Mobile and Cloud Application Development with Android SDK and GAE/Python

Islam Almusaly, Faezeh Bahmani, Sanchit Karve, Michael Tichenor
Oregon State University
Corvallis, OR 97331

SECTION 1(a): Installing and Configuring Google App Engine.

The cloud platform we will be using is the Google Application Engine/Python (GAE/P) SDK. This requires Python v2.7. Follow the steps below:

1) Go to http://www.python.org/getit/releases/2.7
2) Download the self-extracting executable or compressed file for your preferred development platform.
3) Run the self-extracting executable or extract the compressed file for your preferred development platform into your preferred development directory.
4) Go to https://developers.google.com/appengine/downloads
5) Click the link for Google App Engine SDK for Python and download the self-extracting executable or compressed file for your preferred development platform.
6) Install the GAE/P SDK executable, or extract the compressed file into your preferred development directory.
7) Run the GAE Launcher executable.

SECTION 2(b): Reading Data from the Browser + Saving Data to the GAE Data Store.

1) Inside the GAE Launcher, click File → Create New Application (Ctrl+N).
2) Specify the Application Name.
3) Click Browse and navigate to the directory you wish to do development in.

This will automatically create a folder for the application, the python script for the application, and the .yaml configuration file for the application. It will also automatically start up a server on localhost to perform development using GAE without having to upload the application to the Google servers.

4) Click Edit → Open In Explorer (Ctrl+Shift+E).
5) Open “main.py” file in your favorite text editor.

Datastore entities are created from Python objects and the object's attributes become properties for the entity. This is the main python script that is running on the web server to handle requests. Here's a simple python script that reads data from the browser using POST method, and stores the data on the GAE datastore. You can copy and paste this text into the “main.py” file.

```
import cgi
import webapp2
from google.appengine.ext import db
from google.appengine.api import users

# Create the GuestbookEntry class specifying database schema
class GuestbookEntry(db.Model):
    author = db.StringProperty()
    email = db.StringProperty()
    date = db.DateTimeProperty(auto_now_add=True)
    message = db.StringProperty(multiline=True)

# Define the HTMLbook class for the WSGIApplication '/' Request Handler
class HTMLbook(webapp2.RequestHandler):
    def get(self):
        self.response.out.write("""
        <!DOCTYPE HTML>
        <html>
        <title>Google App Engine for Python - Guestbook Demo</title>
        <body>
        <form action="sign" method="post" name="entry">
        Name: <input type="text" name="author" maxlength="100" />
        Email: <input type="text" name="email" maxlength="100" />
        Message: <textarea name="message" rows="2" cols="100"></textarea>
        <input type="submit" value="Sign Guestbook" />
        </form>
        <hr />
        """)
        #Display Guestbook Entries Here
        self.response.out.write("""
        </body>
        </html>
        """)

# Define the POSTbook class for the WSGIApplication '/sign' Request Handler
class POSTbook(webapp2.RequestHandler):
    def post(self):
       self.response.out.write('Author: %s' % self.request.get('author') )
       self.response.out.write('Email: %s' % self.request.get('email') )
       self.response.out.write('Message: %s' % self.request.get('message') )

       Entry = GuestbookEntry()
       Entry.author = self.request.get('author')
       Entry.email = self.request.get('email')
       Entry.message = self.request.get('message')
       Entry.put()
       self.redirect('/')

class XMLbook(webapp2.RequestHandler):
      def get(self):
      self.response.headers['Content-Type'] = 'text/xml'
      self.response.out.write("""
      <?xml version="1.0"?>
      <guestbook>
      """)
      GuestbookEntries = db.GqlQuery("SELECT * "
                                     "FROM GuestbookEntry "
```

```
                                                     "ORDER BY date DESC")

        for Entry in GuestbookEntries:
                self.response.out.write("""
        <entry>
                """)
                self.response.out.write('<author>%s</author>' % Entry.author )
                self.response.out.write('<email>%s</email>' % Entry.email )
                self.response.out.write('<date>%s</date>' % Entry.date )
                self.response.out.write('<message>%s</message>' % cgi.escape(
Entry.message ) )
                self.response.out.write("""
        </entry>
        """)
        self.response.out.write("""
</guestbook>
        """)

# Create the WSGIApplication object that handles GAE HTTP requests.
app = webapp2.WSGIApplication([('/', HTMLbook),('/xml', XMLbook),('/sign',
POSTbook)],debug=True)
```

6) Save the changes to the python script.
7) Click Browse in the Google App Engine Launcher.

SECTION 1(c): Retrieving Data from the GAE Data Store.

In order to retrieve data from the datastore, you need to execute a Google Query Language query (GqlQuery). The results are retrieved as an array, which are accessed individually in a for loop.

1) Replace “`#Display Guestbook Entries Here`” with the following code:

```
        GuestbookEntries = db.GqlQuery("SELECT * "
                                       "FROM GuestbookEntry "
                                       "ORDER BY date DESC")

        for Entry in GuestbookEntries:
            self.response.out.write('<a target="_blank"
href="mailto:%s">' % Entry.email )
            self.response.out.write('%s</a>' % Entry.author )
            self.response.out.write(' posted the following message
on %s:<br />' % Entry.date )
            self.response.out.write('%s' % cgi.escape( Entry.message
) )
            self.response.out.write('<hr />')
```

Now, if you know anything about python, you’ll realize that the code above is not correctly indented.

Because python does not have brackets to indicate function, it uses indenting to keep track of class and function hierarchy. You'll need to make sure that all of the above lines need to be moved to the ends of the preceding lines.
2) Save the changes to the python script.
3) Click Browse in the Google App Engine Launcher.

SECTION 2(a): Installing and Configuring Android Development Platform.

We'll be using the Android development platform (along with Eclipse IDE) on Windows to write mobile applications and communicate with the cloud. The instructions are identical for setting up the development platform on Linux and Mac OSX as long as the appropriate installers and archives (zip, tarballs, etc.) are downloaded and installed.

Note: The installation process is much shorter if you proceed with the installation wizard encountered while installing the ADT Plugin. However, it did not work every time we tested it, so we present a manual installation procedure that is guaranteed to work.

Part I: Setting up the Android SDK

1) Visit the Android SDK Page. http://developer.android.com/sdk/index.html
2) Download android-sdk_r17-windows.zip
3) Extract the contents of the zip archive anywhere on your file system.
4) Start the SDK Manager executable file from the android-sdk-windows folder.
5) You should see a window pop up with listings of android components.
Initially, only Android SDK Tools is installed by default. You need to add an Android Platform as well as some "platform tools" that help to transfer android applications from your computer to your phone or emulator.
6) Select "Android SDK Platform-Tools" and everything placed under "Android 4.0.3 (API 15)".
7) Click the "Install x packages..." button. (where x is the number of selected packages)
8) Accept the license agreements and proceed with the installation. Once everything is installed, the SDK Manager will refresh the package list. Confirm that all the relevant packages are installed and close the SDK Manager.

Part II: Setting up the Android Emulator

The SDK is set up but you need to specify some hardware information for a device that you'd like the emulator to simulate.

1) Run the AVD Manager executable from the Android SDK directory.
2) A window appears with a listing of all the AVDs that have been created so far.
There should be no AVDs listed in this window but there are any, the SDK Manager probably installed some default phone configurations for you, in which case you don't need to do anything else. Skip this section unless you want to create your own custom device configuration.

3) Click the "New..." button.
4) Enter any name for the "Name" field.
Choose ANY development platform for the "Target" field.
The emulated device needs to have some space allocated for its own memory. Type in any value greater than 100 MB for the "SD Card \ Size" field.
Enable the Snapshot field.
By default, WVGA800 should be selected as the built-in "Skin". If you'd like, you can enter a custom resolution for your phone.
You can add more specific hardware features by clicking the "New..." button but it's not required.

Click "Create AVD".
5) You should see a confirmation dialog. Dismiss it to see your new AVD listed in the AVD Manager window.
6) Select your AVD and click Start. Uncheck “Load from snapshot” and ensure that “Save to Snapshot” is checked.
Find something to do for the next 2-10 minutes as this can (and probably will) take a long time to boot up for the first time.
You should see a boot animation before the Android lock screen appears.
When it does, go back to the AVD Manager window, select your AVD and click the "Edit..." button.
Enable the Snapshot checkbox and save your settings by clicking "Edit AVD".
Now, close the emulator window. This will create a snapshot before closing the emulator so the next time you start up your AVD, it will boot up in a few seconds.

Part III: Setting up the ADT Plugin for Eclipse

1) Download Eclipse Classic 3.7.2 from here. http://www.eclipse.org/downloads/
2) Extract the eclipse ZIP file anywhere on your PC and run eclipse.exe
3) Once Eclipse has opened, click Help->Install New Software.
4) Click the "Add..." button and type this URL for the Location field. http://dl-ssl.google.com/android/eclipse
You can assign it any name you like. Click OK.
5) The android repository should now be selected by the "Work with" dropdown box. If it's not, select it manually. You should now see a "Developer Tools" checkbox appear under the "Name" tab. Select it and click Next.
6) Proceed with the ADT Plugin installation.
7) After ADT is installed, it will ask you to select an option to install the Android SDK automatically or specify the location of an SDK that's already set up.
Select the Android SDK folder from your hard drive and proceed with the wizard.

SECTION 2(b): Retrieving an XML Document from the Cloud.

Note: Development on Android can be a lot different than programming for a platform that you’re used to. If you’re familiar with Java or C++, you should be able to understand the code as the code is highly documented. However, it is recommended that you read the Android Programming Notes located at the end of this document before continuing further.

1) In Eclipse IDE, click File→New→Project.
2) Select Android→Android Project and click Next.
3) Enter a Project Name (E.g.: CloudXMLTest) and click Next.
4) Choose any Build Target and click Next. (Example: Android 4.0.3)
5) Enter your Package Name (E.g.: com.sanchitkarve.cloudxmltest). and click Next. All the project files should now be created.
6) In the Package Explorer Tab, navigate to Project Name (CloudXMLTest) → res → layout and open main.xml.
7) Switch from the Graphical Layout to the main.xml tab.
8) Replace the pre-existing XML content with the following code and save the file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical" >
 <TextView
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/firstTextViewText" />

 <Button
  android:id="@+id/btnGetXML"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="@string/buttonText" />

 <ScrollView android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:paddingTop="20dp" >

 <TextView
  android:id="@+id/lblXmlResult"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:gravity="center"
  android:text="@string/resultTextViewText" />
 </ScrollView>

</LinearLayout>
```

9) Navigate to Project Name (CloudXMLTest) → res → values and open strings.xml.
10) Switch from the resources tab to strings.xml and replace the pre-existing XML content with the following code and save the file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">CloudXMLTest</string>
  <string name="firstTextViewText">Click the button to download the XML
file.</string>
  <string name="buttonText">Get XML From Cloud.</string>
  <string name="resultTextViewText">Parsed XML appears here.</string>
</resources>
```

11) Navigate to Project Name (CloudXMLTest) → src → Namespace (com.sanchitkarve.cloudxmltest) and open the Activity source file (CloudXMLTestActivity.java).
12) Replace the pre-existing code with the following Java code and save the file.

```
package com.sanchitkarve.cloudxmltest;

import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class CloudXMLTestActivity extends Activity {

  private Button btnGetXML;
  private TextView lblXmlResult;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    // Call Parent constructor.
    super.onCreate(savedInstanceState);
    // Compile UI from main.xml and set it as the UI for this activity.
    setContentView(R.layout.main);
    // Get a reference to the Button from main.xml
    btnGetXML = (Button)findViewById(R.id.btnGetXML);
    // Add a onClickListner to the button.
    btnGetXML.setOnClickListener(btnGetXMLOnClick);
    // Get a reference to the TextView from main.xml
    lblXmlResult = (TextView)findViewById(R.id.lblXmlResult);
  }

  /** Used as the onClickListener for the button. */
  View.OnClickListener btnGetXMLOnClick = new View.OnClickListener() {

    @Override
    public void onClick(View v)
    {
      // Display a "toast" notification to inform the user that the file is
being downloaded.
Toast.makeText(getBaseContext(), "Downloading XML File",
Toast.LENGTH_SHORT).show();
      // Clear the text present in lblXmlResult TextView.
      lblXmlResult.setText("");
      // Download and Parse the XML present at the cloud's XML web service
end-point.
      new CloudAccessTask().execute("http://fantastic4guestbook.appspot.com/
xml");
    }
  };

  /** Opens a connection to a URL and returns an InputStream from the
connection. */
  private InputStream OpenHttpConnection(String urlString) throws
IOException {
      InputStream in = null;
      int response = -1;

      // Open connection to URL
      URL url = new URL(urlString);
      URLConnection conn = url.openConnection();

      // Throw exception if connection is not valid HTTP.
      if (!(conn instanceof HttpURLConnection)) {
      throw new IOException("Not an HTTP connection");
      }

      try {
      // Set the appropriate request headers and make the request.
      HttpURLConnection httpConn = (HttpURLConnection) conn;
      httpConn.setAllowUserInteraction(false);
      httpConn.setInstanceFollowRedirects(true);
      httpConn.setRequestMethod("GET");
      httpConn.connect();

      // Get Response from HTTP Request.
      response = httpConn.getResponseCode();
      // Get InputStream only if response is OK else throw exception.
      if (response == HttpURLConnection.HTTP_OK) {
            in = httpConn.getInputStream();
      }
      }
      catch (Exception ex) {
      throw new IOException(ex.getMessage());
      }
      // Return InputStream
      return in;
  }

  /** Retrieves XML from Cloud End-point, parses it using DOM and returns a
string with formatted data. */
  public String GetAndParseXMLFromCloud(String url) {
    InputStream in = null;
    String result = "";
    try  {
      // Get InputStream from URL
      in = OpenHttpConnection(url);
      // Set up DOM objects
      Document doc = null;
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = null;
      try  {
        // Create an instance of DocumentBuilder
        db = dbf.newDocumentBuilder();
        // Parse XML from InputStream
        doc = db.parse(in);
      }
      catch(Exception e) {
        e.printStackTrace();
      }
      // Normalizes XML file (i.e., places all text-based attribute nodes in
the full sub-tree.)
```

```
      // Not necessary in HowTo example as all elements are child nodes, but
it's good practice to normalize it anyway.
      doc.getDocumentElement().normalize();

      // #INSERT-CODE#

    }
    catch(Exception e) {
      e.printStackTrace();
    }
    // Return formatted string
    return result;
  }

/** Class used to download and parse XML from the cloud server
asynchronously. */
  private class CloudAccessTask extends AsyncTask<String, Void, String> {

    /** Called when CloudAccessTask::execute() is called. Returns formatted
XML. */
    @Override
    protected String doInBackground(String... urls)  {
      return GetAndParseXMLFromCloud(urls[0]);
    }

    /** Called when the background task is complete. */
    @Override
    protected void onPostExecute(String result) {
      // Display a "toast" notification to inform the user that the XML file
has been downloaded and parsed.
Toast.makeText(getBaseContext(), "XML File Parsed",
Toast.LENGTH_LONG).show();
      // Sets the result string as the text of the lblXmlResult TextView.
      lblXmlResult.setText(result);
    }

  }
}
```

13) Open AndroidManifest.xml located under the Project folder.
14) Switch to the AndroidManifest.xml tab.
15) Insert a uses-permission tag with an INTERNET Permission under the uses-sdk tag so that AndroidManifest.xml looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ...
  ... >

  <uses-sdk android:minSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET"/>

  ...

</manifest>
```

SECTION 2(c): Parsing XML to Display Some Elements to the User.
Android provides three parsers to parse XML. We have chosen to parse XML using the DOM parser so that those not familiar with Android or Java can understand the code more easily due to its similarity with Javascript DOM methods. Alternatively, the SAX Parser or the XMLPull Parser can also be used.

1) Find the `#INSERT-CODE#` comment in the java file and replace it with the following code.

```
       // Get a List of all "entry" elements.
       NodeList contactsElements = doc.getElementsByTagName("entry");
       // For each entry element,
       for(int i=0; i < contactsElements.getLength(); i++)  {
         // Get the ith Entry.
         Element personElement = (Element)contactsElements.item(i);
         // Get the Email from the ith element
String email =
personElement.getElementsByTagName("email").item(0).getChildNodes().item(0).
getNodeValue();
         // Only consider email addresses that end with @gmail.com
         if(email.toLowerCase().endsWith("@gmail.com"))
         {
           // Get the Author, Date and Message for the ith elements.
String author =
personElement.getElementsByTagName("author").item(0).getChildNodes().item(0)
.getNodeValue();
String date =
personElement.getElementsByTagName("date").item(0).getChildNodes().item(0).g
etNodeValue();
String message =
personElement.getElementsByTagName("message").item(0).getChildNodes().item(0
).getNodeValue();
           //Append values from the "author", "email", "date" and "message"
elements into a result string.
           result += "Element #" + String.valueOf(i+1) + "\n";
           result += "Author : " + author + ".\n";
           result += "Email : " + email + ".\n";
           result += "Date : " + date + ".\n";
           result += "Message : " + message + ".\n\n";
         }
       }
```

2) Run the program by clicking Run → Run or by pressing Ctrl + F11.
3) Select Run As “Android Application” when prompted.
4) The Android Emulator should open in a few seconds and run the application within the emulator.

SECTION 3: Individual Contributions to this Document.

Section 1 was written by Michael, working with Sanchit and Faezah. Section 2 was written by Sanchit. Presentation on Section 1 will be given by Islam and Michael. Presentation on Section 2 will be given by Sanchit and Faezah.

ANDROID PROGRAMMING NOTES FOR C / C++ / JAVA DEVELOPERS:

1) All Android (Java) applications require the following files/directories:

- AndroidManifest.xml: Contains information about the application including the permissions needed for the application.
- res\drawable-xxx: All images used need to be placed in an appropriate directory for the target phone. Images for phones with High-Density Screens should be placed in drawable-hdpi, low-density screens in drawable-ldpi, medium-density screens in drawable-mdpi and extra-high-density screens (tablets) in drawable-xhdpi.
- The UI for an Android application is written in XML and is stored under the res → layout directory.

2) Information about Android UI and string resources:

- Although not enforced, literals should be defined in strings.xml located in res → values, and referenced from the UI XML files using "@string/id". This is recommended for localization.
- All ids specified in XML files are automatically included in R.java and need to be referenced from R.java. (For example, a button with id btnGetXML is accessible through R.id.btnGetXML)
- UI Controls or Widgets are known as Views in Android.
- Labels and TextBoxes are referred to as TextView and EditText in Android.
- Layouts are containers that house multiple UI widgets.
- A Window (in Desktop programming) is referred to as an Activity in Android. A class that displays UI needs to inherit from the Activity Class.

3) For the main Activity, the onCreate() function is called every time the application is opened.
4) setContentView() compiles and sets an XML UI file as the UI for the current activity.
5) findViewById() returns a View Object from an XML UI file.
6) The latest Android platform (4.x) requires all time-consuming tasks to be carried out in an Asynchronous thread.
7) A Toast is a non-intrusive notification that is displayed at the bottom of the screen by default.
