Android Content Providers: Programming with Content Resolvers

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand how to program an App that uses a Content Resolver to retrieve & manipulate names in a phone user’s Contacts Content Provider
Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people
- A raw contact represents a person's data coming from a single account type & account name

developer.android.com/guide/topics/providers/contacts-provider.html
Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people
  - A raw contact represents a person's data coming from a single account type & account name
  - The Contacts Provider allows multiple raw contacts for the same person

developer.android.com/guide/topics/providers/contacts-provider.html
Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people.

- It provides three tables used in a device's "Contacts" App.
  - ContactsContract.Contacts -- Rows represent different people.

Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people.
- It provides three tables used in a device's “Contacts” App:
  - ContactsContract.Contacts – Rows represent different people.
  - ContactsContract.RawContacts – Rows contain a summary of a person's data.
    - e.g., specific to a user account & type.

[Code Snippet]

```
public static final Summary: Nested Classes | Constants | Inherited class
  Constants | Fields | Methods | Inherited Methods | [Expand All]
  Added in API level 5

ContactsContract.RawContacts
extends Object
  implements BaseColumns ContactsContract.ContactNameColumns
  ContactsContract.ContactOptionsColumns
  ContactsContract.RawContactsColumns ContactsContract.SyncColumns

java.lang.Object
  android.provider.ContactsContract.RawContacts

Class Overview

Constants for the raw contacts table, which contains one row of contact information for each person in each synced account. Sync adapters and contact management apps are the primary consumers of this API.
```

Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people.
- It provides three tables used in a device's “Contacts” App:
  - ContactsContract.Contacts – Rows represent different people.
  - ContactsContract.RawContacts – Rows contain a summary of a person's data.
  - ContactsContract.Data – Rows contain details for raw contact.
    - e.g., email addresses or phone numbers.

Overview of the Contacts Content Provider

- Contacts Provider is a powerful & flexible component that manages the device's central repository of data about people.
- It provides three tables used in a device's "Contacts" App.
- Android defines a Content URI for retrieving & modifying an Android Contacts Content Provider database.

```java
public final class ContactsContract {
    public static final String AUTHORITY = "com.android.contacts";
    public static final Uri AUTHORITY_URI = Uri.parse("content://" + AUTHORITY);

    public static final Uri CONTENT_URI = Uri.withAppendedPath(AUTHORITY_URI, "contacts");
}
```
Simple Example of Listing Contacts

- This simple App lists the names of all the entries in the Contacts Content Provider.
Simple Example of Listing Contacts

public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        ContentResolver cr = getContentResolver();

        Cursor c = cr.query(ContactsContract.Contacts.CONTENT_URI,
                            new String[] { ContactsContract.Contacts.DISPLAY_NAME },
                            null, null, null);

        ...
        if (c.moveToFirst()) {
            do {
                ...
            } while (c.moveToNext());
        }
    }
    ...

Determine the data to display

Populate list view widget with data to display
Simple Example of Listing Contacts

- The Cursor returned by query() provides an iterator for accessing the retrieved results.

```
public interface Cursor
    implements Closeable
android.database.Cursor

- Known Indirect Subclasses
  AbstractCursor, AbstractWindowedCursor, CrossProcessCursor,
  CrossProcessCursorWrapper, CursorWrapper, MatrixCursor, MergeCursor,
  MockCursor, SQLiteCursor

Class Overview

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass AbstractCursor.
```
The Cursor returned by `query()` provides an iterator for accessing the retrieved results.

Some useful methods:
- `boolean moveToFirst()`  
- `boolean moveToNext()`  
- `int getColumnIndex(String columnName)`  
- `String getString(int columnIndex)`

This interface provides random read-write access to the result set returned by a database query.

Cursor implementations are not required to be synchronized so code using a Cursor from multiple threads should perform its own synchronization when using the Cursor.

Implementations should subclass `AbstractCursor`.

Simple Example of Listing Contacts

public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        Cursor c = ...

        List<String> contacts = new ArrayList<String>();
        if (c.moveToFirst())
            do {
                contacts.add(c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));
            } while (c.moveToNext());

        ...}
    }

    Populate & display list view widget

    Extract a column with contact name from the cursor

    Store column we’re interested in for each contact

Simple Example of Listing Contacts

public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        Cursor c = ...

        List<String> contacts = new ArrayList<String>();
        if (c.moveToFirst())
            do {
                contacts.add(c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));
            } while (c.moveToNext());

        ...

    }
}
Simple Example of Listing Contacts

public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        Cursor c = ...
    }

    List<String> contacts = new ArrayList<String>();
    if (c.moveToFirst())
        do {
            ...
        } while (c.moveToNext());

    ArrayAdapter<String> adapter =
        new ArrayAdapter<String>(this, R.layout.list_item, contacts);
    setListAdapter(adapter);
}

ArrayAdapter can handle any Java object as input & maps data of this input to a TextView in the layout

ArrayAdapter uses toString() method of data input object to determine String to display

[developer.android.com/reference/android/widget/ArrayAdapter.html]
A More Sophisticated Example of Listing Contacts

• This App lists the _id & names of all entries in the Contacts Content Provider
A More Sophisticated Example of Listing Contacts

```java
public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...

        Columns to retrieve

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

        Columns to display

        String colsToDisplay[] = new String[] {
            "_id",
            ContactsContract.Contacts.DISPLAY_NAME
        };

        Layout for columns to display

        int[] colResIds = new int[] { R.id.idString, R.id.name }
        ...
    }
}
```
A More Sophisticated Example of Listing Contacts

```java
public class ContactsListExample extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...,
        
        ContentResolver cr = getContentResolver();

        Query for columns
        Cursor c = cr.query(
            ContactsContract.Contacts.CONTENT_URI, columns,
            ContactsContract.Contacts.STARRED + " = 0", null, null);

        Only select “non-starred” contacts

        setListAdapter(new SimpleCursorAdapter
            (this, R.layout.list_layout,
             c, colsToDisplay, colResIds));
    }
}
```

Map columns from a cursor to TextViews, specifying which columns are wanted & which views to display the columns

Example of Deleting an Entry from Contacts

```java
public class ContactsListDisplayActivity extends ListActivity {
    ...

    private void deleteContact(String name) {
        getContentResolver().delete
           (ContactsContract.RawContacts.CONTENT_URI,
            ContactsContract.Contacts.DISPLAY_NAME + "=?",
            new String[] {name});
    }

    private void deleteAllContacts() {
        getContentResolver().delete
           (ContactsContract.RawContacts.CONTENT_URI, null, null);
    }

    ...
}
```

**Delete a particular contact**

**Delete all contacts (be careful!)**
Example of Inserting an Entry into Contacts

public class ContactsListDisplayActivity extends ListActivity {
    ...
    private void insertContact(String name) {
        ArrayList<ContentProviderOperation> ops =
            new ArrayList<ContentProviderOperation>();
        try {
            getContentResolver().applyBatch
                (ContactsContract.AUTHORITY, ops);
        } finally {
        }
    }

    ... Create new RawContacts (see below)

    ...
}

Apply the batch operation to add the new contact
Example of Inserting an Entry into Contacts

```java
public class ContactsListDisplayActivity extends ListActivity {

    private void insertContact(String name) {

        ops.add(ContentProviderOperation
            .newInsert(RawContacts.CONTENT_URI)
            .withValue(RawContacts.ACCOUNT_TYPE, "com.google")
            .withValue(RawContacts.ACCOUNT_NAME, "douglas.schmidt@gmail.com")
            .build());  /* Create a new RawContact */

        // Add additional contact details...

        ops.add(ContentProviderOperation.newInsert(Data.CONTENT_URI)
            .withValueBackReference(Data.RAW_CONTACT_ID,0)
            .withValue(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE)
            .withValue(StructuredName.DISPLAY_NAME, name)
            .build());

        // Additional operations...
    }

```
Summary

- The Android Contacts Provider accommodates a wide range of data sources & tries to manage as much data as possible for each person
- Not surprisingly, the implementation is large & complex!

Summary

- The Android Contacts Provider accommodates a wide range of data sources & tries to manage as much data as possible for each person.
- The provider's API includes an extensive set of contract classes & interfaces that facilitate both retrieval & modification of contact data.

[developer.android.com/guide/topics/providers/contacts-provider.html](developer.android.com/guide/topics/providers/contacts-provider.html)
Android Content Providers: Designing & Implementing a Content Provider

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

- Understand the steps involved in designing & implementing a Content Provider
Analyze Your Requirements

- Before making the effort to create a ContentProvider make sure you really need one!
Analyze Your Requirements

Before making the effort to create a ContentProvider make sure you really need one!

Some considerations include:
- You want to offer complex data or files to other Apps
  - e.g., a user-customizable spell checker

developer.android.com/guide/topics/providers/content-provider-creating.html
Analyze Your Requirements

• Before making the effort to create a ContentProvider make sure you really need one!

• Some considerations include:
  • You want to offer complex data or files to other Apps
  • You want to allow users to copy complex data from your App into other Apps
    • e.g., contact data

developer.android.com/guide/topics/providers/content-provider-creating.html
Analyze Your Requirements

• Before making the effort to create a ContentProvider make sure you really need one!

• Some considerations include:
  • You want to offer complex data or files to other Apps
  • You want to allow users to copy complex data from your App into other Apps
  • You want to provide custom search suggestions using the search framework
  • Many Android Apps provide this capability

developer.android.com/guide/topics/search/adding-custom-suggestions.html
Creating a ContentProvider

- Steps to creating a ContentProvider
  - Implement a storage system for the data
    - e.g., structure data vs. file vs. remotely accessed data, etc.

[developer.android.com/guide/topics/providers/content-provider-creating.html]
Creating a ContentProvider

Steps to creating a ContentProvider

- Implement a storage system for the data
- Determine the format of the Content URI for accessing the contents of the data managed by the provider

developer.android.com/guide/topics/providers/content-provider-creating.html
Creating a ContentProvider

- Steps to creating a ContentProvider
  - Implement a storage system for the data
  - Determine the format of the Content URI for accessing the contents of the data managed by the provider
  - Implement a provider as one or more classes in an Android App, along with <provider> element in manifest file
  - Subclass ContentProvider to define the interface between the provider & other Apps

developer.android.com/guide/topics/providers/content-provider-creating.html
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row

```plaintext
Content Provider

onCreate()
insert()
query()
update()
delete()
getType()
```
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row
  - `query()` selects table to query, rows & columns to return via Cursor, & sort order of result
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row
  - `query()` selects table to query, rows & columns to return via Cursor, & sort order of result
  - `update()` selects table & rows to update & to get updated column values

![Content Provider Diagram]

- Methods:
  - `onCreate()`
  - `insert()`
  - `query()`
  - `update()`
  - `delete()`
  - `getType()`
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row
  - `query()` selects table to query, rows & columns to return via Cursor, & sort order of result
  - `update()` selects table & rows to update & to get updated column values
  - `delete()` selects table & rows to delete

```
Content Provider
onCreate()
insert()
query()
update()
delete()
getType()
```
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row
  - `query()` selects table to query, rows & columns to return via Cursor, & sort order of result
  - `update()` selects table & rows to update & to get updated column values
  - `delete()` selects table & rows to delete
  - `getType()` returns MIME type corresponding to a content URI
Required ContentProvider Methods

- Abstract methods that subclasses must implement
  - `onCreate()` initializes a provider (called immediately after creating a provider)
  - `insert()` selects table & column values to use to insert a new row
  - `query()` selects table to query, rows & columns to return via Cursor, & sort order of result
  - `update()` selects table & rows to update & to get updated column values
  - `delete()` selects table & rows to delete
  - `getType()` returns MIME type corresponding to a content URI

Methods have same signature as identically named ContentResolver methods
Define the Content Provider Data Model

- A content provider typically presents data to external Apps as one or more tables.

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>freq</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr_FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>

The user dictionary is provider in Android is that stores spellings of non-standard words a user wants to keep.
Define the Content Provider Data Model

- A content provider typically presents data to external Apps as one or more tables.
- A row represents an instance of some type of data the provider manages.

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>freq</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr_FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>

The user dictionary is provided in Android, storing spellings of non-standard words a user wants to keep.
Define the Content Provider Data Model

- A content provider typically presents data to external Apps as one or more tables
- A row represents an instance of some type of data the provider manages
- Each column in a row represents an individual piece of data collected for an instance

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>freq</th>
<th>locale</th>
<th>_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapreduce</td>
<td>user1</td>
<td>100</td>
<td>en_US</td>
<td>1</td>
</tr>
<tr>
<td>precompiler</td>
<td>user14</td>
<td>200</td>
<td>fr_FR</td>
<td>2</td>
</tr>
<tr>
<td>applet</td>
<td>user2</td>
<td>225</td>
<td>fr_CA</td>
<td>3</td>
</tr>
<tr>
<td>const</td>
<td>user1</td>
<td>255</td>
<td>pt_BR</td>
<td>4</td>
</tr>
<tr>
<td>int</td>
<td>user5</td>
<td>100</td>
<td>en_UK</td>
<td>5</td>
</tr>
</tbody>
</table>

The user dictionary is provider in Android is that stores spellings of non-standard words a user wants to keep.
Define the Content URI(s)

- A Content URI identifies data in a provider.
- Each ContentProvider method uses a content URI to determine which table, row, and/or file to access.

Diagram:

- Contacts Application
  - URI1
  - URI2
  - URI3

- ContactsProvider
  - insert()
  - query()
  - update()
  - delete()

- SQLite Database
  - Data Table 1 (Accounts)
  - Data Table 2 (Contacts)
  - Data Table 3 (Calls)
Define the Content URI(s)

- A Content URI identifies data in a provider
- Content URIs have several parts
  - The symbolic name of the entire provider (its authority)

```
content://com.android.contacts/contacts/directory=0/photo_id
```
Define the Content URI(s)

- A Content URI identifies data in a provider
- Content URIs have several parts
  - The symbolic name of the entire provider (its authority)
  - A name that points to a table or file (a path)

```java
content://com.android.contacts/contacts/directory=0/photo_id
```

![Diagram showing Android Content Providers and their interactions with a SQLite Database with Data Tables 1, 2, and 3 representing Accounts, Contacts, and Calls respectively.](image)/
Define the Content URI (s)

- A Content URI identifies data in a provider
- Content URIs have several parts
  - The symbolic name of the entire provider (its authority)
  - A name that points to a table or file (a path)
  - An optional id part that points to an individual row in a table

content://com.android.contacts/contacts/directory=0/photo_id
Define the Content URI(s)

- A Content URI identifies data in a provider
- Content URIs have several parts
- Define unique data members that represent each Content URI part
  - The AUTHORITY_URI for the provider

```java
public final class ContactsContract {
    public static final String AUTHORITY =
        "com.android.contacts";
    public static final Uri AUTHORITY_URI =
        Uri.parse("content://" + AUTHORITY);
    ...
}
```
Define the Content URI(s)

- A Content URI identifies data in a provider
- Content URIs have several parts
- Define unique data members that represent each Content URI part
  - The AUTHORITY_URI for the provider
  - The CONTENT_URI for each table
  - If provider has subtables, define CONTENT_URI constants for each one

```
class ContactsContract {
    public static final String AUTHORITY =
        "com.android.contacts";
    public static final Uri AUTHORITY_URI =
        Uri.parse("content://" + AUTHORITY);
    ...  
    public static class Contacts implements ... {
        ...  
        public static final Uri CONTENT_URI =
            Uri.withAppendedPath(AUTHORITY_URI,
                "contacts");
    }
```

Frameworks/base/core/java/android/provider/ContactsContract.java
Define the Content URI(s)

- A Content URI identifies data in a provider
- Content URIs have several parts
- Define unique data members that represent each Content URI part
- The UriMatcher class maps content URI “patterns” to integer values using wildcard characters:
  - * Matches a string of any valid characters of any length
  - # Matches a string of numeric characters of any length

```java
private static final UriMatcher um;
    um.addURI(
        "com.example.app.provider",
        "table3", 1);
    um.addURI(
        "com.example.app.provider",
        "table3/#", 2);
...
public Cursor query(Uri uri, ...){
    ...
    switch (um.match(uri)) {
        // If URI’s for all of table3
        case 1: ...; break;

        // If URI’s for a single row
        case 2: ...; break;
    }
}
```

---

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

#ContentURI
Define a Contract Class

- A contract class is a public final class containing constant definitions for the URIs, column names, MIME types, & other meta-data that pertain to a Content Provider.

[developer.android.com/guide/topics/providers/content-provider-creating.html#ContractClass](developer.android.com/guide/topics/providers/content-provider-creating.html#ContractClass)
Define a Contract Class

- A contract class is a public final class containing constant definitions for the URIs, column names, MIME types, & other meta-data that pertain to a Content Provider.
- The class establishes a contract between the provider & other Apps by ensuring that the provider can be correctly accessed even if there are changes to the actual values of URIs, column names, and so forth.

developer.android.com/guide/topics/providers/content-provider-creating.html#ContractClass
Define the Column Names

- Define column names
  - Typically identical to the SQL database column names

```java
public static final String _ID = "_id", DATA = "data";

private static final String[] columns = new String[]
{ _ID, DATA };
```
Define the Column Names

• Define column names

• Also define public static String constants that clients can use to specify the columns

• Consider implementing a “Contracts class” to document the data type of each column so clients can read the data

```java
public static final String _ID = "_id", DATA = "data";

private static final String[] columns = new String[]{
    _ID, DATA
};
```
Define the Column Names

- Define column names
- Also define public static String constants that clients can use to specify the columns
- Be sure to include an integer column named "_id" (with the constant _ID) for the IDs of the records
- If you use an SQLite database, the _ID field should be of type INTEGER PRIMARY KEY AUTOINCREMENT

```java
public static final String _ID = "_id", DATA = "data";
private static final String[] columns = new String[]
    { _ID, DATA };
```
Define the Column Names

- Define column names
- Also define public static String constants that clients can use to specify the columns
- Be sure to include an integer column named "_id" (with the constant _ID) for the IDs of the records
- Define the MIME types for items & directories

```java
public static final String _ID = "_id", DATA = "data";

private static final String[] columns = new String[]{
    _ID, DATA
};

private static final String contentTypeSingle =
    "vnd.android.cursor.item/MyCP.data.text";

private static final String contentTypeMultiple =
    "vnd.android.cursor.dir/MyCP.data.text";
```
Define the MIME Types for Tables

- `ContentProvider.getType()` returns a String in MIME format
- This string describes the type of data returned by the content URI argument

```
public abstract String getType (Uri uri)
```

Implement this to handle requests for the MIME type of the data at the given URI. The returned MIME type should start with `vnd.android.cursor.item` for a single record, or `vnd.android.cursor.dir/` for multiple items. This method can be called from multiple threads, as described in Processes and Threads.

Note that there are no permissions needed for an application to access this information; if your content provider requires read and/or write permissions, or is not exported, all applications can still call this method regardless of their access permissions. This allows them to retrieve the MIME type for a URI when dispatching intents.

**Parameters**
- `uri` the URI to query.

**Returns**
- a MIME type string, or `null` if there is no type.
Define the MIME Types for Tables

- `ContentProvider.getType()` returns a string in MIME format.
- For content URIs that point to row(s) of table data, `getType()` should return a MIME type in Android's vendor-specific MIME format.

[developer.android.com/guide/topics/providers/content-provider-creating.html#MIMETypes](developer.android.com/guide/topics/providers/content-provider-creating.html#MIMETypes)
Define the MIME Types for Tables

- `ContentProvider.getType()` returns a String in MIME format
- For content URIs that point to row(s) of table data, `getType()` should return a MIME type in Android's vendor-specific MIME format
  - e.g., if the content provider authority is `com.example.app.provider` & it exposes `table1`, the MIME types will be as follows:
    - multiple rows in `table1`:
      `vnd.android.cursor.dir/vnd.com.example.provider.table1`
Define the MIME Types for Tables

- ContentProvider.getType() returns a String in MIME format
- For content URIs that point to row(s) of table data, getType() should return a MIME type in Android's vendor-specific MIME format
- e.g., if the content provider authority is com.example.app.provider & it exposes table1, the MIME types will be as follows:
  - multiple rows in table1:
    vnd.android.cursor.dir/vnd.com.example.provider.table1
  - a single row of table1:
    vnd.android.cursor.item/vnd.com.example.provider.table1

developer.android.com/guide/topics/providers/content-provider-creating.html#MIMETypes
Define the MIME Types for Files

- If a provider offers files, implement `getStreamTypes()`
- Returns a String array of MIME types for files your provider can return for a given content URI

```java
public String[] getStreamTypes(Uri uri, String mimeTypeFilter)
```

Called by a client to determine the types of data streams that this content provider supports for the given URI. The default implementation returns `null`, meaning no types. If your content provider stores data of a particular type, return that MIME type if it matches the given `mimeTypeFilter`. If it can perform type conversions, return an array of all supported MIME types that match `mimeTypeFilter`.

**Parameters**

- `uri` The data in the content provider being queried.
- `mimeTypeFilter` The type of data the client desires. May be a pattern, such as `*/*` to retrieve all possible data types.

**Returns**

- Returns `null` if there are no possible data streams for the given `mimeTypeFilter`. Otherwise returns an array of all available concrete MIME types.

**See Also**

- `getType(Uri)`
- `openTypedAssetFile(Uri, String, Bundle)`
- `compareMimeTypes(String, String)`

[developer.android.com/reference/android/content/ContentProvider.html #getType(android.net.Uri)]
Define the MIME Types for Files

- If a provider offers files, implement `getStreamTypes()`
  - Returns a String array of MIME types for files your provider can return for a given content URI
  - Filter MIME types offered by MIME type filter argument, so return only MIME types that a client wants

```java
public String[] getStreamTypes(Uri uri, String mimeTypeFilter)
```

Called by a client to determine the types of data streams that this content provider supports for the given URI. The default implementation returns `null`, meaning no types. If your content provider stores data of a particular type, return that MIME type if it matches the given `mimeTypeFilter`. If it can perform type conversions, return an array of all supported MIME types that match `mimeTypeFilter`.

**Parameters**
- `uri` The data in the content provider being queried.
- `mimeTypeFilter` The type of data the client desires. May be a pattern, such as `*/*` to retrieve all possible data types.

**Returns**
Returns `null` if there are no possible data streams for the given `mimeTypeFilter`. Otherwise returns an array of all available concrete MIME types.

**See Also**
- `getType(Uri)`
- `openTypedListAssetFile(Uri, String, Bundle)`
- `compareMimeTypes(String, String)`

[developer.android.com/reference/android/content/ContentProvider.html](developer.android.com/reference/android/content/ContentProvider.html) #getType(android.net.Uri)
If a provider offers files, implement `getStreamTypes()`

- e.g., consider a provider that offers photo images as files in .jpg, .png, & .gif format
- If `getStreamTypes()` is called by an App with the filter string "image/*" then return array `{ "image/jpeg", "image/png", "image/gif" }`

```
public String[] getStreamTypes (Uri uri, String mimeTypeFilter)
```

Called by a client to determine the types of data streams that this content provider supports for the given URI. The default implementation returns `null`, meaning no types. If your content provider stores data of a particular type, return that MIME type if it matches the given `mimeTypeFilter`. If it can perform type conversions, return an array of all supported MIME types that match `mimeTypeFilter`.

- **Parameters**
  - `uri` The data in the content provider being queried.
  - `mimeTypeFilter` The type of data the client desires. May be a pattern, such as `*\/*` to retrieve all possible data types.

- **Returns**
  - `null` if there are no possible data streams for the given `mimeTypeFilter`. Otherwise returns an array of all available concrete MIME types.

- **See Also**
  - `getType(Uri)`
  - `openTypedAssetFile(Uri, String, Bundle)`
  - `compareMimeTypes(String, String)`

`developer.android.com/reference/android/content/ContentProvider.html #getType(android.net.Uri)`
If a provider offers files, implement `getStreamTypes()`

- e.g., consider a provider that offers photo images as files in `.jpg`, `.png`, & `.gif` format

- If `getStreamTypes()` is called by an App with the filter string `"image/*"` then return array `{ "image/jpeg", "image/png", "image/gif" }`

- If `getStreamTypes()` is called by an App with filter string `"*/jpeg"` then just return `{"image/jpeg"}`

See Also
- `getType(Uri)`
- `openTypedAssetFile(Uri, String, Bundle)`
- `compareMimeTypes(String, String)`

developer.android.com/reference/android/content/ContentProvider.html
#`getType(android.net.Uri)`
Declaring the Provider in AndroidManifest.xml

• Declare ContentProvider with `<provider>` in the file AndroidManifest.xml

```
<provider
    android:name="ContactsProvider2"
    android:authorities="contacts;com.android.contacts"
...
```

developer.android.com/guide/topics/manifest/provider-element.html
Declaring the Provider in AndroidManifest.xml

• Declare ContentProvider with `<provider>` in the file AndroidManifest.xml

• The Authorities attribute omits the *path* part of a content:// URI

• e.g., Tables defined by ContactsProvider2 are not defined in the manifest

content://com.android.contacts/

contacts/
Declaring the Provider in AndroidManifest.xml

• Declare ContentProvider with `<provider>` in the file AndroidManifest.xml

• The Authorities attribute omits the `path` part of a content:// URI

• The authority is what identifies a Content Provider, not the path

• A Content Provider implementation can interpret the path part of the URI in any way it chooses

```xml
<provider
    android:name="ContactsProvider2"
    android:authorities="contacts;com.android.contacts"
...`
```
Handling Large Data in Content Providers

- Although the data model for Content Provider has a blob type, it’s only appropriate for “small” binary objects
  - e.g., a small icon or short audio clip

stackoverflow.com/questions/5406429/cursor-size-limit-in-android-sqlitedatabase
Handling Large Data in Content Providers

- Although the data model for Content Provider has a blob type, it’s only appropriate for “small” binary objects, rather than “large” binary objects
  - e.g., large photographs or complete songs/videos, etc.

stackoverflow.com/questions/5406429/cursor-size-limit-in-android-sqlitedatabase
Handling Large Data in Content Providers

- Although the data model for Content Provider has a blob type, it’s only appropriate for “small” binary objects, rather than “large” binary objects.

- If binary data is small then enter it directly into the SQLite table & read from cursor via Cursor.getBlob()

- getBlob() returns a byte array
Handling Large Data in Content Providers

- Although the data model for Content Provider has a blob type, it’s only appropriate for “small” binary objects, rather than “large” binary objects.
- If binary data is small then enter it directly into the SQLite table & read from cursor via Cursor.getBlob().
- For large binary data put content:// URIs in a table.
  - The content:// URI specifies a file that should not be read directly by the client via the Cursor object.
  - Instead, call ContentResolver openInputStream() to get an InputStream object to read the data.

For more information, visit:
[developer.android.com/reference/android/content/ContentResolver.html](developer.android.com/reference/android/content/ContentResolver.html)
Summary

• A ContentProvider instance manages access to a structured set of data by handling requests from other applications.
Summary

- A ContentProvider instance manages access to a structured set of data by handling requests from other applications.
- All forms of access eventually call ContentResolver, which then calls a concrete method of ContentProvider to get access.
Android Content Providers:
Programming a Content Provider

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand how to program a synchronous Content Provider
Synchronous Content Provider Example

- Show how to implement a simple ContentProvider
Synchronous Content Provider Example

- Show how to implement a simple ContentProvider
- Stores the DataRecord objects in a HashMap
- “Real” solutions typically use an SQLite database
Synchronous Content Provider Example

- Show how to implement a simple ContentProvider
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider “CRUD” operations
  - All of which are implemented as synchronized Java methods
Synchronous Content Provider Example

- Show how to implement a simple ContentProvider
- Stores the DataRecord objects in a HashMap
- Supports all the ContentProvider “CRUD” operations
- Client Activity accesses the ContentProvider using synchronous two-way calls made via a ContentResolver
Synchronous Content Provider

public class MyCP extends ContentProvider {
    public static final Uri CONTENT_URI =
        Uri.parse("content://course.examples.ContentProviders.myCP/");

    public static final String _ID = "_id", DATA = "data";

    private static final String[] columns =
        new String[]{_ID, DATA};

    private static final Map<Integer, DataRecord> db =
        new HashMap<Integer, DataRecord>();

    ...

    Define the Content URI

    Define column names

    Define the HashMap that associates numbers with records
Synchronous Content Provider

```java
public class MyCP extends ContentProvider {
    ...

    private static final String contentTypeSingle =
            "vnd.android.cursor.item/MyCP.data.text";

    private static final String contentTypeMultiple =
            "vnd.android.cursor.dir/MyCP.data.text";

    ...

    // Define MIME type info for individual items & groups of items
```
Synchronous Content Provider

```java
class MyCP extends ContentProvider {
    ...
    public synchronized Uri insert(Uri uri, ContentValues values) {
        if (values.containsKey(Data)) {
            DataRecord tmp =
                new DataRecord(values.getAsString(Data));
            db.put(tmp.get_id(), tmp);
            return Uri.parse(CONTENT_URI +
                String.valueOf(tmp.get_id()));
        }
        return null;
    }
    ...
}
```
Insert a new record & return the new URI
public class MyCP extends ContentProvider {

    ...

    public synchronized int delete(Uri uri, String selection, String[] selectionArgs) {
        String requestIdString = uri.getLastPathSegment();
        if (null == requestIdString) {
            for (DataRecord dr : db.values())
                db.remove(dr.get_id());
        }
        else {
            Integer requestId = Integer.parseInt(requestIdString);
            if (db.containsKey(requestId))
                db.remove(requestId);
        }
        return // # of records deleted;
    }

    ...
}
Synchronous Content Provider

public class MyCP extends ContentProvider {
    
    public synchronized Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String order) {
        String requestIdString = uri.getLastPathSegment();

        MatrixCursor cursor = new MatrixCursor(columns);

        if (null == requestIdString)
            for (DataRecord dr : db.values())
                cursor.addRow(new Object[]
                    { dr.get_id(),
                    dr.get_data();
                });

    }

    //...
public class MyCP extends ContentProvider {
    ...

    public synchronized Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String order) {
        ...
        else {
            Integer requestId = Integer.parseInt(requestIdString);

            if (db.containsKey(requestId)) {
                DataRecord dr = db.get(requestId);
                cursor.addRow(new Object[] {
                    dr.get_id(),
                    dr.get_data();
                });
            }
        }
        return cursor;
    }
}
public class ContactProviderActivity extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ContentResolver cr = getContentResolver();
        ContentValues values = new ContentValues();

        values.put("data", "Record1");
        cr.insert(MyCP.CONTENT_URI, values);

        values.clear(); values.put("data", "Record2");
        cr.insert(MyCP.CONTENT_URI, values);

        values.clear(); values.put("data", "Record3");
        cr.insert(MyCP.CONTENT_URI, values);

        ...
Synchronous Contact Provider Activity

```java
public class ContactProviderActivity extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        cr.delete(Uri.parse(MyCP.CONTENT_URI + "/1"),
                  (String) null, (String[]) null);
        values.clear(); values.put("data", "Record4");
        cr.update(Uri.parse(MyCP.CONTENT_URI + "/2"), values, (String) null, (String[]) null);
        ...
    }
}
```

Delete & update various records into the Content Provider
Synchronous Contact Provider Activity

```java
public class ContactProviderActivity extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ... 

        Cursor c = cr.query
            (MyCP.CONTENT_URI, null, null, null, null);

        setListAdapter(new SimpleCursorAdapter(this,
            R.layout.list_layout, c, new String[] { "_id", "data" },
            new int[] { R.id.idString, R.id.data }));

        // Query the Content Provider & display the results
    }
```
AndroidManifest.xml File

<manifest ...
    package="course.examples.ContentProviders.myCP" ...
"
    <application... >
        <activity android:name=".CustomContactProviderDemo" ...>
            ...
        </activity>
        <provider android:name=".MyCP"
            android:authorities="course.examples.contentproviders.mycp"
            android:process=":remote">
            ...
        </provider>
    </application>
</manifest>
Motivating ContentProviderClient

- A ContentResolver provides a mapping from String contentAuthority to ContentProvider

[Diagram showing ContentProvider with methods: insert(), query(), update(), delete().]

[Arrow diagram showing interactions between Apps and ContentProvider through ContentResolver.]

stackoverflow.com/questions/5084896/using-contentproviderclient-vs-contentresolver-to-access-content-provider
Motivating ContentProviderClient

- A ContentResolver provides a mapping from String contentAuthority to ContentProvider
- This mapping is expensive
  - e.g., when you call ContentResolver query(), update(), etc., the URI is parsed apart into its components, the contentAuthority string is identified, & contentResolver must search that map for a matching string, & direct the query to the right provider

[Diagram: Persistent Data Store, Content Provider, Apps with ContentResolvers, insert(), query(), update(), delete()]

[Link: stackoverflow.com/questions/5084896/using-contentproviderclient-vs-contentresolver-to-access-content-provider]
Motivating ContentProviderClient

- A ContentResolver provides a mapping from String contentAuthority to ContentProvider.
- This mapping is expensive.
- Moreover, this expensive search occurs during every method call since the URI might differ from call to call, with a different contentAuthority.

stackoverflow.com/questions/5084896/using-contentproviderclient-vs-contentresolver-to-access-content-provider
Querying via ContentProviderClient

- The `acquireContentProviderClient()` factory method on `ContentResolver` returns a `ContentProviderClient`.
- This object is associated with the `ContentProvider` that services the content at the designated URI.

developer.android.com/reference/android/content/ContentProviderClient.html
Querying via ContentProviderClient

- The acquireContentProviderClient() factory method on ContentResolver returns a ContentProviderClient
  - This object is associated with the ContentProvider that services the content at the designated URI
  - The identified Content Provider is started if necessary

```
public class ContentProviderClient extends Object

java.lang.Object
  L android.content.ContentProviderClient

Class Overview

The public interface object used to interact with a ContentProvider. This is obtained by calling acquireContentProviderClient(Uri). This object must be released using release() in order to indicate to the system that the ContentProvider is no longer needed and can be killed to free up resources.

Note that you should generally create a new ContentProviderClient instance for each thread that will be performing operations. Unlike ContentResolver, the methods here such as query(Uri, String[], String, String[], String) and openFile(Uri, String) are not thread safe – you must not call release() on the ContentProviderClient those calls are made from until you are finished with the data they have returned.
```

developer.android.com/reference/android/content/ContentProviderClient.html
Querying via ContentProviderClient

- The `acquireContentProviderClient()` factory method on `ContentResolver` returns a `ContentProviderClient`.
- `ContentProviderClient` is a direct link to the `ContentProvider`.
- Needn’t constantly re-compute "which provider do I want?"

[developer.android.com/reference/android/content/ContentProviderClient.html](developer.android.com/reference/android/content/ContentProviderClient.html)
Querying via ContentProviderClient

- The `acquireContentProviderClient()` factory method on `ContentResolver` returns a `ContentProviderClient`.
- `ContentProviderClient` is a direct link to the `ContentProvider`.
- The `ContentProviderClient` has essentially the same interface as `ContentProvider`.
- Don’t forget to call `release()` when you’re done.

---

Public class `ContentProviderClient` extends `Object`.

Java class `android.content.ContentProviderClient`.

Class Overview

The public interface object used to interact with a `ContentProvider`. This is obtained by calling `acquireContentProviderClient(Uri)`. This object must be released using `release()` in order to indicate to the system that the `ContentProvider` is no longer needed and can be killed to free up resources.

Note that you should generally create a new `ContentProviderClient` instance for each thread that will be performing operations. Unlike `ContentResolver`, the methods here such as `query(Uri, String[], String, String[], String)` and `openFile(Uri, String)` are not thread safe – you must not call `release()` on the `ContentProviderClient` those calls are made from until you are finished with the data they have returned.

[developer.android.com/reference/android/content/ContentProviderClient.html](developer.android.com/reference/android/content/ContentProviderClient.html)
Querying via ContentProviderClient

• The acquireContentProviderClient() factory method on ContentResolver returns a ContentProviderClient.
• ContentProviderClient is a direct link to the ContentProvider.
• The ContentProviderClient has essentially the same interface as ContentProvider.
• Don’t forget to call release() when you’re done.
• Also, the methods aren’t thread-safe.

developer.android.com/reference/android/content/ContentProviderClient.html
ContactProviderClientActivity Example

public class ContactProviderClientActivity
    extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ContentResolver cr = getContentResolver();
        ContentResolver cr = getContentResolver();
    }

    ContentProviderClient cpc =
            cr.acquireContentProviderClient(MyCP.CONTENT_URI);

    Factory method that obtains a ContentProviderClient
ContactProviderClientActivity Example

```java
public class ContactProviderClientActivity
    extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ... 
        try {
            ContentValues values = new ContentValues();
            values.put("data", "Record1");
            cpc.insert(MyCP.CONTENT_URI, values);

            values.clear(); values.put("data", "Record2");
            cpc.insert(MyCP.CONTENT_URI, values);

            values.clear(); values.put("data", "Record3");
            cpc.insert(MyCP.CONTENT_URI, values);
        ... 
```

Essentially the same code as before, just using a ContentProviderClient
public class ContactProviderClientActivity  
    extends ListActivity {  
    public void onCreate(Bundle savedInstanceState) {  
        ...  
        cpc.delete(Uri.parse(MyCP.CONTENT_URI + 
                   "/1"), (String) null, (String[]) null);  
        values.clear(); values.put("data", "Record4");  
        cpc.update(Uri.parse(MyCP.CONTENT_URI + 
                   "/2"), values, (String) null, (String[]) null);  
        ...  

        Essentially the same code as before, just using a ContentProviderClient
ContactProviderClientActivity Example

```
public class ContactProviderClientActivity extends ListActivity {
    public void onCreate(Bundle savedInstanceState) {
        ...

        Cursor c = cpc.query(MyCP.CONTENT_URI, null,
                             null, null, null);

        setListAdapter(new SimpleCursorAdapter(this,
                                                R.layout.list_layout, c,
                                                new String[] {"_id", "data"},
                                                new int[] { R.id.idString, R.id.data }));

        } catch (RemoteException e) { /* ... */ }
    finally { cpc.release(); }
}
```

Essentially the same code as before, just using a ContentProviderClient
Summary

• Implementing synchronous Content Providers is relatively straightforward

• However, there are issues associated with two-way blocking method calls on the main UI thread
Summary

• Implementing synchronous Content Providers is straightforward

• ContentProviderClients optimize performance
  • However, they need to be released & are not thread-safe