# 58. Understanding Android Content Providers

The previous chapter worked through the creation of an example application designed to store data using a SQLite database. When implemented in this way, the data is private to the application and, as such, inaccessible to other applications running on the same device. While this may be the desired behavior for many types of application, situations will inevitably arise whereby the data stored on behalf of an application could be of benefit to other applications. A prime example of this is the data stored by the built-in Contacts application on an Android device. While the Contacts application is primarily responsible for the management of the user's address book details, this data is also made accessible to any other applications that might need access to this data. This sharing of data between Android applications is achieved through the implementation of *content providers*.

## 58.1 What is a Content Provider?

A content provider provides access to structured data between different Android applications. This data is exposed to applications either as tables of data (in much the same way as a SQLite database) or as a handle to a file. This essentially involves the implementation of a client/server arrangement whereby the application seeking access to the data is the client and the content provider is the server, performing actions and returning results on behalf of the client.

A successful content provider implementation involves a number of different elements, each of which will be covered in detail in the remainder of this chapter.

## 58.2 The Content Provider

A content provider is created as a subclass of the *android.content.ContentProvider* class. Typically, the application responsible for managing the data to be shared will implement a content provider to facilitate the sharing of that data with other applications.

The creation of a content provider involves the implementation of a set of methods to manage the data on behalf of other, client applications. These methods are as follows:

### 58.2.1onCreate()

This method is called when the content provider is first created and should be used to perform any initialization tasks required by the content provider.

### 58.2.2query()

This method will be called when a client requests that data be retrieved from the content provider. It is the responsibility of this method to identify the data to be retrieved (either single or multiple rows), perform the data extraction and return the results wrapped in a Cursor object.

### 58.2.3insert()

This method is called when a new row needs to be inserted into the provider database. This method must identify the destination for the data, perform the insertion and return the full URI of the newly added row.

### 58.2.4update()

The method called when existing rows need to be updated on behalf of the client. The method uses the arguments passed through to update the appropriate table rows and return the number of rows updated as a result of the operation.

### 58.2.5delete()

Called when rows are to be deleted from a table. This method deletes the designated rows and returns a count of the number of rows deleted.

### 58.2.6getType()

Returns the MIME type of the data stored by the content provider.

It is important when implementing these methods in a content provider to keep in mind that, with the exception of the *onCreate()* method, they can be called from many processes simultaneously and must, therefore, be thread safe.

Once a content provider has been implemented, the issue that then arises is how the provider is identified within the Android system. This is where the *content URI* comes into play.

## 58.3 The Content URI

An Android device will potentially contain a number of content providers.

The system must, therefore, provide some way of identifying one provider from another. Similarly, a single content provider may provide access to multiple forms of content (typically in the form of database tables). Client applications, therefore, need a way to specify the underlying data for which access is required. This is achieved through the use of content URIs.

The content URI is essentially used to identify specific data within a specific content provider. The *Authority* section of the URI identifies the content provider and usually takes the form of the package name of the content provider. For example:

```
com.example.mydbapp.myprovider
```

A specific database table within the provider data structure may be referenced by appending the table name to the authority. For example, the following URI references a table named *products* within the content provider:

```
com.example.mydbapp.myprovider/products
```

Similarly, a specific row within the specified table may be referenced by appending the row ID to the URI. The following URI, for example, references the row in the products table in which the value stored in the _ID column equals 3:

```
com.example.mydbapp.myprovider/products/3
```

When implementing the insert, query, update and delete methods in the content provider, it will be the responsibility of these methods to identify whether the incoming URI is targeting a specific row in a table, or references multiple rows, and act accordingly. This can potentially be a complex task given that a URI can extend to multiple levels. This process can, however, be eased significantly by making use of the *UriMatcher* class as will be outlined in the next chapter.

## 58.4 The Content Resolver

Access to a content provider is achieved via a ContentResolver object. An application can obtain a reference to its content resolver by making a call to the *getContentResolver()* method of the application context.

The content resolver object contains a set of methods that mirror those of the content provider (insert, query, delete etc.). The application simply makes calls to the methods, specifying the URI of the content on which the operation is to be performed. The content resolver and content provider objects then communicate to perform the requested task on behalf of the application.

## 58.5 The <provider> Manifest Element

In order for a content provider to be visible within an Android system, it must be declared within the Android manifest file for the application in which it resides. This is achieved using the <*provider*> element, which must contain the following items:

- **android:authority** – The full authority URI of the content provider. For example com.example.mydbapp.mydbapp.myprovider.
- **android:name** – The name of the class that implements the content provider. In most cases, this will use the same value as the authority.

Similarly, the <provider> element may be used to define the permissions that must be held by client applications in order to qualify for access to the underlying data. If no permissions are declared, the default behavior is for permission to be allowed for all applications.

Permissions can be set to cover the entire content provider, or limited to specific tables and records.

## 58.6 Summary

The data belonging to an application is typically private to the application and inaccessible to other applications. In situations where the data needs to be shared, it is necessary to set up a content provider. This chapter has covered the basic elements that combine to enable data sharing between applications, and outlined the concepts of the content provider, content URI and content resolver.

In the next chapter, the Android Studio Database example application created previously will be extended to make the underlying product data available via a content provider.