Module: Mobile Application development (Android)

Session 6: Developing and running first Android application (Practice)

The goal of this practical is to learn the fundamentals of developing Android Applications, from project creation to installation on a physical device. More specifically, you should gain the knowledge of how to use basic development tools to support the application development process, as well as the key components of an Android application itself.

**Objectives**

1. Setup the Development Environment
2. Create a "Hello World" Android Application
3. Understand the various parts of an Android Project
4. Use the Android Emulator
5. Install and run the application on a physical device
6. Create a simple User Interface

**Activities**

**1. Setting Up the Development Environment**

**1.1 Download/Install the SDK**

For in-depth instructions, visit [Android Installation Documentation](http://developer.android.com/sdk/installing.html). Otherwise, perform the following steps.

* Go to[http://developer.android.com/sdk/index.html](http://developer.android.com/sdk/index.html#_blank).
* Unpack to a convenient location - Remember the full path to this location, we will refer to it as ***<*android\_sdk\_dir*>*** for the rest of the lab.

*For example, on Linux your home directory is a convenient location.*

***<android\_sdk\_dir>*** *would then be /home/<username>/android\_dir.*

* Add the path to the <android\_sdk\_dir>/tools directory to your system PATH

Linux (Lab Machines Running Fedora):

Using your favorite text editor, open the .mycshrc file in your home directory.

Add the following text to the end of the file:  
set path=($path <android\_sdk\_dir>/tools)

Open up a terminal, navigate to your home directory and execute the following command:  
 source .mycshrc

Windows:

Right-click **My Computer**.

Click **Properties**.

Click the **Advanced** tab.

Click the **Environment Variables** button.

Double Click **Path** under **System Variables**.

Add  ;<android\_sdk\_dir>/tools  to the end of the **Variable Values** text field.

* Test your installation by running **adb** from the command line. If you did everything right, you should get a long list of help instructions.

**1.2 Download/Install the Eclipse Plugin**

* It is recommended that you use Eclipse 3.4 or later

**Lab Machines** - Fedora Eclipse based on 3.4.2  
The version of Eclipse used by the lab machines is missing a vital component and requires adding an additional Eclipse plugin in order to use the Android plugin:

Click the menu **Help**->**Software Updates**.

Click the tab **Available Software**->**Add Site** button.

Enter  http://download.eclipse.org/releases/ganymede  into the **Location**field.

Click **OK** button.

Enter  WST Common UI  into the search/text box at the top of the window (give it a second, it tries to search as you type and its kind of slow).

Click checkbox next to **WST Common UI**.

Click the **Install** button.

Click the **Next** button.

Accept the terms, click **Finish**.

Restart Eclipse.

Follow the steps in the next bullet **3.4 Ganymede.**

3.4 Ganymede:

Click the menu **Help** → **Software Updates**.

Click **Available Software** tab -> **Add Site** button.

Enter  https://dl-ssl.google.com/android/eclipse/  into the "**Location**" field.

Click the **OK** button.

Click checkbox next to **Developer Tools**.

Click the **Install** button.

Click the **Next** button.

Accept the terms, click **Finish**.

Restart Eclipse.

3.5 Galileo:

Click **Help** → **Install New Software**.

Click **Add...**button.

Enter a name for the site into the **Name** field.

Enter  https://dl-ssl.google.com/android/eclipse/  into the **Location** field.

Click the **OK** button.

Click the checkbox next to **Developer Tools**.

Click the **Next** button.

Accept the terms, click **Finish**.

Restart Eclipse.

* Point Eclipse to <android\_sdk\_dir>:

Click the menu **Window** -> **Preferences**.

Click **Android** from the Hierarchy view on the left hand side.

Enter <android\_sdk\_dir> into the **SDK Location** field.

Click the **Apply** button.

Click the **OK** button.

**1.3 Download/Install the SDK Platform Components**  
At the time of writing this lab there are are five different versions of the Android Platform available, ranging from 1.1 to 2.0.1. It is a best practice to develop for the oldest platform available that still provides the functionality you need. This way you can be assured that your application will be supported by as many devices as possible. However, you will still want to download newer versions of the platforms so that you can test your applications against these as well. Due to the size of each platform component you will only be required to download and develop on one platform for the whole class. We will target Android 1.6, API 4. Before we can begin developing we must download and install this platform:

* Select the menu **Window** -> "**Android SDK and AVD Manager**", or click on the black phone shaped icon in the toolbar.
* Select **Available Packages** on the left hand side.
* Expand the Google Android site in the "**Site, Packages, and Archives**" Tree.
* Check the following items:

**SDK Plaform Android 1.6, API 4 Revision 4**

**Google APIs by Google Inc., Android API 4, Revision 4**

* Click **Install Selected**.
* Accept the Terms for all packages and click **Install Accepted**.

We're now ready to develop our application.

**2. Create "Hello World" Application**

**2.1 Create a new Android Project**

* Open Eclipse.
* Click the menu **File** -> **New** -> **Project**.
* Expand the **Android** folder and select **Android Project**.
* Name the project lab1<userID>

For instance, if you userID is *RAM*, you would name your project lab1RAM .

* You can change the location of where you would like to save the project by un-selecting the "Default Location" check box, and supplying your own location.
* Check "**Android 4.0**" from the Build Target List.

*This identifies that the project is being built to be compatible with Android versions 1.5 and later.*

*Its generally preferred that you choose the lowest build number possible, so as to be compatible with the largest number of existing systems in place.*

*This build target can be changed any time later on through the Project Properties menu.*

* Fill in the Properties:

**Application Name** = Hello World!

*This is the freindly name of the application, that shows up on the device.*

**Package Name** = edu.calpoly.android.lab1<userID>

*This is the namespace for the project, follows standard Java conventions.*

**Create Activity** = HelloWorld

*This optional field will automatically create a "Main Activity" class for the project. You can think of the Main Activity as the Home Page for your application.*

**Min SDK** **Version** = 4

*This specifies the minimum API Level on which your application can run. By default this is set to the API Level of the Build Target Platform. As new API's are added to newer Versions, their API levels increase as well. A Program that uses an API Level of four, won't be able to run on a Platform that has a lower API Level.*

* Click "**Finish**".

**3. Take a Tour of the Application**

The application you've just created is very similar to other java applications you may have created in eclipse. Look in the **Package Explorer** side bar. Notice that the Android Development Toolkit(ADT) has generated a number of folders and files for you:

* **src**: If you expand this out you'll see the package hiearchy you previously entered. This is where your source code files will go.

**HelloWorld.java**: This is the auto-generated stub Activity Class with the name you entered into the project creation wizard. We'll add some code to this later.

* **Android 4.0**: This is the version of the library you had chosen in the project creation wizard. The application will be built using this version of 'android.jar'
* **res**: This folder will contain all of the resources (a.k.a. external data files) that your application may need. There are three main types of resources that you will be using and the ADT has created a subdirectory for each.

**drawable**: This folder will hold image and animations files that you can use in you application.

*It already contains a file called icon.png which represents the icon that Android will use for your application once it is intalled*

**layout**: This folder will hold xml layout files that the application can use to construct user interfaces. You will learn more about this later, but using a layout resource file is the preferred way to layout out your UI.

*It already contians a file called main.xml which defines the user interface for your 'HelloWorld.java' Activity class. Double clicking on this file will open up the Android UI Editor that you can use to help generate the xml layout files.*

* + **values**: This folder will hold files that contain value type resources, such as string and integer constants.
    - *It already contains a file called strings.xml. Double clicking on this file will open up the Android Resource Editor. Notice that there are two strings in there already, one of which is named 'app\_name'. If you select this value, on the right hand side of the editor you should see the Application Name you entered in the project creation wizard. You can use this editor to add new resources to your application.*
* **gen**: This folder will contain Java files that get auto-generated by ADT. Notice that it already contains one file called "R.java".
  + **R.java**: Is a special static class that is used for referencing the data contained in your resource files. If you open this file you will see a number of static inner classes for each of the resource types, as well as static constant integers within them. Notice that the names of the member variables are the same as the names of the values in your resource files. Each value in a resource file is associated with an integer ID, and that ID is stored in a member variable of the same name, within a static class named after its data type.
    - *The 'app\_name' resource value has an ID and is of value type 'string'. The ADT automatically adds an integer constant to the R.string class and names it 'app\_name'.*
* **assets**: This folder is for asset files, which are quite similar to resources. The main difference being that anything stored in the 'assets' folder has to be accessed in the classic 'file' manipulation style. For instance, you would have to use the AssetManager class to open the file, read in a stream of bytes, and process the data. You will not be using assets quite as extensively as you will be using resources.
* **AndroidManifest.xml**: Every project has a file with this exact name in the root directory. It contains all the information about the application that Android will need to run it:
  + Package name used to identify the application.
  + List of Activities, Services, Broadcast Recievers, and Content Provider classes and all of their necessary information, including permissions.
  + System Permissions the application must define in order to make use of various system resources, like GPS.
  + Application defined permissions that other applications must have in order to interact with this application.
  + Aplication profiling information.
  + Libraries and API levels that the application will use.
* **default.propeties**: Ths file contains all of the project settings, such as the build target you chose in the project creation wizard. If you open the file up, you should see 'target=4', which is your build target. You should never edit this file manually. If you wish to edit the project properties, do so by right-clicking the project in the 'Package Explorer' panel, and selecting 'Properties'.

The project creation wizard has written the 'Hello World' application for you already. A string resource containing the display text has been placed into the res\values\strings.xml file. The value is named 'hello'. The xml UI layout has been added to res\layout\Main.xml. While you can use the Android Layout Editor to create your xml layout, you can also code them manually yourself. Lets take a look at this file:

* **Right-Click** on the file.
* Select **Open With** -> **Text Editor**.

Notice the Top Level node, **Linear Layout**, which defines the style of layout this file will be using. This 'Linear Layout' is perhaps the most basic, and speicifies that UI elements will be laid out in a continuous line. It has three properties, orientation, width, and height.

Notice the Second Level node, **Text Element**, which defines a UI element for displaying text. It has three properties, width, height, and the text to display. Notice that the text property is not set to "Hello World!". Instead it is set to reference the resource value which contains the text we want to display. In this case we are choosing to display the contents of the 'hello' value. We do this by using the '@' sybmol, followed by the value type of the resource (which is a 'string'), followed by the name of the value (which is 'hello').

**4. Run "Hello World" on the Emulator**

**4.1 On the Emulator**

Before we can run the application we need to setup an Android Vitual Device(AVD), or emulator, to run it on:

* Select the menu **Window** -> "**Android SDK and AVD Manager**", or click on the black phone shaped icon in the toolbar.
* Select **Virtual Devices** on the left hand side.
* Click the **New...** button.
* Give your AVD a name.
* Select the target build that we would like to run the application on, "**Android 1.6 - API Level 4**".
* Click **Create AVD** and close out the SDK/AVD Manager.

We're now ready to run our application.

* Select the menu **Run** -> **Run**.
  + *Note: The android may take a long time to start up.*

Contratulations! You"ve just created and an Android Application.

**4.2 On a Physical Device**

Before we can run the application on a physical device we need to modify the project, make a configuration change on the phone, and install some drivers for the phone on our development machine. We begin by making your project declare itself as debuggable. It's possible to do this through the Android Manifest Editor that the ADT provides, however doing this manually helps you understand the manifest better:

* Project Modifications
  + From the Package Explorer double-click the file **AndroidManifest.xml**.
  + Select the tab labeled **AndroidManifest.xml** along the bottom.
  + Add android:debuggable="true" to the inside of the opening <application> tag.
  + Save the file and close it.
* Phone Modifications
  + Turn the phone on.
  + Navigate to the **Home** screen.
  + Press **MENU** (the physical button on the device).
  + Select **Settings** -> **Applications** -> **Development**.
  + Enable the **USB debugging**"option.
* Installing the Android USB drivers
  + Mac OS X: Don't need to install drivers, it should just work.
  + Windows: [Follow instructions here](http://developer.android.com/sdk/win-usb.html)
  + Linux:[Follow instructions here](http://developer.android.com/guide/developing/device.html#setting-up)

Ensure the device is properly connected. Run the application as you would normally. The "Hello World!" app should start on the phone.

**5. Simple Activity Classes**

There are four major types of component classes in any Android application:

* **Activities:** Much like a Form for a web page, activities display a user interface for the purpose of performing a single task. An example of an Activity class would be one which displays a Login Screen to the user.
* **Services:** These differ from Activities in that they have no user interface. Services run in the background to perform some sort of task. An example of a Service class would be one which fetches your email from a web server.
* **Broadcast Receivers:** The sole purpose of components of this type is to receive and react to broadcast announcements which are either initiated by system code or other applications. If you've ever done any work with Java Swing, you can think of these like Event Handlers. For example, a broadcast announcement may be made to signal that a WiFi connection has been established. A Broadcast Receiver for an email application listening for that broadcast may then trigger a Service to fetch your email.
* **Content Providers:** Components of this type function to provide data from their application to other applications. Components of this type would allow an email application to use the phone's existing contact list application for looking up and retrieving email address.

In this lab we will be focusing on what Activities are and how they are used. We will cover the other components in later labs if you would like more information on these components visit the Android overview page for [Application Components](http://developer.android.com/guide/topics/fundamentals.html#appcomp).

In case you haven't figured it out by now, you have already created one of these component classes. That's right, the HelloWorld class is an Activity Class. Its a simple user interface designed to greet the user. In the section that follows we'll make our application more personal by adding a new Activity class to ask for the user's name. We'll then update the existing HelloWorld greeting Activity to display that name.

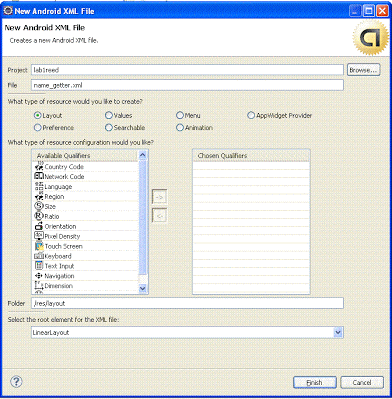
**5.1 Getting the User's Name**

To get the user's name you will be creating an Activity class which will allow the user to enter their name into a text field and press a button when finished to proceed to the HelloWorld greeting Activity. There are three separate steps to accomplish here. You must first layout your user interface in XML. Then you must create the Activity class to parse the input from the user and initiate the HelloWorld Activity. Finally you will have to reconfigure the application to use your new name retrieval Activity on startup.

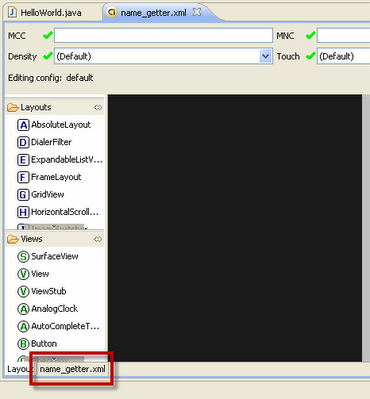
**5.1.1 Create the User Interface**

Android allows you to layout your user interfaces using a simple XML specification. We will go into more depth on this topic in the next lab, so for now you will be setting up a basic interface using four different GUI elements. Begin by creating a new Android XML File:

* Select the menu **File** -> **New -> Android XML File**.*If* ***Android XML File*** *does not appear in the menu:*
  + Select **Other.**
  + Expand the **Android** folder.
  + Select **Android XML File** and click **Next**.
* Ensure the Project matches the name of your project and that the folder is **/res/layout**.
  + *Layout files should always go in this folder.*
* Enter "**name\_getter.xml**"as the file name
  + *The name of your layout files must only contain lower case letters, the numbers 0-9, underscores '\_', or periods '.'*
  + *[a-z0-9\_.]*
* Select the **Layout** radio button.
* Select **LinearLayout** from the "Select the root element..." drop down and click **Finish**.



* By default the file will be opened to the Layout Editor tab. Select the tab labeled **name\_getter.xml** to switch to the XML Editor.
  + *This should be located in the bottom left corner of the Layout Editor.*

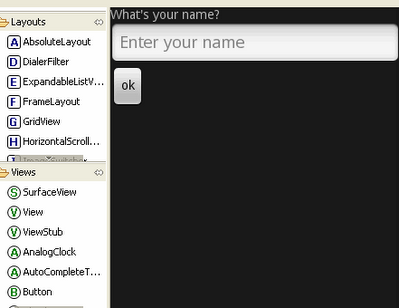


Each GUI element derives from the View base class. The first element was added for you when you created the XML layout file and selected LinearLayout from the dropdown menu. You should be able to see an XML opening and closing tag labeled LinearLayout in the editor. Each XML layout file must have a single root view tag, inside which all other view tags are nested. The LinearLayout tag tells Android to arrange elements contained inside it in a straight line in the order in which they appear. Lets make a few modifications to the LinearLayout by editing the attributes contained in the opening LinearLayout tag:

* Set the attributes labeled android:layout\_width and android:layout\_height to "fill\_parent" *(Include the quotes).*
  + *This tells Android that the LinearLayout should take up all the available width and height on the screen.*
* Add an attribute labeled android:orientation and set it to "vertical" .
  + *This tells Android that elements nested inside the LinearLayout should be laid out in a column, as opposed to a single row as indicated by "horizontal".*

Lets add the other three UI elements to our XML layout file:

* Switch back to the Layout Editor tab.
  + *The three elements we will add all reside under the folder icon labeled "Views"*
  + *This can be seen along the left hand side of the previous figure, about halfway down*
* Scroll down to item labeled **TextView**.
  + Click and drag the **TextView** onto the black canvas.
  + *The Layout Editor will pre-populate the label with its auto-generated id, which may look somewhat strange.*
* Repeat the previous step for the **EditText** and **Button** labels.
  + *Remember, order matters for the LinearLayout.*
* This is what you want your UI to look like. However it may not resemble this quite yet:



* Switch back to the XML Editor to change the attributes of the elements you just added.
  + *Notice that all the UI elements you added are nested within the LinearLayout Element.*
  + *There will always be only one root element. You may, however, nest other Layout elements within each other.*
* Editing the **TextView** element:
  + This is the label that prompts the user to enter their name. It displays the value contained in the android:text attribute.
    - Set this attribute to ask the user what their name is.
  + The android:id attribute provides a variable name for referencing this element from within the code.
    - Id's are specified with the syntax of the @+id/MyId01 .
    - MyId01 is the handle used to identify the element from within your application code via the Static R class.
* Editing the **EditText** element:
  + This is the text field where the user will input their name. It will default to contain the value set by the android:text attribute.
    - Remove this attribute, we don't need it.
  + The android:hint attribute provides a hint to the user when the field is empty, and disappears when text is entered.
    - Set this attribute to instruct the user to enter their name.
  + Either make a mental note of the android:id attribute or provide your own variable name which you will use to reference this element from within the code.
* Editing the **Button** element:
  + This is the button that will allow the user to continue to the next HelloWorld greeting screen.
  + It displays the value contained in the android:text attribute.
    - Set this attribute to something like "ok", "next", or "submit".
  + Either make a mental note of the current value for the android:id attribute or provide your own variable name which you will use to reference this element from within the code.

**5.1.2 Create the Activity Class**

Using the HelloWorld Class from section 2.1 as an example, create a new class that extends android.app.Activity class and implements android.view.View.OnClickListener interface.

* Implement the OnClickListener interface by creating a method stub with the following signature: public void onClick(View v) .
  + *We'll fill this method later*
* Declare a member variable of the type android.widget.EditText
  + *This will hold a reference to the text field in which the user will enter their name, the same one that you added to the name\_getter.xml layout file.*
* Add a method with the following signature: public void onCreate(Bundle savedInstanceState) .
  + *This method will be called when the Activity starts and is where initialization of local and member data will be done.*
* Inside this method perform the following:
  + make a call to super.onCreate(savedInstanceState)
    - This should always be done and is to ensure that any necessary parent class initializations are performed.
  + make a call to this.setContentView(R.layout.name\_getter)
    - When you created the XML layout file earlier, the Android Eclipse Plugin automatically added a static constant to the static R.layout class in the R.java file under the /gen folder. This constant variable has the same name of the file and its value is used to identify the the layout file.
    - This call tells Android to create a screen based off of the layout file.
  + Make a call to this.findViewById(R.id.**<EditText id>**) and set your EditText member variable equal to the return value.
    - **<EditText id>** should be replaced with the android:id that was specified in the name\_getter.xml layout file for the EditText element.
    - You will have to explicitly cast the return value to the type of EditText as this method only returns objects of type Object.
    - *This static constant value was added in the same way as it was done for the R.layout.name\_getter value and serves the same purpose.*
  + Make a call to this.findViewById(R.id.**<Button id>**) and set a local android.widget.Button reference equal to the return value.
    - **<Button id>** should be replaced with the android:id that was specified in the name\_getter.xml layout file for the Button element.
  + Make a call to **button**.setOnClickListener(this) .
    - **button** should be replaced with the local Button reference you just retrieved.
* Fill in the onClick method stub:
  + Retrieve the user entered text from the text field and keep it in a local String variable
  + Create an android.content.Intent object: new Intent(this, HelloWorld.class) .
    - We'll use the Intent object to start the HelloWorld greeting activity and pass it information
    - *We'll discuss Intents in more depth in later labs, essentially we use them to interact with other application components.*
  + You will use the Intent.putExtra(<key>, <value>) method to pass the user entered name to the HelloWorld greeting Activity. This method functions like a hashmap, where values can be stored by associating them with a string key and later retrieved with the same key. You can retrieve this hashmap later by calling getExtras().
    - Make a call to **<intent>**.putExtra(**<key>**, <**value>**) ,
    - **<intent>** should be replaced with the intent object you just created.
    - **<key>** should be replaced with a string you will use to access the user's name later.
    - **<value>** should be replaced with the users name obtained from the text field. To obtain the text from the EditText object, you must call to **<editText object>.**getText().toString() .
  + Make a call to this.startActivity(**<intent>**)
    - *This command will initiate the switch to the HelloWorld greeting Activity.*

**5.1.3 Reconfigure the HelloWorld Application**

The Android Manifest contains information on all of the Applications components, including which component should be used in different scenarios.We need to tell our application to use the new activity on startup instead of the HelloWorld Activity which it was previously using. Begin by Double-Clicking the AndroidManifest.xml file to open it

* Like the Layout, file there is also a Manifest Editor. Switch to the XML Editor by clicking the **AndroidManifest.xml** tab along the bottom of the editor
* Find the first opening **<activity ... >** tag and change the attribute labeled android:name equal from ".HelloWorld" to "**<New Activity Class>**"
  + Replace **<New Activity Class>** with the full name of your new Activity Class.
  + Look at the android:package attribute in the **<manifest>** tag, this declares the top level package for the application. If your activity resides in a sub-package then you must also include the sub-package in the name of your Activity class.
  + For example:
    - application package name in the manifest tag: android:package="**my.app.basepackage"**
    - "Activity1"
      * fully qualified classpath: **my.app.basepackage.Activity1**
      * activity tag: <activity android:package="**Activity1"**></activity>
    - "Activity2"
      * fully qualified classpath: **my.app.basepackage.subpackage.Activity2**
      * activity tag: <activity android:package="**subpackage .Activity2"**></activity>
  + *The Intent Filter tag you see nested inside the Activity tag is what tells the application that this Activity is the Main, or startup, Activity.*
* Add the following opening and closing Activity tag pair underneath the Activity tag pair you just modified, nested inside Application tag:
  + <activity android:name="HelloWorld" ></activity>
  + *This declares to the Android device that the application has an Activity component named HelloWorld. If you don't add this tag Android will not let you launch this Activity*

At this point, you should be able to try running the application. The application should start up and display the name retrieval activity you just created. You should be able to enter your name in the text field and hit the button, after which the old HelloWorld greeting activity should appear.

**5.2 Greeting the User**

Now that we've got the name retrieval activity completed lets update the HelloWorld greeting Activity to print out the name the user entered. In the OnCreate method of HelloWorld.java:

* Make a call to this.getIntent.getExtras() to retrieve the hashmap in which you placed the user entered name.
  + This will return an android.os.Bundle object which acts like a hashmap.
  + You should check to ensure that this value is not null.
* Retrieve user entered name from the Bundle object.
  + Make a call to the bundle's getString(**<key>**) method.
  + **<key>** should be replaced with the key String you used to put the user entered name into the hashmap in the name retrieval Activity's onClick method.
  + You should check to ensure that this value is not null.
* Get a reference to the TextView object in your main.xml layout file.
* Set the text of the TextView object to say Hello **<name>**!
  + **<name>** should be replaced with the user entered name that you retrieved from the bundle object.
* Run your application.

**6. Exporting Your Application**

In order for your application to be run on a physical device, it must be digitally signed with a certificate. The certificate does not need to be signed by a certificate authority like Verisign. It's acceptable to use a self-signed certificate. As of now, you've only been executing your application on your physical device by launching it through Eclipse. By doing it this way, the Eclipse ADT Plugin has been signing your application with its own "debug" certificate. Signing with the "debug" certificate is not acceptable for release to the general public. In this short section you will learn how to generate a new keystore, compile your application as a ".apk" file, create a certificate, and sign it. For more details on the Application signing you can view the documentation on the Android Developer Site [[click here]](http://developer.android.com/guide/publishing/app-signing.html#overview).

For those of you who haven't taken a Security class or are unfamiliar with the idea of Keys, Certificates and how they work, you can read more information about it on Wikipedia.

The Eclipse ADT plugin provides a simple Export Wizard that automates the entire process for you. Follow the instructions on the Android Developer Site on how to [Compile and sign with Eclipse ADT](http://developer.android.com/guide/publishing/app-signing.html#ExportWizard). Make sure to read the short section following this one on [Securing Your Private Key](http://developer.android.com/guide/publishing/app-signing.html#secure-key). While it's not that necessary for this lab, it will be important when you release an application to the public.

Make sure to keep track of your ".apk" file, you will be handing it in at the end of the lab as proof that you completed the Lab.

### 7. Deliverables

### Put your entire project directory into a .zip file and submit the archive. This effectively provides time-stamped evidence that you submitted the lab on time. The name of your archive should be lab1<yourname>.zip. So if your username is RAM and you created a zip file, then your file would be named lab1RAM.zip.