Content Providers

Content Providers

- Content providers store and retrieve data and make it accessible to all applications.
- They're the only way to share data across applications; there's no common storage area that all Android packages can access.
- If you want to make your own data public, you have two options: You can create your own content provider (a ContentProvider subclass) or you can add the data to an existing provider.

Content Provider Basics

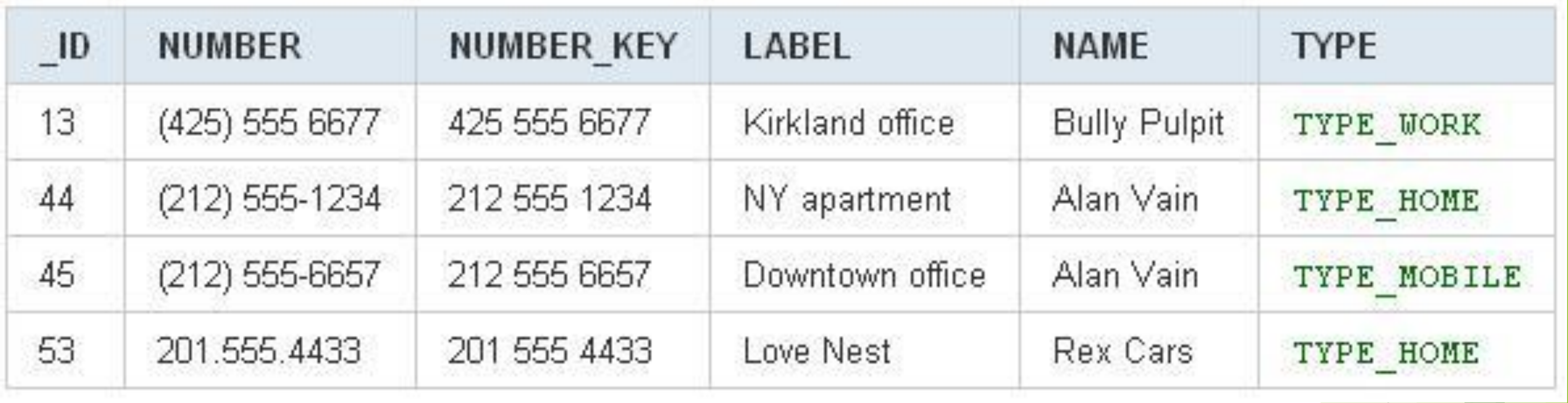
- All content providers implement a common interface for querying the provider and returning results — as well as for adding, altering, and deleting data.
- It's an interface that clients use indirectly, most generally through ContentResolver objects.
  - ContentResolver cr = getContentResolver();
- You can then use the ContentResolver's methods to interact with whatever content providers you're interested in.
- When a query is initiated, the Android system identifies the content provider that's the target of the query and makes sure that it is up and running.
- The system instantiates all ContentProvider objects. There is a single instance of each ContentProvider object.

- Here are some of Android's most useful built-in content providers

Content Provider	*Intended Data*
Contacts	Contact details
Browser	Browser bookmarks, browser history, etc.
CallLog	Missed calls, call details, etc.
MediaStore	Media files such as audio, video and images
Settings	Device settings and preferences

The Data Model

- Content providers expose their data as a simple table on a database model, where each row is a record and each column is data of a particular type and meaning.



_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

- A query returns a Cursor object .

URIs

- Each content provider exposes a public URI (wrapped as a Uri object) that uniquely identifies its data set.
- A content provider that controls multiple data sets (multiple tables) exposes a separate URI for each one.
- All URIs for providers begin with the string "content://".
  - android.provider.Contacts.Phones.CONTENT_URI
  - android.provider.Contacts.Photos.CONTENT_URI
- The URI constant is used in all interactions with the content provider.
- Every ContentResolver method takes the URI as its first argument.
- It's what identifies which provider the ContentResolver should talk to and which table of the provider is being targeted.

Querying a Content Provider

- You need three pieces of information to query a content provider:
  - I. The URI that identifies the provider
  - II. The names of the data fields you want to receive
  - III. The data types for those fields
- To query a content provider, you can use either the ContentResolver.query() method or the Activity.managedQuery() method.
- Both methods take the same set of arguments, and both return a Cursor object.

- managedQuery() causes the activity to manage the life cycle of the Cursor.
- A managed Cursor handles all of the niceties, such as unloading itself when the activity pauses, and requerying itself when the activity restarts.
- The first argument to either query() or managedQuery() is the provider URI — the CONTENT_URI constant that identifies a particular ContentProvider and data set.
- To restrict a query to just one record, you can append the _ID value for that record to the URI
  - content://. . . ./23

- There are some helper methods, particularly ContentUris.withAppendedId() and Uri.withAppendedPath(), that make it easy to append an ID to a URI.

```
// Use the ContentUris method to produce the base URI for the contact with _ID
//== 23.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

- The other arguments to the query() and managedQuery() methods delimit the query in more detail. They are:
  - The names of the data columns that should be returned. A null value returns all columns.
  - an SQL WHERE clause (excluding the WHERE itself).
  - Selection arguments.
  - A sorting order for the rows that are returned.

```
import android.provider.Contacts.People;
import android.database.Cursor;
// Form an array specifying which columns to return.
    String[] projection = new String[] {
                    People._ID,
                    People._COUNT,
                    People.NAME,
                    People.NUMBER
                  };

    // Get the base URI for the People table in the Contacts content provider.
    Uri contacts =  People.CONTENT_URI;
```

Example

```
// Make the query.
Cursor managedCursor = managedQuery(contacts,
                        projection, // Which columns to return
                        null,       // Which rows to return (all rows)
                        null,       // Selection arguments (none)
                        // Put the results in ascending order by name
                        People.NAME + " ASC");
```

The constants for the names of the columns are defined in various interfaces — _ID and _COUNT in BaseColumns, NAME in PeopleColumns, and NUMBER in PhoneColumns. The Contacts.People class implements each of these interfaces

Modifying Data

- Data kept by a content provider can be modified by:
  - Adding new records
  - Adding new values to existing records
  - Batch updating existing records
  - Deleting records
- All data modification is accomplished using ContentResolver methods.

Adding records

- To add a new record to a content provider, first set up a map of key-value pairs in a ContentValues object.
- Then call ContentResolver.insert() and pass it the URI of the provider and the ContentValues map.
- This method returns the full URI of the new record.

Adding Records - Example

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

    ContentValues values = new ContentValues();

    // Add Abraham Lincoln to contacts and make him a favorite.
    values.put(People.NAME, "Abraham Lincoln");
    // 1 = the new contact is added to favorites
    // 0 = the new contact is not added to favorites
    values.put(People.STARRED, 1);

    Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

Adding new values

```
Uri phoneUri = null;
    Uri emailUri = null;

    phoneUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);

    values.clear();
    values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
    values.put(People.Phones.NUMBER, "1233214567");
    getContentResolver().insert(phoneUri, values);

    // Now add an email address in the same way.
    emailUri = Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);

    values.clear();
    values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
    values.put(People.ContactMethods.DATA, "test@example.com");
    values.put(People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
    getContentResolver().insert(emailUri, values);
```

Creating Content Provider

- To create a content provider, you must:
  - Set up a system for storing the data.
    Most content providers store their data using Android's file storage methods or SQLite databases.
  - Extend the ContentProvider class to provide access to the data.
  - Declare the content provider in the manifest file for your application (AndroidManifest.xml).

Extending the ContentProvider class

- You define a ContentProvider subclass to expose your data to others using the conventions expected by ContentResolver and Cursor objects.
- Implement six abstract methods declared in the ContentProvider class:
  - query() - must return a Cursor object that can iterate over the requested data.
  - insert()
  - update()
  - delete()
  - getType()
  - onCreate()

- Because these ContentProvider methods can be called from various ContentResolver objects in different processes and threads, they must be implemented in a thread-safe manner.

Creating Content Provider : Additionals

- Define a public static final Uri named CONTENT_URI.

```
public static final Uri CONTENT_URI =
            Uri.parse("content://com.example.codelab.transporationprovider");
```

- If the provider has subtables, also define CONTENT_URI constants for each of the subtables.

```
content://com.example.codelab.transporationprovider/train
content://com.example.codelab.transporationprovider/air/domestic
content://com.example.codelab.transporationprovider/air/international
```

Creating Content Provider : Additionals

- Define the column names that the content provider will return to clients.
- Be sure to include an integer column named "_id" (with the constant _ID) for the IDs of the records.
  - If you're using the SQLite database, the _ID field should be the following type:

  INTEGER PRIMARY KEY AUTOINCREMENT

Declaring the content provider

- Declare Content Provider with a <provider> element in the application's AndroidManifest.xml file.
- Content providers that are not declared in the manifest are not visible to the Android system.
  - The name attribute is the fully qualified name of the ContentProvider subclass.
  - The authorities attribute is the authority part of the content: URI that identifies the provider.

```
<provider name="com.example.autos.AutoInfoProvider"
          authorities="com.example.autos.autoinfoprovider"
          . . . />
</provider>
```

Declaring the content provider

- Other <provider> attributes can set permissions to read and write data, provide for an icon and text that can be displayed to users, enable and disable the provider, and so on.
- Set the multiprocess attribute to "true" if data does not need to be synchronized between multiple running versions of the content provider.