61. An Android Storage Access Framework Example

As previously discussed, the Storage Access Framework considerably eases the process of integrating cloud based storage access into Android applications. Consisting of a picker user interface and a set of new intents, access to files stored on document providers such as Google Drive and Box can now be built into Android applications with relative ease. With the basics of the Android Storage Access Framework covered in the preceding chapter, this chapter will work through the creation of an example application which uses the Storage Access Framework to store and manage files.

61.1 About the Storage Access Framework Example

The Android application created in this chapter will take the form of a rudimentary text editor designed to create and store text files remotely onto a cloud based storage service. In practice, the example will work with any cloud based document storage provider that is compatible with the Storage Access Framework, though for the purpose of this example the use of Google Drive is assumed.

In functional terms, the application will present the user with a multi-line text view into which text may be entered and edited, together with a set of buttons allowing storage based text files to be created, opened and saved.

61.2 Creating the Storage Access Framework Example

Create a new project in Android Studio, entering *StorageDemo* 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 19: Android 4.4 (KitKat). Continue to proceed through the screens, requesting the creation of an Empty Activity named *StorageDemoActivity* with a corresponding layout named *activity_storage_demo*.

61.3 Designing the User Interface

The user interface will need to be comprised of three Button views and a

single EditText view. Within the Project tool window, navigate to the *activity_storage_demo.xml* layout file located in *app -> res -> layout* and double-click on it to load it into the Layout Editor tool. With the tool in Design mode, select and delete the *Hello World!* TextView object.

Drag and position a Button widget in the top left-hand corner of the layout so that both the left and top dotted margin guidelines appear before dropping the widget in place. Position a second Button such that the center and top margin guidelines appear. The third Button widget should then be placed so that the top and right-hand margin guidelines appear.

Change the text attributes on the three buttons to "New", "Open" and "Save" respectively. Next, position a Plain Text widget so that it is centered horizontally and positioned beneath the center Button so that the user interface layout matches that shown in Figure 61-1. Use the Infer Constraints button in the Layout Editor toolbar to add any missing constraints.

Select the Plain Text widget in the layout, delete the current text property setting so that the field is initially blank and set the ID to *fileText*, remembering to extract all the string attributes to resource values:

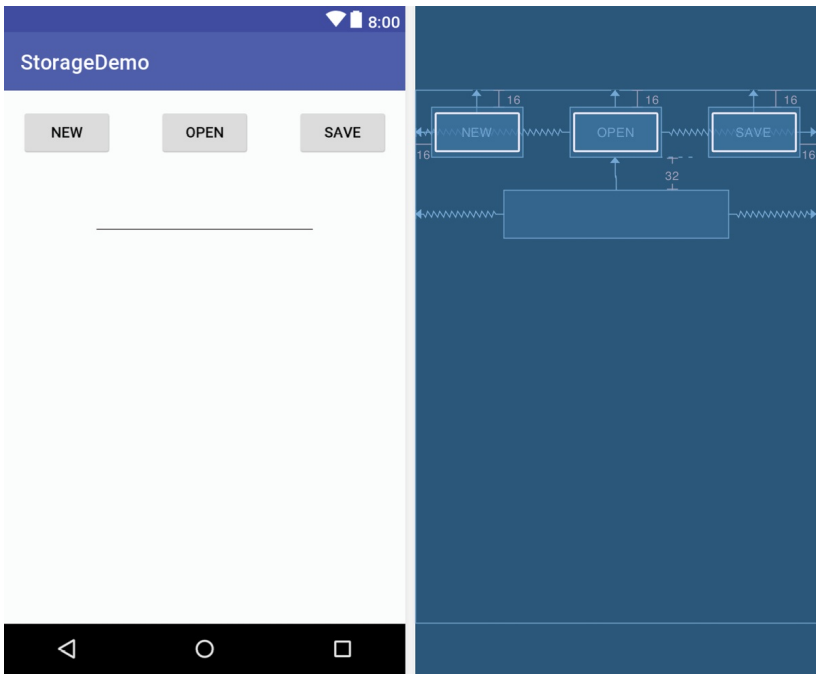

Figure 61-1

Using the Attributes tool window, configure the onClick property on the Button widgets to call methods named *newFile*, *openFile* and *saveFile* respectively.

61.4 Declaring Request Codes

Working with files in the Storage Access Framework involves triggering a variety of intents depending on the specific action to be performed. Invariably this will result in the framework displaying the storage picker user interface so that the user can specify the storage location (such as a directory on Google Drive and the name of a file). When the work of the intent is complete, the application will be notified by a call to a method named *onActivityResult()*.

Since all intents from a single activity will result in a call to the same *onActivityResult()* method, a mechanism is required to identify which intent triggered the call. This can be achieved by passing a request code through to the intent when it is launched. This code is then passed on to the *onActivityResult()* method by the intents, enabling the method to identify which action has been requested by the user. Before implementing the onClick handlers to create, save and open files, the first step is to declare some request codes for these three actions.

Locate and load the *StorageDemoActivity.java* file into the editor and declare constant values for the three actions to be performed by the application. Also, add some code to obtain a reference to the multi-line EditText object which will be referenced in later methods:

```
package com.ebookfrenzy.storagedemo;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;

public class StorageDemoActivity extends AppCompatActivity {

    private static EditText textView;

    private static final int CREATE_REQUEST_CODE = 40;
    private static final int OPEN_REQUEST_CODE = 41;
    private static final int SAVE_REQUEST_CODE = 42;

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

```
        textView = (EditText) findViewById(R.id.fileText);
    }
}
```

61.5 Creating a New Storage File

When the New button is selected, the application will need to trigger an *ACTION_CREATE_DOCUMENT* intent configured to create a file with a plain-text MIME type. When the user interface was designed, the New button was configured to call a method named *newFile()*. It is within this method that the appropriate intent needs to be launched.

Remaining in the *StorageDemoActivity.java* file, implement this method as follows:

```
package com.ebookfrenzy.storagedemo;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.content.Intent;
import android.view.View;

public class StorageDemoActivity extends AppCompatActivity {

public class StorageDemoActivity extends Activity {

    private static EditText textView;

    private static final int CREATE_REQUEST_CODE = 40;
    private static final int OPEN_REQUEST_CODE = 41;
    private static final int SAVE_REQUEST_CODE = 42;
.
.
.
    public void newFile(View view)
    {
        Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);

        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TITLE, "newfile.txt");
```

```
        startActivityForResult(intent, CREATE_REQUEST_CODE);
    }
.
.
}
```

This code creates a new ACTION_CREATE_INTENT Intent object. This intent is then configured so that only files that can be opened with a file descriptor are returned (via the Intent.CATEGORY_OPENABLE category setting).

Next the code specifies that the file to be opened is to have a plain text MIME type and a placeholder filename is provided (which can be changed by the user in the picker interface). Finally, the intent is started, passing through the previously declared *CREATE_REQUEST_CODE*.

When this method is executed and the intent has completed the assigned task, a call will be made to the application's *onActivityResult()* method and passed, amongst other arguments, the Uri of the newly created document and the request code that was used when the intent was started. Now is an ideal opportunity to begin to implement this method.

61.6 The onActivityResult() Method

The *onActivityResult()* method will be shared by all of the intents that will be called during the lifecycle of the application. In each case, the method will be passed a request code, a result code and a set of result data which contains the Uri of the storage file. The method will need to be implemented such that it checks for the success of the intent action, identifies the type of action performed and extracts the file Uri from the results data. At this point in the tutorial, the method only needs to handle the creation of a new file on the selected document provider, so modify the *StorageDemoActivity.java* file to add this method as follows:

```
public void onActivityResult(int requestCode, int resultCode,
        Intent resultData) {

       if (resultCode == Activity.RESULT_OK)
       {
              if (requestCode == CREATE_REQUEST_CODE)
              {
               if (resultData != null) {
                      textView.setText("");
```

```
                    }
                }

            }
}
```

The code in this method is largely straightforward. The result of the activity is checked and, if successful, the request code is compared to the CREATE_REQUEST_CODE value to verify that the user is creating a new file. That being the case, the edit text view is cleared of any previous text to signify the creation of a new file.

Compile and run the application and select the New button. The Storage Access Framework should subsequently display the "Save to" storage picker user interface as illustrated in Figure 61-2.

From this menu, select the *Drive* option followed by *My Drive* and navigate to a suitable location on your Google Drive storage into which to save the file. In the text field at the bottom of the picker interface, change the name from "newfile.txt" to a suitable name (but keeping the *.txt* extension) before selecting the *Save* option.

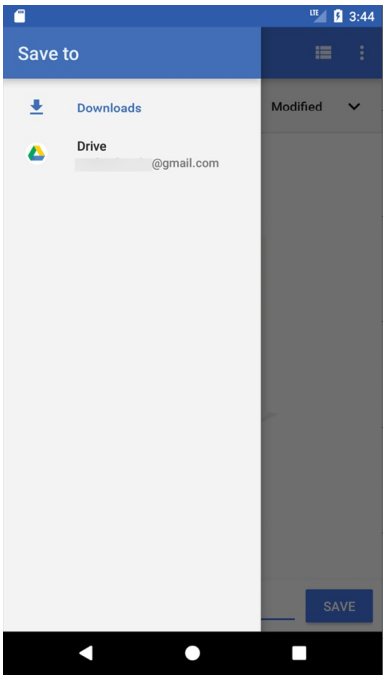

Figure 61-2

Once the new file has been created, the app should return to the main activity and a notification will appear within the notifications panel which reads "1

file uploaded".

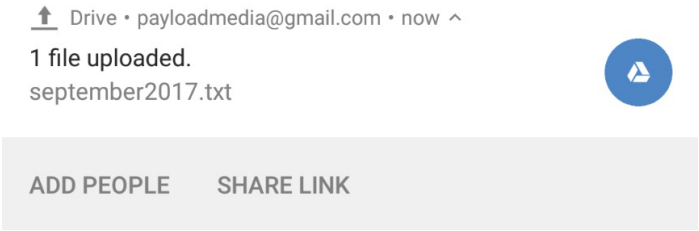

Figure 61-3

At this point, it should be possible to log into your Google Drive account in a browser window and find the newly created file in the requested location. In the event that the file is missing, make sure that the Android device on which the application is running has an active internet connection. Access to Google Drive on the device may also be verified by running the Google *Drive* app, which is installed by default on many Android devices, and available for download from the Google Play store.

61.7 Saving to a Storage File

Now that the application is able to create new storage based files, the next step is to add the ability to save any text entered by the user to a file. The user interface is configured to call the *saveFile()* method when the Save button is selected by the user. This method will be responsible for starting a new intent of type *ACTION_OPEN_DOCUMENT* which will result in the picker user interface appearing so that the user can choose the file to which the text is to be stored. Since we are only working with plain text files, the intent needs to be configured to restrict the user's selection options to existing files that match the text/plain MIME type. Having identified the actions to be performed by the *saveFile()* method, this can now be added to the *StorageDemoActivity.java* class file as follows:

```
public void saveFile(View view)
{
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/plain");

        startActivityForResult(intent, SAVE_REQUEST_CODE);
}
```

Since the SAVE_REQUEST_CODE was passed through to the intent, the

onActivityResult() method must now be extended to handle save actions:

```
package com.ebookfrenzy.storagedemo;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.content.Intent;
import android.view.View;
import android.net.Uri;

public class StorageDemoActivity extends AppCompatActivity {
.
.
       public void onActivityResult(int requestCode, int resultCode,
        Intent resultData) {

              Uri currentUri = null;

              if (resultCode == Activity.RESULT_OK)
              {
                     if (requestCode == CREATE_REQUEST_CODE)
                     {
                      if (resultData != null) {
                             textView.setText("");
                      }
                     } else if (requestCode == SAVE_REQUEST_CODE) {

                            if (resultData != null) {
                                   currentUri =
                                          resultData.getData();
                                   writeFileContent(currentUri);
                            }
                     }
              }
       }
}
```

The method now checks for the save request code, extracts the Uri of the file selected by the user in the storage picker and calls a method named *writeFileContent()*, passing through the Uri of the file to which the text is to be written. Remaining in the *StorageDemoActivity.java* file, implement this

method now so that it reads as follows:

```
package com.ebookfrenzy.storagedemo;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.content.Intent;
import android.view.View;
import android.net.Uri;
import android.os.ParcelFileDescriptor;

public class StorageDemoActivity extends AppCompatActivity {
.
.
       private void writeFileContent(Uri uri)
       {
              try{
                     ParcelFileDescriptor pfd =
                            this.getContentResolver().
                              openFileDescriptor(uri, "w");

                     FileOutputStream fileOutputStream =
                         new FileOutputStream(
                            pfd.getFileDescriptor());

                     String textContent =
                            textView.getText().toString();

                     fileOutputStream.write(textContent.getBytes());

                     fileOutputStream.close();
                     pfd.close();
              } catch (FileNotFoundException e) {
                     e.printStackTrace();
              } catch (IOException e) {
                     e.printStackTrace();
              }
```

```
        }
.
.
}
```

The method begins by obtaining and opening the file descriptor from the Uri of the file selected by the user. Since the code will need to write to the file, the descriptor is opened in write mode ("w"). The file descriptor is then used as the basis for creating an output stream that will enable the application to write to the file.

The text entered by the user is extracted from the edit text object and written to the output stream before both the file descriptor and stream are closed. Code is also added to handle any IO exceptions encountered during the file writing process.

With the new method added, compile and run the application, enter some text into the text view and select the *Save* button. From the picker interface, locate the previously created file from the Google Drive storage to save the text to that file. Return to your Google Drive account in a browser window and select the text file to display the contents. The file should now contain the text entered within the StorageDemo application on the Android device.

61.8 Opening and Reading a Storage File

Having written the code to create and save text files, the final task is to add some functionality to open and read a file from the storage. This will involve writing the *openFile()* onClick event handler method and implementing it so that it starts an ACTION_OPEN_DOCUMENT intent:

```
public void openFile(View view)
{
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/plain");
        startActivityForResult(intent, OPEN_REQUEST_CODE);
}
```

In this code, the intent is configured to filter selection to files which can be opened by the application. When the activity is started, it is passed the open request code constant which will now need to be handled within the *onActivityResult()* method:

```
public void onActivityResult(int requestCode, int resultCode,
```

```
        Intent resultData) {

        Uri currentUri = null;

        if (resultCode == Activity.RESULT_OK)
        {

               if (requestCode == CREATE_REQUEST_CODE)
               {
                if (resultData != null) {
                       textView.setText("");
                }
               } else if (requestCode == SAVE_REQUEST_CODE) {

                      if (resultData != null) {
                             currentUri = resultData.getData();
                             writeFileContent(currentUri);
                      }
               } else if (requestCode == OPEN_REQUEST_CODE) {

                      if (resultData != null) {
                              currentUri = resultData.getData();

                try {
                String content =

readFileContent(currentUri);
                       textView.setText(content);
                } catch (IOException e) {
                              // Handle error here
                              }
                      }
               }
        }
}
```

The new code added above to handle the open request obtains the Uri of the file selected by the user from the picker user interface and passes it through to a method named *readFileContent()* which is expected to return the content of the selected file in the form of a String object. The resulting string is then assigned to the text property of the edit text view. Clearly, the next task is to implement the *readFileContent()* method:

```
package com.ebookfrenzy.storagedemo;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.content.Intent;
import android.view.View;
import android.net.Uri;
import android.os.ParcelFileDescriptor;

public class StorageDemoActivity extends AppCompatActivity {
.
.
.
       private String readFileContent(Uri uri) throws IOException {

              InputStream inputStream =
                           getContentResolver().openInputStream(uri);
              BufferedReader reader =
                     new BufferedReader(new InputStreamReader(
                      inputStream));
              StringBuilder stringBuilder = new StringBuilder();
              String currentline;
              while ((currentline = reader.readLine()) != null) {
               stringBuilder.append(currentline + "\n");
              }
              inputStream.close();
              return stringBuilder.toString();
       }
.
.
}
```

This method begins by extracting the file descriptor for the selected text file and opening it for reading. The input stream associated with the Uri is then

opened and used as the input source for a BufferedReader instance. Each line within the file is then read and stored in a StringBuilder object. Once all the lines have been read, the input stream and file descriptor are both closed, and the file content is returned as a String object.

61.9 Testing the Storage Access Application

With the coding phase complete the application is now ready to be fully tested. Begin by launching the application on a physical Android device and selecting the "New" button. Within the resulting storage picker interface, select a Google Drive location and name the text file *storagedemo.txt* before selecting the Save option located to the right of the file name field.

When control returns to your application look for the file uploading notification, then enter some text into the text area before selecting the "Save" button. Select the previously created *storagedemo.txt* file from the picker to save the content to the file. On returning to the application, delete the text and select the "Open" button, once again choosing the *storagedemo.txt* file. When control is returned to the application, the text view should have been populated with the content of the text file.

It is important to note that the Storage Access Framework will cache storage files locally in the event that the Android device lacks an active internet connection. Once connectivity is re-established, however, any cached data will be synchronized with the remote storage service. As a final test of the application, therefore, log into your Google Drive account in a browser window, navigate to the *storagedemo.txt* file and click on it to view the content which should, all being well, contain the text saved by the application.

61.10 Summary

This chapter has worked through the creation of an example Android Studio application in the form of a very rudimentary text editor designed to use cloud based storage to create, save and open files using the Android Storage Access Framework.