

1

Introduction to Android Programming

Introduction

The goal of this lab 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

- Download Android Studio software.
- Create a "Hello World" Android Application
- Install and run the application on Android Emulator and physical device
- Understand the various parts of an Android Project
- Create a simple User Interface

Download Android Studio Software

Setting up Android Studio takes just a few clicks. (You should have already [downloaded](#) Android Studio.)

To install Android Studio on Windows, proceed as follows:

1. Launch the .exe file you downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

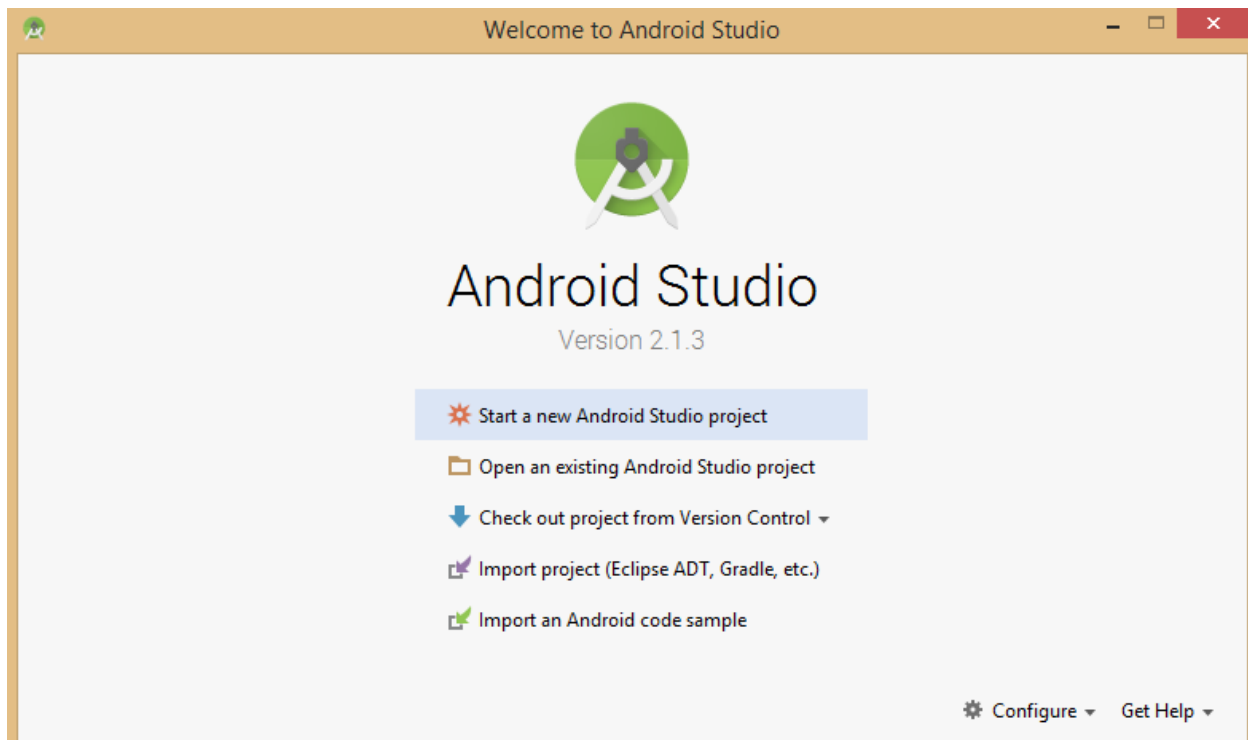


Figure 1 Welcome to Android Studio window

Create "Hello World" Application

This section shows you how to create a new Android project with Android Studio.

1. In Android Studio, create a new project:
 - If you don't have a project opened, in the **Welcome to Android Studio** window, click **Start a new Android Studio project**.
 - If you have a project opened, select **File > New Project**.
2. In the New Project screen, enter the following values:
 - Application Name: "My First App"
 - Company Domain: "example.com"

Android Studio fills in the package name and project location for you, but you can edit these if you'd like.

3. Click **Next**.
4. In the **Target Android Devices** screen, keep the default values and click **Next**.


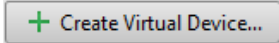
The **Minimum Required SDK** is the earliest version of Android that your app supports, which is indicated by the **API level**. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the core feature set, enable that feature only when running on the versions that support it.

5. In the **Add an Activity to Mobile** screen, select **Empty Activity** and click **Next**.
6. In the **Customize the Activity** screen, keep the default values and click **Finish**.

After some processing, Android Studio opens and displays a "Hello World" app with default files. You will add functionality to some of these files in the following sections.

Run "Hello World" on the Emulator

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

- Click on **AVD Manger** icon in the toolbar. 
- Click on **+ Create Virtual Device...** 
- Select Virtual Devices on the left hand side.
- Click **Next** button.
- Click **Next** button.
- Give your AVD a name.
- Click **finish** and close out the SDK/AVD Manager.

We're now ready to run our application.

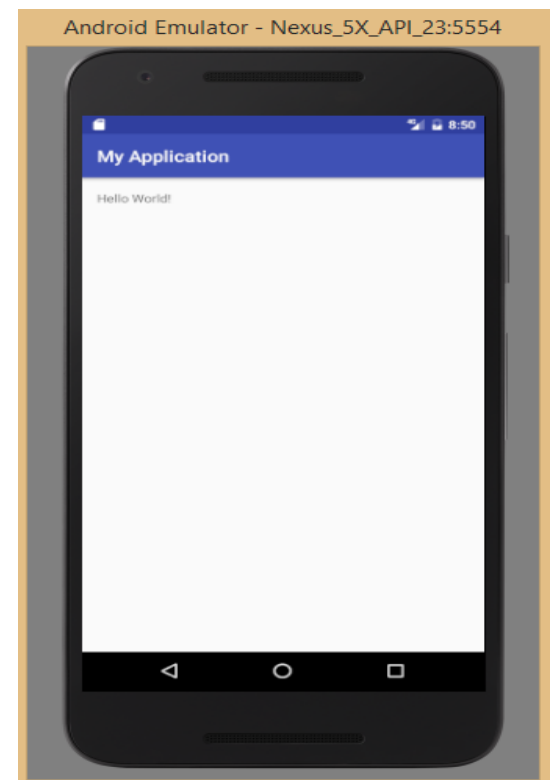


Figure 2 Emulator

Take a Tour of the Application

Now take a moment to review the most important files. First, be sure that the Project window is open (select **View > Tool Windows > Project**) and the Android view is selected from the drop-down list at the top. You can then see the following files:

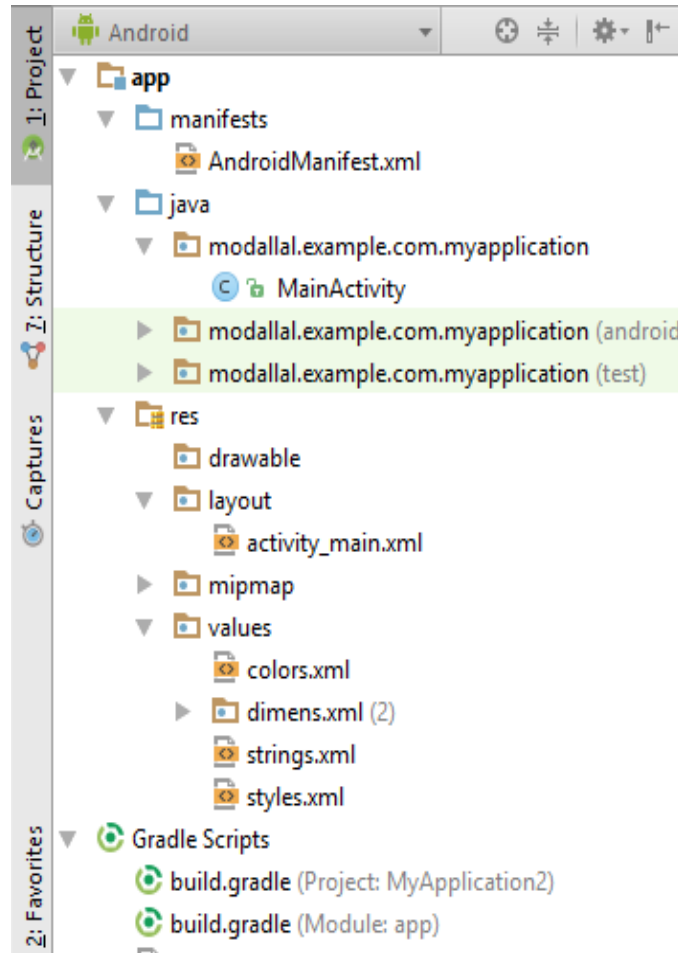


Figure 3 Project files

- ✚ **manifests > AndroidManifest.xml**: Every project has a file with this exact name in the root directory (see Figure 4). 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 Receivers, 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.
 - Application profiling information.
 - Libraries and API levels that the application will use.



Figure 4 AndroidManifest.xml file

✚ java > com.example.myfirstapp > MainActivity.java:

This file appears in Android Studio after the New Project wizard finishes. It contains the class definition for the activity you created earlier. When you build and run the app, the Activity starts and loads the layout file that says "Hello World!"

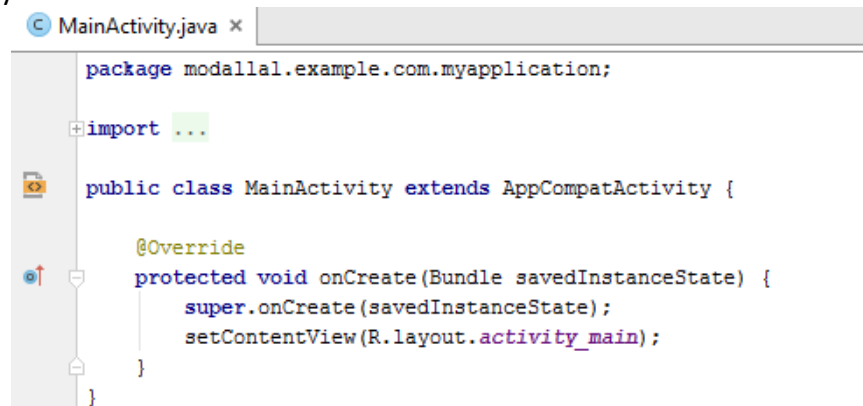


Figure 5 MainActivity.java file

super.onCreate(savedInstanceState);, you tell the Dalvik VM to run your code in addition to the existing code in the onCreate() of the parent class. If you leave out this line, then only your code is run. The existing code is ignored completely. However, you must include this super call in your method, because if you don't then the onCreate() code in Activity is never run, and your app will run into all sorts of problem like having no Context assigned to the Activity (though you'll hit a SuperNotCalledException before you have a chance to figure out that you have no context).

setContentView (int layoutResID) method is used to set the activity content from a layout resource. The resource will be inflated, adding all top-level views to the activity.

- ✚ **res > drawable or mipmap:** This folder will hold image and animation files that you can use in your application.
It already contains a file called **ic_launcher.png** which represents the icon that Android will use for your application once it is installed.
- ✚ **Res > 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 your UI.
 - It already contains a file called **Activitymain.xml** (see Figure 6) 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.

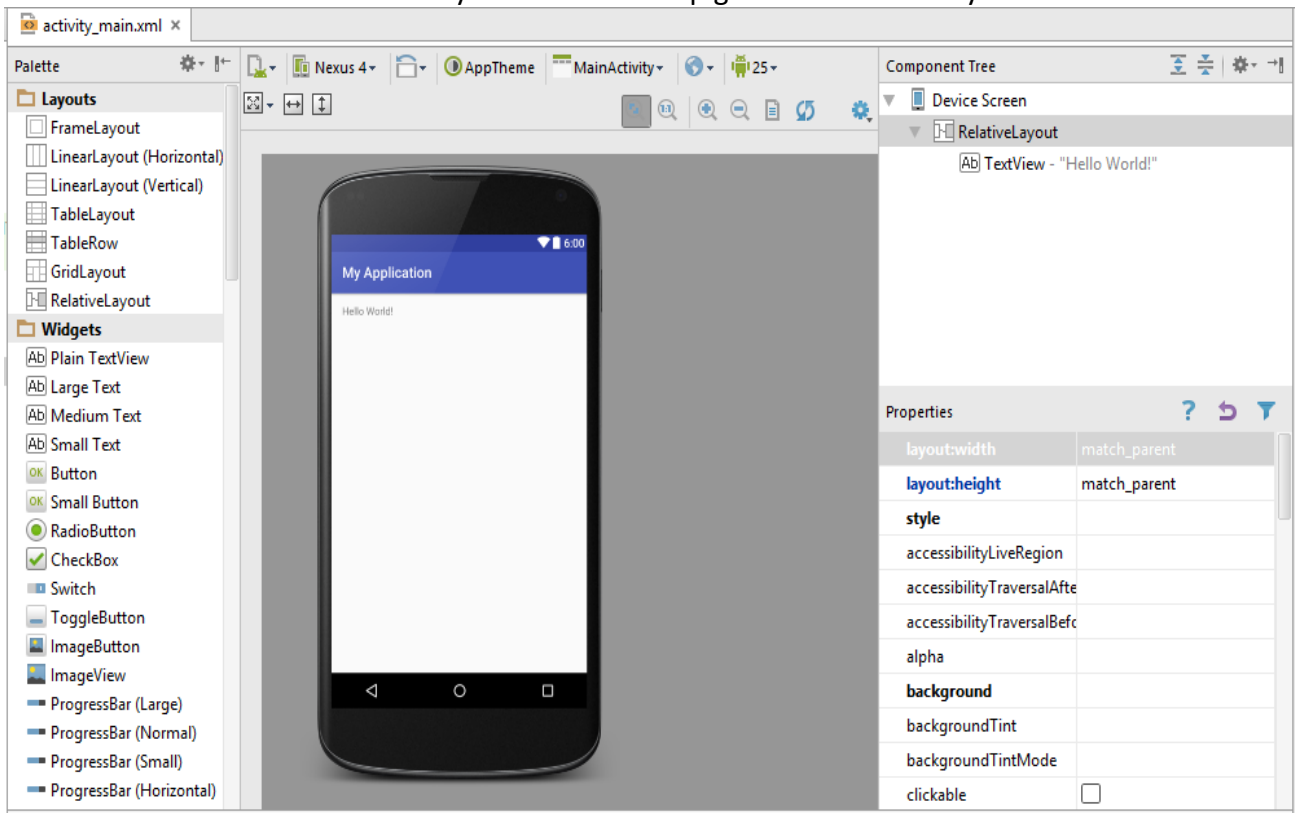



Figure 6 Activity_main.xml file

- ✚ **res > 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.
- ✚ **Gradle Scripts > build.gradle:** Android Studio uses Gradle to compile and build your app. There is a **build.gradle** file for each module of your project, as well as a **build.gradle** file for the entire project. Usually, you're only interested in the **build.gradle** file for the module. in this case the app or application module.

 **R.java:** This 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'.
- A debugging hint: Occasionally, an Android project will report errors in Eclipse that do not show up in any source code file. Sometimes you can fix this by deleting R.java. When you rebuild your project, R.java gets generated, and perhaps your mysterious errors will disappear.

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 an email address.

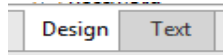
In this lab, we will be focusing on what Activities are and how they are used. We will cover the other components in later labs.

Getting the User's Name Example

In this example, we want to build a simple application to get the user's name from text field and then display it on label. 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. To do that, follow the following steps:

1. Open **Activitymain.xml** file (figure 6).
2. Delete HelloWorld label by pressed on it and click delete.
3. Drag and Drop **LinearLayout(Vertical)** view on empty XML content.
4. Find and drag **Text Field** which exists in Text Fields.
5. Find and drag **Button** which exists in Form Widgets.
6. Find **TextView** in Form Widgets, then drag and drop it.

You can see the code by click on **Text** button in the left corner of **Activitymain.xml** file.



```
<?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"
    tools:context="modallal.example.com.myapplication.MainActivity"
    android:orientation="vertical"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true"
    android:weightSum="1">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:text="Your Name"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button" />

    <TextView
        android:layout_width="206dp"
        android:layout_height="wrap_content"
        android:text="Result"
        android:id="@+id/textView"
        android:layout_weight="0.11" />

</LinearLayout>
```

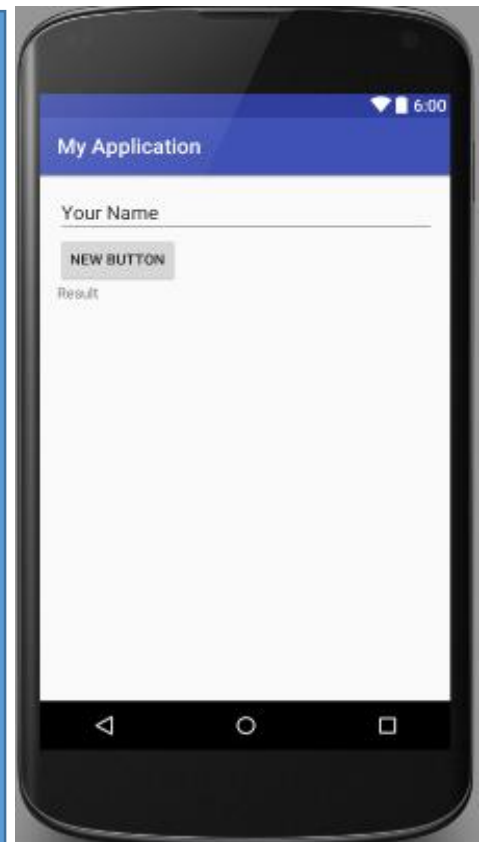


Figure 7 Activitymain.xml file

a) Test

b) Design

7. Open **MainActivity.java** file. The following code is generated when the project was created :

```
package modallal.example.com.myapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

8. Add the following code to existing code.

```
Button btnGetName = (Button) findViewById(R.id.button);
btnGetName.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {
        TextView txtName = (TextView) findViewById(R.id.textView);
        EditText edtName = (EditText) findViewById(R.id.editText);
        txtName.setText(edtName.getText());
    }
});
```

The MainActivity.java file full code:

```
package modallal.example.com.myapplication;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnGetName = (Button) findViewById(R.id.button);
        btnGetName.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0) {
                TextView txtName = (TextView) findViewById(R.id.textView);
                EditText edtName = (EditText) findViewById(R.id.editText);
                txtName.setText(edtName.getText());
            }
        });
    }
}
```

Note: You have to import all necessary libraries. You can do this in easy way by click on the Red word in the code and then press **Alt+Enter**.

9. Run the application.



ToDo

This part will be given to you by the teacher assistant in the lab time.