by Lars Vogel

Using Fragments in Android - Tutorial

Based on Android 4.2

Lars Vogel

Version 11.2

17.06.2013

Revision History			
Revision 0.1	04.07.2009	Lars Vogel	Created
Revision 0.2 - 11.2	07.07.2009 - 17.06.2013		bug fixing and enhancements

Using Fragments in Android Applications

This tutorial describes how to use Fragments in Android applications to create multi-pane layouts, i.e. applications which scale to the available width of the device. It is based on Eclipse 4.2 (Juno), Java 1.6 and Android 4.2 (Jelly Bean).

Table of Contents

1. Android Basics

The following description assumes that you have already basic knowledge in Android development.

Please check the **Android development tutorial** to learn the basics. Also see **Android development tutorials** for more information about Android development.

2. Fragments

2.1. Fragments Overview

A Fragments is an independent component which can be connected to an activity. A Fragment typically defines a part of a user interface but it is possible to define headless Fragments, i.e. without user interface.

Fragments can be dynamically or statically added to a layout. A Fragment encapsulate functionality so that it is easier to reuse within activity and layouts.

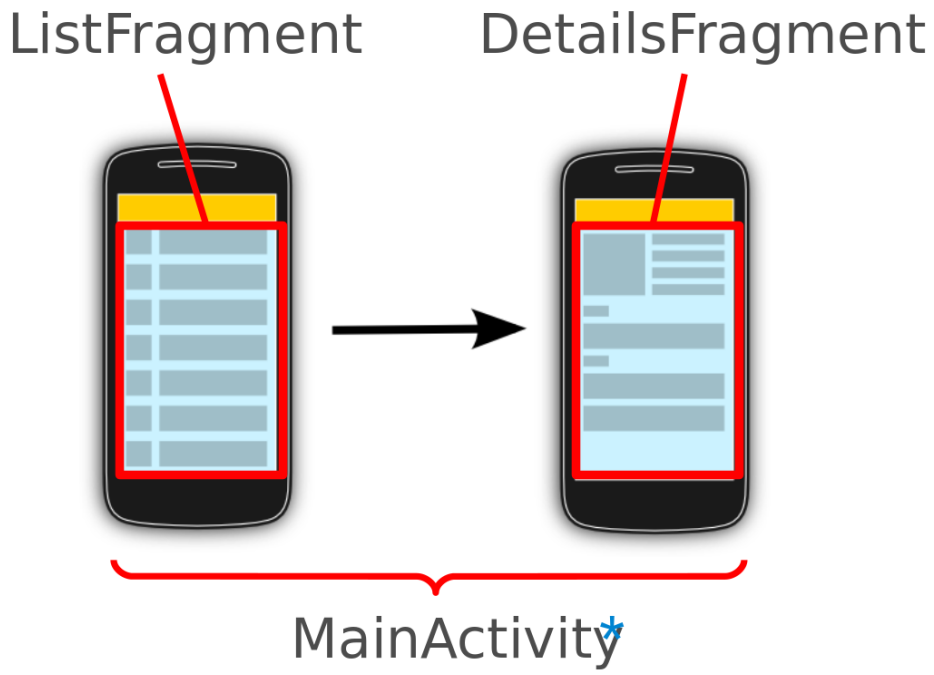
A Fragment component runs in the context of an activity but it has its own lifecycle and their own user interface.

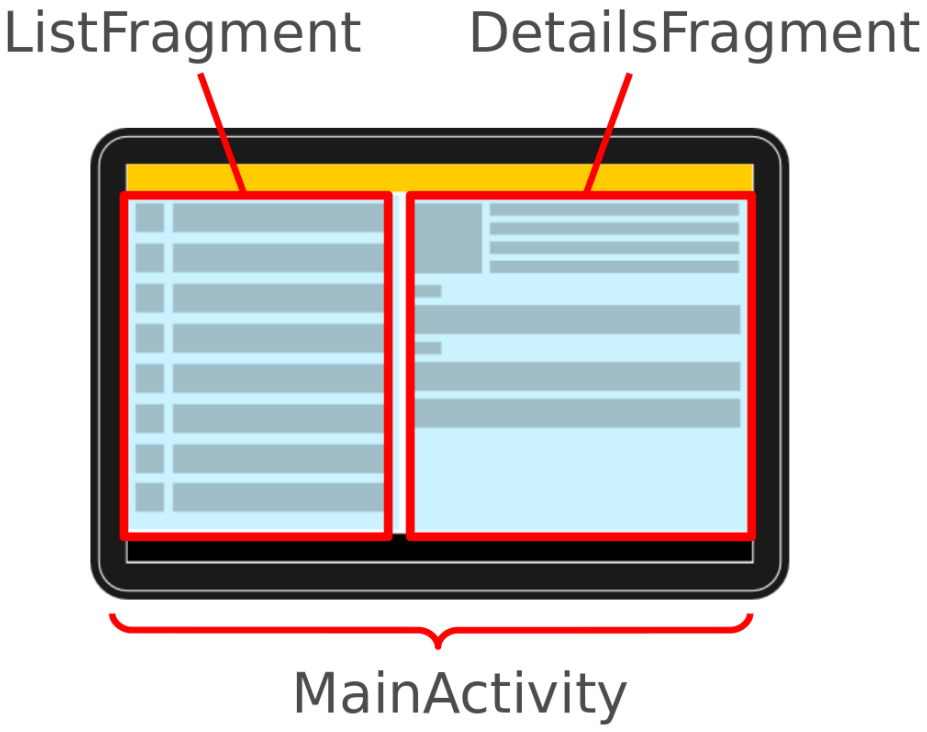
2.2. When to use Fragments

Fragments make it easy to re-use components in different layouts, e.g. you can build single-pane layouts for handsets (phones) and multi-pane layouts for tablets. This is not limited to tablets; for example you can use Fragments also to support different layout for landscape and portrait orientation.

The typical example is a list of items in an activity. On a tablet you see the details immediately on the same screen on the right hand side if you click on item. On a handset you jump to a new detail screen. The following discussion will assume that you have two Fragments (main and detail) but you can also have more. We will also have one main activity and one detailed activity. On a tablet the main activity contains both Fragments in its layout, on a handheld it only contains the main fragment.

The following screenshots demonstrates this usage.





2.3. How to work with Fragments

To create different layouts with Fragments you can:

- Use one activity, which displays two Fragments for tablets and only one on handsets devices. In this case you would switch the Fragments in the activity whenever necessary. This requires that the fragment is not declared in the layout file as such Fragments cannot be removed during runtime. It also requires an update of the action bar if the action bar status depends on the fragment.
- Use separate activities to host each fragment on a handset. For example, when the tablet UI uses two Fragments in an activity, use the same activity for handsets, but supply an alternative layout that includes just one fragment. When you need to switch Fragments, start another activity that hosts the other fragment.

The second approach is the most flexible and in general preferable way of using Fragments. In this case the main activity checks if the detail fragment is available in the layout. If the detailed fragment is there, the main activity tells the fragment that it should update itself. If the detail fragment is not available the main activity starts the detailed activity.

3. Defining and using Fragments

3.1. Defining Fragments

To define a new fragment you extend either the `android.app.Fragment` class or one of its subclasses, for example `ListFragment`, `DialogFragment`, `PreferenceFragment` or `WebViewFragment`. The following code shows an example Fragment class.

```
package com.example.android.rssfeed;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_rssitem_detail,
        container, false);
    return view;
  }

  public void setText(String item) {
    TextView view = (TextView) getView().findViewById(R.id.detailsText);
    view.setText(item);
  }
}
```

To use your new fragment you can statically add it to an XML layout file. Alternatively you can also modify fragments at runtime.

To check if the fragment is already part of your layout you can use the `FragmentManager` class.

```
DetailFragment fragment = (DetailFragment) getFragmentManager().
   findFragmentById(R.id.detail_frag);
if (fragment==null || ! fragment.isInLayout()) {
  // start new Activity
  }
else {
  fragment.update(...);
}
```

3.2. Adding Fragments to layout files

If a Fragment component is defined in an XML layout file, the `android:name` attribute points to the corresponding class.

3.3. Fragment lifecycle

A fragment is always connected to an activity.

If an activity stops, its fragments are also stopped; if an activity is destroyed its fragments are also destroyed.

The `onCreateView()` method is called by Android once the `Fragment` should create its user interface. Here you can inflate a layout via the `inflate()` method call of the `Inflator` object passed as a parameter to this method. There is no need to implement this method for headless `fragments`.

The `onActivityCreated()` is called after the `onCreateView()` method when the host activity is created. Here you can instantiate objects which require a `Context` object.

Fragments don't subclass the `Context` you have to use the `getActivity()` method to get the parent activity.

The `onStart()` method is called once the `fragment` gets visible.

3.4. Application communication with Fragments

To increase reuse of Fragments they should not directly communicate with each other. Every communication of the Fragments should be done via the host activity.

For this purpose a `Fragment` should define an interface as an inner type and require that the activity which uses it, must implement this interface. This way you avoid that the `Fragment` has any knowledge about the activity which uses it. In its `onAttach()` method it can check if the activity correctly implements this interface.

For example, assume you have a Fragment which should communicate a value to its parent activity. This can be implemented like the following.

```
package com.example.android.rssfeed;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MyListFragment extends Fragment {

  private OnItemSelectedListener listener;

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_rsslist_overview,
        container, false);
    Button button = (Button) view.findViewById(R.id.button1);
    button.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        updateDetail();
      }
    });
    return view;
  }

  public interface OnItemSelectedListener {
    public void onRssItemSelected(String link);
  }

  @Override
  public void onAttach(Activity activity) {
    super.onAttach(activity);
    if (activity instanceof OnItemSelectedListener) {
      listener = (OnItemSelectedListener) activity;
    } else {
      throw new ClassCastException(activity.toString()
          + " must implemenet MyListFragment.OnItemSelectedListener");
    }
  }

  @Override
  public void onDetach() {
    super.onDetach();
    listener = null;
  }

  // May also be triggered from the Activity
  public void updateDetail() {
    // Create a string, just for testing
    String newTime = String.valueOf(System.currentTimeMillis());

    // Inform the Activity about the change based
    // interface defintion
    listener.onRssItemSelected(newTime);
  }
}
```

4. Persisting data in Fragments

4.1. Persisting data between application restarts

In fragments you also need to store your application data. For this you can persists the data in a central place. For example

- SQlite database
- File

- The application object, if this case the application need to handle the storage

4.2. Persisting data between configurations changes

If you want to persists data between configuration changes you can also use the application object.

In addition to that you can use the `setRetainState(true)` method call on the fragments. This retains the instance of the fragments between configuration changes but only works if the fragments is not added to the backstack. Using this method is not recommend by Google for fragments which have an user interface. In this case the data must be stored as member (field).

If the data which should be stored is supported by the `Bundle` class, you can use the `onSaveInstanceState()` method to place the data in the Bundle, and retrieve that data the `onActivityCreated()` method.

5. Modifying Fragments at runtime

The `FragmentManager` class and the `FragmentTransaction` class allow you to add, remove and replace fragments in the layout of your activity.

Fragments can be dynamically modified via transactions. To dynamically add Fragments to an existing layout you typically define a container in the XML layout file in which you add a Fragment. For this you can use for example a `FrameLayout` element.

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.replace(R.id.your_placehodler, new YourFragment());
ft.commit();
```

A new Fragment will replace an existing Fragment that was previously added to the container.

If you want to add the transaction to the backstack of Android you use the `addToBackStack()` method. This will add the action to the history stack of the activity, i.e. this will allow to revert the Fragment changes via the back button.

6. Animations for Fragment transition

During a Fragment transaction you can define animations which should be used based on the Property Animation API via the `setCustomAnimations()` method.

You can also use several standard animations provided by Android via the `setTransition()` method call. These are defined via the constants starting with `FragmentTransaction.TRANSIT_FRAGMENT_*`.

Both methods allow you to define an entry animation and an exist animation.

7. Adding Fragments transition to the backstack

You can add a `FragmentTransition` to the backstack to allow the user to use the back button to reverse the transition.

For this you can use the `addToBackStack()` method on the `FragmentTransition` object.

8. Fragments for background processing

8.1. Headless Fragments

Fragments can be used without defining a user interface.

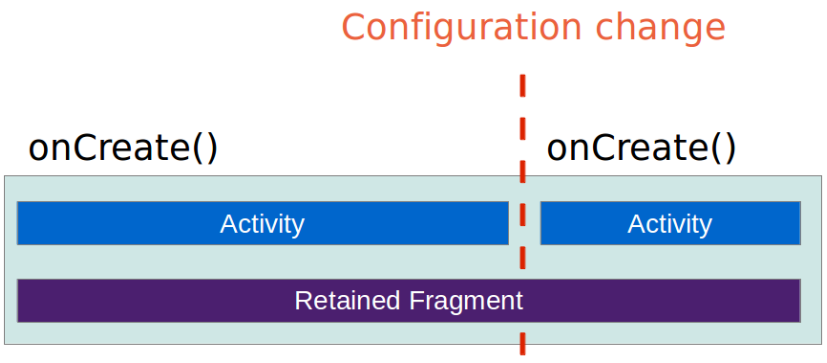
To implement a headless fragment simply return `null` in the `onCreateView()` method of your fragment.

> **Tip**
>
> It is recommended to use headless Fragments for your background processing in combination with the `setRetainInstance()` method. This

way you don't have to handle the configuration changes during your asynchronous processing yourself.

8.2. Retained headless fragments to handle configuration changes

Headless fragment are typically used to encapsulate some state across configuration changes or for a background processing task. For this purpose you would set your headless fragment to be retained. A retained fragment is not destroyed during configuration changes.



To set your fragment to retained, call its `setRetainInstance()` method.

To add such a Fragment to an activity you use the `add()` method of the `FragmentManager` class. If you need to refer to this Fragment later, you need to add it with a tag to be able to search for it via the `findFragmentByTag()` method of the `FragmentManager`.

Warning

The usage of the `onRetainNonConfigurationInstance()` is deprecated and should be replaced by retained headless fragments.

9. Contributing to the ActionBar

Fragments can also contribute entries to the ActionBar. To do this, call `setHasOptionsMenu()` in the `onCreate()` method of the fragment. The Android framework calls in this case the `onCreateOptionsMenu()` method in the Fragment class and adds its menu items to the ones added by the activity.

10. Fragments Tutorial

10.1. Overview

The following tutorial demonstrates how to use Fragments. The application will use layouts with different fragments depending on portrait and landscape mode.

In portrait mode the RssfeedActivity will show one Fragment. From this Fragments the user can navigate to another activity which contains another Fragment.

In landscape mode RssfeedActivity will show both Fragments side by side.

10.2. Create Project

Create a new Android project with the following data.

Table 1. Android project

Property	Value
Application Name	RSS Reader

Project Name	com.example.android.rssfeed
Package name	com.example.android.rssfeed
Template	BlankActivity
Activity	RssfeedActivity
Layout	activity_rssfeed

10.3. Create standard layouts

Create or change the following layout files in the *res/layout/* folder.

Create a new layout file called *fragment_rssitem_detail.xml*. This layout file will be used by the `DetailFragment`.

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

    <TextView
        android:id="@+id/detailsText"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="center_horizontal|center_vertical"
        android:layout_marginTop="20dip"
        android:text="Default Text"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textSize="30dip" />

</LinearLayout>
```

Create a new layout file called *fragment_rsslist_overview.xml*. This layout file will be used by the `MyListFragment`.

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

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Press to update"
         />

</LinearLayout>
```

Change the existing *activity_rssfeed.xml* file. This layout is the default layout for `RssfeedActivity` and shows two Fragments.

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

    <fragment
        android:id="@+id/listFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="match_parent"
        android:layout_marginTop="?android:attr/actionBarSize"
        class="com.example.android.rssfeed.MyListFragment" ></fragment>

    <fragment
        android:id="@+id/detailFragment"
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"
        class="com.example.android.rssfeed.DetailFragment" >
        <!-- Preview: layout=@layout/details -->
    </fragment>

</LinearLayout>
```

10.4. Create Fragment classes

You create now the `Fragment` classes. Start with the `DetailFragment` class.

```
package com.example.android.rssfeed;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_rssitem_detail,
        container, false);
    return view;
  }

  public void setText(String item) {
    TextView view = (TextView) getView().findViewById(R.id.detailsText);
    view.setText(item);
  }
}
```

Create the `MyListFragment` class. Despite its name it will not display a list of items, it will just have a button which allow to send the current time to the details fragment.

```
package com.example.android.rssfeed;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MyListFragment extends Fragment {

  private OnItemSelectedListener listener;

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_rsslist_overview,
        container, false);
    Button button = (Button) view.findViewById(R.id.button1);
    button.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        updateDetail();
      }
    });
    return view;
  }

  public interface OnItemSelectedListener {
      public void onRssItemSelected(String link);
    }

  @Override
    public void onAttach(Activity activity) {
      super.onAttach(activity);
      if (activity instanceof OnItemSelectedListener) {
        listener = (OnItemSelectedListener) activity;
      } else {
        throw new ClassCastException(activity.toString()
            + " must implemenet MyListFragment.OnItemSelectedListener");
      }
    }


  // May also be triggered from the Activity
  public void updateDetail() {
    // Create fake data
    String newTime = String.valueOf(System.currentTimeMillis());
    // Send data to Activity
    listener.onRssItemSelected(newTime);
  }
}
```

10.5. RssfeedActivity

Change the `RssfeedActivity` class to the following code.

```
package com.example.android.rssfeed;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class RssfeedActivity extends Activity implements MyListFragment.OnItemSelect
edListener{

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

    // if the wizard generated an onCreateOptionsMenu you can delete
    // it, not needed for this tutorial

  @Override
  public void onRssItemSelected(String link) {
    DetailFragment fragment = (DetailFragment) getFragmentManager()
            .findFragmentById(R.id.detailFragment);
        if (fragment != null && fragment.isInLayout()) {
          fragment.setText(link);
        }
  }

}
```

10.6. Run

Run your example. Both fragments should be displayed both in landscape and portrait mode. If you press the button in the `ListFragment` the `DetailFragment` should get updated.

11. Fragments Tutorial - layout for portrait mode

11.1. Create layouts for portrait mode

The RssfeedActivity should use a special layout file in portrait mode. In portrait mode Android will check the *`layout-port`* folder for fitting layout files. If Android does not find a fitting layout file it uses the *`layout`* folder.

For this reason create the *`res/layout-port`* folder. Afterwards create the following *`activity_rssfeed.xml`* layout file in the *`res/layout-port`* folder.

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

    <fragment
        android:id="@+id/listFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="?android:attr/actionBarSize"
        class="com.example.android.rssfeed.MyListFragment" />
</LinearLayout>
```

Also create the *`activity_detail.xml`* layout file. This layout will be used in the `DetailActivity`. Please note that we could have create this file also in the *`res/layout`* folder, but it is only used in portrait mode hence we place it into this folder.

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

    <fragment
        android:id="@+id/detailFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.example.android.rssfeed.DetailFragment" />

</LinearLayout>
```

11.2. DetailActivity

Create a new activity called `DetailActivity` with the following class.

```
package com.example.android.rssfeed;

import android.app.Activity;
import android.content.res.Configuration;
import android.os.Bundle;
import android.widget.TextView;

public class DetailActivity extends Activity {

  public static final String EXTRA_URL = "url";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Need to check if Activity has been switched to landscape mode
    // If yes, finished and go back to the start Activity
    if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LAND
SCAPE) {
      finish();
      return;
    }
    setContentView(R.layout.activity_detail);
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
      String s = extras.getString(EXTRA_URL);
      TextView view = (TextView) findViewById(R.id.detailsText);
      view.setText(s);
    }
  }
}
```

Ensure that you also register this activity in the *AndroidManifest.xml* file.

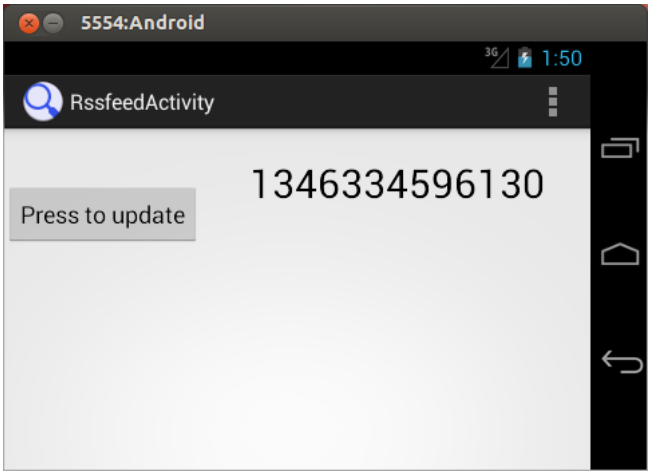
11.3. Adjust the RssfeedActivity activity

Adjust the RssfeedActivity to display the `DetailActivity` in case the other Fragment is not present in the layout.

```
package com.example.android.rssfeed;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;

public class RssfeedActivity extends Activity implements
    MyListFragment.OnItemSelectedListener {

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

  @Override
  public void onRssItemSelected(String link) {
    DetailFragment fragment = (DetailFragment) getFragmentManager()
        .findFragmentById(R.id.detailFragment);
    if (fragment != null && fragment.isInLayout()) {
      fragment.setText(link);
    } else {
      Intent intent = new Intent(getApplicationContext(),
          DetailActivity.class);
      intent.putExtra(DetailActivity.EXTRA_URL, link);
      startActivity(intent);

    }
  }

}
```

11.4. Run

Run your example. If you run the application in portrait mode you should see only one Fragment. Use the **Ctrl**+**F11** shortcut to switch the orientation. In horizontal mode you see two Fragments. If you press the button in portrait mode the a new `DetailActivity` is started and shows the current time. In horizontal mode you see both Fragments.



12. Tutorial: Use resource qualifiers

Adjust the RssFeed reader so that the second column is used in case you have more then 600dp available width. Remove the selection based on orientation mode.

13. Thank you

Please help me to support this article:

14. Questions and Discussion

Before posting questions, please see the **vogella FAQ**. If you have questions or find an error in this article please use the **www.vogella.com Google Group**. I have created a short list **how to create good questions** which might also help you.

15. Links and Literature

15.1. Source Code

Source Code of Examples

15.2. Android Resources

Android Animations

Android Background Processing

15.3. Resources

15.3.1. Android Online Resources

vogella Android tutorials

Android Developer Homepage

Android Issues / Bugs

Android Google Groups

Source code of all vogella examples

Google Android Add-ons, e.g. Maps

15.4. vogella Resources

vogella Training Android and Eclipse Training from the vogella team

Android Tutorial Introduction to Android Programming

GWT Tutorial Program in Java and compile to JavaScript and HTML

Eclipse RCP Tutorial Create native applications in Java

JUnit Tutorial Test your application

Git Tutorial Put everything you have under distributed version control system