Activity Lifecycle in Android

**Activity Lifecycle:** Activity is one of the building blocks of Android OS. In simple words Activity is a screen that user interact with. Every Activity in android has lifecycle like created, started, resumed, paused, stopped or destroyed. These different states are known as Activity Lifecycle. In other words we can say Activity is a class pre-written in Java Programming.

**Below is Activity Lifecycle Table:**

Short description of Activity Lifecycle example:

**onCreate()** – Called when the activity is first created

**onStart()** – Called just after it’s creation or by restart method after onStop(). Here Activity start becoming visible to user

**onResume()** – Called when Activity is visible to user and user can interact with it

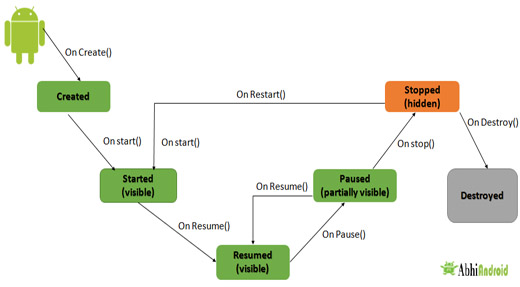
**onPause()** – Called when Activity content is not visible because user resume previous activity

**onStop()** – Called when activity is not visible to user because some other activity takes place of it

**onRestart()** – Called when user comes on screen or resume the activity which was stopped

**onDestroy()** – Called when Activity is not in background

**Below Activity Lifecycle Diagram Shows Different States:**



**Different Types of Activity Lifecycle States:**

Activity have different states or it’s known as Activity life cycle. All life cycle methods aren’t required to override but it’s quite important to understand them. Lifecycles methods can be overridden according to requirements.

**LIST OF ACTIVITY LIFECYCLE METHODS OR STATES:**

**Activity Created: onCreate(Bundle savedInstanceState):**

**onCreate()** method is called when activity gets memory in the OS. To use create state we need to override onCreate(Bundle savedInstanceState) method. Now there will be question in mind what is Bundle here, so Bundle is a data repository object that can store any kind of primitive data and this object will be null until some data isn’t saved in that.

When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.

When ever orientation(i.e. from horizontal to vertical or vertical to horizontal) of activity gets changed or when an Activity gets forcefully terminated by any Operating System then savedInstanceState i.e. object of Bundle Class will save the state of an Activity.

It is best place to put initialization code.

**Activity Started: onStart():**

onStart() method is called just after it’s creation. In other case Activity can also be started by calling restart method i.e after activity stop. So this means onStart() gets called by Android OS when user switch between applications. For example, if a user was using Application A and then a notification comes and user clicked on notification and moved to Application B, in this case Application A will be paused. And again if a user again click on app icon of Application A then Application A which was stopped will again gets started.

**Activity Resumed:.onResume():**

Activity resumed is that situation when it is actually visible to user means the data displayed in the activity is visible to user. In lifecycle it always gets called after activity start and in most use case after activity paused (onPause).

**Activity Paused: onPause():**

Activity is called paused when it’s content is not visible to user, in most case onPause() method called by Android OS when user press Home button (Center Button on Device) to make hide.

Activity also gets paused before stop called in case user press the back navigation button. The activity will go in paused state for these reasons also if a notification or some other dialog is overlaying any part (top or bottom) of the activity (screen). Similarly, if the other screen or dialog is transparent then user can see the screen but cannot interact with it. For example, if a call or notification comes in, the user will get the opportunity to take the call or ignore it.

**Activity Stopped: onStop():**

Activity is called stopped when it’s not visible to user. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

Every activity gets stopped before destroy in case of when user press back navigation button. So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user. In this case application will not present anything useful to the user directly as it’s going to stop.

**Activity Restarted: onRestart():**

Activity is called in restart state after stop state. So activity’s onRestart() function gets called when user comes on screen or resume the activity which was stopped. In other words, when Operating System starts the activity for the first time onRestart() never gets called. It gets called only in case when activity is resumes after stopped state.

**Activity Destroyed: onDestroy():**

Any activity is known as in destroyed state when it’s not in background. There can different cases at what time activity get destroyed.

First is if user pressed the back navigation button then activity will be destroyed after completing the lifecycle of pause and stop.

In case if user press the home button and app moves to background. User is not using it no more and it’s being shown in recent apps list. So in this case if system required resources need to use somewhere else then OS can destroy the Activity.

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again. Another use case is with Splash Screens if there is call to finish() method from onCreate() of an activity then OS can directly call onDestroy() with calling onPause() and onStop().

**Activity Lifecycle Example:**

In the below example we have used the below JAVA and Android topics:

JAVA Topics Used: Method Overriding, static variable, package, Inheritance, method and class.

Android Topic Used: We have used Log class which is used to printout message in Logcat. One of the important use of Log is in debugging.

First we will create a new Android Project and name the activity as HomeActivity. In our case we have named our App project as Activity Lifecycle Example.

We will initialize a static String variable with the name of the underlying class using getSimpleName() method. In our case HOME\_ACTIVITY\_TAG is the name of the String variable which store class name HomeActivity.

|  |
| --- |
| private static final String HOME\_ACTIVITY\_TAG = HomeActivity.class.getSimpleName(); |

Now we will create a new method which will print message in Logcat.

|  |
| --- |
| private void showLog(String text){  Log.d(HOME\_ACTIVITY\_TAG, text);  } |

Now we will override all activity lifecycle method in Android and use showLog() method which we creating for printing message in Logcat.

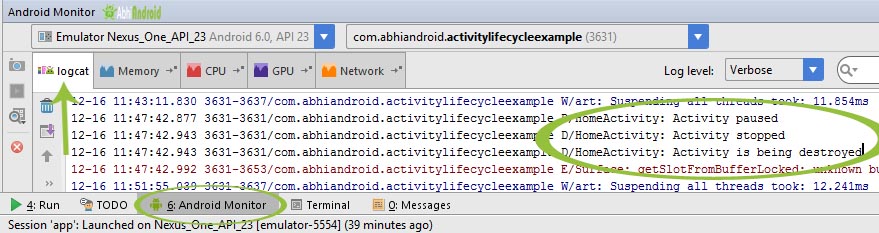
|  |
| --- |
| @Override  public void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  showLog("Activity Created");  }  @Override  protected void onRestart(){  super.onRestart();//call to restart after onStop  showLog("Activity restarted");  }  @Override  protected void onStart() {  super.onStart();//soon be visible  showLog("Activity started");  }  @Override  protected void onResume() {  super.onResume();//visible  showLog("Activity resumed");  }  @Override  protected void onPause() {  super.onPause();//invisible  showLog("Activity paused");  }  @Override  protected void onStop() {  super.onStop();  showLog("Activity stopped");  }  @Override  protected void onDestroy() {  super.onDestroy();  showLog("Activity is being destroyed");  } |

When creating an Activity we need to register this in AndroidManifest.xml file. Now question is why need to register? **It's actually because manifest file has the information which Android OS read very first.** When registering an activity other information can also be defined within manifest like Launcher Activity (An activity that should start when user click on app icon).

Here is declaration example in AndroidManifest.xml file

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.abhiandroid.homeactivity">  <application  android:allowBackup="true"  android:icon="@mipmap/ic\_launcher"  android:label="@string/app\_name"  android:supportsRtl="true"  android:theme="@style/AppTheme">  <activity  android:name=".HomeActivity"  android:label="@string/app\_name"  android:theme="@style/AppTheme.NoActionBar">  <intent-filter>  <action android:name="android.intent.action.MAIN" />  <category android:name="android.intent.category.LAUNCHER" />  </intent-filter>  </activity>  </application>  </manifest> |

Go to Logcat again and scroll down to bottom. You will see 3 more methods were called: Activity paused, Activity stopped and Activity is being destroyed.



So this clears:

* onPause() method was called when user resume previous activity
* onStop() method was called when activity is not visible to user
* Last onDestroy() method was called when Activity is not in background

**Important Note:** In the above example onRestart() won’t be called because there was no situation when we can resume the onStart() method again. In future example we will show you onRestart() in action as well.

**Importance Of Activity Life Cycle:**

Activity is the main component of Android Application, as every screen is an activity so to create any simple app first we have to start with Activities. Every lifecycle method is quite important to implement according to requirements, However onCreate(Bundle state) is always needed to implement to show or display some content on screen.

## Intent Tutorial in Android With Example And Types

Android uses [Intent](https://abhiandroid.com/programming/intent-in-android/) for communicating between the components of an Application and also from one application to another application.

[Intent](https://abhiandroid.com/programming/intent-in-android/) are the objects which is used in android for passing the information among Activities in an Application and from one app to another also. [Intent](https://abhiandroid.com/programming/intent-in-android/) are used for communicating between the Application components and it also provides the connectivity between two apps.

For example: Intent facilitate you to redirect your activity to another activity on occurrence of any event. By calling, startActivity() you can perform this task.

Intent intent = new Intent(getApplicationContext(), SecondActivity.class);

startActivity(intent);

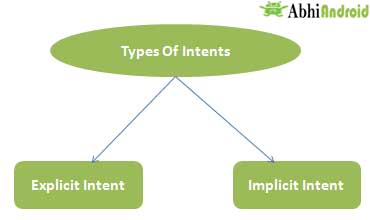
In the above example, foreground activity is getting redirected to another activity i.e. SecondActivity.[java](https://abhiandroid.com/java/). getApplicationContext() returns the context for your foreground activity.

**Table Of Contents**[[hide](https://abhiandroid.com/programming/intent-in-android)]

* [1 Types of Intents:](https://abhiandroid.com/programming/intent-in-android#Types_of_Intents)
* [2 Intent Example In Android:](https://abhiandroid.com/programming/intent-in-android#Intent_Example_In_Android)
* [3 Intent Uses In Android:](https://abhiandroid.com/programming/intent-in-android#Intent_Uses_In_Android)
* [4 Importance of using Intents in Android Applications:](https://abhiandroid.com/programming/intent-in-android#Importance_of_using_Intents_in_Android_Applications)

#### ****Types of Intents:****

Intent are of two types: Explicit Intent and Implicit Intent



**Explicit Intent:**

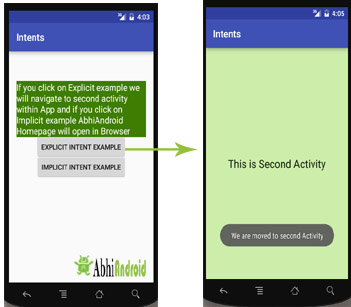
* Explicit Intents are used to connect the application internally.
* In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.

Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

Intent intent = new Intent(getApplicationContext(), SecondActivity.class);

startActivity(intent);

Here SecondActivity is the [JAVA](https://abhiandroid.com/java/" \o "JAVA " \t "_self) class name where the activity will now be navigated. Example with code in the end of this post will make it more clear.



**Implicit Intent:**

* In Implicit Intents we do need to specify the name of the component. We just specify the Action which has to be performed and further this action is handled by the component of another application.
* The basic example of implicit Intent is to open any web page

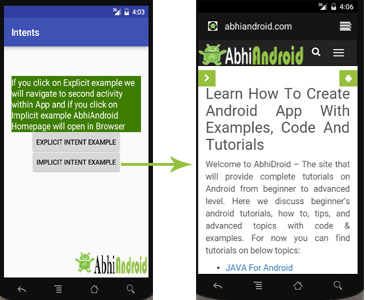
Let’s take an example to understand Implicit Intents more clearly. We have to open a website using intent in your application. See the code snippet given below

Intent intentObj = new Intent(Intent.ACTION\_VIEW);

intentObj.setData(Uri.parse("https://www.abhiandroid.com"));

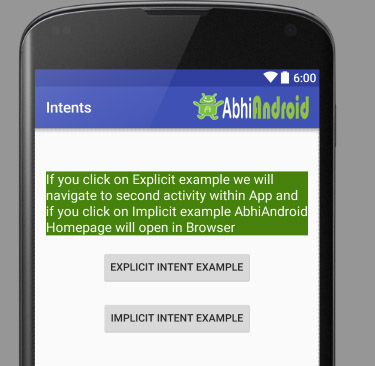
startActivity(intentObj);

Unlike Explicit Intent you do not use any class name to pass through Intent(). In this example we has just specified an action. Now when we will run this code then Android will automatically start your web browser and it will open AbhiAndroid home page.



#### ****Intent Example In Android:****

Let’s implement Intent for a very basic use. In the below example we will Navigate from one Activity to another and open a web homepage of AbhiAndroid using Intent. **The example will show you both implicit and explicit Intent together. Below is the final output:**



Create a project in [Android Studio](https://abhiandroid.com/androidstudio/" \o "Android Studio " \t "_self) and named it “Intents”. Make an activity, which would consists [Java](https://abhiandroid.com/java/" \o "JAVA " \t "_self) file; MainActivity.java and an [xml](https://abhiandroid.com/ui/xml/" \o "XML in Android" \t "_self) file for User interface which would be activity\_main.[xml](https://abhiandroid.com/ui/xml/" \o "XML in Android" \t "_self)

**Step 1: Let’s design the UI of activity\_main.xml:**

* First design the [text view](https://abhiandroid.com/ui/textview/" \o "Text View With Example" \t "_self) displaying basic details of the App
* Second design the two [button](https://abhiandroid.com/ui/button/" \o "Button Tutorial" \t "_self) of Explicit Intent Example and Implicit Intent Example

**Below is the complete code of activity\_main.xml**

<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:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textAppearance="?android:attr/textAppearanceMedium"

android:text="If you click on Explicit example we will navigate to second activity within App and if you click on Implicit example AbhiAndroid Homepage will open in Browser"

android:id="@+id/textView2"

android:clickable="false"

android:layout\_alignParentTop="true"

android:layout\_alignParentStart="true"

android:layout\_marginTop="42dp"

android:background="#3e7d02"

android:textColor="#ffffff" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Explicit Intent Example"

android:id="@+id/explicit\_Intent"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:layout\_marginTop="147dp" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Implicit Intent Example"

android:id="@+id/implicit\_Intent"

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true" />

</RelativeLayout>

**Step 2: Design the UI of second activity activity\_second.xml**

Now lets design UI of another activity where user will navigate after he click on Explicit Example [button](https://abhiandroid.com/ui/button/" \o "Button Tutorial" \t "_self). Go to layout folder, create a new activity and name it activity\_second.[xml](https://abhiandroid.com/ui/xml/" \o "XML in Android" \t "_self).

* In this activity we will simply use [TextView](https://abhiandroid.com/ui/textview/" \o "TextView" \t "_self) to tell user he is now on second activity.

**Below is the complete code of activity\_second.xml**

<?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"

android:background="#CCEEAA"

tools:context="com.example.android.intents.SecondActivity">

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textAppearance="?android:attr/textAppearanceLarge"

android:text="This is Second Activity"

android:id="@+id/textView"

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true" />

</RelativeLayout>

**Step 3: Implement onClick event for Implicit And Explicit Button inside MainActivity.java**

Now we will use setOnClickListener() method to implement OnClick event on both the [button](https://abhiandroid.com/ui/button/" \o "Button Tutorial" \t "_self). Implicit button will open AbhiAndroid.com homepage in browser and Explicit button will move to SecondActivity.java.

**Below is the complete code of MainActivity.java**

package com.example.android.intents;

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 explicit\_btn, implicit\_btn;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

explicit\_btn = (Button)findViewById(R.id.explicit\_Intent);

implicit\_btn = (Button) findViewById(R.id.implicit\_Intent);

//implement Onclick event for Explicit Intent

explicit\_btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(getBaseContext(), SecondActivity.class);

startActivity(intent);

}

});

//implement onClick event for Implicit Intent

implicit\_btn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(Intent.ACTION\_VIEW);

intent.setData(Uri.parse("https://www.abhiandroid.com"));

startActivity(intent);

}

});

}

}

**Step 4: Create A New JAVA class name SecondActivity**

Now we need to create another SecondActivity.java which will simply open the layout of activity\_second.xml . Also we will use [Toast](https://abhiandroid.com/programming/custom-toast-tutorial-example.html/" \o "Toast" \t "_self) to display message that he is on second activity.

**Below is the complete code of SecondActivity.java:**

package com.example.android.intents;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.widget.Toast;

public class SecondActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_second);

Toast.makeText(getApplicationContext(), "We are moved to second Activity",Toast.LENGTH\_LONG).show();

}

}

**Step 5: Manifest file:**

Make sure Manifest file has both the MainActivity and SecondActivity listed it. Also here MainActivity is our main activity which will be launched first. So make sure intent-filter is correctly added just below MainActivity.

**Below is the code of Manifest file:**

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

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

package="com.example.android.intents" >

<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>

<activity android:name=".SecondActivity" >

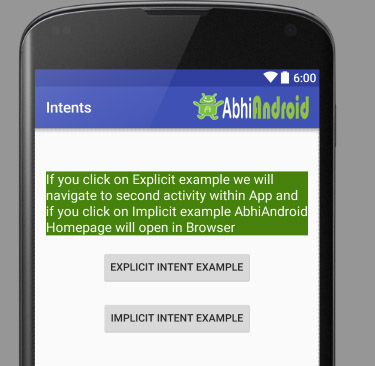
</activity>

</application>

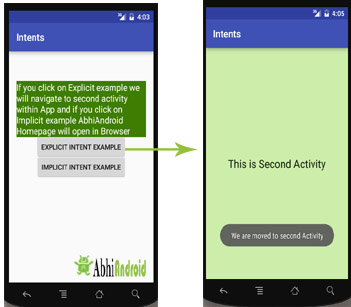
</manifest>

**Output:**

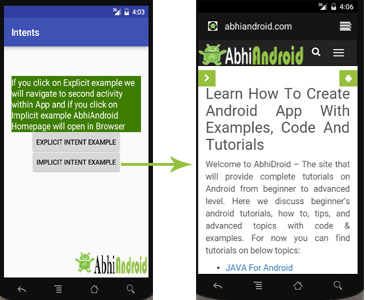
Now run the above program in your Emulator. The App will look like this:



First Click on Explicit Intent Example. The SecondActivity will be open within the App:



Now go back in Emulator and click on Implicit Intent Example. The AbhiAndroid.com homepage will open in Browser (make sure you have internet):



#### ****Intent Uses In Android:****

Android uses Intents for facilitating communication between its components like Activities, Services and Broadcast Receivers.

**Intent for an Activity:**

Every screen in Android application represents an activity. To start a new activity you need to pass an Intent object to startActivity() method. This Intent object helps to start a new activity and passing data to the second activity.

**Intent for Services:**

Services work in background of an Android application and it does not require any user Interface. Intents could be used to start a Service that performs one-time task(for example: Downloading some file) or for starting a Service you need to pass Intent to startService() method.

**Intent for Broadcast Receivers:**

There are various message that an app receives, these messages are called as Broadcast Receivers. (For example, a broadcast message could be initiated to intimate that the file downloading is completed and ready to use). Android system initiates some broadcast message on several events, such as System Reboot, Low Battery warning message etc.

#### ****Importance of using Intents in Android Applications:****

Whenever you need to navigate to another activity of your app or you need to send some information to next activity then we can always prefer to Intents for doing so.

Intents are really easy to handle and it facilitates communication of components and activities of your application. Moreover you can communicate to another application and send some data to another application using Intents.

## 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.My Application.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.My Application.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.My Application.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.