
globsyn
Taking People To The Next Level

globsyn


globsyn
finishing school

Building the first App

Building the first App

- In this session you will learn to:
  - Create an Android project
  - Run the application
  - Build a simple user interface
  - Start another activity

To Create a Project with Eclipse

- Select the **New** button from the toolbar.
- In the window that appears, open the **Android** folder and select the **Android Application Project** then click **Next**.
- In the window fill up the following tabs:
  - **Application Name** is the app name that appears to users.
  - **Project Name** is the name of project directory and the name visible in Eclipse.
  - **Package Name** is the package namespace for the app and must be unique across all packages.

To Create a Project with Eclipse

- Minimum Required SDK is the lowest version of Android that the app supports, indicated using the API level. To support as many devices as possible, the lowest version available should be selected.
- Target SDK indicates the highest version of Android with which the application has been tested.
- Compile With is the platform version against where the apps is compiled. By default, this is set to the latest version of Android available in your SDK.
- Click Next.

To Create a Project with Eclipse

- The next screen appears to configure the project, leave the default selections and click Next.
- The next screen helps create a launcher icon for the app.
- Then click **Next**.
- An activity template is then selected from which the app can be built.
- For this project, select BlankActivity and click Next.
- The details for the activity are left blank in their default state and now click **Finish**.

Running the Apps

- The directories to be known include the following:
- AndroidManifest.xml The manifest file describes the fundamental characteristics of the app and defines each of its components
- Src Directory for your app's main source file
- Res Contains several sub-directories for app resources.
- In order to run on a real android device:
- The device has to be plugged into the development machine with a USB cable.

Running the Apps

- Now, enable **USB debugging** on the device.
  - On Android 3.2 or older, **Settings > Applications > Development**.
  - On Android 4.0 and newer, **Settings > Developer options**.
- To run the app from Eclipse:
- Open one of the project's files and click **Run** from the toolbar.
- In the **Run as** window that appears, select **Android Application** and click **OK**.
- Eclipse installs the app on the connected device and starts it.

Build a Simple User Interface

- Open the activity_main.xml file from the res/layout/ directory.
- The BlankActivity template you chose when you created this project includes the activity_main.xml file with a RelativeLayout root view and a TextView child view.
- First, delete the TextView element and change the <RelativeLayout> element to <LinearLayout>. Then add the android:orientation attribute and set it to "horizontal"

Build a Simple User Interface

- To create a user-editable text field, add an <EditText>element inside the <LinearLayout>
- Like every View object, you must define certain XML attributes to specify the EditText object's properties. Here's how you should declare it inside the <LinearLayout> element:

Build a Simple User Interface

- When text has to be added to the user interface, each string has to be specified as a resource.
- String resources allow to manage all UI text in a single location, which makes it easier to find and update text.
- Externalizing the strings also allows to localize the app to different languages by providing alternative definitions for each string resource.

Build a Simple User Interface

- Now add a <Button> to the layout, immediately following the <EditText> element. The height and width are set to "wrap_content" so the button is only as big as necessary to fit the button's text
- This layout is applied by the default Activity class that the SDK tools generated when you created the project, so you can now run the app to see the results:
- In Eclipse, click Run from the toolbar.

Complete Layout Coding

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal">
      <EditText android:id="@+id/edit_message"
          android:layout_weight="1"
          android:layout_width="0dp"
          android:layout_height="wrap_content"
          android:hint="@string/edit_message" />
      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/button_send" />
  </LinearLayout>
```

Starting Another Activity

- To respond to the button's on-click event, open theactivity_main.xml layout file and add theandroid:onClick attribute to the <Button> element:
- <Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/button_send"
  android:onClick="sendMessage" />

Starting Another Activity

- An Intent is an object that provides runtime binding between separate components (such as two activities). The Intent represents an app's "intent to do something." You can use intents for a wide variety of tasks, but most often they're used to start another activity.
- Inside the sendMessage() method, create an Intent to start an activity called DisplayMessageActivity:
- Intent intent = new Intent(this, DisplayMessageActivity.class);

Starting Another Activity

- To start an activity, call startActivity() and pass it your Intent. The system receives this call and starts an instance of the Activity specified by the Intent.
- To start an activity, call startActivity() and pass it your Intent . The system receives this call and starts an instance of the Activity specified by the Intent.
- All activities must be declared in your manifest file, AndroidManifest.xml, using an <activity> element.
- In the DisplayMessageActivity class's onCreate() method, get the intent and extract the message delivered by MainActivity:

Starting Another Activity

To show the message on the screen, create a TextViewwidget and set the text using setText(). Then add theTextViewas the root view of the activity's layout by passing it to setContentView().

Reference

http://developer.android.com/training/basics/firstapp/starting-activity.html


Taking People To The Next Level ...