

Connecting Apple's iPhone to Google's cloud computing offerings

Skill Level: Intermediate

[Noah Gift \(noah.gift@giftcs.com\)](mailto:noah.gift@giftcs.com)
Founder GiftCS, LLC
GiftCS

[Jonathan Saggau \(jonathan@jonathansaggau.com\)](mailto:jonathan@jonathansaggau.com)
Founder and CEO of Sounds Broken, Inc.
Sounds Broken inc

20 Jan 2009

Cloud computing and software development for handheld devices are two very hot technologies that are increasingly being combined to create hybrid solutions. With this article, learn how to connect Google App Engine, Google's cloud computing offering, with the iPhone, Apple's mobile platform. You'll also see how to use the open source library, TouchEngine, to dynamically control application data on the iPhone by connecting to the App Engine cloud and caching that data for offline use.

Introduction

The past couple of years have seen an explosion in innovation, making 2008 an interesting year for technology. Two of the exciting innovations have been cloud computing and mobile application development. In this article, we explore a communication methodology that takes advantage of these two technologies to create a synergistic developers' dream. Within the article, we will use Google App Engine, Google's cloud computing platform, and the iPhone, Apple's mobile platform, to develop an application that synchronizes its data from "The Cloud."

We utilize a simple method of pulling data from App Engine on the iPhone; one that lets python and App Engine do the heavy lifting. The common method of using RSS, ATOM, or REST to syndicate data to the iPhone is fairly simple, but you will have to write a parser. It is far easier to do using XML property lists or *plists*. According to

the manual page for property lists (see [Resources](#)): "Property lists organize data into named values and lists of values using several Core Foundation types: CFString, CFNumber, CFBoolean, CFDate, CFData, CFArray, and CFDictionary. These types give you the means to produce data that is meaningfully structured, transportable, storable, and accessible, but still as efficient as possible."

plists remove the headaches of parsing XML on the iPhone because they are an XML file format that Cocoa Touch will parse and convert into meaningful objects for you. Using the plist library in Python on App Engine, you can send most any simple Python dictionary object to the iPhone with minimal effort, provided the data types in the Python dictionary are those allowed in a plist. This article demonstrates this technique using an open source library called TouchEngine to develop an application to view sonnets by William Shakespeare. See [Resources](#) for a link to the project on Google Code.

Background

First, let's start with some background information on the iPhone SDK and Google App Engine.

iPhone SDK

The Native iPhone SDK is available through the Objective-C language. While quite similar to Cocoa programming on Mac OS X®, it includes APIs that exploit the unique features of the iPhone, such as GPS, multi-touch, the accelerometer, and the on-screen keyboard. The future roadmap includes support for technologies such as push notification. You can find more information about the iPhone Native SDK in [Resources](#).

For mobile app developers, the iPhone offers a rich development environment. Objective-C, until recently a fairly esoteric language for many developers, as it was almost exclusively used at NeXT and Apple, has gained a wider following through the Cocoa Touch SDK. With the iPhone, Objective-C becomes front and center to a whole new generation of mobile app developers.

What is Google App Engine?

Cloud computing has enjoyed a recent boost in visibility with Amazon's offerings of S3 storage and EC2 elastic computing services. A new player in the service-based cloud computing market is Google App Engine. Google App Engine offers an API in Python (other languages are forthcoming) to Google's famously scalable data center. This is a remarkable game changer that lets software developers ignore the inherent complexity of managing the scalability of an application, and lets them focus on writing applications.

Generating plist files from Google App Engine

To get started, take a look at how to generate plist files from Google App Engine, which you will later consume on the iPhone using the iPhone Cocoa Touch SDK. Because App Engine is initially free, it can serve as an interesting prototyping option for mobile app developers who need to solve a problem. In addition, the API is in Python, which has a reputation for fast development; it is a highly productive interpreted language. Outsourcing the heavy lifting and data storage of your iPhone application to the cloud, through App Engine and Python, can be a very rewarding experience.

To follow along, you will need to download the App Engine SDK. (See [Resources](#) to find the latest version.) Getting a prototype working in just a few minutes is simple with App Engine. Note that you can also download this example from the source code distributed with the article.

In order to serve plist files for the iPhone application to consume, you simply need to include `plistlib.py` from the App Engine project directory, slightly modify the `main.py` script, and also include `sonnet.py`. `Sonnet.py` is a Python source file that contains a dictionary with the text of all Shakespearean sonnets. Listing 1 shows the `main.py` file.

Listing 1. main.py

```
#!/usr/bin/env python
#Python sonnet maker

import wsgiref.handlers
from google.appengine.ext import webapp
from google.appengine.ext.webapp.util import run_wsgi_app

#external imports
import sonnet
import plistlib

class MainHandler(webapp.RequestHandler):
    """Returns sonnets dictionary as a converted plist"""
    def get(self):
        plist = plistlib.writeplistToString(sonnet.verses)
        self.response.out.write(plist)

def main():
    application = webapp.WSGIApplication([('/plists/sonnets', MainHandler),
                                         ],
                                         debug=True)

    run_wsgi_app(application)

if __name__ == '__main__':
    main()
```

This small code snippet takes the contents of the dictionary containing Shakespeare's sonnets, converts it to an XML plist, and serves it to any client that requests the `/plists/sonnets` URL. Believe it or not, that's the bulk of our Google App Engine Application. Listing 2 shows a small portion of `sonnet.py`.

Listing 2. Sample of sonnet.py

```

        verses={"verses":[[ "I", "" "FROM fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the ripper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light'st flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel.
Thou that art now the world's fresh ornament
And only herald to the gaudy spring,
Within thine own bud buriest thy content
And, tender churl, makest waste in niggarding.
    Pity the world, or else this glutton be,
    To eat the world's due, by the grave and thee." "" ]}]

```

[NOTE: EDITED FOR SPACE]

The `main` function routes the URL `/plists/sonnets` to the class `MainHandler`. If the client requests data through a HTTP `GET`, it will return a result that looks something like Listing 3.

Listing 3. HTTP Get results

```

        <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD plist 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>verses</key>
    <array>
        <array>
            <string>I</string>
            <string>FROM fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the ripper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light'st flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel.
Thou that art now the world's fresh ornament
And only herald to the gaudy spring,
Within thine own bud buriest thy content
And, tender churl, makest waste in niggarding.
    Pity the world, or else this glutton be,
    To eat the world's due, by the grave and thee.</string>
        </array>
    </array>

```

[NOTE: EDITED FOR SPACE]

You can take a look at this output yourself if you go to the [isonnet project page](#) (see [Resources](#) for a link). This URL gives you an edited, complete plist representation of Shakespeare's sonnets. Note, however, that your browser may display this as a giant flat text file. An XML plist is valid XML and most browsers will try to display it. View the source of the page to see the formatted plist.

Take a look at [Resources](#) for more detailed Python and App Engine examples and

links to deep tutorials on Google App Engine. But, for now, let's look at how an iPhone app ingests this plist data to change the application data.

Creating an iPhone application that dynamically reads and caches XML plist files from Google App Engine

TouchEngine includes a set of objects that make downloading and caching XML plists on the iPhone a simple affair. The object that we'll use to download and cache the sonnet plist is described in its header file, GRplistController.h, which is shown in Listing 4.

Listing 4. GRplistController.h

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
#import "Reachability.h"
#import "GRplistControllerDelegateProtocol.h"

typedef enum {
    kGRplistDownloadCannotInitiate = 0,
    kGRplistConnectionFailure,
    kGRplistFileFormatFailure
} GRplisErrorCode;

#define GR_ERROR_DOMAIN @"GRplistController_Error_Domain"

@class GRplistModel;

// This class will grab a remote plist from the server whenever update
// is called.

// Also registers with the Reachability object for notifications when
// the remote host changes availability and updates accordingly
// asking the delegate first if downloading new data is desirable.

@interface GRplistController : NSObject {
    NSURL *remoteURL;
    NSObject <GRplistControllerDelegate> *delegate;

    NetworkStatus remoteHostStatus;
    NetworkStatus internetConnectionStatus;
    NetworkStatus localWiFiConnectionStatus;
    BOOL loadingData;
@private
    GRplistModel *_model;
    BOOL hostIsReachable;
    NSMutableDictionary *plistIndex;
    NSMutableData *receivedData;
    NSURLConnection *connection;
}

@property(n nonatomic, retain)NSURL *remoteURL;
@property(n nonatomic, assign)NSObject *delegate;
@property(n nonatomic, readonly)GRplistModel *model;

//date of last download
@property(n nonatomic, retain)NSDate *lastUpdate;
@property(n nonatomic, getter=isLoadingData)BOOL loadingData;
@property NetworkStatus remoteHostStatus;
@property NetworkStatus internetConnectionStatus;
@property NetworkStatus localWiFiConnectionStatus;

//designated initializer
- (id)initWithRemoteURL:(NSURL *)aRemoteURL;
```

```
- (void)updateDataFromDisk;  
- (void)download;  
- (void)cancelDownload;  
@end
```

To use this class, we provide an `NSURL`, where an XML plist resides, and tell it to download or pull data from the disk. `GRplistController` generates a plist dictionary file in the user's data storage directory to store the locations and last download dates of cached plists keyed to the remote URL. After a plist has been downloaded once, subsequent data loads can be made directly from the disk. You can also query the `GRplistController` object regarding the date and time of the last download for a given URL using the `lastUpdate` property and decide when to try to refresh data from the Web. `GRplistController` always downloads data asynchronously using `NSURLConnection`, so the user interface will not freeze up while waiting for new data. If new data is not available or accessible, you can continue to use cached data until new data becomes available and is fully downloaded. You can also set an object as a delegate of `GRplistController`, which allows for fine-grained control of data downloading, provides notification of the arrival of updated data, and provides detailed error reporting when remote data proves inaccessible. The delegate methods for `GRplistController` are defined in `GRplistControllerDelegateProtocol.h`, shown in Listing 5.

Listing 5. `GRplistControllerDelegateProtocol.h`

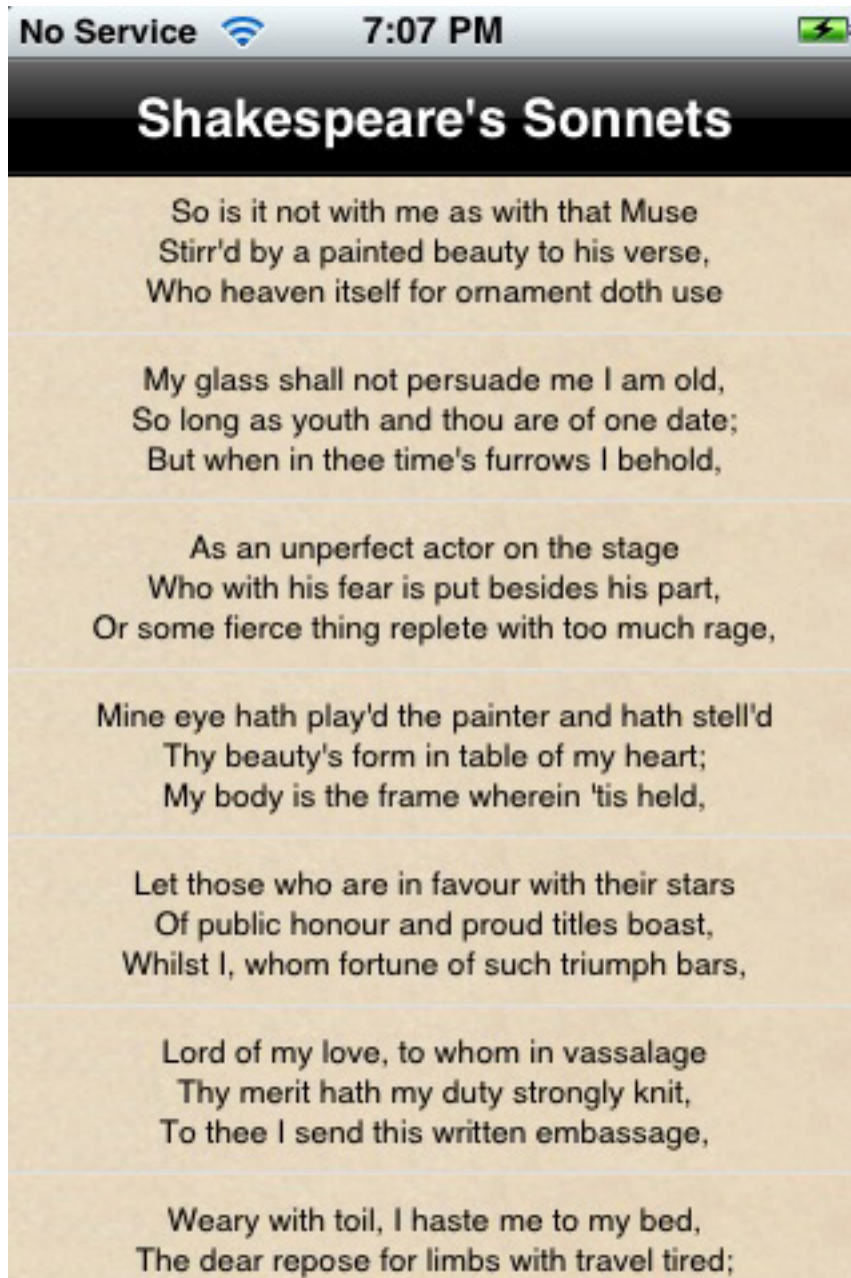
```
@class GRplistController;  
  
@protocol GRplistControllerDelegate  
  
@optional  
  
// the list controller will automatically try to update data when the network status  
// changes, so it's asking permission.  
- (BOOL)listControllerShouldDownloadRemoteData:(GRplistController *)listController;  
- (void)listController:(GRplistController *)  
listController downloadDidFailWithError:(NSError *)err;  
  
// if the data from the server has changed...  
- (void)listControllerDataWillChange:(GRplistController *)listController;  
- (void)listControllerDataDidChange:(GRplistController *)listController;  
  
@end
```

Our iPhone demo application, *Sonnet*, the source code for which is included in the [download](#) for this article, pulls all of Shakespeare's sonnets from our project on the App Engine servers. This lets us upload corrections to the sonnets occasionally (typos, inaccuracies, and so on) that would normally require a recompile and an application update if the data shipped with the application. We like to update common application data from time to time, but prefer to avoid an application recompile because our UI has not changed. This separates application data updates from feature additions and bug fixes. Furthermore, the user can always have the latest version of our data in hand by simply starting the application while connected to the Internet, rather than waiting for our application update to appear in the iPhone

application store, which can take some time.

The user interface for Sonnet is quite simple. The application first loads a `RootViewController` with a `UITableView`, which displays the first three lines of each sonnet from any available cached data immediately, and later displays any updated data.

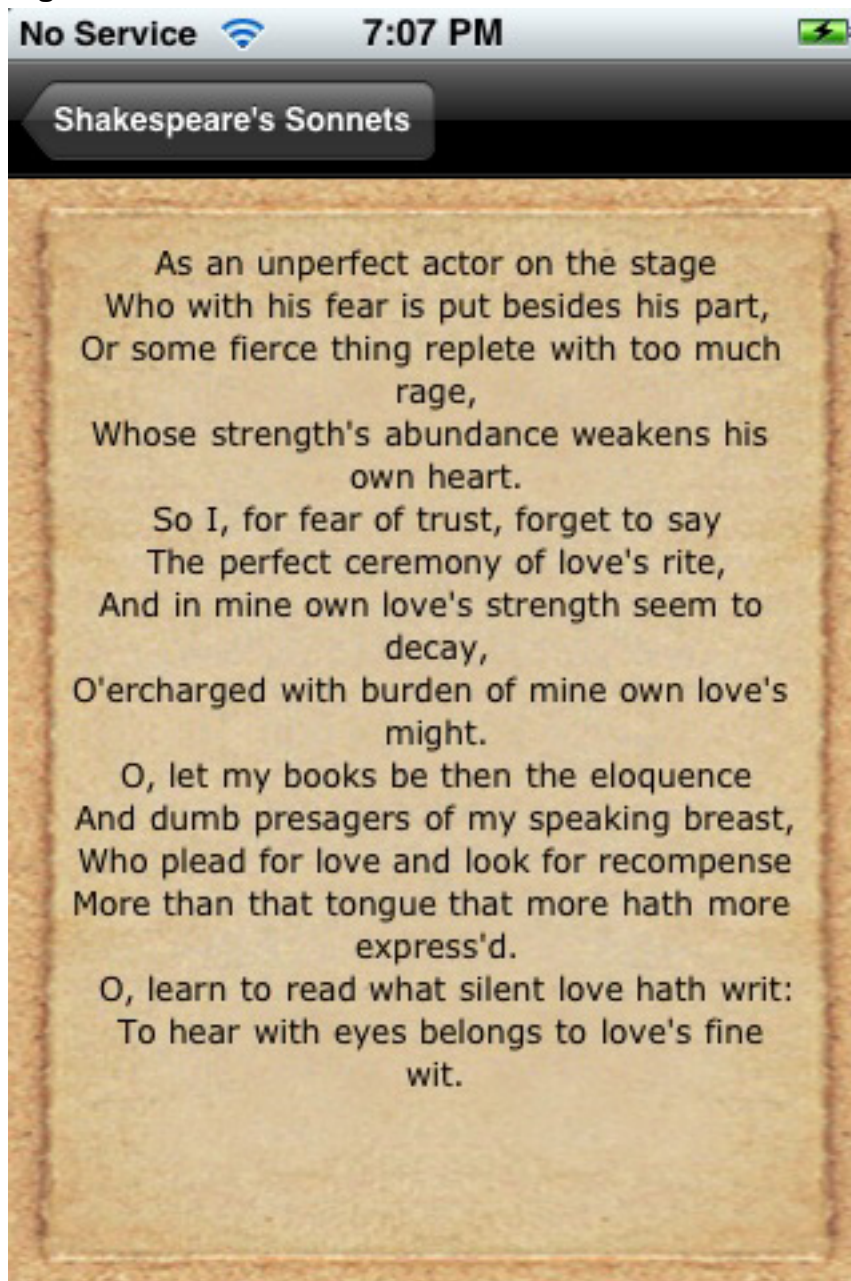
Figure 1. Table view iSonnet



When the user touches a cell in this table view, the `RootViewController` pushes `GRSonnetViewController` onto the screen. The `GRSonnetViewController`

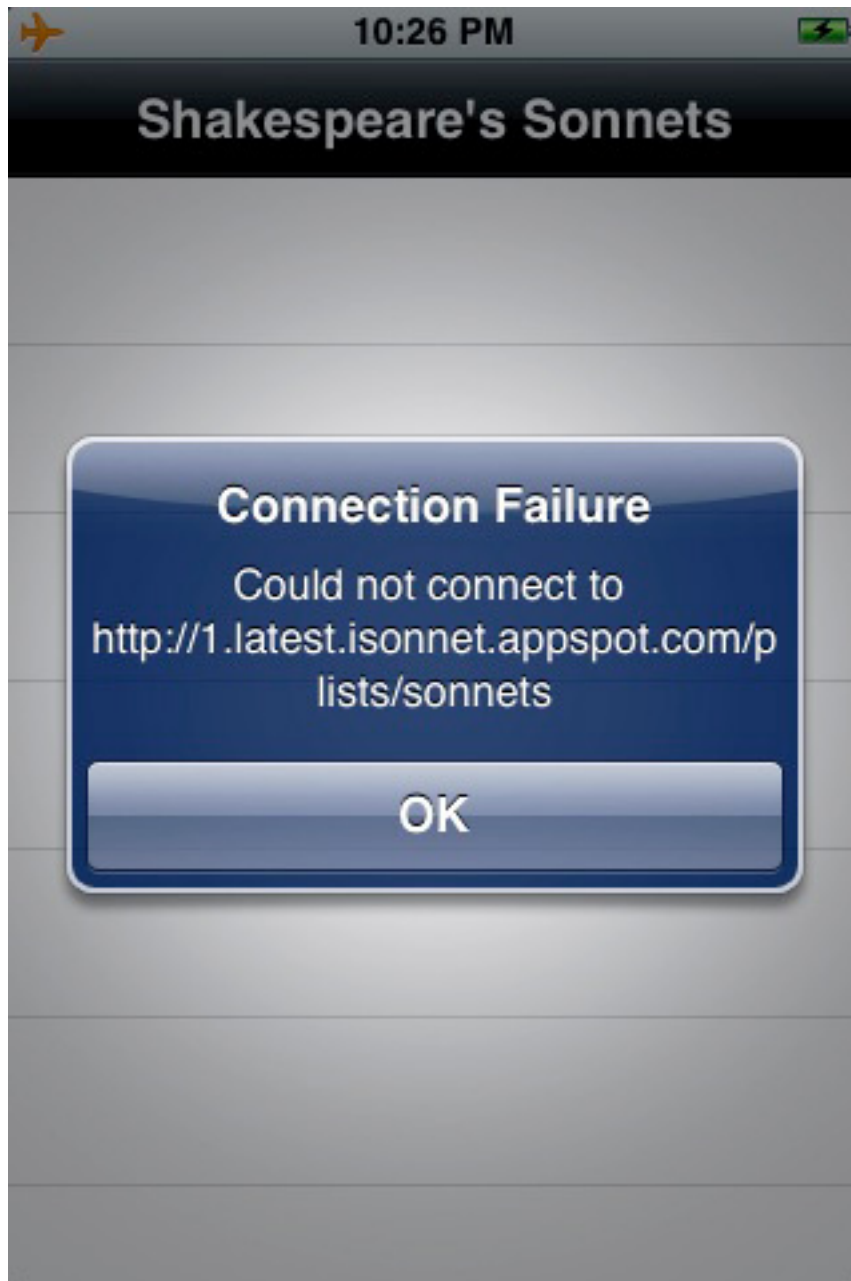
then displays the entire corresponding sonnet.

Figure 2. Sonnet view



On first run, the application has no data to show the user, so it is best to show an error if you are trying to pull data from the App Engine servers, but can't. The user would otherwise be staring at a blank table view. (The shipping version of the app will include a version of the data in the application bundle, but the demo does not.)

Figure 3. Showing an error



This is the only time that the application will show a network communication error to the user because the application will have downloaded and cached data under all other uses. You can look at the [sample code](#) for UI implementation details.

In *Sonnet*, the `RootViewController` object acts as a delegate for the `GRPlistController`. The initialization code for `RootViewController` is shown in Listing 6.

Listing 6 - `RootViewController` initialization code

```
(void)viewDidLoad {
```

```
[super viewDidLoad];
// Add the following line if you want the list to be editable
NSString *rootURL = [[self class] defaultURL];
self.sonnetsController = [[[GRplistController alloc]
    initWithRemoteURL:[NSURL URLWithString:rootURL]] autorelease];
self.sonnetsController.delegate = self;

[self.sonnetsController updateDataFromDisk];
self.sonnets = nil;
[self updateSonnetsFromModel];

//hit the web for new information
[self updateSonnets];
}
```

When Sonnet starts, the `RootViewController` first attempts to load cached plist data from the disk. If cached data is available, it is immediately loaded into the application so that the application becomes usable very quickly. After any cached data loads, the application queries the App Engine site for new data asynchronously. (See `GRplistController.m` below the "#pragma mark Downloading of data" line for the `NSURLConnection` implementation details). If there is new data, it is compared to the cached data and, if changed, the `GRplistController` notifies the `RootViewController` through the `listControllerDataDidChange` method. The `RootViewController` then reloads the table of sonnets with the new data.

Conclusion

Mixing App Engine with iPhone development is a powerful tool for writing and prototyping mobile apps. This sample Web application powers a small iPhone application, Sonnet, that should be available as a free download in Apple's Application Store. The power of combining Google App Engine with iPhone development comes from the option to rapidly prototype parts of an application in Python, which otherwise would be tedious to code in Objective-C, coupled with enhanced flexibility in the storage of data, both online and offline. TouchEngine offers the best of multiple worlds. Developers of hybrid applications can easily write software using TouchEngine that caches data locally on the iPhone while asynchronously updating data from the cloud. This lets the application respond quickly to user input while maintaining data online for timely updates.

Both the iPhone and the Google App Engine developer communities offer a rich set of resources to get up to speed quickly. If you are interested in developing for either platform or combining both offerings as you have seen in this article, we suggest that you go through some of the official documentation. Both Google and Apple have great written tutorials, and both have video-based tutorials that are exceptional. Google App Engine holds "Hack-A-Thon" events all over the world, so you may want to check with Google to see if there is an event in your area. Apple's WWDC conference has also proven an invaluable resource for many iPhone SDK programmers.

Downloads

Description	Name	Size	Download method
Sample iPhone and Google App Engine Code	iPhone_Google_App_Engine_Code	201.9KB	HTTP iPhone_Google_App_Engine_Code.zip

[Information about download methods](#)

Resources

Learn

- The [manual page](#) for property lists can be found on the Apple Web site.
- The [isonnet project page](#) contains the verses from Shakespeare's sonnets.
- Explore the Google App Engine Example with "[Python For Unix and Linux System Administration](#)".
- The article "[Getting Started With Google App Engine O'Reilly Article](#)" gives information on Google's App Engine and then helps you build an application.
- [Google App Engine in Action](#) provides an introduction to building Web apps using the Google App Engine infrastructure.
- See how with [Google's GData APIs](#) you can get access to user-specific data stored in Google services.
- Learn the basic concepts and features of the Python language and system using this [Python Tutorial](#).
- Learn more about the [TouchEngine Open Source Project: Synch plist files to App Engine](#).
- Get an introduction to [learning Objective-C 2](#) with this Developer Connection article.

Get products and technologies

- Download the latest [App Engine SDK](#).
- Start your development with the [iPhone SDK](#).

About the authors

Noah Gift

[Noah Gift](#) is the co-author of [Python For Unix and Linux System Administration](#) by O'Reilly, and is also working on [Google App Engine In Action](#) for Manning. He is an author, speaker, consultant, and community leader, writing for publications such as IBM developerWorks, [Red Hat Magazine](#), [O'Reilly](#), and [MacTech](#). His consulting company's Web site is <http://www.giftcs.com>, and much of his writing can be found at <http://noahgift.com>. Noah has a master's degree in CIS from Cal State Los Angeles, a B.S. degree in Nutritional Science from Cal Poly San Luis Obispo, is an Apple and LPI certified SysAdmin, and has worked at companies such as, Caltech, Disney Feature Animation, Sony Imageworks, and Turner Studios. He is currently working at [Weta Digital](#) in New Zealand. In his free time he enjoys spending time with his wife,

Leah, and their son, Liam, composing for the piano, running marathons, and exercising religiously.

Jonathan Saggau

Jonathan Saggau is the founder and CEO of Sounds Broken inc, a Mac OS X and iPhone software contracting shop as well as a technical and business process consultancy. When he is not flying airplanes or reverse-engineering hardware and software, he works with clients such as Equity Audio, Innovative Audio, and the [Big Nerd Ranch](#) to develop outstanding products, services, and processes. He blogs at jonathansaggau.com/blog/ and can be followed on twitter [@jonmarimba](#).