

[Ads by Google](#)

[IOS 4 Applications](#)

[IOS Objective C](#)

[Iphone Application](#)

[IOS App De](#)

iOS 4 iPhone Data Persistence using Archiving



One like. [Sign Up](#) to see what your friends like.



[Previous](#)

[Table of Contents](#)

[Next](#)

iOS 4 iPhone Directory
Handling and File I/O – A
Worked Example

iOS 4 iPhone Database
Implementation using
SQLite



Purchase and download the fully updated iOS 5 Edition of this book in Print (\$27.99) or eBook (\$12.99) format.

[iPhone iOS](#) 5 Development Essentials Print and eBook (ePub/PDF/Kindle) editions contain 61 chapters.

[Buy eBook](#)

[Buy Print](#)

In the previous chapters of this book we have looked at some basic file and directory handling operations that can be performed within an iOS 4 iPhone application. In the chapter entitled [Working with Files on iOS 4](#) we looked at creating files and reading and writing data from within an [iOS application](#). In this chapter we will look at another form of data persistence on the iPhone using a more object oriented approach known as [Core Data](#).

[Get a Free 14-day Videoconferencing Trial!](#)

[Contents](#) [\[hide\]](#)

- 1 An Overview of Archiving
- 2 The Archiving Example Application

[Node.js Webinar 7/25](#)

Learn how to master the basics of Node.js.
Sign up to attend now!
www.vmwareevents.webex.com/js

- 3 Implementing the Actions and Outlets
- 4 Releasing Memory
- 5 Designing the iPhone User Interface
- 6 Checking for the Existence of the Archive File on Startup
- 7 Archiving Object Data in the Action Method
- 8 Testing the Application
- 9 Summary

Objective C Xcode


Try AppCode — a New Objective-C IDE
Better Refactorings, Debugger & VCS
JetBrains.com/AppCode

iOS Objective-C Training

Intensive Objective-C 2.0 Training. 5-10
Day Workshops. View Schedule!
AboutObjects.com

Ultimate DDoS Protection

Defend Your Site & Defeat Hackers. Guards
24/7, Free Security Audit!
Neustar.biz/DDoS-Attack-Protection

AdChoices 



An Overview of Archiving

iPhone iOS applications are inherently object oriented in so much as they are developed using Objective-C and consist of any number of objects designed to work together to provide the required functionality. As such, it is highly likely that any data created or used within an application will be held in memory encapsulated in an object. It is also equally likely that the data encapsulated in an object may need to be saved to the iPhone's file system so that it may be restored on future invocations of the application. One approach might be to write code that extracts each data element from an object and writes it to a file. Similarly, code would need to be written to read the data from the file, create an instance of the original object and then assign the data to that object accordingly. Whilst this can be achieved, it can quickly become complex and time consuming to implement.

An alternative is to use a mechanism called archiving. Archiving involves encoding objects into a format that is written to a file. Data may subsequently decoded (or unarchived) and used to automatically rebuild the object. This concept is somewhat analogous to serialization as supported by languages such as Java.

A number of approaches to archiving are supported by the Foundation [Framework](#). Arguably the most flexible option is that provided by the NSKeyedArchiver class. This class provides the ability to encode an object into the form of a binary property list that is written to file and may subsequently be decoded to recreate the object using the NSKeyedUnarchiver class.

An alternative option is to use the writeToFile:atomically method available with a subset Foundation classes. This mechanism writes the object data to file in the form of an XML property list file. This approach, however, is limited to NSArray, NSData,

NSDate, NSDictionary, NSNumber and NSString based objects.

In the remainder of this chapter we will work through an example of archiving using the NSKeyedArchiver and NSKeyedUnarchiver classes.

The Archiving Example Application

The end product of this chapter is an application that prompts the user for a name, address and phone number. Once this data has been entered, pressing a button causes the contact data to be stored in an array object which is then archived to a binary property file. On a subsequent reload of the application this data is unarchived and used to recreate the array object. The restored data is then extracted from the array object and presented to the user.

Begin by launching Xcode and create a new iOS iPhone project called archive using the View-based application template. Once the main Xcode project window appears populated with the template files, it is time to start writing some code.

Implementing the Actions and Outlets

In order to connect the [user interface](#) objects with our back end code we will need to declare some outlets for our name, address and phone text fields and an action for the button object. We will also need an NSString variable to hold the path to the archive data file. With these requirements in mind, select the archiveViewController.h file from the file list in the [Xcode](#) project window, modify the file so that it reads as follows and then save the file:

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

@interface archiveViewController : UIViewController {
    UITextField      *name;
    UITextField      *address;
    UITextField      *phone;
    NSString          *dataFilePath;
}
@property (nonatomic, retain) IBOutlet UITextField *name;
@property (nonatomic, retain) IBOutlet UITextField *address;
@property (nonatomic, retain) IBOutlet UITextField *phone;
@property (nonatomic, retain) NSString *dataFilePath;
- (IBAction) saveData;
@end
```

Having made the appropriate declarations in the interface file, the next step is to move to the archiveViewController.m implementation file where we will synthesize the access methods for our instance variables and create a template method for the saveData action:

```
#import "archiveViewController.h"
```

```
@implementation archiveViewController
```

Are you a developer? Try out the [HTML to PDF API](#)

```
@synthesize name, address, phone, dataFilePath;

- (void) saveData
{
}

:
:
@end
```

Next, make sure that we free up memory that we have allocated when the application exists:

Releasing Memory

Having allocated memory in implementing the above outlets, it is important that we add code to free up any resources that were allocated during execution of the application. To do so, edit the `archiveViewController.m` file again and modify the `viewDidLoad` and `dealloc` methods as follows:

```
- (void)viewDidLoad {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
    self.name = nil;
    self.address = nil;
    self.phone = nil;
    self.dataFilePath = nil;
}

- (void)dealloc {
    [name release];
    [address release];
    [phone release];
    [dataFilePath release];
    [super dealloc];
}
```

We will need to do some more coding later but at this point it makes sense to design the user interface and establish the connections between the user interface object and the outlets and action we have declared so far.

Designing the iPhone User Interface

The user interface for our application is going to consist of three UILabels, three UITextFields and single UIButton. Launch the Interface Builder tool by double clicking on the `archiveViewController.xib` in the main Xcode project window. Display the tool's Library window if it is not already visible by selecting the *Tools -> Library* menu option or by pressing Command+L. Drag, drop, resize, position and configure objects on the View window canvas until your design approximates that illustrated in the following figure:



The next step is to establish the connections to our action and outlets. Beginning with the outlets, hold down the Ctrl key and click and drag from the File's Owner item in the documents window to the text field component located to right of the Name label object. Release the Ctrl key and mouse button and select the name outlet from the resulting menu. Repeat these steps for the address and phone text fields, connecting them to the corresponding outlets.

To connect the action, select the Save button object in the view window and display the Connections Inspector window (*Tools -> Connections Inspector* or Command+2). [Click](#) with the mouse within the small round circle to the right of the *Touch Up Inside* event and drag the blue line to the *File's Owner* entry in the documents window. Release the mouse button and select the

Are you a developer? Try out the [HTML to PDF API](#)

saveData method from the resulting window.

Having designed the user interface and established the necessary connections save the design and exit from Interface Builder.

Checking for the Existence of the Archive File on Startup

Each time the application is launched by the user, [the code](#) will need to identify whether the archive data file exists from a previous session. In the event that it does exist, the archive will need to be read and the contents therein used to recreate the original array object from which the archive as created. Using this newly recreated array object, the array element will then be extracted and used to populate the name, address and phone text fields.

The traditional location for placing such initialization code is in the viewDidLoad method of the view controller class. Within [the project](#) window, select the archiveViewController.m file and scroll down the contents of this file until you reach the viewDidLoad method. This method is commented out by default so remove the leading and trailing /* and */ markers before entering the following code:

```
- (void)viewDidLoad {
    NSFileManager *filemgr;
    NSString *docsDir;
    NSArray *dirPaths;

    filemgr = [NSFileManager defaultManager];

    // Get the documents directory

    dirPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

    docsDir = [dirPaths objectAtIndex:0];

    // Build the path to the data file

    dataFilePath = [[NSString alloc] initWithString: [docsDir stringByAppendingPathComponent: @"data.archive"]];

    // Check if the file already exists

    if ([filemgr fileExistsAtPath: dataFilePath])
    {
        NSMutableArray *dataArray;

        dataArray = [NSKeyedUnarchiver unarchiveObjectWithFile: dataFilePath];

        name.text = [dataArray objectAtIndex:0];
        address.text = [dataArray objectAtIndex:1];
        phone.text = [dataArray objectAtIndex:2];
    }
    [filemgr release];
    [super viewDidLoad];
}
```

Within this method a number of variables are declared before creating an instance of the `NSFileManager` class.

A call is then made to the `NSSearchPathForDirectoriesInDomains` function and the path to the application's Documents directory extracted from the returned array object. This path is then used to construct the full pathname of the archive data file, which in turn is stored in the `dataFilePath` instance variable we previously added to the view controller class interface file. Having identified the path to the archive data file, the [file manager](#) object is used to check for the existence of the file. If it exists, the file is "unarchived" into a new array object using the `unarchiveObjectWithFile` method of the `NSKeyedUnarchiver` class. The data is then extracted from the array and displayed in the corresponding text fields.

With this code implemented, select the Build and Run toolbar button to compile and execute the application in the simulator. Assuming no problems are encountered, the next step is to implement the action method. If problems are encountered, check the details reported by Xcode and correct any syntax errors that may have been introduced into the code. Once the app has launched successfully, exit from the iOS Simulator and return to the main Xcode project window.

Archiving Object Data in the Action Method

The Save button in the user interface design is connected to the `saveData` method of the view controller class. Edit the `archiveViewController.m` file and modify the template action method as follows:

```
- (void) saveData
{
    NSMutableArray *contactArray;

    contactArray = [[NSMutableArray alloc] init];

    [contactArray addObject:name.text];
    [contactArray addObject:address.text];
    [contactArray addObject:phone.text];

    [NSKeyedArchiver archiveRootObject: contactArray toFile:dataFilePath];

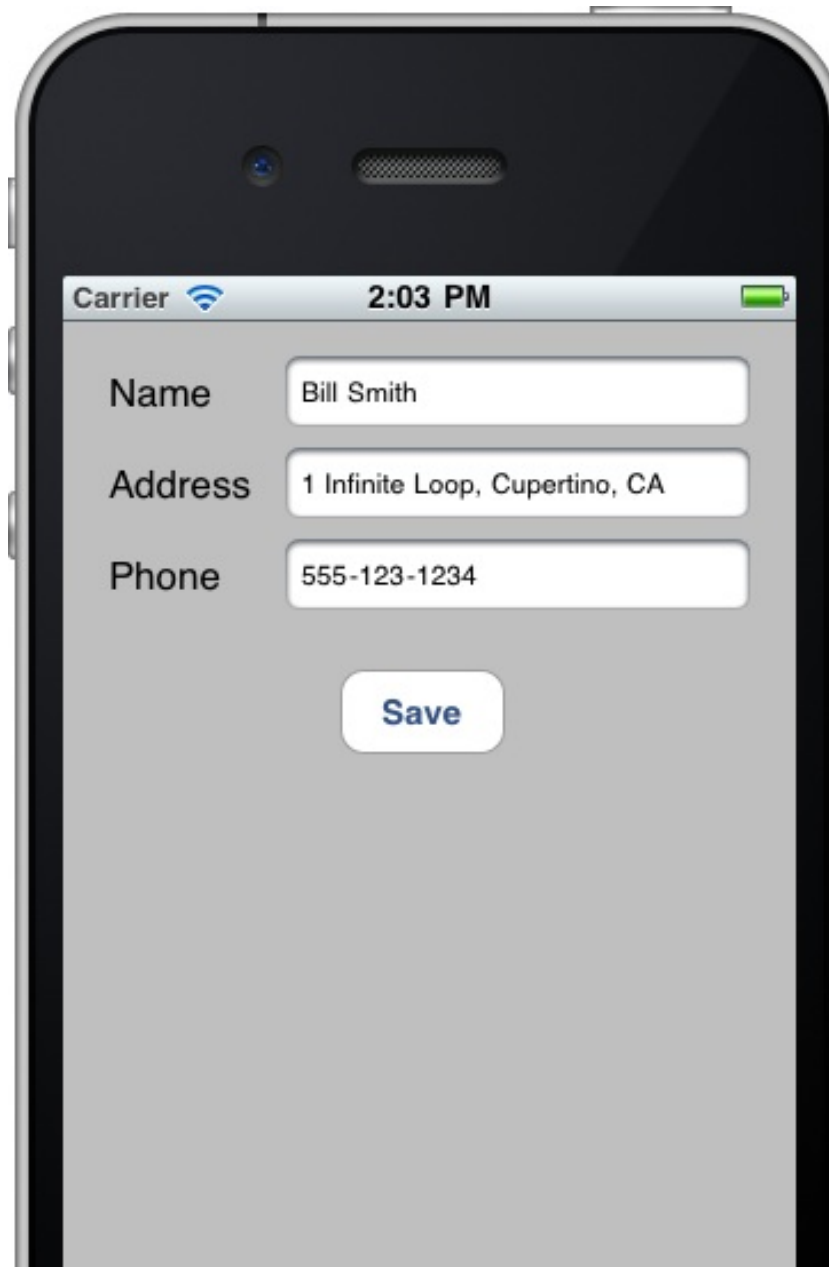
    [contactArray release];
}
```

When triggered, this method creates a new array and assigns the content of each text field to an element of that array. The array object is then archived to the predetermined data file using the `archiveRootObject` method of the `NSKeyedArchiver` class before releasing the memory allocated to the array. The instance data of the array object is now saved to the archive ready to be loaded next time the application is executed.

Testing the Application

Save the code changes and build and run the application in the simulator environment. Enter a name

Save the code changes and build and run the application in the simulator environment. Enter a name, address and phone number into the respective text fields and press the save button. Exit the iOS Simulator (*iOS Simulator -> Quit iOS Simulator*) and then relaunch the application (*Run -> Run*). The application should re-appear with the text fields primed with the contact information saved during the previous session:



iPhone iOS 5
Development
Essentials eBook

\$12.99

Buy eBook

eBookFrenzy.com



Summary

Whilst data can be written to files on the iPhone using a variety of mechanisms, archiving provides the ability to save the instance data of an object to file at a particular point and then restore the object to that state at any time in the future. This implements an object-oriented approach to data persistence on iOS iPhone based applications.



Purchase and download the fully updated iOS 5 Edition of this book in Print (\$27.99) or eBook (\$12.99) format.

iPhone iOS 5 Development Essentials Print and eBook (ePub/PDF/Kindle) editions contain 61 chapters.

[Buy eBook](#)



[Buy Print](#)



[Previous](#)

iOS 4 iPhone Directory
Handling and File I/O – A
Worked Example

[Table of Contents](#)

[Next](#)

iOS 4 iPhone Database
Implementation using
SQLite



navigation

- [Home](#)
- [iOS / iPhone / iPad](#)
- [Objective-C](#)
- [PowerShell](#)
- [Hyper-V](#)
- [VMM 2008](#)
- [VMware Server](#)
- [Xen Virtualization](#)
- [Windows Server 2008](#)
- [Security+](#)
- [Red Hat Linux](#)
- [Linux eBooks](#)
- [Ubuntu Linux](#)
- [Fedora Linux](#)
- [Fedora Desktop](#)
- [OpenSUSE Desktop](#)
- [C#](#)
- [Visual Basic](#)
- [MySQL](#)
- [PHP](#)
- [JavaScript](#)
- [Ruby](#)
- [Windows](#)
- [Networking](#)
- [Web Development](#)
- [Feedback](#)
- [Linuxtopia.org](#)
- [Virtuatopia.com](#)
- [eBook Store](#)



Find us on
Facebook



Search



iPhone iOS 5
Development Essentials
Book

\$12.99 (eBook)

\$27.99 (Print)

Buy eBook

Buy Print

eBookFrenzy.com



Cocoa
Flavanols
are
Good
for
You.



You. Renewed.

LEARN MORE ►

the new
Samsung
Galaxy S III
only \$199⁹⁹



Get It Now
free shipping



Requires new 2-yr agreement with
qualifying voice and data plans.

