Android Content Providers

Using the database approach is the recommended way to save structured and complex data. Sharing data between applications is a challenge because the database is accessible to only the package that created it.

A *content provider* allows you to share data across packages. Think of a content provider as a data service. How it stores its data is not relevant to the application using it. What is important is how packages can access the data stored in it using a consistent programming interface.

A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete content. The data can be stored in a database, in files, or over a network.

http://developer.android.com/guide/topics/providers/content-providers.html

Common Provided Android Content Providers

Android ships with many content providers, including:

- Browser — Stores data such as browser bookmarks, browser history, etc.
- CallLog — Stores data such as missed calls, call details, etc.
- Contacts — Stores contact details
- MediaStore — Stores media files such as audio, video, and images
- Settings — Stores the device's settings and preferences.

Besides the many built-in content providers, you can also create your own content providers.

ContentResolver and ContentProvider

Use the **ContentResolver** object in your application's **Context** to communicate with the **ContentProvider** as a client.

The **ContentResolver** object communicates with the provider object, an instance of a class that implements **ContentProvider**.

The provider object receives data requests from clients, performs the requested action, and returns the results.

Note: You don't need to develop your own provider if you don't intend to share your data with other applications. However, you do need your own provider to provide custom search suggestions in your own application. You also need your own provider if you want to copy and paste complex data or files from your application to other applications.

Example

Get a list of the words and their locales from the User Dictionary Provider, you call **ContentResolver.query().**

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

Content Provider *versus* SQL

query() argument	SELECT keyword/parameter	Notes
`Uri`	`FROM table_name`	`Uri` maps to the table in the provider named *table_name*.
`projection`	`col,col,col,...`	`projection` is an array of columns that should be included for each row retrieved.
`selection`	`WHERE col = value`	`selection` specifies the criteria for selecting rows.
`selectionArgs`	(No exact equivalent. Selection arguments replace `?` placeholders in the selection clause.)	
`sortOrder`	`ORDER BY col,col,...`	`sortOrder` specifies the order in which rows appear in the returned `Cursor`.

Content URIs

A **content URI** is a URI that identifies data in a provider.

Content URIs include the symbolic name of the provider and a name that points to a table (a **path**).

Example:

//Uri allContacts = Uri.parse("content://contacts/people");

Uri allContacts = ContactsContract.Contacts.CONTENT_URI;

Example:

content://user_dictionary/words

Requesting read access permission

To retrieve data from a provider, your application needs to specify that you need this permission in your manifest, using the **<uses-permission>** element and the exact permission name defined by the provider.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.provider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="12" android:targetSdkVersion="16"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".ProviderActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Note: When you specify this element in your manifest, you are in effect "requesting" this permission for your application. When users install your application, they implicitly grant this request.

To find the exact name of the read access permission for the provider you're using, as well as the names for other access permissions used by the provider, look in the provider's documentation.

Constructing the query

http://developer.android.com/reference/android/content/ContentResolver.html

public final Cursor **query** (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

Added in API level 1

Query the given URI, returning a `Cursor` over the result set.

For best performance, the caller should follow these guidelines:

- Provide an explicit projection, to prevent reading data from storage that aren't going to be used.
- Use question mark parameter markers such as 'phone=?' instead of explicit values in the `selection` parameter, so that queries that differ only by those values will be recognized as the same for caching purposes.

Parameters

uri	The URI, using the content:// scheme, for the content to retrieve.
projection	A list of which columns to return. Passing null will return all columns, which is inefficient.
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given URI.
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in the order that they appear in the selection. The values will be bound as Strings.
sortOrder	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.

Returns

A Cursor object, which is positioned before the first entry, or null

See Also

`Cursor`

Preparing query parameters:

```
        //Uri allContacts = Uri.parse("content://contacts/people");
        Uri allContacts = ContactsContract.Contacts.CONTENT_URI;

        // A "projection" defines the columns that will be returned for each row
//        String[] mProjection =
//                {ContactsContract.Contacts._ID,
//                 ContactsContract.Contacts.DISPLAY_NAME,
//                 ContactsContract.Contacts.HAS_PHONE_NUMBER};

        String[] mProjection = new String[] {
                ContactsContract.Contacts.DISPLAY_NAME,
                ContactsContract.Contacts._ID};

        // Defines a string to contain the selection clause
        String mSelectionClause = null;

        // Initializes an array to contain selection arguments
        String[] mSelectionArgs = {""};

        // Gets a word from the UI
        //mSearchString = mSearchWord.getText().toString();
        //String mSearchString = "Jay";
        String mSearchString = "";

        // If the word is the empty string, gets everything
        if (TextUtils.isEmpty(mSearchString)) {
            // Setting the selection clause to null will return all words
            mSelectionClause = null;
            mSelectionArgs = null;

        } else {
            // Constructs a selection clause that matches the word that the user entered.
            mSelectionClause = ContactsContract.Contacts.DISPLAY_NAME + " = ?";

            // Moves the user's input string to the selection arguments.
            mSelectionArgs[0] = mSearchString;
        }

        String mSortOrder = null;
```

Creating and executing the query:

```
        Cursor mCursor;

        // query against the table and returns a Cursor object
        mCursor = getContentResolver().query(
            ContactsContract.Contacts.CONTENT_URI,  // The content URI of the words table
            mProjection,                        // The columns to return for each row
            mSelectionClause,                   // Either null, or the word the user entered
            null,                               // Either empty, or the string the user entered
            mSortOrder);                        // The sort order for the returned rows

    // Some providers return null if an error occurs, others throw an exception
        if (null == mCursor) {
            android.util.Log.e(null, null);

        // If the Cursor is empty, the provider found no matches
        } else if (mCursor.getCount() < 1) {
            CharSequence msg = "No contacts";
            Toast toast = Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT);
            toast.show();
        } else { // create cursor, iterate through results

            int[] views = new int[] {R.id.contactName, R.id.contactID};

            SimpleCursorAdapter adapter;

            adapter = new SimpleCursorAdapter(
                    this, R.layout.main, mCursor, mProjection, views,
                    CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);

            this.setListAdapter(adapter);

            PrintContacts(mCursor);
        }
    }
```

http://developer.android.com/reference/android/widget/SimpleCursorAdapter.html

public **SimpleCursorAdapter** (Context context, int layout, Cursor c, String[] from, int[] to, int flags) Added in API level 1

Standard constructor.

Parameters

context	The context where the ListView associated with this SimpleListItemFactory is running
layout	resource identifier of a layout file that defines the views for this list item. The layout file should include at least those named views defined in "to"
c	The database cursor. Can be null if the cursor is not available yet.
from	A list of column names representing the data to bind to the UI. Can be null if the cursor is not available yet.
to	The views that should display column in the "from" parameter. These should all be TextViews. The first N views in this list are given the values of the first N columns in the from parameter. Can be null if the cursor is not available yet.
flags	Flags used to determine the behavior of the adapter, as per `CursorAdapter(Context, Cursor, int)`.

Processing the results:

```
    private void PrintContacts(Cursor c)
    {
        if (c.moveToFirst()) {
            do{
            String contactID = c.getString(c.getColumnIndex(
                   ContactsContract.Contacts._ID));
            String contactDisplayName =
                  c.getString(c.getColumnIndex(
                      ContactsContract.Contacts.DISPLAY_NAME));
                Log.v("Content Providers", contactID + ", " +
                   contactDisplayName);
            } while (c.moveToNext());
        }
    }

}
```

Creating a Content Provider

A content provider manages access to a central repository of data.

A content provider is implemented by extending the **ContentProvider abstract class**, which provides an interface between your provider and other applications.

Although content providers are intended to make data available to other applications, you may of course use the content provider within your application as a data access object, allowing other activities in your application to query and modify the data managed by your provider.

http://developer.android.com/guide/topics/providers/content-provider-creating.html

Designing Content URIs

The content URI identifies data in a provider.

Content URIs include the symbolic name of the entire provider (its **authority**) and a name that points to a table or file (a **path**).

The optional **id** part points to an individual row in a table. Every data access method of **ContentProvider** has a content URI as an argument; this allows you to determine the table, row, or file to access.

Designing an authority

A provider usually has a single authority, which serves as its Android-internal name.

To avoid conflicts with other providers, you should use Internet domain ownership (in reverse) as the basis of your provider authority.

For example, **com.contentprovider**

Designing a path structure

Create content URIs from the authority by appending paths that point to individual tables.

Example,
URIs **com.example.<appname>.provider/table1**, and **com.example.<appname>.provider/table2**.

Simple non-hierarchical example, **com.contentprovider.Books**

Handling content URI IDs

By convention, providers offer access to a single row in a table by accepting a content URI with an ID value for the row at the end of the URI.

Providers match the ID value to the table's **_ID** column, and perform the requested access against the row that matches.

The app does a query against the provider and displays the resulting **Cursor** in a **ListView** using a **CursorAdapter**. The definition of **CursorAdapter** requires one of the columns in the **Cursor** to be **_ID**.

The user then picks one of the displayed rows from the UI in order to look at or modify the data. The app gets the corresponding row from the **Cursor** backing the **ListView**, gets the **_ID** value for this row, appends it to the content URI, and sends the access request to the provider. The provider can then do the query or modification against the exact row the user picked.

Content URI patterns

To help you choose which action to take for an incoming content URI, the provider API includes the convenience class **UriMatcher**, which maps content URI "patterns" to integer values.

You can use the integer values in a **switch** statement that chooses the desired action for the content URI or URIs that match a particular pattern.

A content URI pattern matches content URIs using wildcard characters:

- ***:** Matches a string of any valid characters of any length.
- **#:** Matches a string of numeric characters of any length.

Example,

content://com.example.app.provider/*

Matches any content URI in the provider.

Implementing the ContentProvider Class

Provider schema:

_id	title	isbn

Add your URI to the manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.contentprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="12" android:targetSdkVersion="15"/>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".ContentProvidersActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name="BooksProvider"
            android:authorities="com.contentprovider.Books">
        </provider>
    </application>

</manifest>
```

Extend **ContentProvider**:

```
public class BooksProvider extends ContentProvider {

    static final String PROVIDER_NAME ="com.contentprovider.Books";

    static final Uri CONTENT_URI =
        Uri.parse("content://"+ PROVIDER_NAME + "/books");

    static final String _ID = "_id";
    static final String TITLE = "title";
    static final String ISBN = "isbn";

    static final int BOOKS = 1;
    static final int BOOK_ID = 2;

    private static final UriMatcher uriMatcher;
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "books", BOOKS);
        uriMatcher.addURI(PROVIDER_NAME, "books/#", BOOK_ID);
    }
```

Implement required abstract class methods

The abstract class **ContentProvider** defines six abstract methods that must implement ed as part of your own concrete subclass.

All of these methods except **onCreate()** are called by a client application that is attempting to access your content provider:

query()

Retrieve data from your provider. Use the arguments to select the table to query, the rows and columns to return, and the sort order of the result. Return the data as a **Cursor** object.

insert()

Insert a new row into your provider. Use the arguments to select the destination table and to get the column values to use. Return a content URI for the newly-inserted row.

update()

Update existing rows in your provider. Use the arguments to select the table and rows to update and to get the updated column values. Return the number of rows updated.

delete()

Delete rows from your provider. Use the arguments to select the table and the rows to delete. Return the number of rows deleted.

getType()

Return the MIME type corresponding to a content URI.

onCreate()

Initialize your provider. The Android system calls this method immediately after it creates your provider.

*Notes: All of these methods except **onCreate()** can be called by multiple threads at once, so they must be thread-safe.*

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    //---add a new book---
    long rowID = booksDB.insert(
            DATABASE_TABLE,
            "",
            values);

    //---if added successfully---
    if (rowID>0)
    {
        Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
        getContext().getContentResolver().notifyChange(_uri, null);
        return _uri;
    }
    throw new SQLException("Failed to insert row into " + uri);
}
```

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
    sqlBuilder.setTables(DATABASE_TABLE);

    if (uriMatcher.match(uri) == BOOK_ID)
        //---if getting a particular book---
        sqlBuilder.appendWhere(
                _ID + " = " + uri.getPathSegments().get(1));

    if (sortOrder==null || sortOrder=="")
        sortOrder = TITLE;

    Cursor c = sqlBuilder.query(
        booksDB,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        sortOrder);

    //---register to watch a content URI for changes---
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
```

```
@Override
public int delete(Uri arg0, String arg1, String[] arg2) {
    // arg0 = uri
    // arg1 = selection
    // arg2 = selectionArgs
    int count=0;
    switch (uriMatcher.match(arg0)){
    case BOOKS:
        count = booksDB.delete(
                DATABASE_TABLE,
                arg1,
                arg2);
        break;
    case BOOK_ID:
        String id = arg0.getPathSegments().get(1);
        count = booksDB.delete(
                DATABASE_TABLE,
                _ID + " = " + id +
                (!TextUtils.isEmpty(arg1) ? " AND (" +
                        arg1 + ')' : ""),
                        arg2);
        break;
    default: throw new IllegalArgumentException("Unknown URI " + arg0);
    }
    getContext().getContentResolver().notifyChange(arg0, null);
    return count;
}
```

Calling Activity:

```
public class ContentProvidersActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onClickAddTitle(View view) {

        ContentValues values = new ContentValues();
        values.put("title", ((EditText)
                findViewById(R.id.txtTitle)).getText().toString());
        values.put("isbn", ((EditText)
                findViewById(R.id.txtISBN)).getText().toString());
        Uri uri = getContentResolver().insert(
                Uri.parse("content://com.contentprovider.Books/books"), values);

        Toast.makeText(getBaseContext(),uri.toString(),
                Toast.LENGTH_LONG).show();
    }

    public void onClickRetrieveTitles(View view) {
        //---retrieve the titles---
        Uri allTitles = Uri.parse("content://com.contentprovider.Books/books");

        Cursor c;
        CursorLoader cursorLoader = new CursorLoader(
                this,
                allTitles, null, null, null,
                "title desc");
        c = cursorLoader.loadInBackground();

        if (c.moveToFirst()) {
            do{
                Toast.makeText(this,
                        c.getString(c.getColumnIndex(
                                BooksProvider._ID)) + ", " +
                                c.getString(c.getColumnIndex(
                                        BooksProvider.TITLE)) + ", " +
                                        c.getString(c.getColumnIndex(
                                                BooksProvider.ISBN)),
                                                Toast.LENGTH_SHORT).show();
            } while (c.moveToNext());
        }
    }
```