating, 335–337
Password display bullets, 84
paste:, 289–290
pasteboardChangedOwner:, 294
Pasteboards
  BigLetterView cut:, **copy:**, and **paste:**, 289–290
  Lazy copying, 293–294
  nil-targeted actions, 290–293
  NSApplication, 291, 293
  NSPasteboard, 287–289
  pasteboardChangedOwner:, 294
  and PDF text, 294
  responder chain, 291–293
  server, 287–288, 294
PDF (portable document format), 2, 262, 283–285, 294, 339
PeopleView, 341–342
performClick:, 348
Persistence framework, 6
Persistent document, 172, 180–181, 356
Person class, 125–127, 136
Person.h file, 126
PICT, 262
Pixels, 247
Plug-in, 10
PNG, 262
Pointers, 21–22, 34
Points (measurement), 247
Polynomials, 388–393
Polynomial View Effects, 388
Polynomial View Identity, 382
po (print-object), 41, 59
Poster, 209–210, 214
Posting a notification, 212–213
postNotification:, 211
postNotificationName:, 211
PostScript, 2
#pragma mark, 137, 251, 256, 271, 305, 342, 343, 351
Precedence of defaults, 201
Preferences panel
  AppController.m, 187–188
  menu setup, 186–187
  NSBundle, 194–195
  NSPanel, 183–184
  NSWindowController, 183–195
  object diagram, 185
  PreferenceController.m, 185, 188, 192–194
  Preferences.nib, 188–192
  Reset Preferences button, 208
prepareRandomNumbers, 48–49, 55
prepareWithInvocationTarget:, 141, 144–145, 147–148
pressure, 250
printDocument:, 341
printf, 38
Printing
  isDrawingToScreen, 344
  NSPrintOperation, 339–342
  pagination, 339–344
print-object (po), 41, 59, 342
Private methods, 297

Progress Indicator Attributes, 312
Progress Indicator Bindings, 313
Project directory, 10–11
`@property`, 24, 118–119
Property attributes, 118–119
Property list classes, 203
Property (**`NSManagedObjectModel`**), 171–172, 357, 367
Protocol, 154–157, 167, 297–298
`@protocol`, 24
Public methods, 26
Python programming language, 4

Q

Quartz, 247, 397
QuartzCore framework, 378, 388
Quicklook, 168
Quit menu item, 32
Quotation marks, 38, 202, 226

R

RaiseMan application
  Interface Builder, 128–132
  `NSArrayController`, 124–132
  `.rsmn` extension, 163–164
  and undo, 140–145
  Xcode, 125–127
`RANDFLOAT()`, 378
`random()`, 26, 48, 55–56, 61, 71, 242, 392
Random number generator application
  build and run, 27–28
  class creation, 17–19
  compiling, 27–28
  implementation file, 25–27
  instance creation, 19–20
  main function, 12–13
  new project creation, 10–12
  objects, 20–23
  troubleshooting, 28–29
  user interface layout, 15–17
Range of characters, 278
Range of numbers, 278
Reachable objects, 375
`readFromData:error:`, 159, 162
`readFromFileWrapper:error:`, 159
`readFromPasteboard:`, 289–290, 296, 306
`readFromURL:error:`, 160, 408–409
`readonly`, 119
`readwrite`, 119
`rectForPage:`, 340, 343
`registerDefaults:`, 201
`registerForDraggedTypes:`, 303–304
Relationships, 120, 142–144, 172
  See also Core data relationships
`release`, 69, 80
`reloadData`, 100
`remove:`, 125, 130, 178, 220, 370–373
`removeAllObjects`, 44
`removeEmployee:`, 218–220
`removeEmployeesObject:`, 369
`removeObject:`, 44
`removeObjectAtIndex:`, 44
`removeObjectForKey:`, 199, 201
`removeObserver:`, 211
Reset Preferences button, 208
`reshape`, 397
`resignFirstResponder`, 265, 272, 276
`resizeAndRedrawPolynomialLayers`, 393–394
Resizing the window, 109, 320–321, 362–363
Responder chain, 291–293, 372–373
`respondsToSelector:`, 105, 109
`retain`, 69, 80, 119
Retain, then Release, 75
Retain counts
  accessor methods, 74–77
  autoreleased objects, 72–74
  deallocation, 70–72
  dog and leash analogy, 66, 68
  `NSAutoreleasePool`, 72
  retain and release balance, 69–70
Return code, 317, 324–325
`rightMouseDown:`, 249
`rightMouseDragged:`, 249
`rightMouseUp:`, 249
Rollovers, 274–275
`.rsmn` extension, 163–164
RTF, 280
RTFD, 280
Ruby programming language, 4
Run loop, 316, 410
`runModalForWindow:`, 325

S

Save panel, 283
`savePDF:`, 284

Saving (adding to apps), 161
sayIt:, 86, 88–89, 97–98, 101, 106
Scanned, 376–377
Scroll view, 234, 242–244
scrollWheel:, 249
Sculley, John, 1
Search strings for substrings, 332
secondNumber, 47–51, 55–57, 71–73
seed: method, 19, 22, 26
Seed button, 22
Selector, 34, 90, 133, 139, 309, 317
@selector, 24, 90
Selector table, 63
Serialization. *See* Archiving/unarchiving
set, 75–77
setBool:forKey:, 201
setCalendarFormat:, 54
setEmployees:, 127
setEnabled:, 81
setEntryDate:, 47–49, 56–57, 77
setExpectedRaise:, 132
setFido:, 113, 119
setFloat:forKey:, 201
setFloatValue:, 82
setFrameLoadDelegate:, 353
setImage:, 260
setInteger:forKey:, 201
setMainFrameURL:, 353
setNeedsDisplay:, 238–240, 247, 257, 282
setNilValueForKey:, 132–133
setObject:forKey:, 199, 201
setObjectValue:, 84, 100, 137, 327
setState:, 82
setString:, 282
setStringValue:, 84, 91
Setter methods, 75–76, 119
setValue:forKey:, 111–113, 132
setValue:forKeyPath:, 120
Shadow text style, 286
Sheet (alert panel window), 218
Sheets
  adding, 318–324
  contextInfo, 324
  modalDelegate, 317
  modal windows, 325
  NSApplication methods, 317
  object diagram, 318
  outlets and actions, 319
showOpenPanel:, 252, 254–255
showSpeedSheet:, 319–320
showWindow:, 187, 193
sigint signal, 411
size, 280
Size Inspector, 109, 237–238, 243–244
sizeWithAttributes:, 281
Slider Attributes, 83, 114, 253, 321, 400
Slider Bindings, 114, 254, 322
Slider Cell Attributes, 400
Slider Inspector, 83
Sliders, 82–83, 114–115, 253–256, 399
Smalltalk (language), 33
Snapshot, 294
Sort descriptors, 135
sortDescriptorsDidChange:, 135
Sorting, 133–135
sortUsingDescriptors:, 134
Speech synthesizer, 85–86, 96–99
speechSynthesizer:, 96
speechSynthesizer:didFinishSpeaking:, 96
speechSynthesizer:willSpeakPhoneme:, 96
speechSynthesizer:willSpeakWord:, 96
speedSheet, 319, 321–324
Split views, 234
srandom(), 26
standardUserDefaults, 200
startButton, 97–99, 101, 106, 412
startStopPing:, 413
state, 80–82
stopButton, 97–99, 101, 106
stopGo:, 311–312, 315, 319
stopModalWithCode:, 325
StretchView, 236–237, 241, 243–244, 251–261, 316
stringByAppendingString:, 45
stringForObjectValue:, 328, 333, 337–338
stringForType:, 289
Strings, 38–39, 44, 278–281
stringValue, 84, 91
stringWithFormat:, 73–74, 91
Strong references, 376
structs, 5, 109, 239
Subclass, 19, 24
Subclassing, 45–46, 79
Subversion, 188
subviews, 233–234
@sum, 120
Superclass, 23–24
superview, 233, 237, 244–245, 291

Symbolic breakpoints, 59–60
Syntax-aware indention, 24–25
`@synthesize`, 24, 118–119

T

Table Column Attributes, 134, 349
Table Column Bindings, 130–131, 180
Table view
  `dataSource`, 99–104, 410
  `delegate`, 105–106
tableView, 100–106, 135–137, 150, 346, 408, 410
Table View Attributes, 102–103, 219
`tableView:objectValueForTableColumn:row:`, 100, 105
Taligent, 79
`.tar` files, 415
`target`, 22, 80
Target/action, 22
  `AppController` class, 88–90
  debugging, 92–93
  menu items, 90
  Nib file, 86–88
  NSControl subclasses, 81–85
  SpeakLine project, 85–86
`taskTerminated:`, 414–415
Temporary objects, 73–74
Terminal tool, 165, 207–208
`terminate:`, 32
Text, 277–286
  See also Drawing text with attributes
Text field, 16–17, 84
Text Field Attributes, 348
Text Field Bindings, 115, 330
Text Field Inspector, 83
textField outlet, 21, 88
Text View Attributes, 151
`.tgz` files, 415
TIFF, 262
`timeIntervalSinceDate:`, 55
Timer. *See* `NSTimer`
`timestamp`, 250
`titleBarFontOfSize:`, 278
Toggles, 80–81
Tokens, 38, 45, 54, 165, 231
To-many relationships, 142, 365
Tool names, 36
`toolTipsFontOfSize:`, 278
To-one relationships, 142, 365
Tracking area for rollovers, 274–275
Type declarations, 168
Types, Objective-C, 23–24
TypingTutor project, 266–274, 309–315
Typographical conventions, 7

U

Unarchived (objects), 14
`unarchiveObjectWithData:`, 162
Unarchiving. *See* Archiving/unarchiving
`unbind:` method, 121
Undo, 167
`undoManager`, 141–145, 151–152
  See also **`NSUndoManager`**
`undoManagerForTextView:`, 151–152
Undo/redo, 140–142, 151–152
Unicode, 38–39, 44
Unicode UTF-8, 228–229
Universal type identifiers (UTIs), 168–169, 406–407
University of California, Berkeley, 1
Unix, 1–3, 405
Unordered-to-many relationships, 142, 365
`updateChangeCount:`, 168
User defaults
  command line tool, 207–208
  name keys, 202
  `NSDictionary`, 198–199
  `NSMutableDictionary`, 198–199
  `NSUserDefaults`, 200–201
  `NSUserDefaultsController`, 207
  prefix of global variables, 202
  registering defaults, 203
  setting background color, 206
  untitled documents, 198, 205–206
  user-edited, 203–205
`userFixedPitchFontOfSize:`, 278
`userFontOfSize:`, 278
`userInfo` dictionary, 214–215, 324
User interface layout, 15–17, 190–192
`UTExportedTypeDeclarations`, 168

V

`valueForKey:`, 111
`valueForKeyPath:`, 120

Version-control systems, 188
View(s)
  coordinate system, 258–260
  flipping, 247
  hierarchy, 233–235
  See also Custom views
View-backing layer, 395
View class, 123
Views & Cells, 15, 128
View subclass, 236
View swapping
  changeViewController:, 356–357, 361–362
  design, 356–362
  displayViewController:, 361–363, 372
  ManagingViewController, 357–358, 360–361
  NSViewController, 355, 357
  object diagram, 356
  resizing the window, 362–363
Visibility specifiers, 26
Voice projects. *See* Speech synthesizer
void, 24, 26

W

Weak references, 376
WebKit framework, 353
Web service
  AmaZone, 346–352
  Amazon Web service, 346–349
  AWS ID, 349
  code, 349–352
  fetchBooks:, 346–349
  interface layout, 347–349
  NSURL, 345
  NSXMLDocument, 345–346, 349–350
  NSXMLNode, 345–346, 348, 351–352
  webView:, 353–354
  XML, 348
Web sites
  Darwin development, 1
  Free Software Foundation, 62
  The Objective-C Language PDF, 5
  this books's, 418
window, 250
Window Attributes, 321, 363
windowControllerDidLoadNib:, 160, 206
windowDidLoad:, 193, 204–205
window (**NSView** method), 233
Window resizing, 192, 320–321, 362–363
Window server, 2, 31
windowWillReturnUndoManager:, 152
Wozniak, Steve, 1
writeToPasteboard:, 289–290, 302
writeToURL:, 159

X

x- and *y*-coordinates, 247, 258
Xcode
  breakpoints, 58–60
  console, 26–28
  and defaults, 207–208
  documentation, 29–30
  document-based application, 125, 172
  Foundation Tool, 35–37
  and garbage collector, 382
  icons and extensions, 163–165
  Info.plist, 157, 163–164, 168
  localization, 224–227
  new controller class, 86
  #pragma mark, 251
  preferences, 25–28
  project directory, 9–11
  RandomApp (random number generator), 9–13
  targets, 66
XIB files, 188
XML, 168, 188, 202–203, 345, 348
XML data, 349–351
XML node, 348
XPath, 348–352
X window server, 2

Z

Zero-length strings, 89, 301, 333, 336
Zip files, 407
ZIPspector, 406–410
Zombies, 92–93