COM413 – Mobile App Development – Workbook Practical 1

Introduction

This Practical provides an opportunity for you to develop a multi-faceted application. The app must be capable of showing web pages and allowing the user a search function (this may have auto-complete). You must also include a button which allows the user to call and a button for sending a text.

Basic:
Create an app with a search bar the app must have:

- Search functionality
- The ability to display web pages

Advanced:
Create two buttons for:

- Sending a text
- Making a phone call

Additional:
When loading your app, it can be nice to have a logo come up on screen before the app actually starts:

- Add a Splash screen to your app using an Activity.

Below is an online resource to help you with the basics of creating an application:

http://tutorials.jenkov.com/android/core-concepts.html

There are also resources on Blackboard to help.

Basic Functionality:

Introduction

Basic Functionality:

We'll start this practical by looking at the **WebView** for adding browser functionality to your app. At its basic level, the **WebView** is a basic HTML viewer.

Advanced Functionality:

We'll then explore Telephony functionality with ***How to make a phone call***. After exploring how to make a call, we'll then move on to SMS messaging with ***How to send SMS Messages.***

Displaying a web page in your application

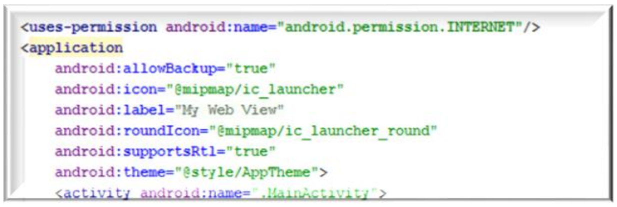
This practical will show how to display a **web page** in your application, you can use the **WebView** for this purpose.

Create a new project in Android Studio using the following details:

- **Application Name**: "My Web View"
- **Activity to Mobile** screen, select **Empty Activity**

How to do it...

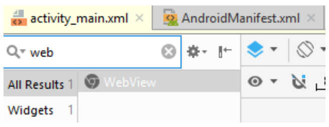
1. We'll start by opening the **Android Manifest** (open **app > manifests > AndroidManifest.xml**) and add the following permission: <uses-permission android:name="android.permission.INTERNET"/>



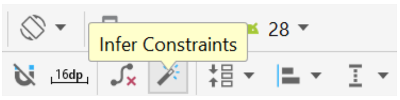
```
<uses-permission android:name="android.permission.INTERNET"/>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="My Web View"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
```

2. Open **activity_main.xml**, you need to remove what's already in the **layout**. So, click **TextView** in the **Component Tree** window, and then press **Delete**.

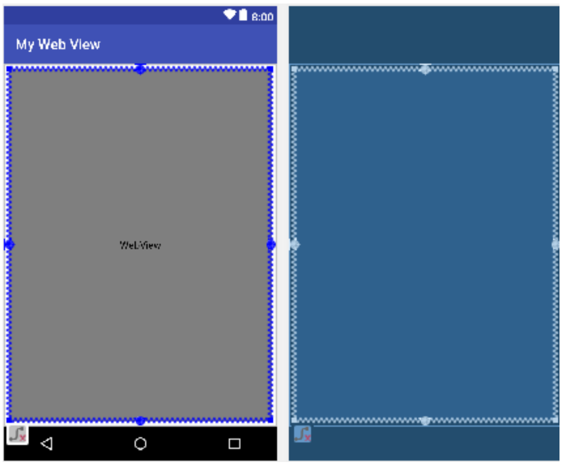
3. Now, we need to add a **WebView**.
   a. Search the word **web** in the **Palette**,



   b. Drag **WebView** into the design editor and drop it.
   c. Click on the **Infer Constraints**



   d. You will find the design view as shown below (or something similar).

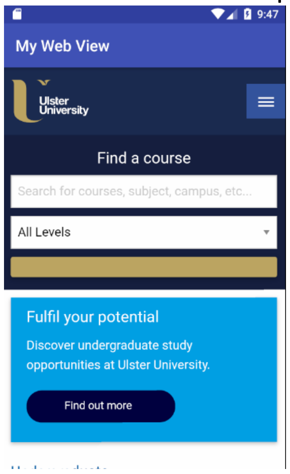


4. In MainActivity, add the following code to the onCreate() method:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    WebView webview = new WebView( context: this);
    setContentView(webview);
    webview.loadUrl("https://www.ulster.ac.uk/");

}
```

5. You're ready to run the application on an Emulator or a mobile device.

How it works...
We create a **WebView** to use as our layout and load our webpage with **loadUrl().** The preceding code works, but at this level, it is very basic and only displays the first page. If you click on any links, the default browser will handle the request.


9:47
My Web View
Ulster University
Find a course
Search for courses, subject, campus, etc...
All Levels
Fulfil your potential
Discover undergraduate study opportunities at Ulster University.
Find out more

Web Search Example

Being able to search the whole world for information is one reason why we use the Internet. After all, it is the World Wide Web. The Android Operating System has web search built into it. Using an Android *ACTION_WEB_SEARCH* Intent, a web search can be started from inside an app.

Putting a Web Search Button in Your App

In this Android web search example, an *ACTION_WEB_SEARCH* Intent is started from a button. This allows an Activity to have a web search button, in this case with the usual magnifying glass icon. This practical covers adding a simple Internet search. If functionality is required to search for items within your app (such as the database) see the Android Developers Search Overview page.

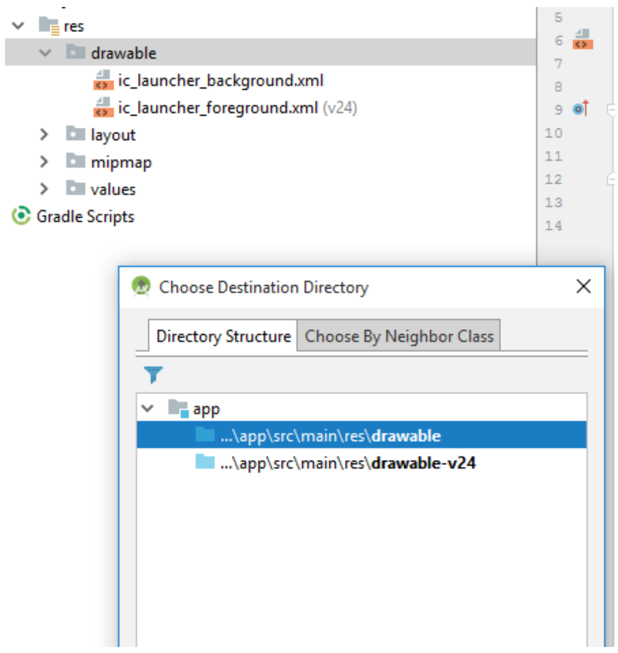
The User Interface Design

For this simple Internet search button practical the user interface (UI) is an EditText (for the search term) and an ImageButton (to start the search). The ImageButton has a magnifying glass icon from a vector image from openclipart.org (a good source of free vector images).

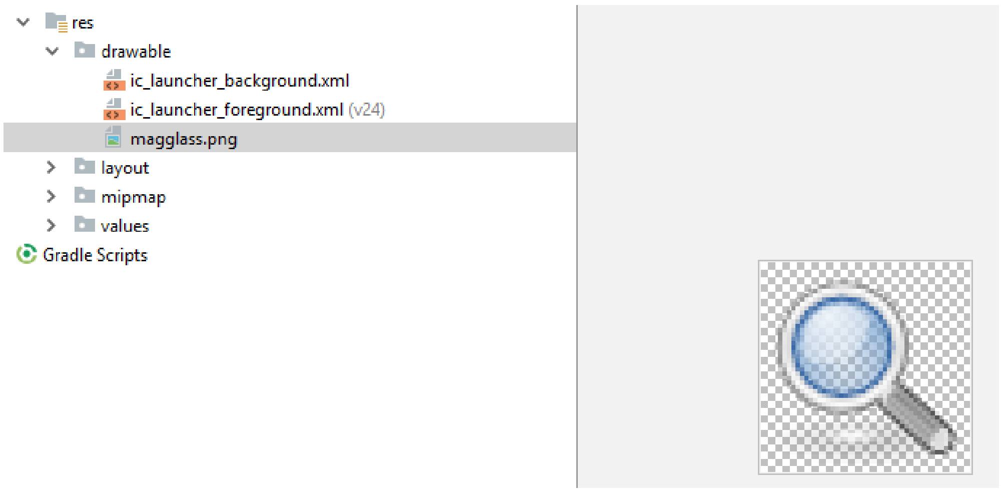
Note: See the tutorial Free Android Icons Using OpenClipArt.org and Paint.NET on how to convert a vector image to the Portable Network Graphics (PNG) formats required for Android apps. The image is placed into the ***res/drawable*** folders for the app. You can of course use you own image for the button or use a simple text Button.

Create a New App Project

In Android Studio, start a new project, here it is called ***Web Search*** and uses an *Empty Activity* with all other starting values left at their default values. Add the icon file for the search button to the ***res/drawable*** folder.


res
drawable
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
mipmap
values
Gradle Scripts
Choose Destination Directory
Directory Structure
Choose By Neighbor Class
app
...\app\src\main\res\drawable
...\app\src\main\res\drawable-v24

When copied into the correct directory, Android Studio will list them in the *Project* explorer (or use *Synchronize* to refresh the project tree).



The ***activity_main.xml*** code should look similar to this:

Text View

ID = textView

Key = text_search

Value = Search For

Plain Text

ID = editText

Key = edit_search

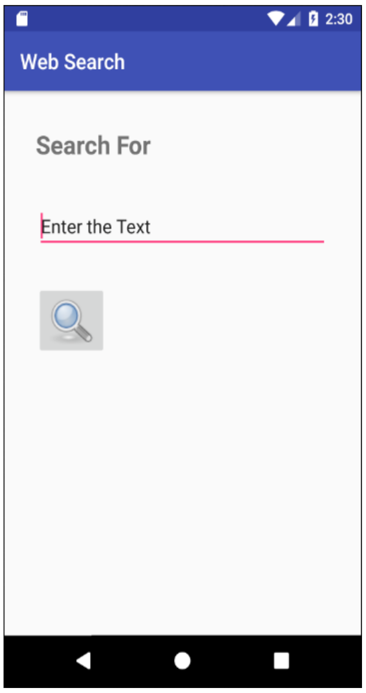
Value = Enter the Text

Image Button

ID = imageButton

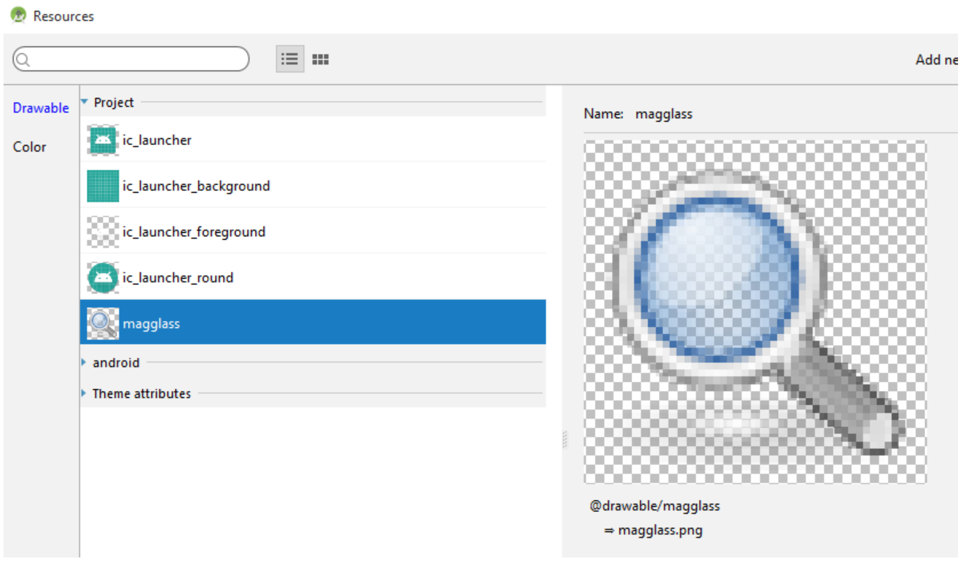
Key = img_button

Value = Image Button

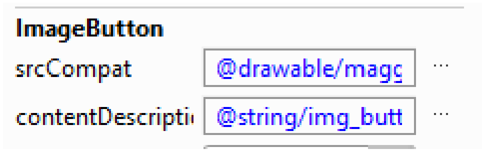


Add an image button

In the **Palette**, click **Buttons** to show the available button controls. Drag **ImageButton** into the design editor and drop it. You need to select an image, select an image from Drawable→Project→magglass



Now click the image button in the layout, locate the **contentDescription** property, click **Pick a Resource** ⋯, select **img_button**

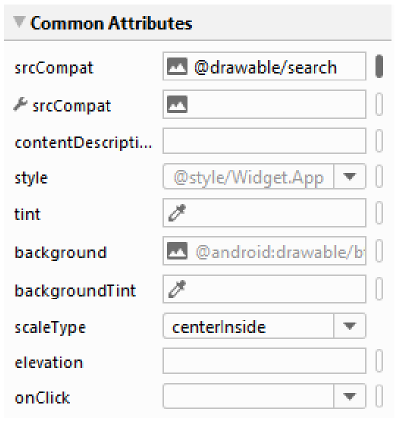


ImageButton

srcCompat	@drawable/magg
contentDescripti	@string/img_butt

Note: After including a **Text View**, a **Plain Text** and an **Image button**, make sure you click the "**Infer Constraints**" button in the Menu bar.

In the attributes make sure you change the **scaleType** to **centerInside:**



Common Attributes

srcCompat	@drawable/search
srcCompat	
contentDescripti...	
style	@style/Widget.App
tint	
background	@android:drawable/b
backgroundTint	
scaleType	centerInside
elevation	
onClick	

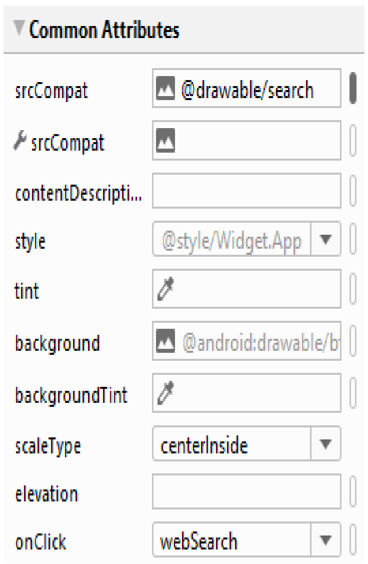
Performing the Search

Intents can reduce the amount of code to be written if that code already exists. For an Internet search, the Android platform supports an Intent of type *ACTION_WEB_SEARCH*. Here the search term is taken from the **EditText** and loaded into the Intent. The Intent is then passed to Android, via ***startActivity()***, to do the browser based search. Here is the code in an ***webSearch()*** that will be linked to the ImageButton (imports for View, EditText, Intent and SearchManager will be required).

Add the Code to Handle the Web Search

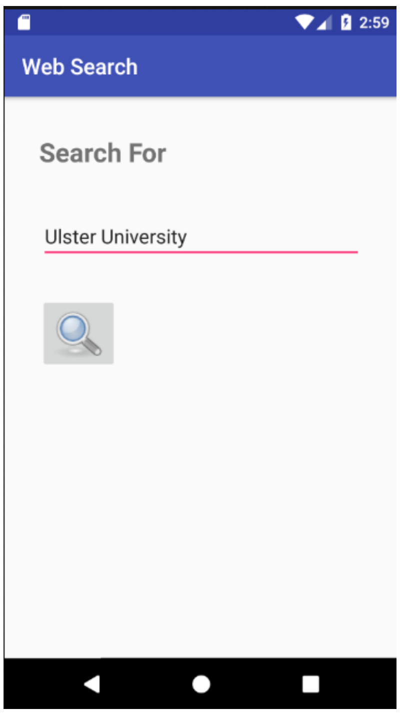
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

public void webSearch(View view) {
    String searchFor = ((EditText) findViewById(R.id.editText)) .getText().toString();
    Intent viewSearch = new Intent(Intent.ACTION_WEB_SEARCH);
    viewSearch.putExtra(SearchManager.QUERY, searchFor);
    startActivity(viewSearch);
}
```

Click on the **image button**, in the attributes, select **webSearch** method from the **onClick** option. You're ready to run the application on a device (Emulator or mobile phone).
This screenshot shows the result of entering *Ulster University* in the **EditText** and pressing the **ImageButton**.



Common Attributes

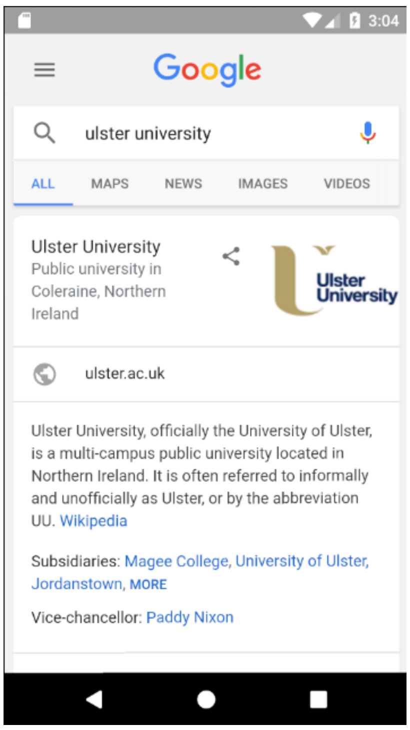
srcCompat	@drawable/search
srcCompat	
contentDescripti...	
style	@style/Widget.App
tint	
background	@android:drawable/b
backgroundTint	
scaleType	centerInside
elevation	
onClick	webSearch



Web Search

Search For

Ulster University



Ulster University
Public university in Coleraine, Northern Ireland

ulster.ac.uk

Ulster University, officially the University of Ulster, is a multi-campus public university located in Northern Ireland. It is often referred to informally and unofficially as Ulster, or by the abbreviation UU. Wikipedia

Subsidiaries: Magee College, University of Ulster, Jordanstown, MORE

Vice-chancellor: Paddy Nixon

Advanced Functionality:

How to make a phone call

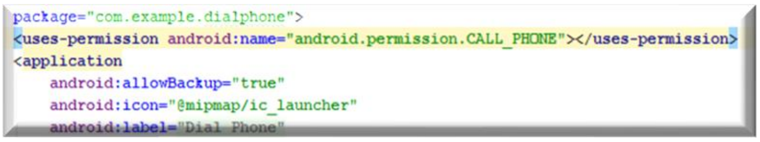
To make a phone call, use Intent.ACTION_DIAL when creating an Intent. You can include a phone number with the setData() method. Here is a sample code that will call up the Dialer app with the phone number specified:

```
Intent intent = new Intent(Intent.ACTION_DIAL);
intent.setData(Uri.parse("tel:" + number));
startActivity(intent);
```

First, we need to add the appropriate permission to make the call. Then, we need to add a **button** to call our Dial method.

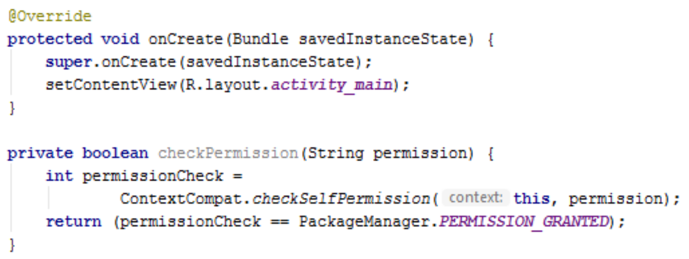
1) Open the **Android Manifest** and add the following permission:

```
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
```



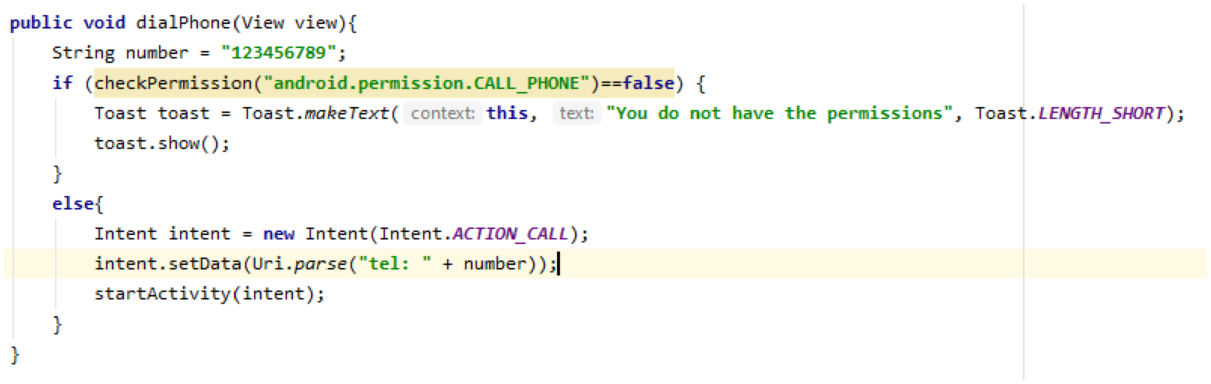
```
package="com.example.dialphone">
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Dial Phone"
```

2) Add this method to **MainActivity.java** that will check whether your app has been granted the CALL_PHONE permission:



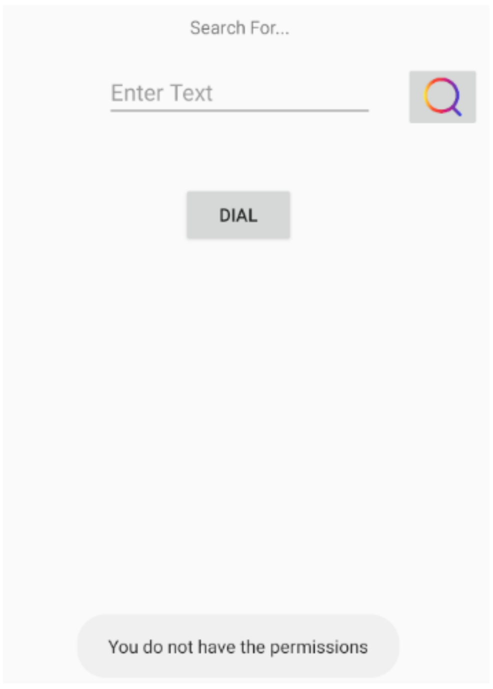
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

private boolean checkPermission(String permission) {
    int permissionCheck =
            ContextCompat.checkSelfPermission( context: this, permission);
    return (permissionCheck == PackageManager.PERMISSION_GRANTED);
}
```

3) Create a method **dialPhone()** and add the code to dial the number
   There are two things to note here:
   1. We need to check the permissions to allow the app access to Android Phone
   2. We need to check if permission is granted



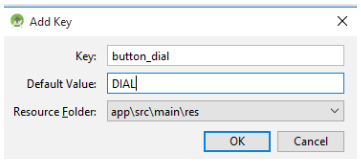
```
public void dialPhone(View view){
    String number = "123456789";
    if (checkPermission("android.permission.CALL_PHONE")==false) {
        Toast toast = Toast.makeText( context: this,  text: "You do not have the permissions", Toast.LENGTH_SHORT);
        toast.show();
    }
    else{
        Intent intent = new Intent(Intent.ACTION_CALL);
        intent.setData(Uri.parse("tel: " + number));
        startActivity(intent);
    }
}
```

So create a new function which will call **checkPermission**(). If the checkPermission is false i.e. the app is denied function we will use a toast to notify the user.



You do not have the permissions

4) Open **activity_main.xml**, you need to add a button to the **layout**.
5) In the **Palette**, click **Buttons**. Drag **Button** into the design editor and drop it near the centre.
    a. Open the **Project** window and then open **app > res > values > strings.xml**. Click **Open editor,** Click **Add Key** to create a new string



Add Key

Key: button_dial

Default Value: DIAL

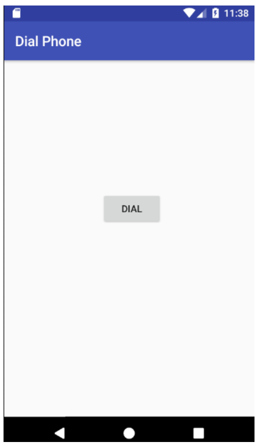
Resource Folder: app\src\main\res

    b. In the **Attributes** window, locate the **onClick** property and select **dialPhone [MainActivity]** from the drop-down list.
    c. Locate the **text** property (currently set to "Button"), click **Pick a Resource** ..., and then select **button_dial**.

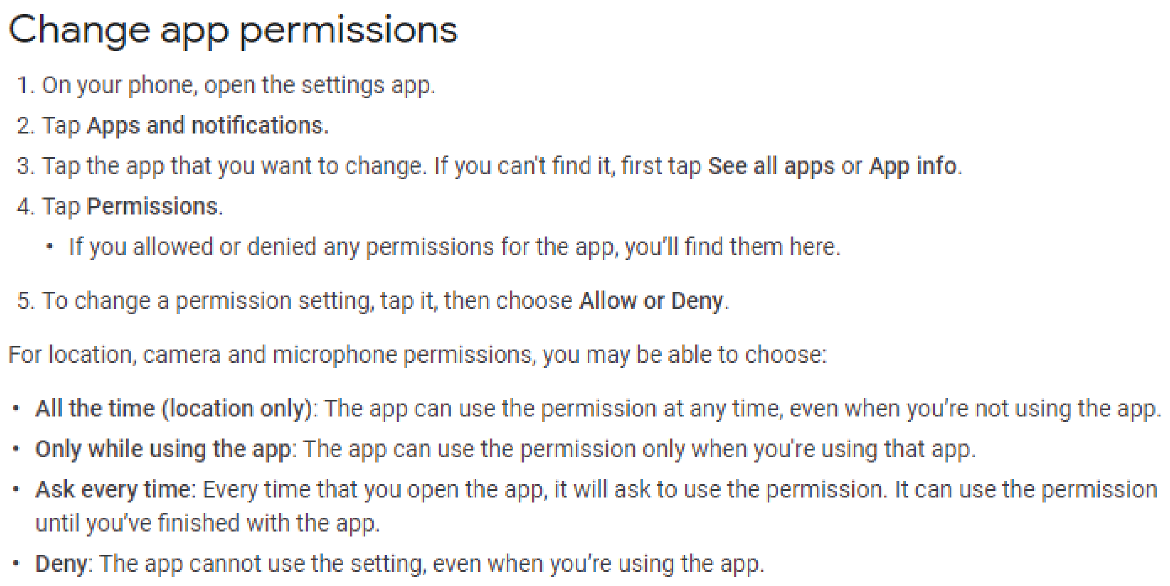
6) The application is ready to run.
7) Before running this app on your device, be sure to replace **123456789** with a valid number.

If we want to dial a number directly, we need to add the **CALL_PHONE** permission.

<u>Note:</u> Starting with Android 6.0 Marshmallow (API 23), permissions are no longer granted during installation, therefore, we need to check whether the application has permission before attempting to dial.



If we don't have the permissions follow these steps and allow the app to access the phone app:

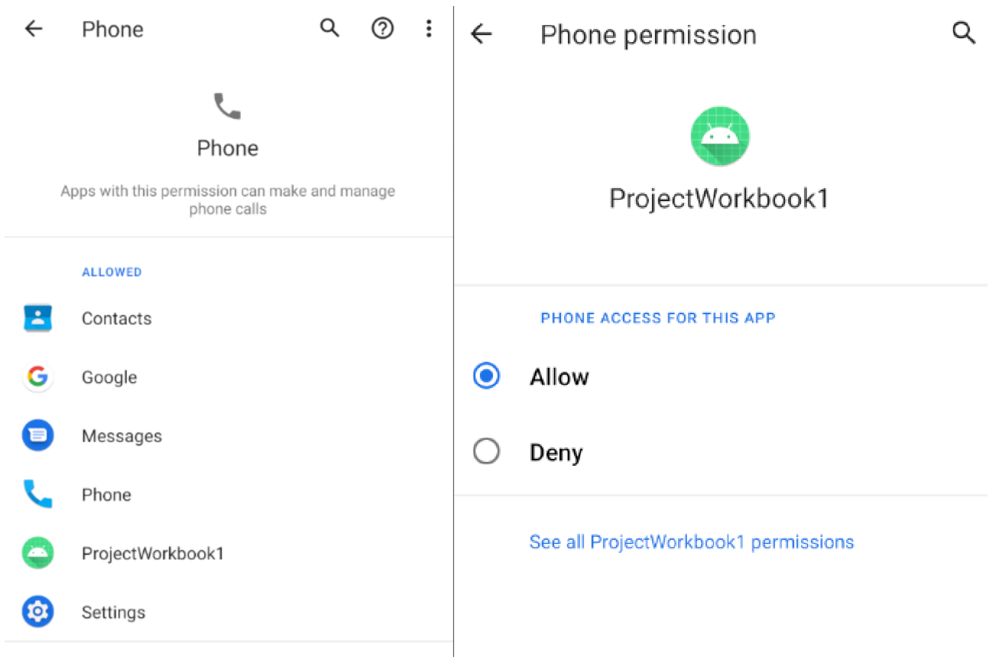


Change app permissions

1. On your phone, open the settings app.
2. Tap **Apps and notifications.**
3. Tap the app that you want to change. If you can't find it, first tap **See all apps** or **App info**.
4. Tap **Permissions**.
   - If you allowed or denied any permissions for the app, you'll find them here.
5. To change a permission setting, tap it, then choose **Allow or Deny**.

For location, camera and microphone permissions, you may be able to choose:

- **All the time (location only)**: The app can use the permission at any time, even when you're not using the app.
- **Only while using the app**: The app can use the permission only when you're using that app.
- **Ask every time**: Every time that you open the app, it will ask to use the permission. It can use the permission until you've finished with the app.
- **Deny**: The app cannot use the setting, even when you're using the app.


Phone

Phone

Apps with this permission can make and manage phone calls

ALLOWED

Contacts

Google

Messages

Phone

ProjectWorkbook1

Settings

Phone permission

ProjectWorkbook1

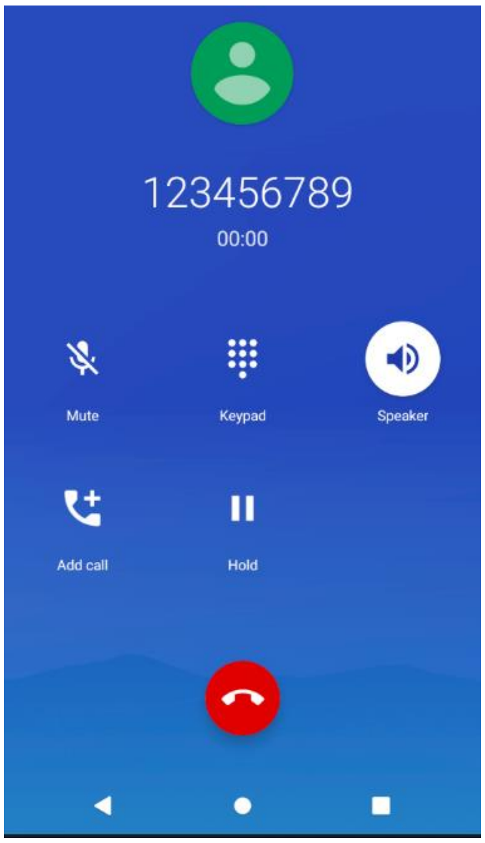
PHONE ACCESS FOR THIS APP

[✓ Allow]

Deny

See all ProjectWorkbook1 permissions

Then when we rerun the app, your app will attempt to dial a number:



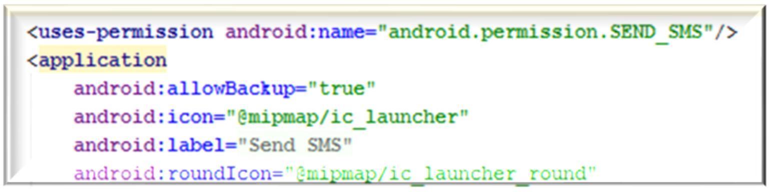
How to send SMS (text) messages

We will demonstrate how to send a SMS Message.

Create a new project.

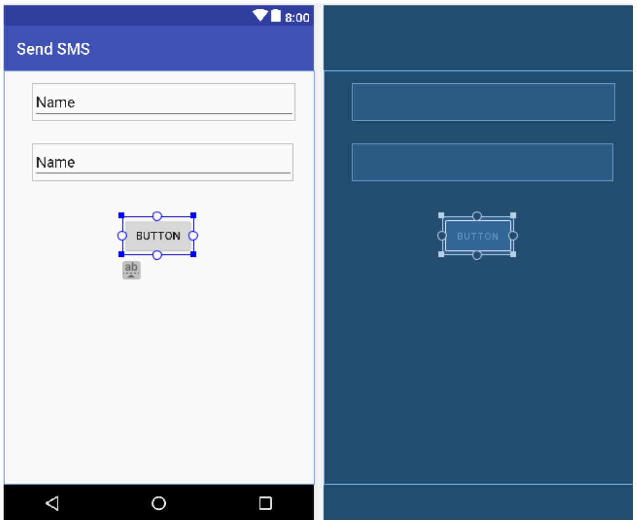
First, we'll add the necessary permissions for sending an SMS. Then, we'll create a layout with a **Phone Number,** a **Message field** and a **Send button**. When the **Send button** is clicked on, we'll create and send the SMS. Here are the steps:

1. Open the Android Manifest and add the following permission:
   <uses-permission android:name="android.permission.SEND_SMS"/>



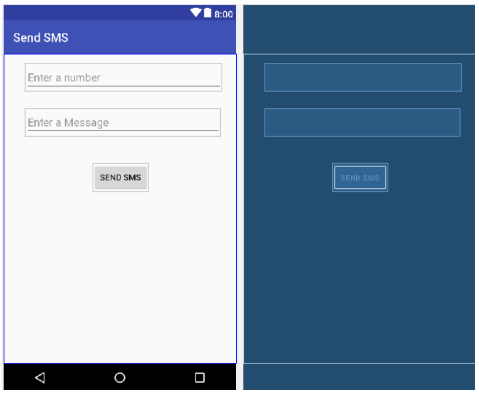
```
<uses-permission android:name="android.permission.SEND_SMS"/>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Send SMS"
    android:roundIcon="@mipmap/ic_launcher_round"
```

2. Open **activity_main.xml**, you need to remove what's already in the **layout**. So, click **TextView** in the **Component Tree** window, and then press **Delete**.
3. Add two **Plain Text** boxes and one **button** from the **Palette**.



4. Open the **Project** window and then open **app > res > values > strings.xml**. Click **Open editor,** Click **Add Key** to create a new string. In the add Key dialog,
   a. For the first text box, enter key "**edit_number**" with the value of "**Enter a number**".
   b. For the second text box, enter key "**edit_message**" with a value of "**Enter a message**".
   c. For the button, enter key "**button_send**" with a value of "**Send SMS**".
5. Now you can set these strings for each view. So, return to the layout file by clicking **activity_main.xml** in the tab bar, and add the strings as follows:

6. Click the text box in the layout and, locate the **text** property (currently set to "**Name**") and delete the value. Locate the **hint** property and then click **Pick a Resource** ··· to the right of the text box.
7. Now click the button in the layout, locate the **text** property (currently set to "**Button**"), click **Pick a Resource**, ···
8. After completing the step 7, you will see the design layout as show below (or something similar)



9. Now we need to change the ID values in the Attributes window:
   - click on "**Enter a number**" Text Box, change the ID to "editTextNumber"
   - click on "**Enter a Message**" Text Box, change the ID to "editTextMsg"
   - click on "**SEND SMS**" Button, change the ID to "buttonSend"

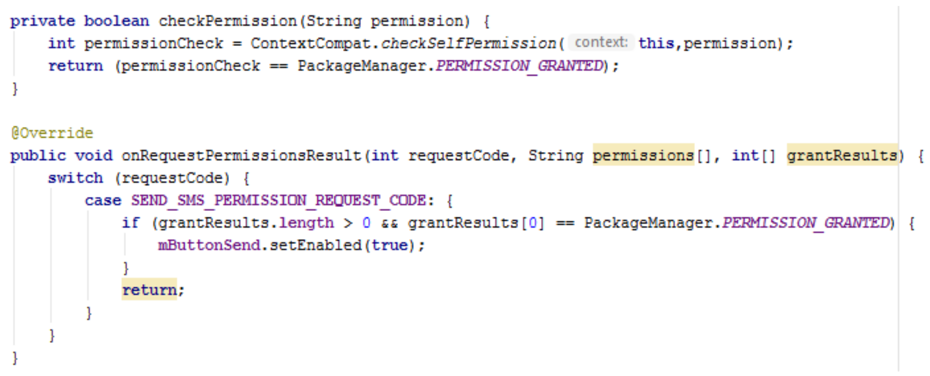
10. Click the **Infer Constraints** from the menu bar to provide appropriate constraints layout.
11. Open MainActivity.java and add the following **global variables**:

```
public class MainActivity extends AppCompatActivity {

    final int SEND_SMS_PERMISSION_REQUEST_CODE =1 ;
    Button mButtonSend;
```

12. Add the following method to check the permission:

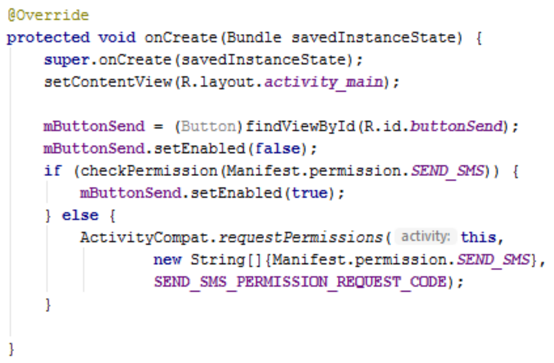
```
final int SEND_SMS_PERMISSION_REQUEST_CODE =1 ;
Button mButtonSend;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

private boolean checkPermission(String permission) {
    int permissionCheck = ContextCompat.checkSelfPermission( context: this,permission);
    return (permissionCheck == PackageManager.PERMISSION_GRANTED);
}
```

13. Override onRequestPermissionsResult() to handle the permission request response:



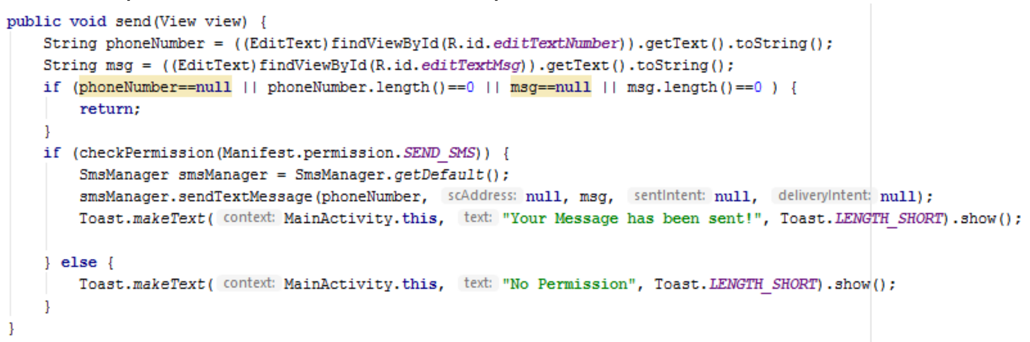
```java
private boolean checkPermission(String permission) {
    int permissionCheck = ContextCompat.checkSelfPermission( context: this,permission);
    return (permissionCheck == PackageManager.PERMISSION_GRANTED);
}

@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    switch (requestCode) {
        case SEND_SMS_PERMISSION_REQUEST_CODE: {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                mButtonSend.setEnabled(true);
            }
            return;
        }
    }
}
```

14. Add the following code to the existing onCreate() callback:



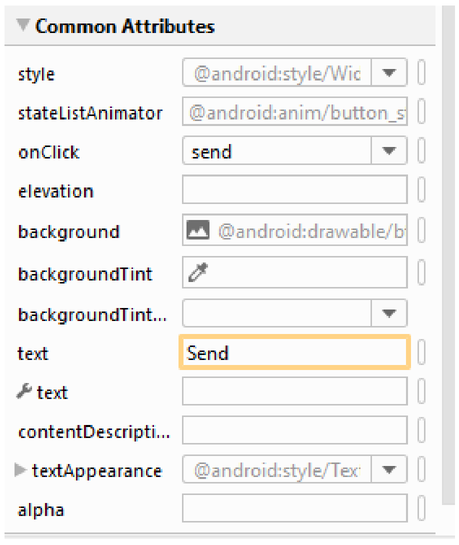
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mButtonSend = (Button)findViewById(R.id.buttonSend);
    mButtonSend.setEnabled(false);
    if (checkPermission(Manifest.permission.SEND_SMS)) {
        mButtonSend.setEnabled(true);
    } else {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{Manifest.permission.SEND_SMS},
                SEND_SMS_PERMISSION_REQUEST_CODE);
    }

}
```

15. And finally, add the method to actually send the SMS:



```java
public void send(View view) {
    String phoneNumber = ((EditText)findViewById(R.id.editTextNumber)).getText().toString();
    String msg = ((EditText)findViewById(R.id.editTextMsg)).getText().toString();
    if (phoneNumber==null || phoneNumber.length()==0 || msg==null || msg.length()==0 ) {
        return;
    }
    if (checkPermission(Manifest.permission.SEND_SMS)) {
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(phoneNumber,  scAddress: null, msg,  sentIntent: null,  deliveryIntent: null);
        Toast.makeText( context: MainActivity.this,  text: "Your Message has been sent!", Toast.LENGTH_SHORT).show();

    } else {
        Toast.makeText( context: MainActivity.this,  text: "No Permission", Toast.LENGTH_SHORT).show();
    }
}
```

16. Click on the "**SEND SMS**" button, in the attributes, select **send** method from the **onClick** option.



Common Attributes

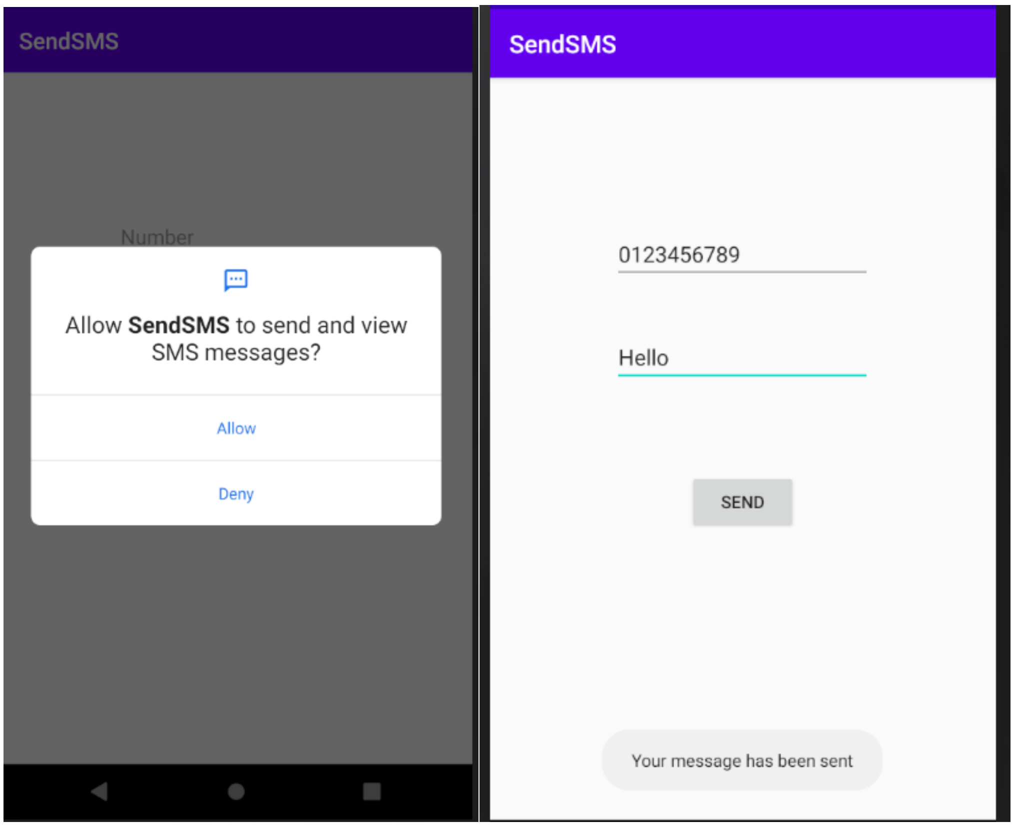
style	@android:style/Wic
stateListAnimator	@android:anim/button_s
onClick	send
elevation	
background	@android:drawable/b
backgroundTint	
backgroundTint...	
text	Send
text	
contentDescripti...	
textAppearance	@android:style/Tex
alpha	

17. You're ready to run the application on a device

How it works...
The code for sending an SMS is only two lines, as shown here:

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNumber, null, msg, null, null);
```

The sendTextMessage() method does the actual sending. Most of the code for this tutorial is to set up the permissions since the permission model was changed in Android 6.0 Marshmallow (API 23).



Your message has been sent

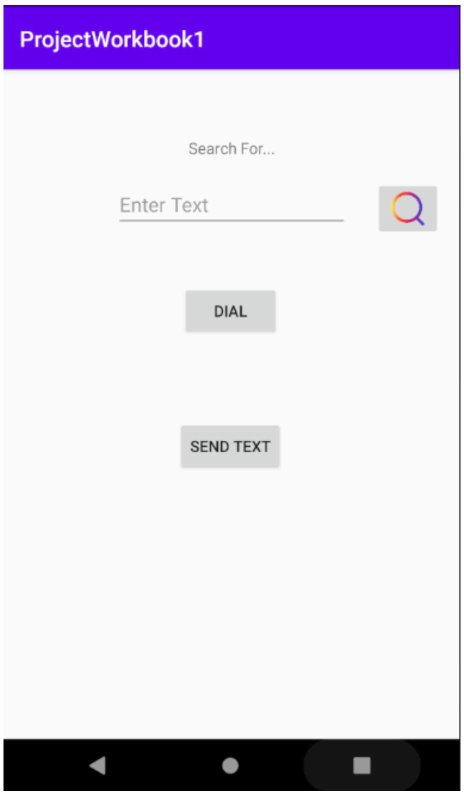
For more details - SMSManager at

https://developer.android.com/reference/android/telephony/SmsManager.html

Additional Functionality:

In the Additional aspect of this practical you have two exercises. First of all, look at the **Adding Another Activity Practical.**

Right now, you should have an App that can perform web searches and dial outwards to make calls. You should also have an app that can send text messages. For the first part of the additional work, combine the two apps you have created, that is create a new button beside **DIAL** called **SMS**, when the user **presses SMS** – start a new activity and in this activity create your Send SMS XML. It should look like this:



The Second Part of this Additional Functionality:
Create a **Splashscreen**. You can do this in a number of ways, first of all Google what a **Splashscreen** is, and secondly add this to your **main app** which should now have **web search** functionality, make **outward calls** and **send SMS** messages.

You can create a new activity which will load a splashscreen XML and when 2/3 seconds has passed, use that activity to automatically call a new activity to the main app.