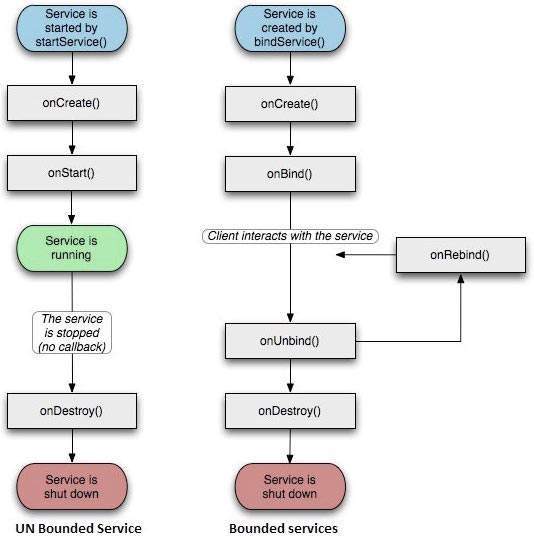
Android - Services

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

|  |  |
| --- | --- |
| **Sr.No.** | **State & Description** |
| 1 | **Started**  A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. |
| 2 | **Bound**  A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). |

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): *(image courtesy : android.com )*



To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

|  |  |
| --- | --- |
| **Sr.No.** | **Callback & Description** |
| 1 | **onStartCommand()**  The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods. |
| 2 | **onBind()**  The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*. |
| 3 | **onUnbind()**  The system calls this method when all clients have disconnected from a particular interface published by the service. |
| 4 | **onRebind()**  The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*. |
| 5 | **onCreate()**  The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up. |
| 6 | **onDestroy()**  The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc. |

The following skeleton service demonstrates each of the life cycle methods −

package com.tutorialspoint;

import android.app.Service;

import android.os.IBinder;

import android.content.Intent;

import android.os.Bundle;

public class HelloService extends Service {

/\*\* indicates how to behave if the service is killed \*/

int mStartMode;

/\*\* interface for clients that bind \*/

IBinder mBinder;

/\*\* indicates whether onRebind should be used \*/

boolean mAllowRebind;

/\*\* Called when the service is being created. \*/

@Override

public void onCreate() {

}

/\*\* The service is starting, due to a call to startService() \*/

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

return mStartMode;

}

/\*\* A client is binding to the service with bindService() \*/

@Override

public IBinder onBind(Intent intent) {

return mBinder;

}

/\*\* Called when all clients have unbound with unbindService() \*/

@Override

public boolean onUnbind(Intent intent) {

return mAllowRebind;

}

/\*\* Called when a client is binding to the service with bindService()\*/

@Override

public void onRebind(Intent intent) {

}

/\*\* Called when The service is no longer used and is being destroyed \*/

@Override

public void onDestroy() {

}

}

## Example

This example will take you through simple steps to show how to create your own Android Service. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter. |
| 2 | Modify main activity file *MainActivity.java* to add *startService()* and *stopService()* methods. |
| 3 | Create a new java file *MyService.java* under the package *com.example.My Application*. This file will have implementation of Android service related methods. |
| 4 | Define your service in *AndroidManifest.xml* file using <service.../> tag. An application can have one or more services without any restrictions. |
| 5 | Modify the default content of *res/layout/activity\_main.xml* file to include two buttons in linear layout. |
| 6 | No need to change any constants in *res/values/strings.xml* file. Android studio take care of string values |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

package com.example.tutorialspoint7.myapplication;

import android.content.Intent;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.os.Bundle;

import android.app.Activity;

import android.util.Log;

import android.view.View;

public class MainActivity extends Activity {

String msg = "Android : ";

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Log.d(msg, "The onCreate() event");

}

public void startService(View view) {

startService(new Intent(getBaseContext(), MyService.class));

}

// Method to stop the service

public void stopService(View view) {

stopService(new Intent(getBaseContext(), MyService.class));

}

}

Following is the content of **MyService.java**. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()* −

package com.example.tutorialspoint7.myapplication;

import android.app.Service;

import android.content.Intent;

import android.os.IBinder;

import android.support.annotation.Nullable;

import android.widget.Toast;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class MyService extends Service {

@Nullable

@Override

public IBinder onBind(Intent intent) {

return null;

}

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

// Let it continue running until it is stopped.

Toast.makeText(this, "Service Started", Toast.LENGTH\_LONG).show();

return START\_STICKY;

}

@Override

public void onDestroy() {

super.onDestroy();

Toast.makeText(this, "Service Destroyed", Toast.LENGTH\_LONG).show();

}

}

Following will the modified content of *AndroidManifest.xml* file. Here we have added <service.../> tag to include our service −

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<service android:name=".MyService" />

</application>

</manifest>

Following will be the content of **res/layout/activity\_main.xml** file to include two buttons −

<RelativeLayout 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:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin" tools:context=".MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Example of services"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point "

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_above="@+id/imageButton"

android:layout\_centerHorizontal="true"

android:layout\_marginBottom="40dp" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/button2"

android:text="Start Services"

android:onClick="startService"

android:layout\_below="@+id/imageButton"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Stop Services"

android:id="@+id/button"

android:onClick="stopService"

android:layout\_below="@+id/button2"

android:layout\_alignLeft="@+id/button2"

android:layout\_alignStart="@+id/button2"

android:layout\_alignRight="@+id/button2"

android:layout\_alignEnd="@+id/button2" />

</RelativeLayout>

**Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)**to communicate with a background Service.

**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.

Intent email = new Intent(Intent.ACTION\_SEND, Uri.parse("mailto:"));

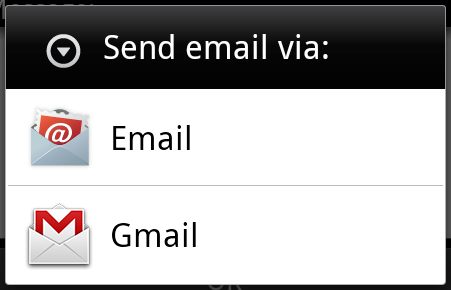
email.putExtra(Intent.EXTRA\_EMAIL, recipients);

email.putExtra(Intent.EXTRA\_SUBJECT, subject.getText().toString());

email.putExtra(Intent.EXTRA\_TEXT, body.getText().toString());

startActivity(Intent.createChooser(email, "Choose an email client from..."));

Above syntax is calling startActivity method to start an email activity and result should be as shown below −



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.

String q = "tutorialspoint";

Intent intent = new Intent(Intent.ACTION\_WEB\_SEARCH );

intent.putExtra(SearchManager.QUERY, q);

startActivity(intent);

Above example will search as **tutorialspoint** on android search engine and it gives the result of tutorialspoint in your an activity

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

|  |  |
| --- | --- |
| **Sr.No** | **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 as well as 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](https://www.tutorialspoint.com/android/android_intent_standard_actions.htm)

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

### Data

Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute), just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its parts −

These attributes that specify the URL format are optional, but also mutually dependent −

* If a scheme is not specified for the intent filter, all the other URI attributes are ignored.
* If a host is not specified for the filter, the port attribute and all the path attributes are ignored.

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 −

|  |  |
| --- | --- |
| **Sr.No.** | **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. |
| 7 | **ACTION\_SET\_WALLPAPER**  Show settings for choosing wallpaper |
| 8 | **ACTION\_SYNC**  It going to be synchronous the data,Constant Value is **android.intent.action.SYNC** |
| 9 | **ACTION\_SYSTEM\_TUTORIAL**  It will start the platform-defined tutorial(Default tutorial or start up tutorial) |
| 10 | **ACTION\_TIMEZONE\_CHANGED**  It intimates when time zone has changed |
| 11 | **ACTION\_UNINSTALL\_PACKAGE**  It is used to run default uninstaller |

### 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](https://www.tutorialspoint.com/android/android_intent_standard_categories.htm).

You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate activity corresponding 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](https://www.tutorialspoint.com/android/android_intent_standard_extra_data.htm)

### 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.

|  |  |
| --- | --- |
| **Sr.No** | **Flags & Description** |
| 1 | **FLAG\_ACTIVITY\_CLEAR\_TASK**  If set in an Intent passed to Context.startActivity(), this flag will cause any existing task that would be associated with the activity to be cleared before the activity is started. That is, the activity becomes the new root of an otherwise empty task, and any old activities are finished. This can only be used in conjunction with FLAG\_ACTIVITY\_NEW\_TASK. |
| 2 | **FLAG\_ACTIVITY\_CLEAR\_TOP**  If set, and the activity being launched is already running in the current task, then instead of launching a new instance of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to the (now on top) old activity as a new Intent. |
| 3 | **FLAG\_ACTIVITY\_NEW\_TASK**  This flag is generally used by activities that want to present a "launcher" style behavior: they give the user a list of separate things that can be done, which otherwise run completely independently of the activity launching them. |

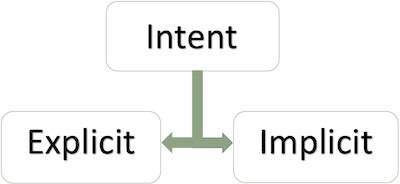
### Component Name

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().

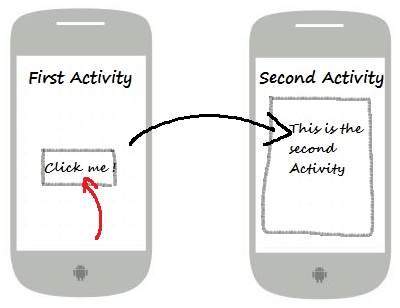
## Types of Intents

There are following two types of intents supported by Android



### Explicit Intents

Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.



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(FirstActivity.this, SecondActivity.class);

// 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 −

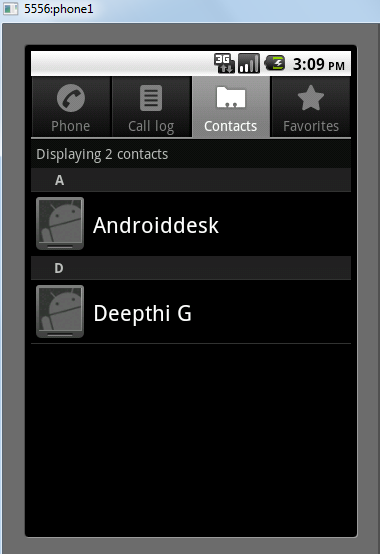
Intent read1=new Intent();

read1.setAction(android.content.Intent.ACTION\_VIEW);

read1.setData(ContactsContract.Contacts.CONTENT\_URI);

startActivity(read1);

Above code will give result as shown below



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 Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira\_000.myapplication*. |
| 2 | Modify *src/main/java/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 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

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

package com.example.saira\_000.myapplication;

import android.content.Intent;

import android.net.Uri;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

Button b1,b2;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

b1=(Button)findViewById(R.id.button);

b1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("http://www.example.com"));

startActivity(i);

}

});

b2=(Button)findViewById(R.id.button2);

b2.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent i = new Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("tel:9510300000"));

startActivity(i);

}

});

}

}

Following will be the content of **res/layout/activity\_main.xml** file −

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

<RelativeLayout 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:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Intent Example"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point"

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_below="@+id/textView1"

android:layout\_centerHorizontal="true" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_below="@+id/textView2"

android:layout\_centerHorizontal="true" />

<EditText

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/editText"

android:layout\_below="@+id/imageButton"

android:layout\_alignRight="@+id/imageButton"

android:layout\_alignEnd="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start Browser"

android:id="@+id/button"

android:layout\_alignTop="@+id/editText"

android:layout\_alignRight="@+id/textView1"

android:layout\_alignEnd="@+id/textView1"

android:layout\_alignLeft="@+id/imageButton"

android:layout\_alignStart="@+id/imageButton" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Start Phone"

android:id="@+id/button2"

android:layout\_below="@+id/button"

android:layout\_alignLeft="@+id/button"

android:layout\_alignStart="@+id/button"

android:layout\_alignRight="@+id/textView2"

android:layout\_alignEnd="@+id/textView2" />

</RelativeLayout>

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">My Applicaiton</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.saira\_000.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>