# 41. An Android Studio Master/Detail Flow Tutorial

This chapter will explain the concept of the Master/Detail user interface design before exploring, in detail, the elements that make up the Master/Detail Flow template included with Android Studio. An example application will then be created that demonstrates the steps involved in modifying the template to meet the specific needs of the application developer.

## 41.1 The Master/Detail Flow

A master/detail flow is an interface design concept whereby a list of items (referred to as the *master list*) is displayed to the user. On selecting an item from the list, additional information relating to that item is then presented to the user within a *detail* pane. An email application might, for example, consist of a master list of received messages consisting of the address of the sender and the subject of the message. Upon selection of a message from the master list, the body of the email message would appear within the detail pane.

On tablet sized Android device displays in landscape orientation, the master list appears in a narrow vertical panel along the left-hand edge of the screen. The remainder of the display is devoted to the detail pane in an arrangement referred to as *two-pane mode*. Figure 41-1, for example, shows the master/detail, two-pane arrangement with master items listed and the content of item one displayed in the detail pane:

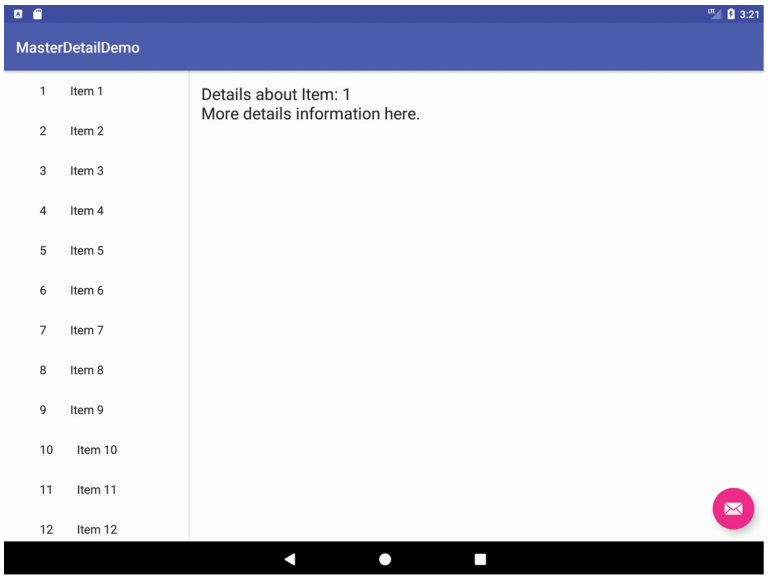


Figure 41-1

On smaller, phone sized Android devices, the master list takes up the entire screen and the detail pane appears on a separate screen which appears when a selection is made from the master list. In this mode, the detail screen includes an action bar entry to return to the master list. Figure 41-2 for example, illustrates both the master and detail screens for the same item list on a 4" phone screen:

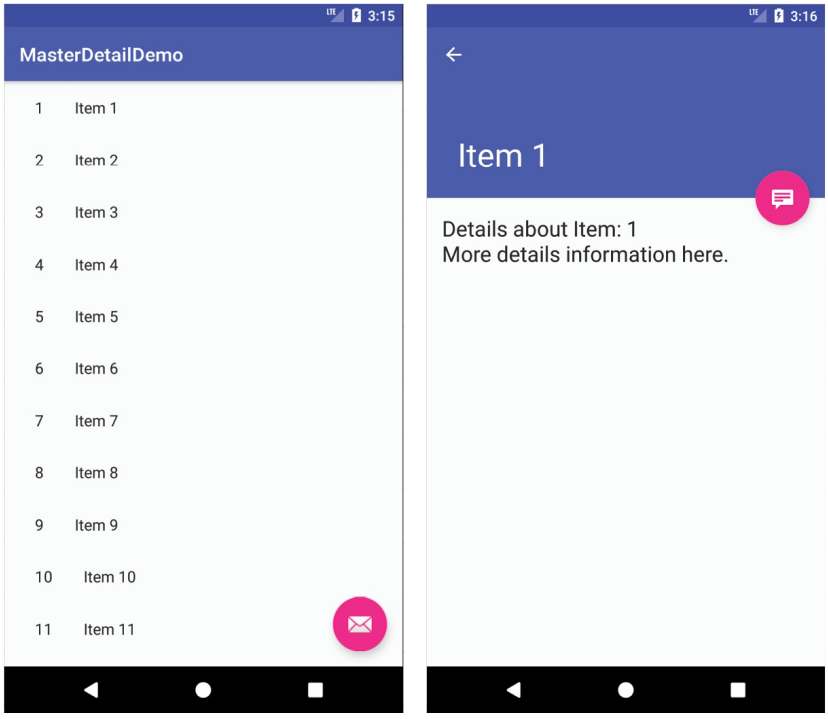


Figure 41-2

## 41.2 Creating a Master/Detail Flow Activity

In the next section of this chapter, the different elements that comprise the Master/Detail Flow template will be covered in some detail. This is best achieved by creating a project using the Master/Detail Flow template to use while working through the information. This project will subsequently be used as the basis for the tutorial at the end of the chapter.

Create a new project in Android Studio, entering *MasterDetailFlow* into the Application name field and *ebookfrenzy.com* as the Company Domain setting before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK setting to API 14: Android 4.0 (IceCreamSandwich). When selecting a minimum SDK of less than API 14, Android Studio creates a Master/Detail Flow project template that uses an outdated and less efficient approach to handling the list of items displayed in the master panel. After the project has been created, the *minSdkVersion* setting in the *build.gradle (module: app)* file located under *Gradle Scripts* in the Project tool window may be changed to target older Android versions if required.

When the activity configuration screen of the New Project dialog appears, select the *Master/Detail Flow* option as illustrated in Figure 41-3 before clicking on *Next* once again:

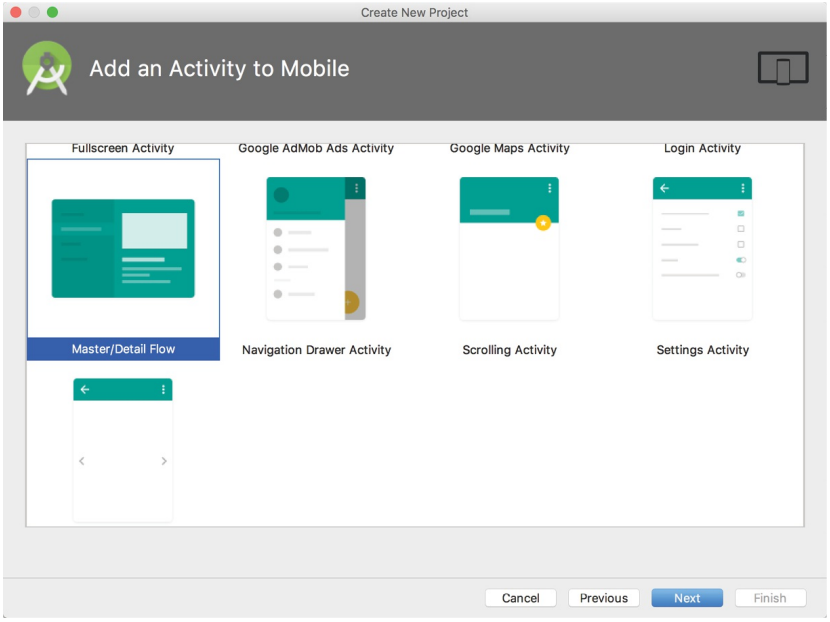


Figure 41-3

The next screen (Figure 41-4) provides the opportunity to configure the objects that will be displayed within the master/detail activity. In the tutorial later in this chapter, the master list will contain a number of web site names which, when selected, will load the chosen web site into a web view within the detail pane. With these requirements in mind, set the *Object Kind* field to "Website", and the *Object Kind Plural* and *Title* settings to "Websites".

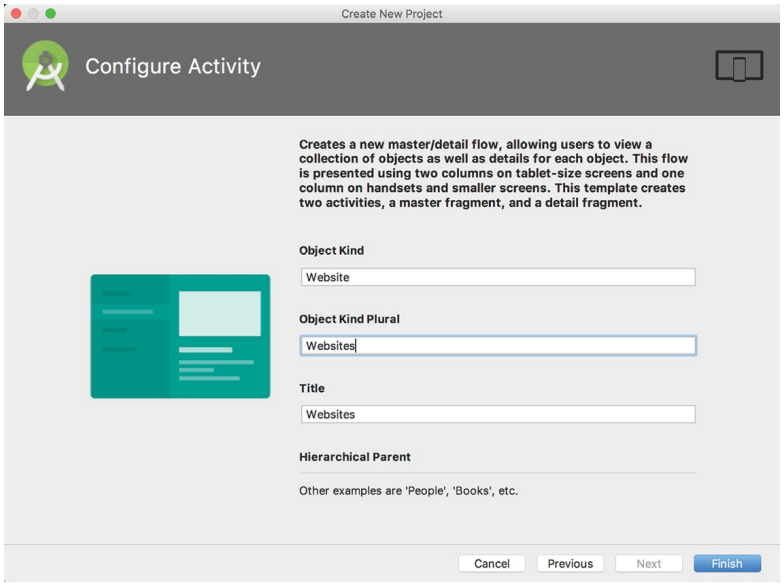


Figure 41-4

Finally, click Finish to create the new Master/Detail Flow based application project.

## 41.3 The Anatomy of the Master/Detail Flow Template

Once a new project has been created using the Master/Detail Flow template, a number of Java and XML layout resource files will have been created automatically. It is important to gain an understanding of these different files in order to be able to adapt the template to specific requirements. A review of the project within the Android Studio Project tool window will reveal the following files, where *<item>* is replaced by the Object Kind name that was specified when the project was created (this being "Website" in the case of the *MasterDetailFlow* example project):

- **activity_<item>_list.xml** – The top level layout file for the master list, this file is loaded by the *<item>ListActivity* class. This layout contains a toolbar, a floating action button and

includes the *<item>_list.xml* file.

- **<item>ListActivity.java** – The activity class responsible for displaying and managing the master list (declared in the *activity_<item>_list.xml* file) and for both displaying and responding to the selection of items within that list.
- **<item>_list.xml** – The layout file used to display the master list of items in single-pane mode where the master list and detail pane appear on different screens. This file consists of a RecyclerView object configured to use the LinearLayoutManager. The RecyclerView element declares that each item in the master list is to be displayed using the layout declared within the *<item>_list_content.xml* file.
- **<item>_list.xml (w900dp)** – The layout file for the master list in the two-pane mode used on tablets in landscape (where the master list and detail pane appear side by side). This file contains a horizontal LinearLayout parent within which resides a RecyclerView to display the master list, and a FrameLayout to contain the content of the detail pane. As with the single-pane variant of this file, the RecyclerView element declares that each item in the list be displayed using the layout contained within the *<item>_list_content.xml* file.
- **<item>_content_list.xml** – This file contains the layout to be used for each item in the master list. By default, this consists of two TextView objects embedded in a horizontal LinearLayout but may be changed to meet specific application needs.
- **activity_<item>_detail.xml** – The top level layout file used for the detail pane when running in single-pane mode. This layout contains an app bar, collapsing toolbar, scrolling view and a floating action button. At runtime this layout file is loaded and displayed by the *<item>DetailActivity* class.

- **<item>DetailActivity.java** – This class displays the layout defined in the *activity_<item>_detail.xml* file. The class also initializes and displays the fragment containing the detail content defined in the *item_detail.xml* and *<item>DetailFragment.java* files.
- **<item>_detail.xml** – The layout file that accompanies the *<item>DetailFragment* class and contains the layout for the content area of the detail pane. By default, this contains a single TextView object, but may be changed to meet your specific application needs. In single-pane mode, this fragment is loaded into the layout defined by the *activity_<item>_detail.xml* file. In two-pane mode, this layout is loaded into the FrameLayout area of the *<item>_list.xml (w900dp)* file so that it appears adjacent to the master list.
- **<item>DetailFragment.java** – The fragment class file responsible for displaying the *<item>_detail.xml* layout and populating it with the content to be displayed in the detail pane. This fragment is initialized and displayed within the *<item>DetailActivity.java* file to provide the content displayed within the *activity_<item>_detail.xml* layout for single-pane mode and the *<item>_list.xml (w900dp)* layout for two-pane mode.
- **DummyContent.java** – A class file intended to provide sample data for the template. This class can either be modified to meet application needs, or replaced entirely. By default, the content provided by this class simply consists of a number of string items.

## 41.4 Modifying the Master/Detail Flow Template

While the structure of the Master/Detail Flow template can appear confusing at first, the concepts will become clearer as the default template is modified in the remainder of this chapter. As will become evident, much of the functionality provided by the template can remain unchanged for many

master/detail implementation requirements.

In the rest of this chapter, the *MasterDetailFlow* project will be modified such that the master list displays a list of web site names and the detail pane altered to contain a WebView object instead of the current TextView. When a web site is selected by the user, the corresponding web page will subsequently load and display in the detail pane.

## 41.5 Changing the Content Model

The content for the example as it currently stands is defined by the *DummyContent* class file. Begin, therefore, by selecting the *DummyContent.java* file (located in the Project tool window in the *app -> java -> com.ebookfrenzy.masterdetailflow -> dummy* folder) and reviewing the code. At the bottom of the file is a declaration for a class named *DummyItem* which is currently able to store two String objects representing a content string and an ID. The updated project, on the other hand, will need each item object to contain an ID string, a string for the web site name, and a string for the corresponding URL of the web site. To add these features, modify the *DummyItem* class so that it reads as follows:

```
public static class DummyItem {
        public String id;
        public String website_name;
        public String website_url;

        public DummyItem(String id, String website_name,
              String website_url)
        {
               this.id = id;
               this.website_name = website_name;
               this.website_url = website_url;
        }

        @Override
        public String toString() {
               return website_name;
        }
}
```

Note that the encapsulating DummyContent class currently contains a *for* loop that adds 25 items by making multiple calls to methods named

*createDummyItem()* and *makeDetails()*. Much of this code will no longer be required and should be deleted from the class as follows:

```
public static Map<String, DummyItem> ITEM_MAP = new HashMap<String,
DummyItem>();

private static final int COUNT = 25;

static {
    // Add some sample items.
    for (int i = 1; i <= COUNT; i++) {
        addItem(createDummyItem(i));
    }
}

private static void addItem(DummyItem item) {
    ITEMS.add(item);
    ITEM_MAP.put(item.id, item);
}

private static DummyItem createDummyItem(int position) {
    return new DummyItem(String.valueOf(position), "Item " +
position, makeDetails(position));
}

private static String makeDetails(int position) {
    StringBuilder builder = new StringBuilder();
    builder.append("Details about Item: ").append(position);
    for (int i = 0; i < position; i++) {
        builder.append("\nMore details information here.");
    }
    return builder.toString();
}
```

This code needs to be modified to initialize the data model with the required w

```
public static final Map<String, DummyItem> ITEM_MAP =
          new HashMap<String, DummyItem>();

static {
       // Add 3 sample items.
        addItem(new DummyItem("1", "eBookFrenzy",
                "http://www.ebookfrenzy.com"));
        addItem(new DummyItem("2", "Amazon",
                "http://www.amazon.com"));
```

```
        addItem(new DummyItem("3", "New York Times",
                "http://www.nytimes.com"));
}
```

The code now takes advantage of the modified DummyItem class to store an ID, web site name and URL for each item.

## 41.6 Changing the Detail Pane

The detail information shown to the user when an item is selected from the master list is currently displayed via the layout contained in the *website_detail.xml* file. By default, this contains a single view in the form of a TextView. Since the TextView class is not capable of displaying a web page, this needs to be changed to a WebView object for this tutorial. To achieve this, navigate to the *app -> res -> layout -> website_detail.xml* file in the Project tool window and double-click on it to load it into the Layout Editor tool. Switch to Text mode and delete the current XML content from the file. Replace this content with the following XML:

```
<WebView 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:id="@+id/website_detail"
    tools:context=
        "com.ebookfrenzy.masterdetailflow.WebsiteDetailFragment">
</WebView>
```

Switch to Design mode and verify that the layout now matches that shown in Figure 41-5:

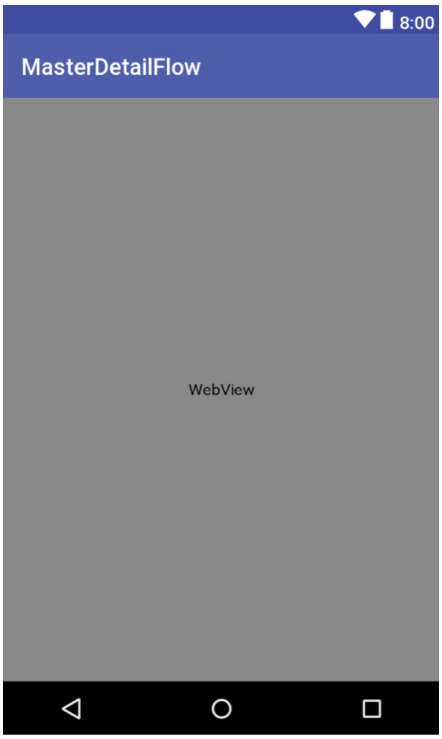


Figure 41-5

## 41.7 Modifying the WebsiteDetailFragment Class

At this point the user interface detail pane has been modified but the corresponding Java class is still designed for working with a TextView object instead of a WebView. Load the source code for this class by double-clicking on the *WebsiteDetailFragment.java* file in the Project tool window.

In order to load the web page URL corresponding to the currently selected item only a few lines of code need to be changed. Once this change has been made, the code should read as follows:

```
package com.ebookfrenzy.masterdetailflow;
import android.app.Activity;
import android.support.design.widget.CollapsingToolbarLayout;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import android.webkit.WebViewClient;
import android.webkit.WebView;
import android.webkit.WebResourceRequest;
```

```
import com.ebookfrenzy.masterdetailflow.dummy.DummyContent;

public class WebSiteDetailFragment extends Fragment {
.
.
.
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getArguments().containsKey(ARG_ITEM_ID)) {
            // Load the dummy content specified by the fragment
            // arguments. In a real-world scenario, use a Loader
            // to load content from a content provider.
            mItem =
DummyContent.ITEM_MAP.get(getArguments().getString(ARG_ITEM_ID));

            Activity activity = this.getActivity();
            CollapsingToolbarLayout appBarLayout =
(CollapsingToolbarLayout) activity.findViewById(R.id.toolbar_layout);
            if (appBarLayout != null) {
                appBarLayout.setTitle(mItem.website_name);
            }
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
              ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
             R.layout.fragment_website_detail, container, false);

        // Show the dummy content as text in a TextView.
        if (mItem != null) {
            ((WebView) rootView.findViewById(R.id.website_detail))
                    .loadUrl(mItem.website_url);
            WebView webView = (WebView)
                 rootView.findViewById(R.id.website_detail);
            webView.setWebViewClient(new WebViewClient(){
                @Override
                public boolean shouldOverrideUrlLoading(
                    WebView view, WebResourceRequest request) {
                    return super.shouldOverrideUrlLoading(
```

```
                                    view, request);
                }
            });
            webView.getSettings().setJavaScriptEnabled(true);
            webView.loadUrl(mItem.website_url);
        }

        return rootView;
    }
}
```

The above changes modify the *onCreate()* method to display the web site name on the app bar:

```
appBarLayout.setTitle(mItem.website_name);
```

The *onCreateView()* method is then modified to find the view with the ID of *website_detail* (this was formally the TextView but is now a WebView) and extract the URL of the web site from the selected item. An instance of the WebViewClient class is created and assigned the *shouldOverrideUrlLoading()* callback method. This method is implemented so as to force the system to use the WebView instance to load the page instead of the Chrome browser. Finally, JavaScript support is enabled on the webView instance and the web page loaded.

## 41.8 Modifying the WebsiteListActivity Class

A minor change also needs to be made to the *WebsiteListActivity.java* file to make sure that the web site names appear in the master list. Edit this file, locate the *onBindViewHolder()* method and modify the *setText()* method call to reference the web site name as follows:

```
public void onBindViewHolder(final ViewHolder holder, int position) {
    holder.mItem = mValues.get(position);
    holder.mIdView.setText(mValues.get(position).id);
    holder.mContentView.setText(mValues.get(position).website_name);
.
.
}
```

## 41.9 Adding Manifest Permissions

The final step is to add internet permission to the application via the manifest file. This will enable the WebView object to access the internet and download web pages. Navigate to, and load the *AndroidManifest.xml* file in the Project

tool window (*app* -> *manifests*), and double-click on it to load it into the editor. Once loaded, add the appropriate permission line to the file:

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >.
.
.
```

## 41.10 Running the Application

Compile and run the application on a suitably configured emulator or an attached Android device. Depending on the size of the display, the application will appear either in small screen or two-pane mode. Regardless, the master list should appear primed with the names of the three web sites defined in the content model. Selecting an item should cause the corresponding web site to appear in the detail pane as illustrated in two-pane mode in Figure 41-6:

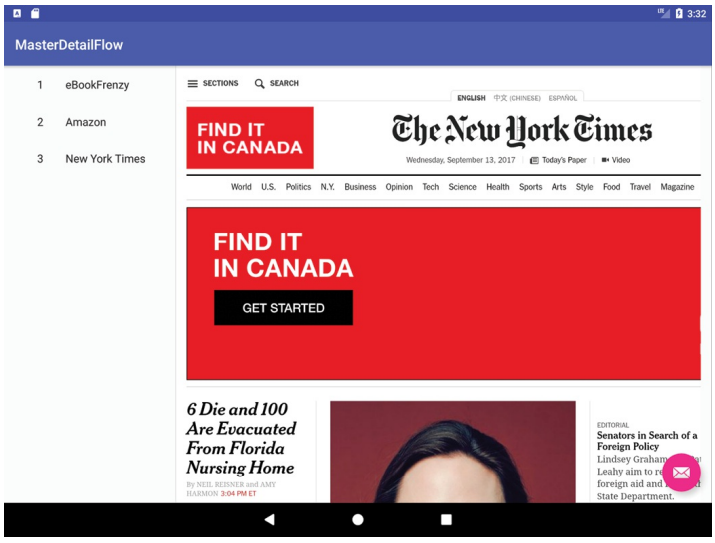


Figure 41-6

## 41.11 Summary

A master/detail user interface consists of a master list of items which, when selected, displays additional information about that selection within a detail pane. The Master/Detail Flow is a template provided with Android Studio that allows a master/detail arrangement to be created quickly and with relative ease. As demonstrated in this chapter, with minor modifications to the default template files, a wide range of master/detail based functionality can be implemented with minimal coding and design effort.