Seneca College of Applied Arts & Technology

PLAIN HTML VERSION: lecture6a.161.txt.html
Xcode Project Archive: lecture6a.161.zip

```
/*
file:    lecture6a.161.swift
author:  Danny Abesdris
date:    February 17, 2016
purpose: MAP523AA/DPS923AA lecture #6a

         Swift programming language topics:
         Data Persistence (defaults and property lists).
*/

/*
Data Persistence:

Saving data across application launches is a requirement that most iOS applications
have, from storing user preferences in the defaults system to managing large data
sets in a relational database.
Virtually every iOS application stores data for later use.
The data stored within your application can range from user preferences to
large relational data sets.


File System and Application Sandboxing

Security on the iOS platform is a top priority for Apple and subsequently, iOS
applications are placed in what is known as a "sandbox". An application's sandbox
not only refers to an application's sandbox directory in the file system, but also
includes controlled and limited access to user data stored on the device, system servi
and hardware.
The operating system installs each iOS application in a sandbox directory that
contains the application bundle directory and three additional directories,
Documents, Library, and tmp.
The application's sandbox directory, often referred to as its home directory, can be
accessed by calling a simple Foundation function, NSHomeDirectory( ).
*/
print(NSHomeDirectory( ))

/*
You can try this yourself. In any Xcode iOS Single View Application, place this code:
print(NSHomeDirectory( )) in any function that is invoked by your application.
Your output should look similar to:
*/
/Users/danny.abesdris/Library/Developer/CoreSimulator/Devices/
C0DFD357-580B-46EC-AD99-E6F4BB21819B/data/Containers/Data/Application/
F938C546-6FF0-45E6-B9A8-9B39C43863FD


// To retrieve the path to the application's Documents directory, the following is req

let directories = NSSearchPathForDirectoriesInDomains(.DocumentDirectory,
                  NSSearchPathDomainMask.UserDomainMask, true)
```

```
if let documents = directories.first {
    print(documents)
}

/*
We invoke the NSSearchPathForDirectoriesInDomains() function, which is defined in the
Foundation framework. As the first argument, we pass in DocumentDirectory of type
NSSearchPathDirectory to indicate that we're only interested in the application's
Documents directory.
The benefit of sandboxing? The primary reason for sandboxing applications is security.
By confining applications to their own sandbox, compromised applications
cannot cause damage to the operating system or other applications.

The tmp directory should only be used for temporarily storing files. The operating
system is free to empty this directory at any time, for example, when the device is
low on disk space.
The Documents directory is meant for user data, whereas the Library directory is used
for application data that isn't strictly tied to the user.

NOTE: Your application isn't supposed to modify the contents of the application bundle
directory. The application bundle directory is signed when the application is installe
By modifying the contents of the application bundle directory in any way, the
aforementioned signature is altered, which means the operating system doesn't allow th
application to launch again. This is another security measure put into place by Apple
to protect customers.

Data Persistence Options:

There are several strategies for storing application data on disk.
In this article, we take a brief look at four common approaches on iOS:

defaults system
property lists
SQLite
Core Data

User Defaults:
The defaults system is something that iOS inherited from OS X. Even though it was creat
and designed for storing user preferences, it can be used for storing any type of data
long as it's a property list type, NSString, NSNumber, NSDate, NSArray,NSDictionary, a
NSData, or any of their mutable variants.

What about Swift data types? Fortunately, Swift is smart enough. It can store strings
numbers by converting them to NSString and NSNumber. The same applies to Swift arrays
dictionaries.

The defaults system is nothing more than a collection of property lists, one property
per application. The property list is stored in a Preferences folder in the applicatio
Library folder, hinting at the property list's purpose and function.

One of the reasons that developers like the defaults system is because it's so easy to
*/

let userDefaults = NSUserDefaults.standardUserDefaults( )

// Setting Values
userDefaults.setBool(true, forKey: "Key1")
userDefaults.setInteger(123, forKey: "Key2")
userDefaults.setObject("Some Object", forKey: "Key3")
userDefaults.setObject([1, 2, 3, 4], forKey: "Key4")
```

```
// Getting Values
userDefaults.boolForKey("Key1")
userDefaults.integerForKey("Key2")
userDefaults.objectForKey("Key3")
userDefaults.objectForKey("Key4")

userDefaults.synchronize( )
/*
By calling standardUserDefaults( ) on NSUserDefaults, a reference to the shared default
object is returned.
The call to synchronize( ) is used to write the shared defaults object to disk.
To see the changes, open a new Finder window and navigate to Library > Developer >
CoreSimulator > Devices > <DEVICE_ID> > data > Containers > Data > Application >
<APPLICATION_ID>.

<DEVICE_ID> and <APPLICATION_ID> are two identifiers unique to the simulator and your
application respectively. The location of the application sandbox for the simulator
depends on the version of Xcode you're using.
*/

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/Propert
<plist version="1.0">
<dict>
    <key>Key1</key>
    <true/>
    <key>Key2</key>
    <integer>123</integer>
    <key>Key3</key>
    <string>Some Object</string>
    <key>Key4</key>
    <array>
        <integer>1</integer>
        <integer>2</integer>
        <integer>3</integer>
        <integer>4</integer>
    </array>
</dict>
</plist>

/*
Property Lists:
The following code snippet demonstrates how to write an array or dictionary to disk.
In theory, property lists can be as complex or as large as you need to make them, howev
property lists were not designed to store tens or hundreds of megabytes of data and
attempting to do so will likely result in degraded performance of your iOS application
*/

   let directories = NSSearchPathForDirectoriesInDomains(.DocumentDirectory,
                     NSSearchPathDomainMask.UserDomainMask, true)

   if let documents = directories.first {
      if let urlDocuments = NSURL(string: documents) {
         let urlLangs = urlDocuments.URLByAppendingPathComponent("langs.plist")
         let urlDictionary = urlDocuments.URLByAppendingPathComponent("dictionary.plist

         // Write Array to Disk
         let langs =
         [ "C", "Perl", "Python", "C++", "Java", "Pascal", "Ruby", "Swift", "Javascript
           "PHP", "Assembler", "Processing.js", "Prolog", "Smalltalk", "Objective-C",
           "Rust", "Basic", "Logo", "Lisp", "D", "Simula", "Haskell", "Go", "Fortran",
```

```
            "Ada", "Bash", "Algol 68", "AppleScript", "ActionScript", "COBOL", "Delphi"

        let dictionary = ["anArray" : langs, "aNumber" : 12345, "aBoolean" : true] as

        langs.writeToFile(urlLangs.path!, atomically: true)

        dictionary.writeToFile(urlDictionary.path!, atomically: true)

        // Load from Disk
        let loadedLangs = NSArray(contentsOfURL: urlLangs)
        if let myLangs = loadedLangs {
            print(myLangs)
        }

        let loadedDictionary = NSDictionary(contentsOfURL: urlDictionary)
        if let dictionary = loadedDictionary {
            print(dictionary)
        }
    }

/*
Writing the array to disk is as easy as calling writeToFile(_:atomically:) on the array
Run the application in the simulator and navigate
to the application's Documents directory. In this directory, you should see the
two property lists just created.

Writing Objects to a Property List
This is what the property list of the dictionary looks like when you open it in a
text editor.
*/
```

langs.plist

langs.plist › No Selection

Key	Type	Value
▼ Root	Array	(34 items)
Item 0	String	C
Item 1	String	Perl
Item 2	String	Python
Item 3	String	C++
Item 4	String	Java
Item 5	String	Pascal
Item 6	String	Ruby
Item 7	String	Swift
Item 8	String	Javascript
Item 9	String	PHP
Item 10	String	Assembler
Item 11	String	Processing.js
Item 12	String	Prolog
Item 13	String	Smalltalk
Item 14	String	Objective-C
Item 15	String	C#
Item 16	String	PL/SQL
Item 17	String	Rust
Item 18	String	Basic
Item 19	String	Logo
Item 20	String	Lisp
Item 21	String	D
Item 22	String	Simula
Item 23	String	Haskell
Item 24	String	Go
Item 25	String	Fortran
Item 26	String	Erlang
Item 27	String	Ada
Item 28	String	Bash
Item 29	String	Algol 68
Item 30	String	AppleScript
Item 31	String	ActionScript
Item 32	String	COBOL
Item 33	String	Delphi

dictionary.plist

dictionary.plist › No Selection

Key	Type	Value
▼ Root	Dictionary	(3 items)
aBoolean	Boolean	YES
aNumber	Number	12,345
▼ anArray	Array	(34 items)
Item 0	String	C
Item 1	String	Perl
Item 2	String	Python
Item 3	String	C++
Item 4	String	Java
Item 5	String	Pascal
Item 6	String	Ruby
Item 7	String	Swift
Item 8	String	Javascript
Item 9	String	PHP
Item 10	String	Assembler
Item 11	String	Processing.js
Item 12	String	Prolog
Item 13	String	Smalltalk
Item 14	String	Objective-C
Item 15	String	C#
Item 16	String	PL/SQL
Item 17	String	Rust
Item 18	String	Basic
Item 19	String	Logo
Item 20	String	Lisp
Item 21	String	D
Item 22	String	Simula
Item 23	String	Haskell
Item 24	String	Go
Item 25	String	Fortran
Item 26	String	Erlang
Item 27	String	Ada
Item 28	String	Bash
Item 29	String	Algol 68
Item 30	String	AppleScript
Item 31	String	ActionScript
Item 32	String	COBOL
Item 33	String	Delphi