Using Content Providers

Access to Content Providers is handled by the ContentResolver class.

The following sections demonstrate how to access a Content Resolver and how to use it to query and transact with a Content Provider. They also demonstrate some practical examples using the native Android Content Providers.

Introducing Content Resolvers

Each application Context has a single ContentResolver, accessible using the getContentResolver method, as shown in the following code snippet:

```
ContentResolver cr = getContentResolver();
```

Content Resolver includes several methods to transact and query Content Providers. You specify the provider to interact using a URI.

A Content Provider's URI is defi ned by its *authority* as defi ned in its application manifest node. An authority URI is an arbitrary string, so most providers expose a CONTENT_URI property that includes its authority.

Content Providers usually expose two forms of URI, one for requests against all the data and another that specifi es only a single row. The form for the latter appends /<rowID> to the standard CONTENT_URI.

Querying for Content

As in databases, query results are returned as Cursors over a result set. You can extract values from the cursor using the techniques described previously within the database section on "Extracting Results from a Cursor."

Content Provider queries take a very similar form to database queries. Using the query method on the ContentResolver object, pass in:

❑ The URI of the content provider data you want to query

❑ A projection that represents the columns you want to include in the result set

❑ A where clause that defi nes the rows to be returned. You can include ? wild cards that will be replaced by the values stored in the selection argument parameter.

❑ An array of selection argument strings that will replace the ?'s in the where clause

❑ A string that describes the order of the returned rows

The following skeleton code demonstrates the use of a Content Resolver to apply a query to a Content Provider:

```
// Return all rows
Cursor allRows = getContentResolver().query(MyProvider.CONTENT_URI, null, null, null, null);
// Return all columns for rows where column 3 equals a set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor someRows = getContentResolver().query(MyProvider.CONTENT_URI, null, where, null, order);
```

You'll see some more practical examples of querying for content later in this chapter when the native Android content providers are introduced.

Adding, Updating, and Deleting Content

To perform transactions on Content Providers, use the delete, update, and insert methods on the ContentResolver object.

Inserts

The Content Resolver offers two methods for inserting new records into your Content Provider — insert and bulkInsert. Both methods accept the URI of the item type you're adding; where the former takes a single new ContentValues object, the latter takes an array.

The simple insert method will return a URI to the newly added record, while bulkInsert returns the number of successfully added items.

The following code snippet shows how to use the insert and bulkInsert methods:

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Assign values for each row.
newValues.put(COLUMN_NAME, newValue);
[ ... Repeat for each column ... ]
Uri myRowUri = getContentResolver().insert(MyProvider.CONTENT_URI, newValues);
// Create a new row of values to insert.
ContentValues[] valueArray = new ContentValues[5];
// TODO: Create an array of new rows
int count = getContentResolver().bulkInsert(MyProvider.CONTENT_URI, valueArray);
```

Deletes

To delete a single record using the Content Resolver, call delete, passing in the URI of the row you want to remove. Alternatively, you can specify a where clause to remove multiple rows. Both techniques are shown in the following snippet:

```
// Remove a specific row.
getContentResolver().delete(myRowUri, null, null);
// Remove the first five rows.
String where = "_id < 5";
getContentResolver().delete(MyProvider.CONTENT_URI, where, null);
```

Updates

Updates to a Content Provider are handled using the update method on a Content Resolver. The update method takes the URI of the target Content Provider, a Content Values object that maps column names to updated values, and a where clause that specifies which rows to update.

When executed, every matching row in the where clause will be updated using the values in the Content Values passed in and will return the number of successful updates.

```
// Create a new row of values to insert.
ContentValues newValues = new ContentValues();
// Create a replacement map, specifying which columns you want to
// update, and what values to assign to each of them.
newValues.put(COLUMN_NAME, newValue);
// Apply to the first 5 rows.
String where = "_id < 5";
getContentResolver().update(MyProvider.CONTENT_URI, newValues, where, null);
```

Accessing Files in Content Providers

Content Providers represent fi les as fully qualifi ed URIs rather than raw fi le data. To insert a file into a Content Provider, or access a saved fi le, use the Content Resolvers openOutputStream or openInputStream methods, respectively. The process for storing a fi le is shown in the following code snippet:

```
// Insert a new row into your provider, returning its unique URI.
Uri uri = getContentResolver().insert(MyProvider.CONTENT_URI, newValues);
try {
// Open an output stream using the new row's URI.
OutputStream outStream = getContentResolver().openOutputStream(uri);
// Compress your bitmap and save it into your provider.
sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
}
catch (FileNotFoundException e) { }
```

Native Android Content Providers

Android exposes many Content Providers that supply access to the native databases.

You can use each of these Content Providers natively using the techniques described previously. Alternatively, the android.provider class includes convenience classes that simplify access to many of the most useful providers, including:

❑ **Browser** Use the browser Content Provider to read or modify bookmarks, browser history, or web searches.

❑ **CallLog** View or update the call history including both incoming and outgoing calls together with missed calls and call details, like caller ID and call durations.

❑ **Contacts** Use the Contacts provider to retrieve, modify, or store your contacts' details.

❑ **MediaStore** The Media Store provides centralized, managed access to the multimedia on your device, including audio, video, and images. You can store your own multimedia within the Media Store and make it globally available.

❑ **Settings** You can access the device's preferences using the Settings provider. Using it, you can view and modify Bluetooth settings, ring tones, and other device preferences.

You should use these native Content Providers wherever possible to ensure that your application integrates seamlessly with other native and third-party applications.

While a detailed description of how to use each of these helpers is beyond the scope of this chapter, the following sections describe how to use some of the more useful and powerful native Content Providers.

Using the Media Store Provider

The Android Media Store provides a managed repository for audio, video, and image fi les. Whenever you add a new multimedia fi le to the Android fi lesystem, it should be added to the Media Store to expose it to other applications.

The MediaStore class includes a number of convenience methods to simplify inserting files into the Media Store. For example, the following code snippet shows how to insert an image directly into the Media Store:

```
android.provider.MediaStore.Images.Media.insertImage(
getContentResolver(),
sourceBitmap,
“my_cat_pic”,
“Photo of my cat!”);
```

Using the Contacts Provider

Access to the Contact Manager is particularly powerful on a communications device. Android does the right thing by exposing all the information available from the contacts database to any application granted the READ_CONTACTS permission.

In the following example, an Activity gets a Cursor to every person in the contact database, creating an array of Strings that holds each contact's name and phone number.

To simplify extracting the data from the Cursor, Android supplies public static properties on the People class that expose the column names.

```
// Get a cursor over every contact.
Cursor cursor = getContentResolver().query(People.CONTENT_URI, null, null, null, null);
// Let the activity manage the cursor lifecycle.
startManagingCursor(cursor);
// Use the convenience properties to get the index of the columns
int nameIdx = cursor.getColumnIndexOrThrow(People.NAME);

int phoneIdx = cursor. getColumnIndexOrThrow(People.NUMBER);
String[] result = new String[cursor.getCount()];
if (cursor.moveToFirst())
do {
// Extract the name.
String name = cursor.getString(nameIdx);
// Extract the phone number.
String phone = cursor.getString(phoneIdx);
result[cursor.getPosition()] = name + “ (“ + phone + “)”;
} while(cursor.moveToNext());
```

To run this code snippet, you need to add the READ_CONTACTS *permission to your application.*
As well as querying the contacts database, you can use this Content Provider to modify, delete, or insert contact records.