:: INSTALLATION AND GETTING STARTED WITH GOOGLE APP ENGINE USING ECLIPSE (java IDE) ::

by Rohan Banerjee and Rohit Prakash
rohanb@cs.uoregon.edu
rohitp@cs.uoregon.edu
University of Oregon.

Download and Install Eclipse:

[http://www.eclipse.org/downloads/]

Eclipse Classic 3.7.1 is the recommended version.

Download SDK:

[http://developer.apple.com/search/index.php?q=java]

Things to do to get started:

- create an App Engine Java project with Eclipse, and without
- use the Google Plugin for Eclipse for App Engine development
- use the App Engine datastore with the Java Data Objects (JDO) standard interface
- integrate an App Engine application with Google Accounts for user authentication
- upload your app to App Engine

[http://code.google.com/appengine/docs/java/gettingstarted/introduction.html]

You develop and upload Java applications for Google App Engine using the App Engine Java software development kit (SDK).

The SDK includes software for a web server that you can run on your own computer to test your Java applications. The server simulates all of the App Engine services, including a local version of the datastore, Google Accounts, and the ability to fetch URLs and send email from your computer using the App Engine APIs.

```
MacBook-Pro:~ rohanbanerjee$ javac -version
javac 1.6.0_29
```

Install the Google Plugin for Eclipse using the Software Update feature of Eclipse:

[http://code.google.com/appengine/docs/java/tools/eclipse.html]

To install the plugin, using **Eclipse 3.7 (Indigo)**:

1. Select the **Help** menu > **Install New Software...**
2. In the **Work with** text box, enter: `http://dl.google.com/eclipse/plugin/3.7`
   Click the **Add...** button. In the dialog that shows, click **OK** (keep the name blank, it will be retrieved from the update site.)
3. Click the plus icon next to "Google Plugin for Eclipse" and "SDKs". Check the boxes next to "Google Plugin for Eclipse 3.7" and "Google App Engine Java SDK". You can also select the "Google Web Toolkit SDK" if you'd like to use Google Web Toolkit with your apps. Click the **Next** button. Follow the prompts to accept the terms of service and install the plugin.
4. When the installation is complete, Eclipse prompts you to restart. Click **Yes**. Eclipse restarts. The plugin is installed.

To create a new App Engine project:

1. Select the **File** menu > **New** > **Web Application Project** *(If you do not see this menu option, select the **Window** menu > **Reset Perspective...**, click **OK**, then try the **File** menu again.)* Alternatively, click the New Web Application Project button in the toolbar: 
2. The "Create a Web Application Project" wizard opens. For "Project name," enter a name for your project, such as `Test 1` for the project described in the Getting Started Guide. For "Package," enter an appropriate package name, such as `testing`.
3. If you're not using Google Web Toolkit, uncheck "Use Google Web Toolkit." Verify that "Use Google App Engine" is checked.
4. If you installed the App Engine SDK using Software Update, the plugin is already configured to use the SDKs that were installed. If you would like to use a separate installation of the App Engine SDK, click **Configure SDKs...**, and follow the prompts to add a configuration with your SDK's `appengine-java-sdk/` directory.
5. Click **Finish** to create the project.

Running the Project:

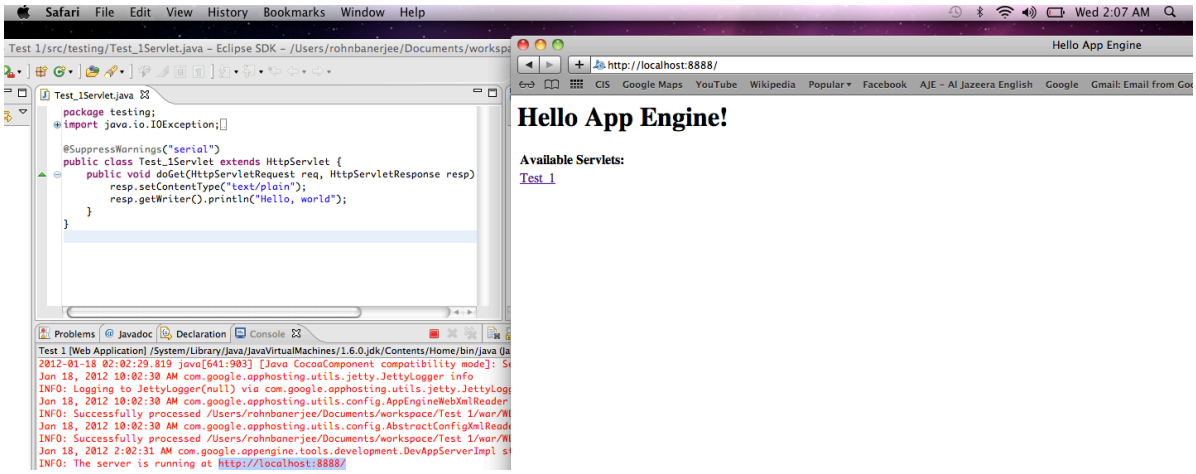
The App Engine SDK includes a web server for testing your application in a simulated environment. The Google Plugin for Eclipse adds new items to the **Run** menu for starting this server.
To run your application in the web server inside the Eclipse debugger, select the **Run** menu, **Debug As** > **Web Application**. Eclipse builds the project, switches to the Debug perspective, and the server starts. If the server starts successfully, the server displays several messages, including a message similar to the following, in the Console:

```
The server is running at http://localhost:8888/
```

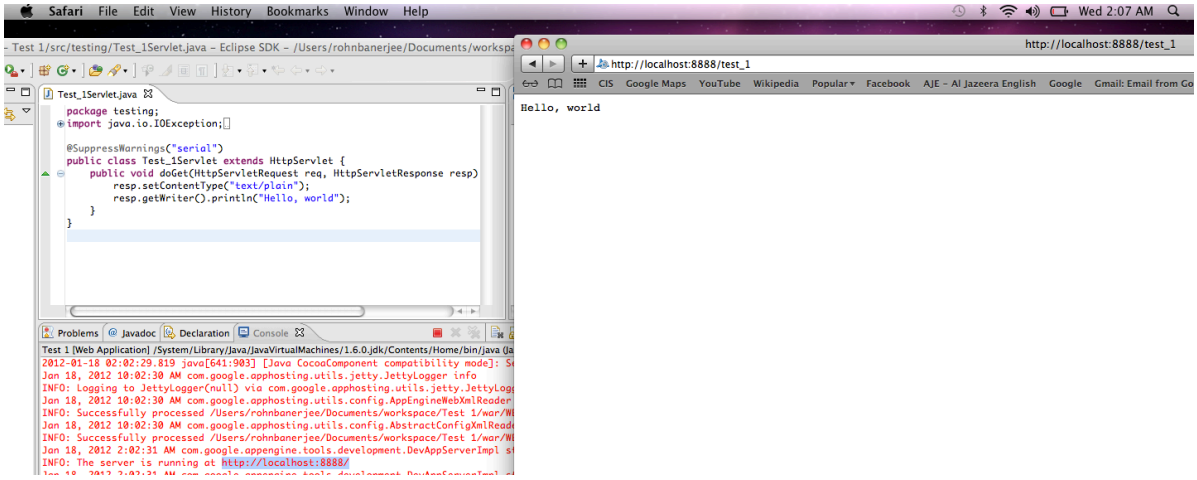
If you want to customize how the server is started, you can create a new Run/Debug configuration of the type "Web Application."

To test the new application that the plugin created, start the server as above, then visit the following URL in your browser (using a URL path appropriate to your application): -

- http://localhost:8888/



- http://localhost:8888/test_1



:: INSTALLATION AND GETTING STARTED WITH GOOGLE APP ENGINE ::

Create and manage App Engine web applications from the App Engine Administration Console, at the following URL:

https://appengine.google.com/

Follow the instructions (prompted).

Uploading an Application to appspot.com (online rather than being only locally accessible)**:**

- ***From Eclipse***: If you are using Eclipse and the Google Plugin, you can upload your application directly from within Eclipse. To upload the app, click the App Engine deploy button in the toolbar:

    + The following URL is where an uploaded app can be accessed at - http://your_app_id.appspot.com/

:: Android Development with Eclipse (java IDE) ::

Downloading the ADT Plugin:

Use the Update Manager feature of your Eclipse installation to install the latest revision of ADT on your development computer.<>

Assuming that you have a compatible version of the Eclipse IDE installed, as described in Preparing for Installation, above, follow these steps to download the ADT plugin and install it in your Eclipse environment.

1. Start Eclipse, then select **Help** > **Install New Software...**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*: **https**://dl-ssl.google.com/android/eclipse/
4. Click **OK** Note: If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**. Note: If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
8. When the installation completes, restart Eclipse.

Configuring the ADT Plugin:

After you've successfully downloaded the ADT as described above, the next step is to modify your ADT preferences in Eclipse to point to the Android SDK directory:

1. Select **Window** > **Preferences...** to open the Preferences panel (Mac OS X: **Eclipse** > **Preferences**).
2. Select **Android** from the left panel.

3. You may see a dialog asking whether you want to send usage statistics to Google. If so, make your choice and click **Proceed**. You cannot continue with this procedure until you click **Proceed**.
4. For the *SDK Location* in the main panel, click **Browse...** and locate your downloaded SDK directory.
5. Click **Apply**, then **OK**.

Create an AVD:

Before launching the emulator, create an Android Virtual Device (AVD). An AVD defines the system image and device settings used by the emulator.

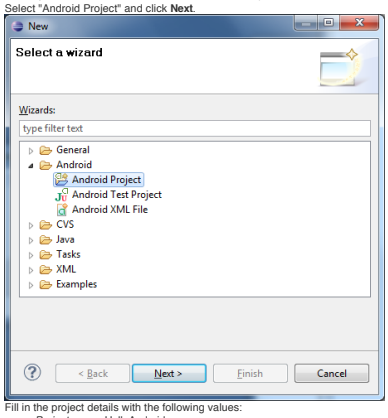
To create an AVD:

1. In Eclipse, select **Window > AVD Manager**.
2. Click **New...**. The **Create New AVD** dialog appears.
3. Type the name of the AVD, such as "my_avd".
4. Choose a target. The target is the platform (that is, the version of the Android SDK, such as 2.3.3) you want to run on the emulator. For this tutorial, choose the latest platform that you have installed and ignore the rest of the fields.
5. Click **Create AVD**.

Create a New Android Project:

After you've created an AVD you can move to the next step and start a new Android project in Eclipse.

1. In Eclipse, select **File > New > Project...**. If the ADT Plugin for Eclipse has been successfully installed, the resulting dialog should have a folder labeled "Android" which should contain "Android Project". (After you create one or more Android projects, an entry for "Android XML File" will also be available.)
2. Select "Android Project" and click **Next**.

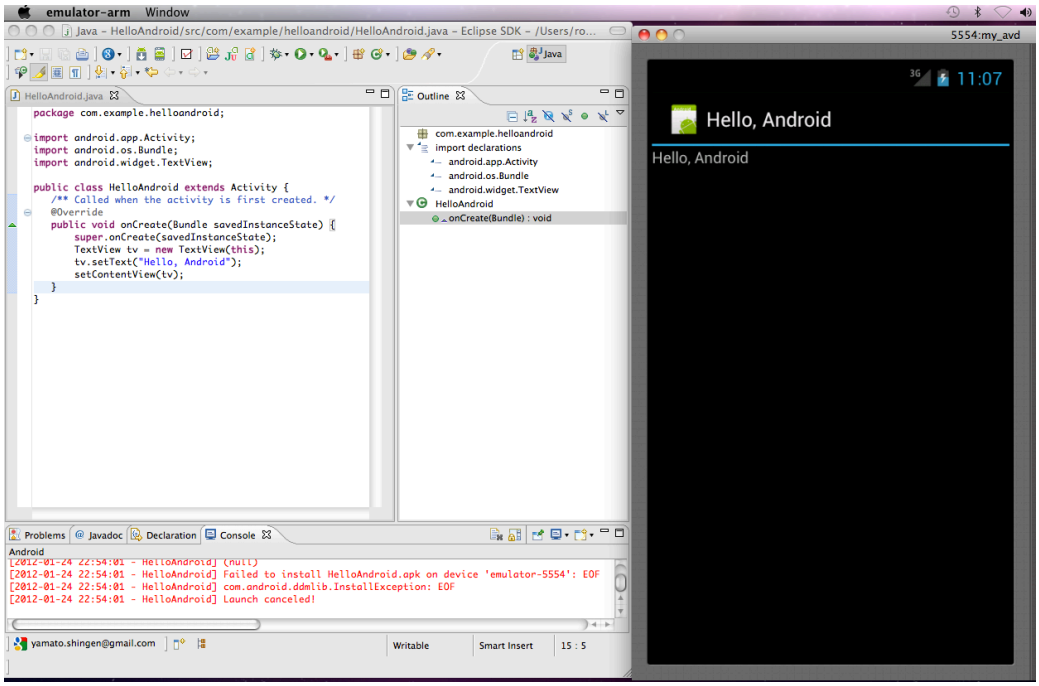


3. Fill in the project details with the following values:
   - *Project name:* HelloAndroid
   - *Build Target:* Select a platform version that is equal to or lower than the target you chose for your AVD.
   - *Application name:* Hello, Android
   - *Package name:* com.example.helloandroid (or your own private namespace)
   - *Create Activity:* HelloAndroid
4. Click **Finish**.

Run the Application:

The Eclipse plugin makes it easy to run your applications:

1. Select **Run > Run**.
2. Select "Android Application".



Upgrade the UI to an XML Layout:

1. In the Eclipse Package Explorer, expand the /res/layout/ folder and open main.xml (once opened, you might need to click the "main.xml" tab at the bottom of the window to see the XML source). Replace the contents with the following XML: -

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/textview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="@string/hello"/>
```

2. Inside the res/values/ folder, open strings.xml. This is where you should save all default text strings for your user interface. If you're using Eclipse, then ADT will have started you with two strings, *hello* and *app_name*. Revise *hello* to something else. Perhaps "Hello, Android! I am a string resource!" The entire file should now look like this: -

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am a string resource!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```

3. Modifying your HelloAndroid class and use the XML layout. Edit the file to look like this: -

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Note: - Instead of passing setContentView() a View object, we give it a reference to the layout resource. The resource is identified as R.layout.main, which is actually a compiled object representation of the layout defined in /res/layout/main.xml. Other than the change to the TextView string (Hello, Android! I am a string resource!), the application looks the same.

Note:- Do not copy paste the code from this text…unintended text formatting errors will occur in Eclipse.

:: C2DM (Cloud to Device Messenging) ::

Understanding and Working with Datastore (Google app engine):

http://code.google.com/appengine/docs/java/datastore/overview.html

Android Cloud to Device Messagenging:

http://code.google.com/android/c2dm/

- It requires devices running Android 2.2 or higher that also have the Market application installed, and it must have at least one logged in Google account… (However, not limited to deploying your applications through Market).
- Uses AlarmManager to keep the connection (Heartbeat).
- It uses an existing connection for Google services. This requires users to set up their Google account on their mobile devices.
- Upper bound on message size 1024 characters.

Requirements - Components vs. Credentials:

Components	
Mobile Device	The device that is running an Android application that uses C2DM. This must be a 2.2 Android device that has Market installed, and it must have at least one logged in Google account.
Third-Party Application Server	An application server that developers set up as part of implementing C2DM in their applications. The third-party application server sends data to an Android application on the device via the C2DM server.
C2DM Servers	The Google servers involved in taking messages from the third-party application server and sending them to the device.
Credentials	
Sender ID	An email account associated with the application's developer. The sender ID is used in the registration process to identify a Android application that is permitted to send messages to the device. This ID is typically role-based rather than being a personal account—- for example, `my-app@gmail.com`.
Application ID	The application that is registering to receive messages. The application is identified by the package name from the manifest. This ensures that the messages are targeted to the correct application.
Registration ID	An ID issued by the C2DM servers to the Android application that allows it to receive messages. Once the application has the registration ID, it sends it to the third-party application server, which uses it to identify each device that has registered to receive messages for a given application. In other words, a registration ID is tied to a particular application running on a particular device.
Google User Account	For C2DM to work, the mobile device must include at least one logged in Google account.
Sender Auth Token	A ClientLogin Auth token that is saved on the third-party application server that gives the application server authorized access to Google services. The token is included in the header of POST requests that send messages. For more discussion of ClientLogin Auth tokens, see ClientLogin for Installed Applications.

A slide-show reference ['**Pushing' data from cloud to mobile device**]:

(+) http://www.slideshare.net/LarsVogel/android-cloud-to-device-messaging-with-the-google-app-engine

Implementation:

The 3 steps

(+) To register with Google C2DM server, we must call Google's until class: ***C2DMessagin.register*** (context, Your.Email@gmail.com).

(+) Next, we must register the Google provided C2DM broadcast receiver in "***AndroidMaifest.xml***".

(+) Finally, Google receiver will forward the information it receives to: ***context.getPackageName()*** + "***.C2DMReceiver***".

Permissions

To use C2DM in your application to have to register for the following permissions -

- ***com.google.android.c2dm.permission.RECEIVE***
- ***android.permission.INTERNET***

Your application should also declare the permission "***applicationPackage + ".permission.C2D_MESSAGE***" with the "***android:protectionLevel***" of "***signature***" so that other applications cannot register and receive message for the application.

android:protectionLevel="signature" ensures that applications which request a permission must be signed with same certificate as the application that declared the permission.

Intent Receiver

Your application must register an intent receiver for the two intents:

- ***com.google.android.c2dm.intent.REGISTRATION***
- ***com.google.android.c2dm.intent.RECEIVE***

The receiver for "***com.google.android.c2dm.intent.RECEIVE***" will be called once a new message is received, while the receiver for "***com.google.android.c2dm.intent.REGISTRATION***" will be called once the registration code for the app is received.

:: C2DM (Cloud to Device Messenging) <continued from week 2>::

How it works: -

- To enable C2DM, an application on the device registers with Google and get a registration ID, and sends the ID to its server.
- When the server needs to push a message to the app on the device, it posts the message via HTTP to Google's C2DM servers.
- The C2DM servers route the message to the device, and an Intent broadcast is sent to the app.
- The app is woken up to process the message in its Intent Receiver.
- The app can unregister with C2DM when the user no longer wants messages to be pushed to it.

:: IMPLEMENTATION ::

Step 0: Signup for C2DM

http://code.google.com/android/c2dm/signup.html

Go through the list of steps and finish the signing-up process.

Signup form
Thank you for signing up for Android Cloud to Device Messaging (C2DM). You will receive an email at your given 'Contact email' address when your registration has been processed.

Please also note that by default, all new sender accounts are held to a development-only quota. This quota is sufficient for development and exploratory purposes. Information on production-level quota will be sent to your 'Contact email' address when your registration has been processed.

Step 1.1: Register Your App for C2DM

Modify the file "**AndroidManifest.xml**" as shown below

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <permission
        android:name="com.example.helloandroid.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />

    <uses-permission android:name="com.example.helloandroid.permission.C2D_MESSAGE" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".C2DMClientActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver
            android:name=".C2DMRegistrationReceiver"
            android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter >
                <action android:name="com.google.android.c2dm.intent.REGISTRATION" >
                </action>

                <category android:name="com.example.helloandroid" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

NOTE - The "com.google.android.c2dm.intent.REGISTRATION" intent includes a registration ID. Each registration ID represents a particular device, e.g. every Android phone will receive its own registration code.

The C2DM may refresh this registration ID periodically but until a refreshment your application should store this ID for later use.

Step 2.1: Application Server Registration

Step 2.2: Registration ID for mobile app

PROBLEMS FACED :- I was trying to build all of this manually but the location of edited code was getting convoluted. I found out that Google had pre-written sample code. I will be restarting the C2DM part of the project with this new code.

:: INSTALLATION AND GETTING STARTED WITH GOOGLE APP ENGINE CONNECTED ANDROID PROJECT ::

Download and Install Eclipse:

[http://www.eclipse.org/downloads/]

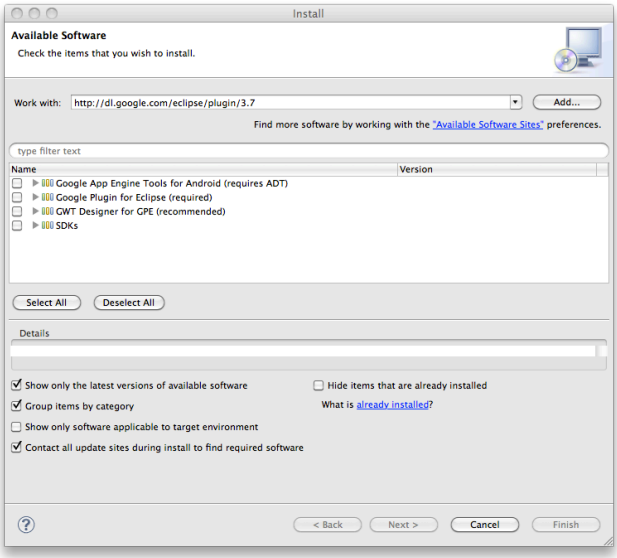
Eclipse Classic 3.7.1 is the recommended version.

Install the Google Plugin for Eclipse using the Software Update feature of Eclipse:

[http://code.google.com/appengine/docs/java/tools/eclipse.html]

To install the plugin, using **Eclipse 3.7 (Indigo)**:

1. Select the **Help** menu > **Install New Software...**
2. In the **Work with** text box, enter: **`http://dl.google.com/eclipse/plugin/3.7`**
   Click the **Add...** button. In the dialog that shows, click **OK** (keep the name blank, it will be retrieved from the update site.)
3. Click the plus icon next to "Google Plugin for Eclipse" and "SDKs". Check the boxes next to "Google Plugin for Eclipse 3.7" and "Google App Engine Java SDK". You can also select the "Google Web Toolkit SDK" if you'd like to use Google Web Toolkit with your apps. Click the **Next** button. Follow the prompts to accept the terms of service and install the plugin.
4. When the installation is complete, Eclipse prompts you to restart. Click **Yes**. Eclipse restarts. The plugin is installed.



Download the Android SDK:

The following link has the downloadable zip file (for MAC OS X, also Windows and Linux): -

http://developer.android.com/sdk/index.html

For further installation and configuration directions refer to the following link: -

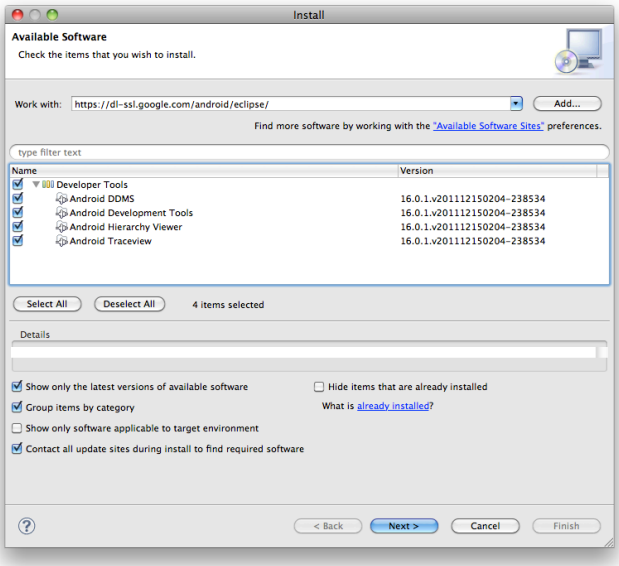
http://developer.android.com/sdk/installing.html

Downloading the ADT Plugin:

Use the Update Manager feature of your Eclipse installation to install the latest revision of ADT on your development computer.<>

Assuming that you have a compatible version of the Eclipse IDE installed, as described in Preparing for Installation, above, follow these steps to download the ADT plugin and install it in your Eclipse environment.

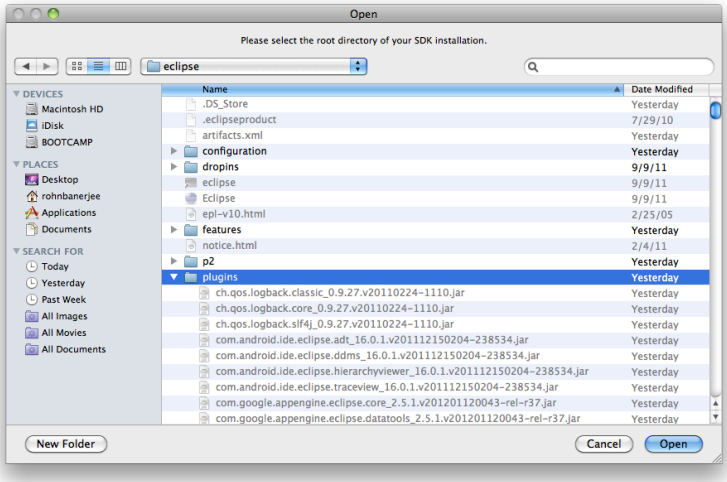
1. Start Eclipse, then select **Help** > **Install New Software...**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*: **https://dl-ssl.google.com/android/eclipse/**
4. Click **OK** Note: If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**. Note: If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
8. When the installation completes, restart Eclipse.



Configuring the ADT Plugin:

After you've successfully downloaded the ADT as described above, the next step is to modify your ADT preferences in Eclipse to point to the Android SDK directory:

1. Select **Window** > **Preferences...** to open the Preferences panel (Mac OS X: **Eclipse** > **Preferences**).
2. Select **Android** from the left panel.
3. You may see a dialog asking whether you want to send usage statistics to Google. If so, make your choice and click **Proceed**. You cannot continue with this procedure until you click **Proceed**.
4. For the *SDK Location* in the main panel, click **Browse...** and locate your downloaded SDK directory.
5. Click **Apply**, then **OK**.

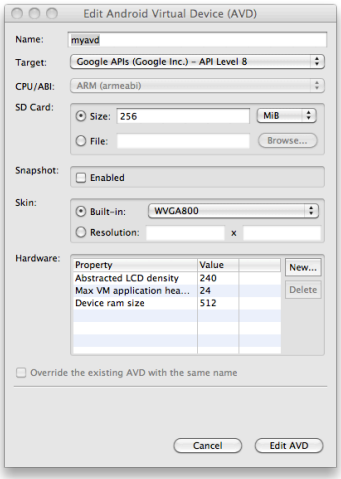


Create an AVD:

Before launching the emulator, create an Android Virtual Device (AVD). An AVD defines the system image and device settings used by the emulator.

To create an AVD:

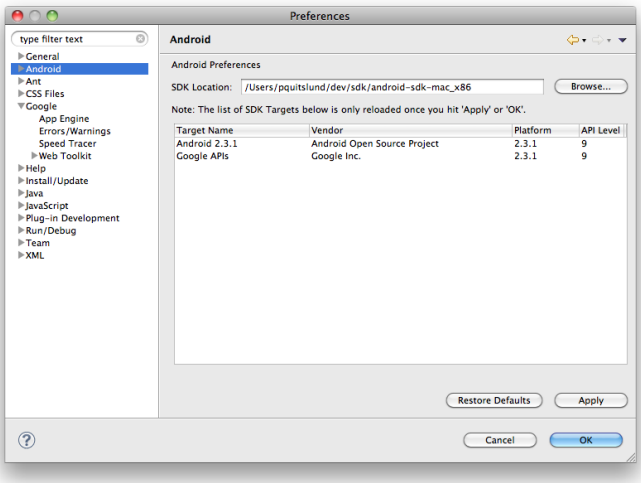
1. In Eclipse, select **Window > AVD Manager**.
2. Click **New...**. The **Create New AVD** dialog appears.
3. Type the name of the AVD, such as "my_avd".
4. Choose a target. The target is the platform (for the purpose of C2DM we will choose the Google API - API level 8) you want to run on the emulator. For this tutorial, choose the latest platform that you have installed and ignore the rest of the fields.
5. Click **Create AVD**.



Setup for Eclipse Preferences:

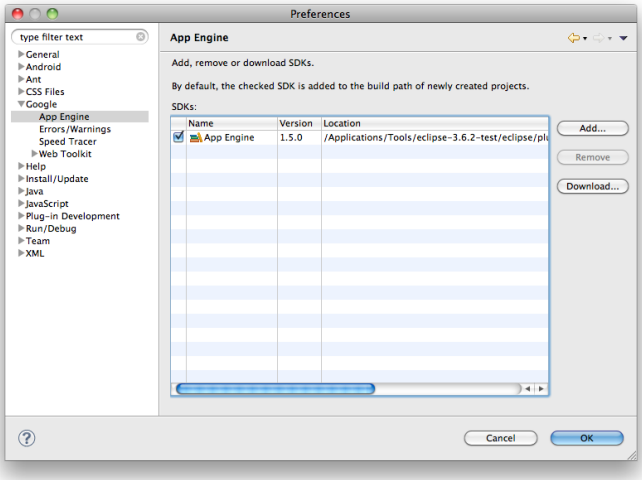
After you install the required software pieces in Eclipse, you need to set Eclipse preferences for the Android, App Engine, and GWT SDKs. If there are multiple versions installed, the wizard uses these preferences to determine which version is used to generate code.

To set Eclipse preferences::



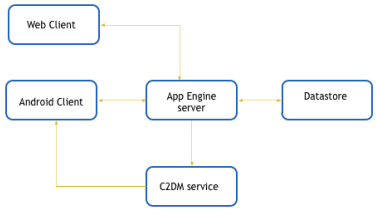
1. Set your Eclipse preferences for Android SDK: **Preferences > Android**
   The first time you set the Android preferences, you may need to browse to the download directory of the Android SDK and click on **Apply** to get the list of available SDK and API levels supported by the install directory you selected. Once you see the list, you can continue. (You don't select anything from the list.)

2. Set your Eclipse preferences to use the desired App Engine SDK version (use 1.5.0 or greater) as the default SDK: **Preferences** > **Google** > **App Engine**



Google App Engine Connected Android Architecture:

The diagram below shows the architecture of the application generated by the App Engine Connected Android wizard:

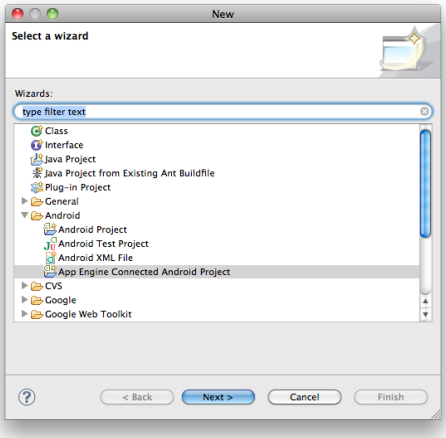


Starting and Deploying theC2DM Project:

After installing required software and configuring Eclipse, you can use the App Engine Connected Android wizard to create a new project in Eclipse.

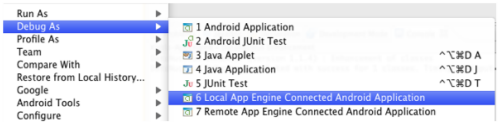
To create a new App Engine Connected Android project::

In Eclipse, choose **File > New > Other... > App Engine Connected Android Project** as shown

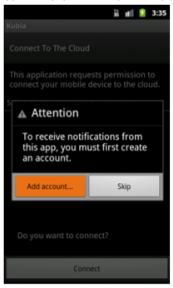


In the lower left of Eclipse, if you see the **Sign in to Google** icon, click on it to log into the Google account used for App Engine and C2DM. If the account is already displayed in that area, you are already logged in and can skip this step. (If you need to log out and then log into a different account, click on the icon.)

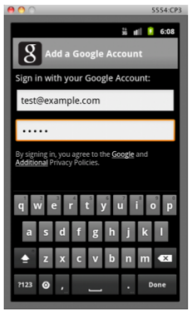
In the Eclipse project explorer pane, select the Android project, right click on it, then select **Debug As > Local App Engine Connected Android Application**. This starts up the App Engine back end in development mode, connects it to the Android application and loads the APK for your Android app into the AVD.



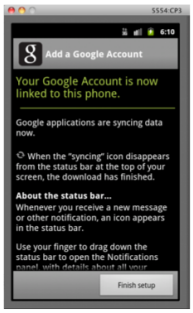
Wait for a few moments until the application finishes launching in the AVD. If this is the first time you are running the app on this AVD, the application pops up and asks you to set up an account. (If you have already set up the AVD for your app and have run the app already, you can skip this step.) To set up the AVD for the app for the first time:



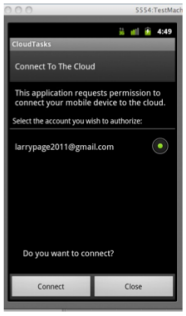
a. Click to create an account
b. In the Add Account prompt, select Google as the account to add
c. In the Add a Google Account form, click Next
d. Click **Sign in** and then supply the login credentials for the Google account that you specified when setting up C2DM for your project. This is needed for the C2DM feature and for authenticated RPC calls to App Engine. Click **Done** on the keypad, then click **Sign in**.



Wait while the Google account is linked to the AVD; then click **Finish setup** when the success page appears.



Authorize the application to connect to the cloud using one of your Google accounts. (If you don't see any accounts, click the AVD's back button.) This registers the AVD (or physical Aroid device) so that it can receive C2DM notifications.



Say "HELLO" and notice the message returned from the app engine::

Running the C2DM APP using a Real Phone, Production APP Engine, and C2DM:

To get the Cloud Tasks starter app running on a real phone with c2dm, you need to do the following:

1. Sign up for c2dm first. Remember the package name and use it when running the wizard! This is required, and once you choose the package name on c2dm sign-up, it is nowhere visible later.
2. Sign in to Eclipse with the c2dm account name you'll use.
3. Run the new project wizard to create the app as described in Create a New Project.
4. To run on a phone (not on an emulator), you must deploy to live GAE, as follows:
    a. Edit Setup.java and change APP_NAME and SENDER_ID to match your GAE app id and c2dm account.
    b. Change app id in appengine-web.xml.
    c. Deploy to app engine with your c2dm account.
    d. Force sign in to GAE app by going to `your_production_url/tasks/`. This triggers the security constraint in web.xml to get logged in to GAE. (After logging in, you'll get an error page, but you can ignore that.)
    e. Go to `your_production_url` and Say Hello to App Engine. If logged in, you should see a success message.
    f. If you've previously connected an account on a phone to this production app, go to the App Engine datastore viewer and delete all records.
    g. In Eclipse, select the project, right-click and select **Debug as > Remote App Engine Connected Android project**.
    h. On the phone, you'll be prompted to connect an account. Use your c2dm account. If you have previously connected a different account, select **Menu > Accounts > Disconnect** in the app and then reconnect.

SharingScripts
How to share your scripts with others.
Currently, the preferred way to share small scripts with others is via barcode. The easiest way to generate a barcode is to use the ZXing project's online QR Code Generator.

1. Open the Contents drop down and choose Text.
2. On the first line of the Text Content, enter the name of the script (e.g. hello_world.py).
3. Below that, paste the script content.
4. Open the Size drop down and choose L.
5. Click Generate.
6. Embed or share the resulting barcode image with your friends.

To download the script to your phone:

1. Launch SL4A or return the scripts list.
2. Press the Menu button.
3. Tap Add.
4. Tap Scan Barcode.
5. Scan the barcode and SL4A will add the script to your list.

A QR code can encode 4,296 characters of content. So, this is only effective for small scripts.

Scripts as APKs

There are several ways to publish your script as an APK. The following steps describe how to do that using Eclipse IDE.

1. Download script template project archive.
2. Import the template project into Eclipse: `File > Import > Existing Projects into Workspace`, click on `Select archive file` and fill in the path to your copy of script_for_android_template.zip.
3. Set the ANDROID_SDK variable, as described in the compilation instructions.
4. Build the project. If Eclipse complains that gen folder is missing and/or there are build path errors, Clean/Build/Refresh should solve the problem.
5. Rename the project and the default package `com.dummy.fooforandroid -> your_package_name` (we suggest using `Refactor > Rename`). It is important that this is unique because Android system uses package name as an identifier of your APK.
6. Update the `package` property in AndroidManifest.xml `package="your_package_name"`.
7. Replace the `script.py` in `res > raw` with your script. You can either use the default name `script.your_extension` or you can name it however you like, in which case make sure to change `R.raw.script` (Script.java:10) to `R.raw.your_script_name`.
8. Place any additional files (ie, html) into 'res > raw' with your script. They'll be unpacked into the same scripts folder.
9. In AndroidManifest.xml uncomment all the permissions that are required to execute your script.
10. Create and sign an APK using ADT Export Wizard as described in the Android's Dev Guide.

The following steps describe how to build an APK using Ant.

1. Download script template project archive.
2. Open a command-line and extract the archive: `unzip -d <path/project_directory> script_for_android_template.zip`.
3. Set the ANDROID_SDK variable to point to the root of your Android SDK directory: `export ANDROID_SDK=<SDK_root>`.
4. Navigate to the root directory of your project and configure your package name: `sh configure_package.sh <your_fully_qualified_package_name>` (`sh configure_package.sh com.dummy.fooforandroid` by default).
5. Rename the project name in build.xml.
6. Replace the `script.py` in `res > raw` with your script. You can either use the default name `script.your_extension` or you can name it however you like, in which case make sure to change `R.raw.script` (Script.java:10) to `R.raw.your_script_name`.
7. Place any additional files (ie, html) into 'res > raw' with your script. They'll be unpacked into the same scripts folder.
8. In AndroidManifest.xml uncomment all the permissions that are required to execute your script.
9. Follow Android's Dev Guide to build and sign your application.

For a really detailed walk through, see John K's Blog.

Updating your Template to the latest libraries

Sl4a is being continually updated, whereas your standalone script will be using a static snapshot of whenever you created your copy from the template.

To get the latest version of the template, open script_for_android_template.zip and extract:

```
libs/script.jar
libs/armeabi/libcom_googlecode_android_scripting_Exec.so
```

into your project.

If using eclipse, remember to refresh and clean your project.

NB: libcom_googlecode_android_scripting_Exec.so has only been updated once so far, whereas script.jar is being constantly updated.

If you have a full version of the SL4A source installed, and wish to make your own version of script.jar etc, run build.xml in ScriptForAndroidTemplate as a Ant Build.

GitHub Repo Setup:

The documentation for setting up the Github repository is the same as given on the github website and can be found at the original site at the following links:

The documentation for help with commands can be found at the following link:

1. http://help.github.com/mac-set-up-git/
2. http://help.github.com/create-a-repo/

The documentation for help with commands along with user manual and cheat sheets can be found at the following links:

1. http://schacon.github.com/git/user-manual.html
2. http://help.github.com/git-cheat-sheets/
3. http://gitref.org/

REFERENCES:

http://www.vineetmanohar.com/2009/04/writing-java-hello-world-for-google-app-engine/

http://developer.android.com/resources/tutorials/hello-world.html