CHAPTER

12

Intents and Filters

An Android **Intent** is an object carrying an *intent* ie. message from one component to another component with-in the application or outside the application. The intents can communicate messages among any of the three core components of an application - activities, services, and broadcast receivers.
The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.
For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

For example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web Browser Activity.

There are separate mechanisms for delivering intents to each type of component - activities, services, and broadcast receivers.

S.N.	Method & Description
1	**Context.startActivity()** The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.
2	**Context.startService()** The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.
3	**Context.sendBroadcast()** The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

Intent Objects

An Intent object is a bundle of information which is used by the component that receives the intent plus information used by the Android system.

An Intent object can contain the following components based on what it is communicating or going to perform:

ACTION

This is mandatory part of the Intent object and is a string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured . The Intent class defines a number of action constants corresponding to different intents. Here is a list of Android Intent Standard Actions

Android Intent Standard Actions:

Following table lists down various important Android Intent Standard Actions. You can check Android Official Documentation for a complete list of Actions:

S.N.	Activity Action Intent & Description
1	**ACTION_ALL_APPS** List all the applications available on the device.
2	**ACTION_ANSWER** Handle an incoming phone call.
3	**ACTION_ATTACH_DATA** Used to indicate that some piece of data should be attached to some other place
4	**ACTION_BATTERY_CHANGED** This is a sticky broadcast containing the charging state, level, and other information about the battery.
5	**ACTION_BATTERY_LOW** This broadcast corresponds to the "Low battery warning" system dialog.
6	**ACTION_BATTERY_OKAY** This will be sent after ACTION_BATTERY_LOW once the battery has gone back up to an okay state.
7	**ACTION_BOOT_COMPLETED** This is broadcast once, after the system has finished booting.
8	**ACTION_BUG_REPORT** Show activity for reporting a bug.
9	**ACTION_CALL** Perform a call to someone specified by the data.
10	**ACTION_CALL_BUTTON** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
11	**ACTION_CAMERA_BUTTON** The "Camera Button" was pressed.
12	**ACTION_CHOOSER** Display an activity chooser, allowing the user to pick what they want to before proceeding.
13	**ACTION_CONFIGURATION_CHANGED** The current device Configuration (orientation, locale, etc) has changed.
14	**ACTION_DATE_CHANGED** The date has changed.
15	**ACTION_DEFAULT** A synonym for ACTION_VIEW, the "standard" action that is performed on a piece of data.

16	**ACTION_DELETE** Delete the given data from its container.
17	**ACTION_DEVICE_STORAGE_LOW** A sticky broadcast that indicates low memory condition on the device.
18	**ACTION_DEVICE_STORAGE_OK** Indicates low memory condition on the device no longer exists.
19	**ACTION_DIAL** Dial a number as specified by the data.
20	**ACTION_DOCK_EVENT** A sticky broadcast for changes in the physical docking state of the device.
21	**ACTION_DREAMING_STARTED** Sent after the system starts dreaming.
22	**ACTION_DREAMING_STOPPED** Sent after the system stops dreaming.
23	**ACTION_EDIT** Provide explicit editable access to the given data.
24	**ACTION_FACTORY_TEST** Main entry point for factory tests.
25	**ACTION_GET_CONTENT** Allow the user to select a particular kind of data and return it.
26	**ACTION_GTALK_SERVICE_CONNECTED** A GTalk connection has been established.
27	**ACTION_GTALK_SERVICE_DISCONNECTED** A GTalk connection has been disconnected.
28	**ACTION_HEADSET_PLUG** Wired Headset plugged in or unplugged.
29	**ACTION_INPUT_METHOD_CHANGED** An input method has been changed.
30	**ACTION_INSERT** Insert an empty item into the given container.
31	**ACTION_INSERT_OR_EDIT** Pick an existing item, or insert a new item, and then edit it.
32	**ACTION_INSTALL_PACKAGE** Launch application installer.
33	**ACTION_LOCALE_CHANGED** The current device's locale has changed.
34	**ACTION_MAIN** Start as a main entry point, does not expect to receive data.
35	**ACTION_MEDIA_BUTTON** The "Media Button" was pressed.

36	**ACTION_MEDIA_CHECKING** External media is present, and being disk-checked.
37	**ACTION_MEDIA_EJECT** User has expressed the desire to remove the external storage media.
38	**ACTION_MEDIA_REMOVED** External media has been removed.
39	**ACTION_NEW_OUTGOING_CALL** An outgoing call is about to be placed.
40	**ACTION_PASTE** Create a new item in the given container, initializing it from the current contents of the clipboard.
41	**ACTION_POWER_CONNECTED** External power has been connected to the device.
42	**ACTION_REBOOT** Have the device reboot. This is only for use by system code.
43	**ACTION_RUN** Run the data, whatever that means.
44	**ACTION_SCREEN_OFF** Sent after the screen turns off.
45	**ACTION_SCREEN_ON** Sent after the screen turns on.
46	**ACTION_SEARCH** Perform a search.
47	**ACTION_SEND** Deliver some data to someone else.
48	**ACTION_SENDTO** Send a message to someone specified by the data.
49	**ACTION_SEND_MULTIPLE** Deliver multiple data to someone else.
50	**ACTION_SET_WALLPAPER** Show settings for choosing wallpaper.
51	**ACTION_SHUTDOWN** Device is shutting down.
52	**ACTION_SYNC** Perform a data synchronization.
53	**ACTION_TIMEZONE_CHANGED** The timezone has changed.
54	**ACTION_TIME_CHANGED** The time was set.
55	**ACTION_VIEW** Display the data to the user.

56	**ACTION_VOICE_COMMAND** Start Voice Command.
57	**ACTION_WALLPAPER_CHANGED** The current system wallpaper has changed.
58	**ACTION_WEB_SEARCH** Perform a web search.

The action in an Intent object can be set by the setAction() method and read by getAction().

DATA

The URI of the data to be acted on and the MIME type of that data. For example, if the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.

The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().

Some examples of action/data pairs are:

S.N.	Action/Data Pair & Description
1	**ACTION_VIEW content://contacts/people/1** Display information about the person whose identifier is "1".
2	**ACTION_DIAL content://contacts/people/1** Display the phone dialer with the person filled in.
3	**ACTION_VIEW tel:123** Display the phone dialer with the given number filled in.
4	**ACTION_DIAL tel:123** Display the phone dialer with the given number filled in.
5	**ACTION_EDIT content://contacts/people/1** Edit information about the person whose identifier is "1".
6	**ACTION_VIEW content://contacts/people/** Display a list of people, which the user can browse through.

CATEGORY

The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object. Here is a list of Android Intent Standard Categories.

Android Intent Standard Categories

Following table lists down various important Android Intent Standard Categories. You can check Android Official Documentation for a complete list of Categories:

S.N.	Categories & Description
1	**CATEGORY_APP_BROWSER**

	Used with ACTION_MAIN to launch the browser application.
2	**CATEGORY_APP_CALCULATOR** Used with ACTION_MAIN to launch the calculator application.
3	**CATEGORY_APP_CALENDAR** Used with ACTION_MAIN to launch the calendar application.
4	**CATEGORY_APP_CONTACTS** Used with ACTION_MAIN to launch the contacts application.
5	**CATEGORY_APP_EMAIL** Used with ACTION_MAIN to launch the email application.
6	**CATEGORY_APP_GALLERY** Used with ACTION_MAIN to launch the gallery application.
7	**CATEGORY_APP_MAPS** Used with ACTION_MAIN to launch the maps application.
8	**CATEGORY_APP_MARKET** This activity allows the user to browse and download new applications.
9	**CATEGORY_APP_MESSAGING** Used with ACTION_MAIN to launch the messaging application.
10	**CATEGORY_APP_MUSIC** Used with ACTION_MAIN to launch the music application.
11	**CATEGORY_BROWSABLE** Activities that can be safely invoked from a browser must support this category.
12	**CATEGORY_CAR_DOCK** An activity to run when device is inserted into a car dock.
13	**CATEGORY_CAR_MODE** Used to indicate that the activity can be used in a car environment.
14	**CATEGORY_DEFAULT** Set if the activity should be an option for the default action (center press) to perform on a piece of data.
15	**CATEGORY_DESK_DOCK** An activity to run when device is inserted into a car dock.
16	**CATEGORY_DEVELOPMENT_PREFERENCE** This activity is a development preference panel.
17	**CATEGORY_EMBED** Capable of running inside a parent activity container.
18	**CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST** To be used as code under test for framework instrumentation tests.
19	**CATEGORY_HE_DESK_DOCK** An activity to run when device is inserted into a digital (high end) dock.
20	**CATEGORY_HOME** This is the home activity, that is the first activity that is displayed when the device boots.

21	**CATEGORY_INFO** Provides information about the package it is in.
22	**CATEGORY_LAUNCHER** Should be displayed in the top-level launcher.
23	**CATEGORY_LE_DESK_DOCK** An activity to run when device is inserted into a analog (low end) dock.
24	**CATEGORY_MONKEY** This activity may be exercised by the monkey or other automated test tools.
25	**CATEGORY_OPENABLE** Used to indicate that a GET_CONTENT intent only wants URIs that can be opened with ContentResolver.openInputStream.
26	**CATEGORY_PREFERENCE** This activity is a preference panel.
27	**CATEGORY_TAB** Intended to be used as a tab inside of a containing TabActivity.
28	**CATEGORY_TEST** To be used as a test (not part of the normal user experience).
29	**CATEGORY_UNIT_TEST** To be used as a unit test (run through the Test Harness).

You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate acivity coressponding to an Intent.

EXTRAS

This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively. Here is a list of Android Intent Standard Extra Data

Android Intent standard Extra Data

Following table lists down various important Android Intent Standard Extra Data. You can check Android Official Documentation for a complete list of Extra Data:

S.N.	Extra Data & Description
1	**EXTRA_ALARM_COUNT** Used as an int extra field in AlarmManager intents to tell the application being invoked how many pending alarms are being delievered with the intent.
2	**EXTRA_ALLOW_MULTIPLE** Used to indicate that a ACTION_GET_CONTENT intent can allow the user to select and return multiple

	items.
3	**EXTRA_ALLOW_REPLACE** Used as a boolean extra field with ACTION_INSTALL_PACKAGE to install a package.
4	**EXTRA_BCC** A String[] holding e-mail addresses that should be blind carbon copied.
5	**EXTRA_CC** A String[] holding e-mail addresses that should be carbon copied.
6	**EXTRA_CHANGED_COMPONENT_NAME_LIST** This field is part of ACTION_PACKAGE_CHANGED, and contains a string array of all of the components that have changed.
7	**EXTRA_DATA_REMOVED** Used as a boolean extra field in ACTION_PACKAGE_REMOVED intents to indicate whether this represents a full uninstall or a partial uninstall
8	**EXTRA_DOCK_STATE** Used as an int extra field in ACTION_DOCK_EVENT intents to request the dock state.
9	**EXTRA_DOCK_STATE_CAR** Used as an int value for EXTRA_DOCK_STATE to represent that the phone is in a car dock.
10	**EXTRA_DOCK_STATE_DESK** Used as an int value for EXTRA_DOCK_STATE to represent that the phone is in a desk dock.
11	**EXTRA_EMAIL** A String[] holding e-mail addresses that should be delivered to.
12	**EXTRA_HTML_TEXT** A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
13	**EXTRA_INTENT** An Intent describing the choices you would like shown with ACTION_PICK_ACTIVITY.
14	**EXTRA_KEY_EVENT** A KeyEvent object containing the event that triggered the creation of the Intent it is in.
15	**EXTRA_LOCAL_ONLY** Used to indicate that a ACTION_GET_CONTENT intent should only return data that is on the local device.
16	**EXTRA_ORIGINATING_URI** Used as a URI extra field with ACTION_INSTALL_PACKAGE and ACTION_VIEW to indicate the URI from which the local APK in the Intent data field originated from.
17	**EXTRA_PHONE_NUMBER** A String holding the phone number originally entered in ACTION_NEW_OUTGOING_CALL, or the actual number to call in a ACTION_CALL.
18	**EXTRA_SHORTCUT_ICON** The name of the extra used to define the icon, as a Bitmap, of a shortcut.
19	**EXTRA_SHORTCUT_INTENT** The name of the extra used to define the Intent of a shortcut.

20	**EXTRA_SHORTCUT_NAME** The name of the extra used to define the name of a shortcut.
21	**EXTRA_STREAM** URI holding a stream of data associated with the Intent, used with ACTION_SEND to supply the data being sent.
22	**EXTRA_SUBJECT** A constant string holding the desired subject line of a message.
23	**EXTRA_TEMPLATE** The initial data to place in a newly created record. Use with ACTION_INSERT.
24	**EXTRA_TEXT** A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
25	**EXTRA_TITLE** A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.
26	**EXTRA_UID** Used as an int extra field in ACTION_UID_REMOVED intents to supply the uid the package had been assigned.

FLAGS

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

COMPONENT NAME

This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class otherwise Android uses other information in the Intent object to locate a suitable target.

The component name is set by setComponent(), setClass(), or setClassName() and read by getComponent().

Types of Intents

There are following two types of intents supported by Android till version 4.1

EXPLICIT INTENTS

These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example:

```
// Explicit Intent by specifying its class name
Intent i = new Intent(this, TargetActivity.class);
i.putExtra("Key1", "ABC");
i.putExtra("Key2", "123");

// Starts TargetActivity
```

```
startActivity(i);
```

IMPLICIT INTENTS

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example:

```
// Implicit Intent by specifying a URI
Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.example.com"));

// Starts Implicit Activity
startActivity(i);
```

The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example:

```
// Get bundle object at appropriate place in your code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Example

Following example shows the functionality of a Android Intent to launch various Android built-in applications.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as *IntentDemo* under a package *com.example.intentdemo*. While creating this project, make sure you *Target SDK*and *Compile With* at the latest version of Android SDK to use higher levels of APIs.
2	Modify *src/MainActivity.java* file and add the code to define two listeners corresponding two buttons ie. Start Browser and Start Phone.
3	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
4	Modify *res/values/strings.xml* to define required constant values
5	Run the application to launch Android emulator and verify the result of the changes done in the aplication.

Following is the content of the modified main activity file**src/com.example.intentdemo/MainActivity.java**.

```
package com.example.intentdemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

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

      Button startBrowser = (Button) findViewById(R.id.start_browser);
      startBrowser.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });
      Button startPhone = (Button) findViewById(R.id.start_phone);
      startPhone.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("tel:9510300000"));
            startActivity(i);
         }
      });
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      // Inflate the menu; this adds items to the action
      // bar if it is present.
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }

}
```

Following will be the content of **res/layout/activity_main.xml** file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button android:id="@+id/start_browser"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_browser"/>

   <Button android:id="@+id/start_phone"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_phone" />

</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants:

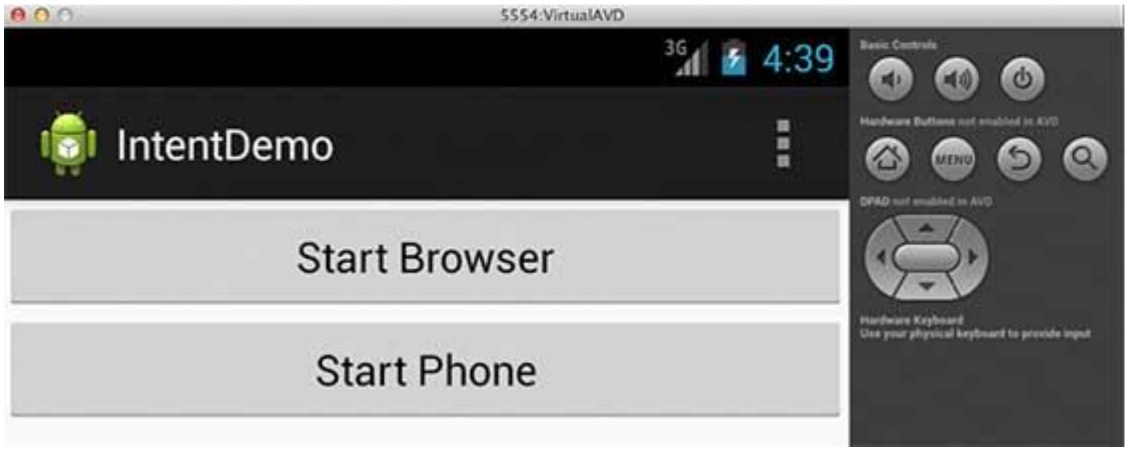
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">IntentDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="start_browser">Start Browser</string>
    <string name="start_phone">Start Phone</string>

```

```
</resources>
```

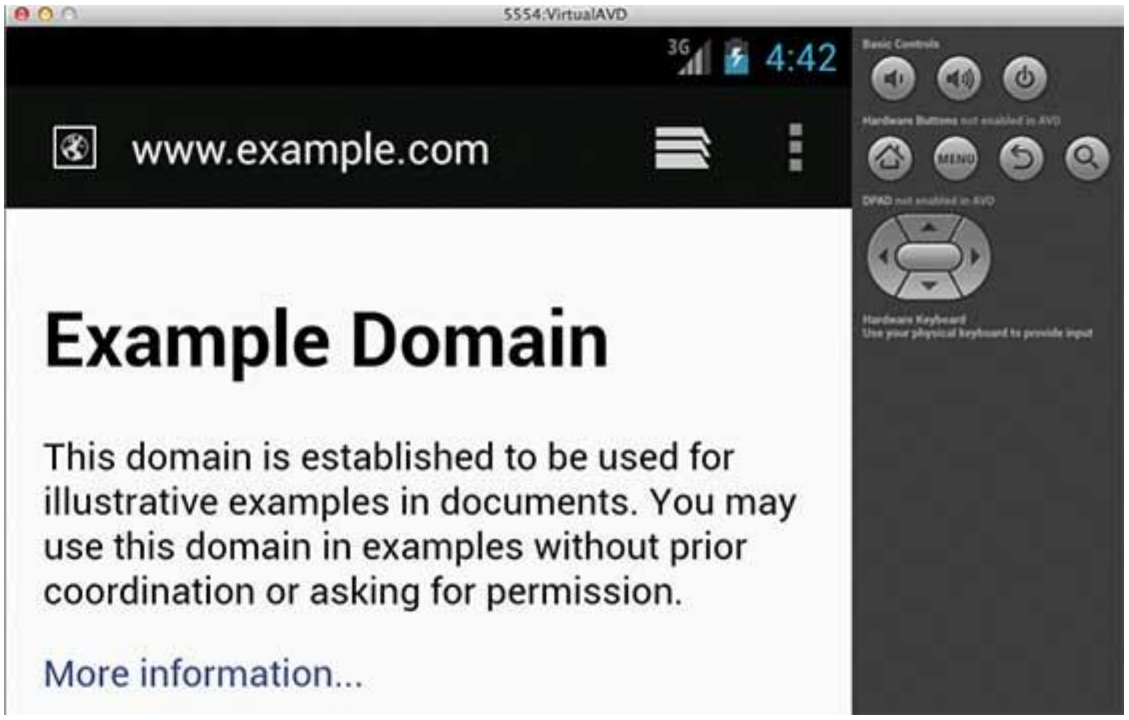
Following is the default content of **AndroidManifest.xml**:

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.intentdemo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Let's try to run your **IntentDemo** application. I assume you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Now click on **Start Browser** button, which will start a browser configured and display http://www.example.com as shown below:



Example Domain

This domain is established to be used for illustrative examples in documents. You may use this domain in examples without prior coordination or asking for permission.

More information...

Similar way you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

Intent Filters

You have seen how an Intent has been used to call an another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>**element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity**com.example.intentdemo.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data:

```
<activity android:name=".CustomActivity"
   android:label="@string/app_name">
   <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <action android:name="com.example.intentdemo.LAUNCH" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="http" />
   </intent-filter>
</activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.intentdemo.LAUNCH**action provided their category is **android.intent.category.DEFAULT**.

The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

There are following test Android checks before invoking an activity:

- A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.

- A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.

- Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

Example

Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use Eclipse IDE to create an Android application and name it as *IntentDemo* under a package *com.example.intentdemo*. While creating this project, make sure you *Target SDK*and *Compile With* at the latest version of Android SDK to use higher levels of APIs.
2	Modify *src/MainActivity.java* file and add the code to define three listeners corresponding to three buttons defined in layout file.
3	Add a new *src/CustomActivity.java* file to have one custom activity which will be invoked by different intents.
4	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
5	Add one layout XML file *res/layout/custom_view.xml* to add a simple <TextView> to show the passed data through intent.
6	Modify *res/values/strings.xml* to define required constant values
7	Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom activity.
8	Run the application to launch Android emulator and verify the result of the changes done in the aplication.

Following is the content of the modified main activity file**src/com.example.intentdemo/MainActivity.java**.

```
package com.example.intentdemo;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

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

      // First intent to use ACTION_VIEW action with correct data
      Button startBrowser_a = (Button) findViewById(R.id.start_browser_a);
      startBrowser_a.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
            Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      // Second intent to use LAUNCH action with correct data
      Button startBrowser_b = (Button) findViewById(R.id.start_browser_b);
      startBrowser_b.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            Intent i = new Intent("com.example.intentdemo.LAUNCH",
            Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      // Third intent to use LAUNCH action with incorrect data
      Button startBrowser_c = (Button) findViewById(R.id.start_browser_c);
      startBrowser_c.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            Intent i = new Intent("com.example.intentdemo.LAUNCH",
            Uri.parse("https://www.example.com"));
            startActivity(i);
         }
      });

   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
   // Inflate the menu; this adds items to the
   // action bar if it is present.
   getMenuInflater().inflate(R.menu.main, menu);
   return true;
   }

}
```

Following is the content of the modified main activity file**src/com.example.intentdemo/CustomActivity.java**.

```
package com.example.intentdemo;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class CustomActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.custom_view);

      TextView label = (TextView) findViewById(R.id.show_data);
      Uri url = getIntent().getData();
      label.setText(url.toString());
   }

}
```

Following will be the content of **res/layout/activity_main.xml** file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button android:id="@+id/start_browser_a"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_browser_a"/>

   <Button android:id="@+id/start_browser_b"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_browser_b"/>

   <Button android:id="@+id/start_browser_c"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_browser_c"/>

</LinearLayout>
```

Following will be the content of **res/layout/custom_view.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"    >

   <TextView android:id="@+id/show_data"
   android:layout_width="fill_parent"
   android:layout_height="400dp"/>

</LinearLayout>
```

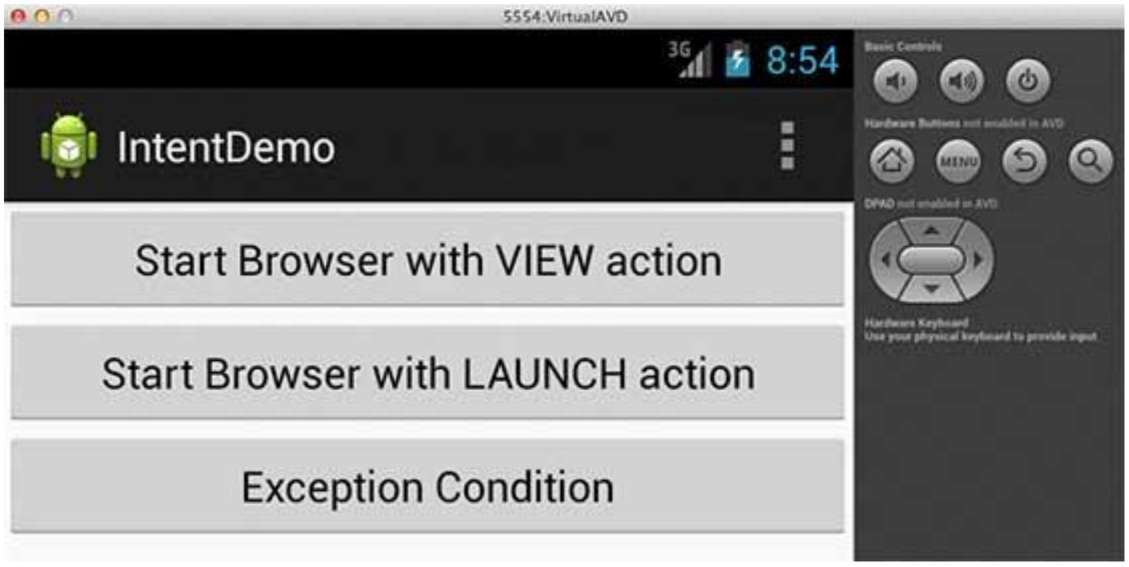
Following will be the content of **res/values/strings.xml** to define two new constants:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">IntentDemo</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="start_browser_a">Start Browser with VIEW action</string>
    <string name="start_browser_b">Start Browser with LAUNCH action</string>
    <string name="start_browser_c">Exception Condition</string>

</resources>
```

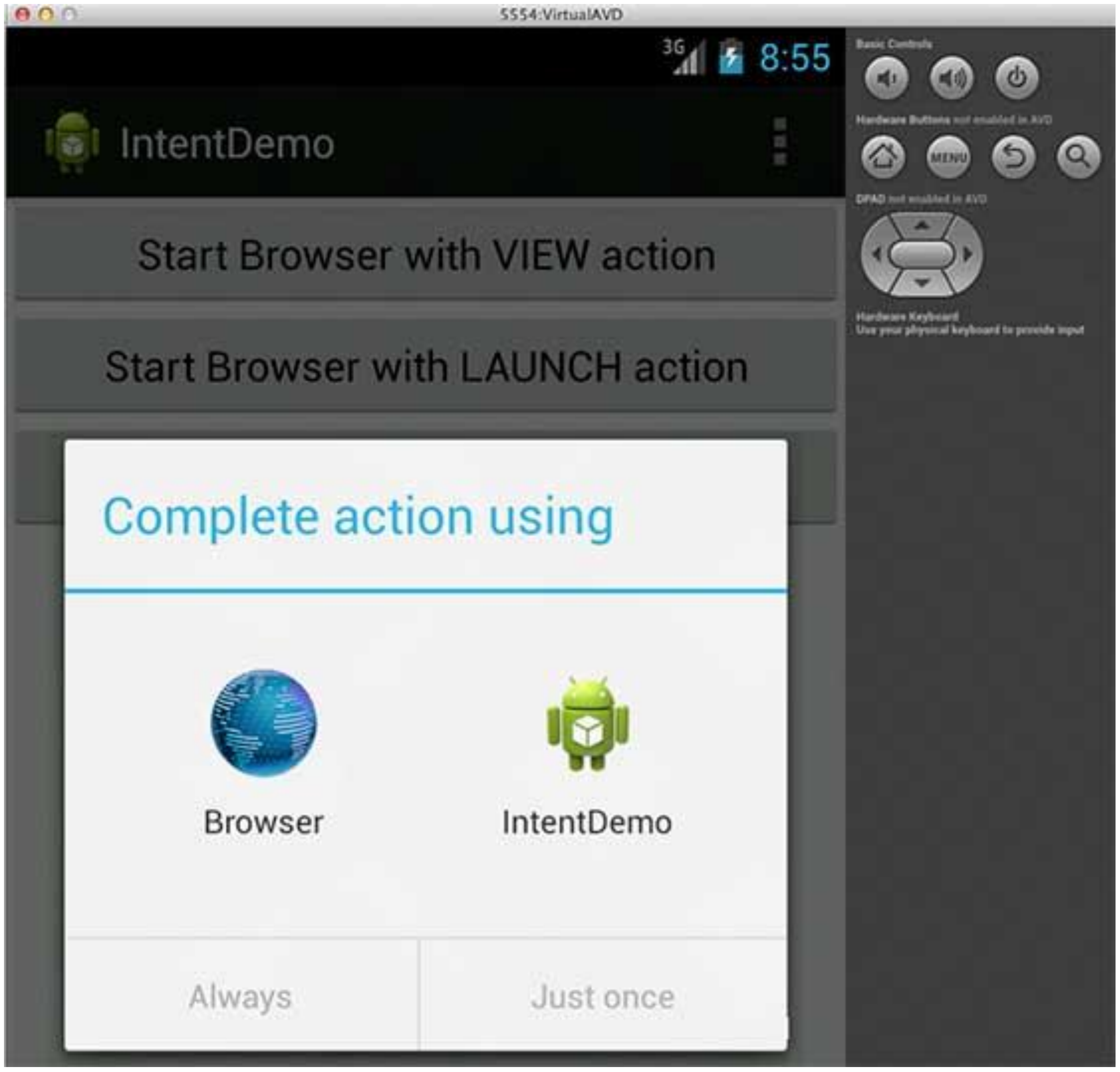
Following is the default content of **AndroidManifest.xml**:

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.intentdemo.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.example.intentdemo.CustomActivity"
            android:label="@string/app_name">
          <intent-filter>
             <action android:name="android.intent.action.VIEW" />
             <action android:name="com.example.intentdemo.LAUNCH" />
             <category android:name="android.intent.category.DEFAULT" />
             <data android:scheme="http" />
          </intent-filter>
        </activity>
    </application>

</manifest>
```

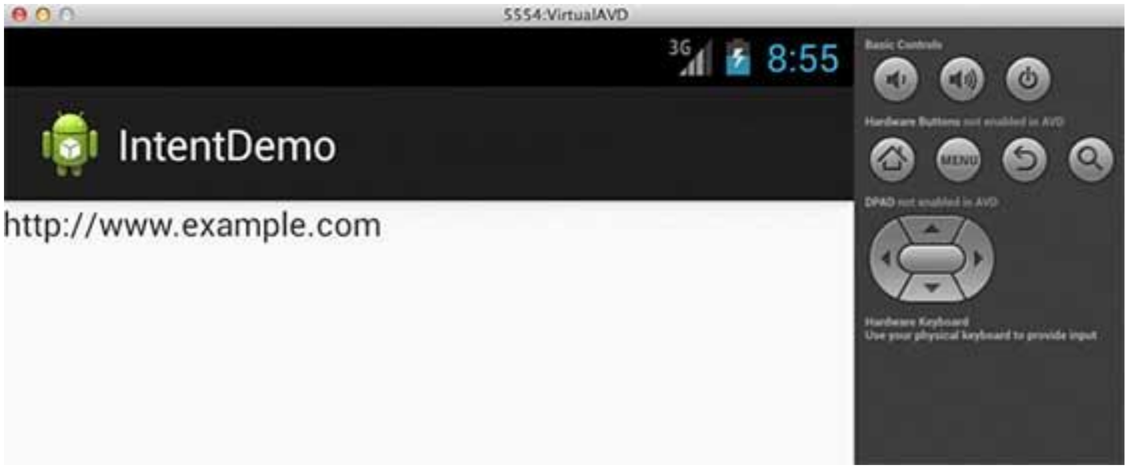
Let's try to run your **IntentDemo** application. I assume you had created your **AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run ▶ icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:


5554:VirtualAVD
8:54
IntentDemo
Start Browser with VIEW action
Start Browser with LAUNCH action
Exception Condition
Basic Controls
Hardware Buttons not enabled in AVD
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.

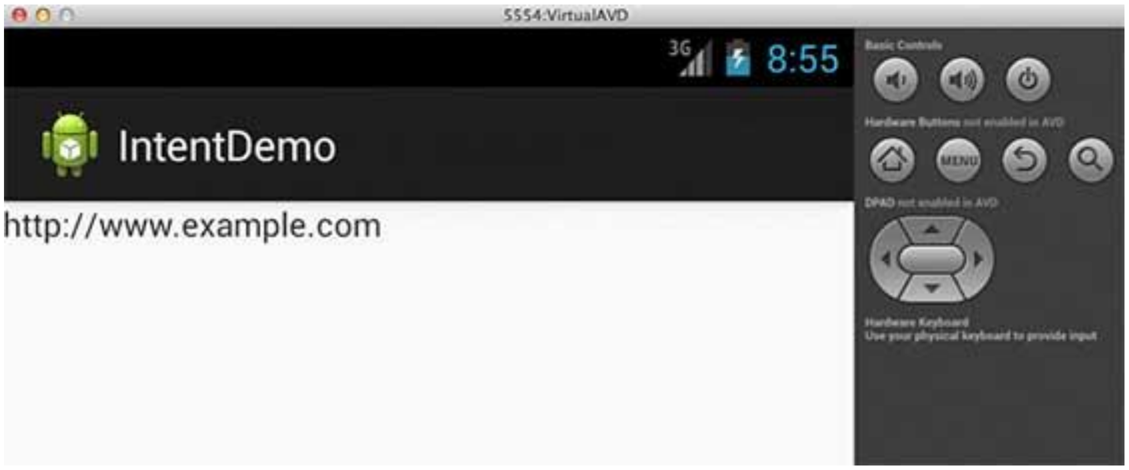


Now if you select Browser, then Android will launch web browser and open example.com website but if you select IndentDemo option then Android will launch CustomActivity which does nothing but just capture passed data and displays in a text view as follows:



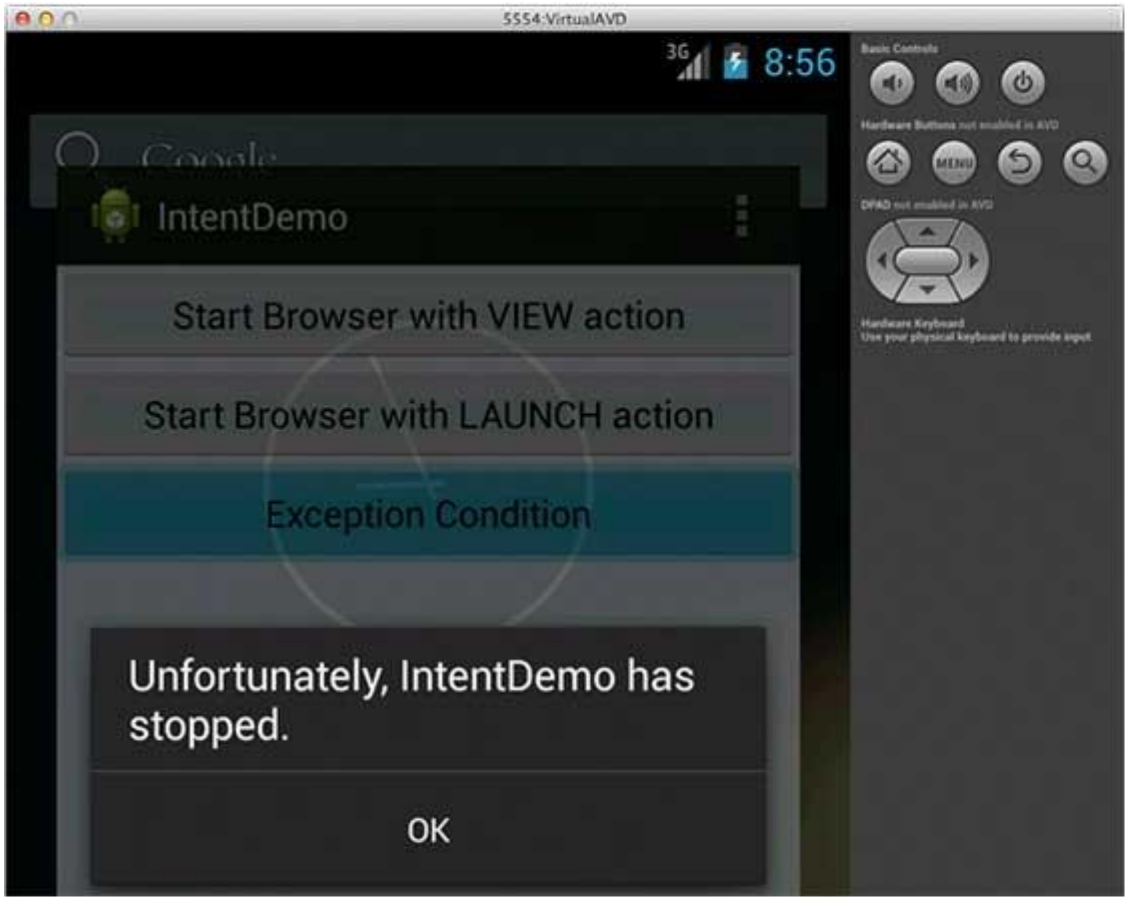
http://www.example.com

Now go back using back button and click on "Start Browser with LAUNCH Action" button, here Android applies filter to choose define activity and it simply launch your custom activity and again it displays following screen:



http://www.example.com

Again, go back using back button and click on "Exception Condition" button, here Android tries to find out a valid filter for the given intent but it does not find a valid activity defined because this time we have used data as **https** instead of **http** though we are giving a correct action, so Android raises an exception and shows following screen:


5554:VirtualAVD
8:56
IntentDemo
Start Browser with VIEW action
Start Browser with LAUNCH action
Exception Condition
OK
Unfortunately, IntentDemo has stopped.