CS371m - Mobile Computing

Content Providers And
Content Resolvers
Content Providers

• One of the four primary application components:
  – activities
  – content providers / content resolvers
  – services
  – broadcast receivers
Android Applications

• Recall...

• Each application runs as a different user on the OS

• private files, user id, separate process with its own instance of the Dalvik VM

• Content Providers and Content Resolvers act as a bridge between applications to share data
Content Providers

• Standard mechanism / interface to allow code in one process (app, content resolver) to access data from another process (app, content provider)
  – example, app to remind you to call certain people
  – content resolver accesses call log to check last contact

• manage access to a structured data

• encapsulate the data and provide mechanisms for defining data security
Content Provider - Content Resolver

Calendar App Data

Call Log Data

Your App
Content Providers

• Many of the built in applications on devices have **content providers** to allow other apps to access data

• Examples of built in content providers
  – AlarmClock
  – CalendarContract (API level 14)
  – CallLog (sent and received calls)
  – ContactsContract
  – MediaStore (audio / visual data)
  – UserDictionary
  – VoicemailContract

Content Providers

• Provide access to a central data repository
  – ability to read and write to centralized data
• data presented by Content Provider in the form of a table
  – like table from relational database
• Each row in data table one "piece" of data in repository
Example Table

• Data from user dictionary

Table 1: Sample user dictionary table.

<table>
<thead>
<tr>
<th>word</th>
<th>app id</th>
<th>frequency</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>

• primary key optional
• _ID column required to bind data from provider to a ListView
ContentProvider Theory

• Abstraction facilitated by having URI for data from content provider
• content://<more to follow> is the URI for the content
• Don't know where data (content is actually stored)
  – sqlite database, flat file, server accessible via network
• content://contacts/people
USING CONTENT PROVIDERS AND CONTENT RESOLVERS
Accessing Data

• Use a ContentResolver client object in your app
• ContentResolver communicates with ContentProvider
  – provides "CRUD" functionality, Create, Retrieve, Update, Delete
• matching methods in Content Resolver / Content Provider
• example: query() method
• Create a cursor via content resolver to move through rows of table
Using Content Providers

• Unlike Activities, Services, and Broadcast Receivers we won't declare ContentResolvers in our AndroidManifest.xml file

• In practice you may not even realize you are using a ContentProvider

• we call the getContentResolver() method inherited from Context and then the query method on the returned ContentResolver
Accessing Content via Provider

• Example: Exploring Images on a device
• MediaStore.Images.Media class presents various Content Providers

• get the cursor:

```java
cursor = getContentResolver().query(
    /* The content URI of the image table*/
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    /* String[] projection, The columns to return for each row
    * if null, get them all*/
    null,
    /* String selection criteria, return rows that match this
    * if null return all rows */
    null,
    /* String[] selectionArgs. ?s from selection
    * ?s replaced by this parameter.*/
    null,
    /* String sortOrder, how to sort row, null unsorted */
    null);
```
Query

- 5 parameters
- uri for content, URI
  - look at class documentation, generally a constant
- projection, String[]
  - what columns to get from the table, null = all, *can be inefficient*
- selection clause, String
  - filter, what rows to include, a SQL WHERE clause
- selection args, String[]
  - replace '?'s in selection with these
- sortOrder, String
  - how to order the rows
Accessing Content via Provider

• After obtaining cursor:

```java
Log.d(TAG, "Image count: " + cursor.getCount());
Log.d(TAG, "Columns: " + cursor.getColumnCount());
String[] columns = cursor.getColumnNames();
Log.d(TAG, "Columns: " + Arrays.toString(columns));
```

• result:

<table>
<thead>
<tr>
<th>ImageContent</th>
<th>Image count: 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageContent</td>
<td>Columns: 20</td>
</tr>
<tr>
<td>ImageContent</td>
<td>Columns: [ _id, _data, _size, _dir, ]</td>
</tr>
</tbody>
</table>
MediaStore.Images.Media

• Columns from table:
• According to Logcat:
• [__id, _data, _size, _display_name, mime_type, title, date_added, date_modified, description, picasa_id, isprivate, latitude, longitude, datetaken, orientation, mini_thumb_magic, bucket_id, bucket_display_name, width, height]
MediaStore.Images.Media

- Columns documented in ContentProvider classes and interfaces

<table>
<thead>
<tr>
<th>String</th>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>DATA</td>
<td>The data stream for the file Type: DATA STREAM</td>
</tr>
<tr>
<td>String</td>
<td>DATE_ADDED</td>
<td>The time the file was added to the media provider Units are seconds since 1970.</td>
</tr>
<tr>
<td>String</td>
<td>DATE_MODIFIED</td>
<td>The time the file was last modified Units are seconds since 1970.</td>
</tr>
<tr>
<td>String</td>
<td>DISPLAY_NAME</td>
<td>The display name of the file Type: TEXT</td>
</tr>
<tr>
<td>String</td>
<td>MIME_TYPE</td>
<td>The MIME type of the file Type: TEXT</td>
</tr>
<tr>
<td>String</td>
<td>SIZE</td>
<td>The size of the file in bytes Type: INTEGER (long)</td>
</tr>
<tr>
<td>String</td>
<td>TITLE</td>
<td>The title of the content Type: TEXT</td>
</tr>
</tbody>
</table>
## MediaStore.Images.Media Columns

### Inherited Constants

<table>
<thead>
<tr>
<th>String</th>
<th>_COUNT</th>
<th>The count of rows in a directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>_ID</td>
<td>The unique ID for a row.</td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th>String</th>
<th>BUCKET_DISPLAY_NAME</th>
<th>The bucket display name of the image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>BUCKET_ID</td>
<td>The bucket id of the image.</td>
</tr>
<tr>
<td>String</td>
<td>DATE_TAKEN</td>
<td>The date &amp; time that the image was taken in units of milliseconds since Jan 1, 1970.</td>
</tr>
<tr>
<td>String</td>
<td>DESCRIPTION</td>
<td>The description of the image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: TEXT</td>
</tr>
<tr>
<td>String</td>
<td>IS_PRIVATE</td>
<td>Whether the video should be published as public or private</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: INTEGER</td>
</tr>
<tr>
<td>String</td>
<td>LATITUDE</td>
<td>The latitude where the image was captured.</td>
</tr>
<tr>
<td>String</td>
<td>LONGITUDE</td>
<td>The longitude where the image was captured.</td>
</tr>
<tr>
<td>String</td>
<td>MINI_THUMB_MAGIC</td>
<td>The mini thumb id.</td>
</tr>
<tr>
<td>String</td>
<td>ORIENTATION</td>
<td>The orientation for the image expressed as degrees.</td>
</tr>
<tr>
<td>String</td>
<td>PICASA_ID</td>
<td>The picasa id of the image</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: TEXT</td>
</tr>
</tbody>
</table>
Selection Columns

• Limit Columns returned with projection argument to query method that creates Cursor

```java
String[] projection = {MediaStore.Images.Media.DATE_TAKEN,
                      MediaStore.Images.Media.SIZE,
                      MediaStore.Images.Images.Media.ORIENTATION};

cursor = getContentResolver().query(
  MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
  projection,
  null,
  null,
  MediaStore.Images.Media.SIZE);
```
Showing Data in Logcat

```
// get column indices
int size
    = cursor.getColumnIndex(MediaStore.Images.Media.SIZE);
int dateTaken
    = cursor.getColumnIndex(MediaStore.Images.Media.DATE_TAKEN);
int orientation
    = cursor.getColumnIndex(MediaStore.Images.Media.ORIENTATION);

SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
cursor.moveToFirst();
while(!cursor.isAfterLast()) {
    Log.d(TAG, "size: " + cursor.getInt(size));
    String sDate = format.format(cursor.getLong(dateTaken));
    Log.d(TAG, "date taken: " + sDate);
    Log.d(TAG, "orientation: " + cursor.getInt(orientation));
cursor.moveToNext();
}
```
Cursor

• The ContentResolver query method creates and returns a **Cursor**
• Similar to a Database Cursor
  – similar to Scanner or Iterator
• Move through the data (rows) one element at a time
• Process data with loop or bind cursor to a ListView with an Adapter
Getting Data from Row

• Must determine what type column data is in, use getX method
• refer to constants from ContentProvider class
• careful - some INTEGERS longs

<table>
<thead>
<tr>
<th>Type</th>
<th>MIME_TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>MIME_TYPE</td>
<td>The MIME type of the file Type: TEXT</td>
</tr>
<tr>
<td>String</td>
<td>SIZE</td>
<td>The size of the file in bytes Type: INTEGER (long)</td>
</tr>
<tr>
<td>String</td>
<td>TITLE</td>
<td>The title of the content Type: TEXT</td>
</tr>
</tbody>
</table>
Using Selection Criteria

• Example gets rows in table
• the selection criteria and selection args allow picking only certain rows
• essentially an SQL WHERE clause
  – http://www.w3schools.com/sql/sql_where.asp
• specify a criteria that must be met
• ? is value filled in with selectionArgs
  – multiple criteria possible, AND, OR, NOT
Using Selection Criteria

• Instead of selecting all rows, only select rows with image size greater than some minimum value
  —recall: null, null returns all rows

```java
String selectionClause = MediaStore.Images.Media.SIZE + " > ?";
String[] selectionArgs = {Integer.toString(MIN_IMAGE_SIZE)};

Cursor imageData = getContentResolver().query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI, columns, selectionClause, selectionArgs, MediaStore.Images.Media.DATE_TAKEN);
```
<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/21/2012 01:17 PM</td>
<td>1897541</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10/30/2012 04:03 AM</td>
<td>1881612</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10/30/2012 04:05 AM</td>
<td>1882039</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10/30/2012 04:06 AM</td>
<td>2026593</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10/30/2012 04:09 AM</td>
<td>1797412</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Why selectionCriteria and selectionArgs??

• Why not just say:
  – selectionCriteria = "MediaStore.Images.Media.SIZE > 1750000"

• SECURITY

• If selection criteria is based on user input, user could insert malicious SQL statements to attempt to delete data base table

• example:
  "MediaStore.Images.Media.SIZE > " + "nothing; DROP TABLE *;"
Displaying Data in ListView

- Specify columns to get from ContentProvider
- Create view that will hold data
  - one row
- Obtain cursor from ContentProvider
- Use ListAdapter to convert data from Cursor to Views
- Sub class adapter to format text
private void populateListView() {
    listView = getListView();

    String[] columns = {MediaStore.Images.Media.DATE_TAKEN,
                        MediaStore.Images.Media.SIZE,
                        MediaStore.Images.Media.ORIENTATION,
                        MediaStore.Images.Media._ID};

    int[] textViewIds = {R.id.date_taken,
                          R.id.size, R.id.orientation};

    Cursor imageData = getContentResolver().query(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        columns,
        null,
        null,
        MediaStore.Images.Media.DATE_TAKEN);
Display Data from ContentProvider

- rest of populateListView from ListActivity

```java
ListAdapter adapter = new MyAdapter(this,
        R.layout.list_item_view,
        imageData, columns, textViewIds);

Log.d(TAG, "count: " + adapter.getCount());

setListAdapter(adapter);
```
private static class MyAdapter extends SimpleCursorAdapter {

    static String format = "MM/dd/yyyy hh:mm a";

    private MyAdapter(Context c, int layout, Cursor cur, String[] from, int[] to) {
        super(c, layout, cur, from, to);
    }

    public void setViewText(TextView v, String text) {
        if (v.getId() == R.id.date_taken) {
            text = getDate(Long.parseLong(text), format);
        }
        v.setText(text);
    }
}
### Results

<table>
<thead>
<tr>
<th>Date/Time</th>
<th>ContentProviderExample</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/28/2012 12:58 AM</td>
<td>1070319 90</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>1506016 90</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>1592671 90</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>1952992 0</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>1398451 90</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>1322717 90</td>
</tr>
<tr>
<td>04/17/2012 03:58 PM</td>
<td>912070 90</td>
</tr>
<tr>
<td>04/17/2012 04:00 PM</td>
<td>1522044 90</td>
</tr>
</tbody>
</table>
Permissions

- Using certain content providers require an application request permission
  - so user aware what content the application will access and possibly modify

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READCALENDAR</td>
<td>Allows an application to read the user's calendar data</td>
</tr>
<tr>
<td>READCALLLOG</td>
<td>Allows an application to read the user's call log.</td>
</tr>
<tr>
<td>READCONTACTS</td>
<td>Allows an application to read the user's contacts data</td>
</tr>
<tr>
<td>WRITECALENDAR</td>
<td>Allows an application to write (but not read) the user’s calendar data</td>
</tr>
<tr>
<td>WRITECALLLOG</td>
<td>Allows an application to write (but not read) the user’s contacts data</td>
</tr>
<tr>
<td>WRITECONTACTS</td>
<td>Allows an application to write (but not read) the user’s contacts data</td>
</tr>
</tbody>
</table>
Content Provider Capabilities

• Possible to update, insert, and delete data via a ContentProvider
  – CRUD

• insert, update, and delete methods part of ContentResolver class

• for example insert new calendar data or modify existing calendar data
ContentProviders and Intents

• Some Content Providers have common activities
• created Intent Filters to handle the operation
• Example
  – Calendar has Intent Filter so other applications can add events
  – opens form with data filled in, finish creation, go back to original app, event added to calendar
private void tryCalendarIntent() {
    Calendar beginTime = Calendar.getInstance();
    beginTime.set(2012, Calendar.NOVEMBER, 9, 8, 00);
    Calendar endTime = Calendar.getInstance();
    endTime.set(2012, Calendar.NOVEMBER, 9, 19, 00);
    Intent intent = new Intent(Intent.ACTION_INSERT)
        .setData(Events.CONTENT_URI)
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,
                   beginTime.getTimeInMillis())
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME,
                   endTime.getTimeInMillis())
        .putExtra(Events.TITLE, "ALPHA RELEASE")
        .putExtra(Events.DESCRIPTION, "Major assignment " +
                   "is due in CS378!!!")
        .putExtra(Intent.EXTRA_EMAIL, "scottm@cs.utexas.edu");
    startActivity(intent);
}
Single Data Elements

• Sometimes you don't want all the data from a Content Provider
  – you want one piece of data, one row from the table
• must have the *ID value* of the row and the Content Provider must support this functionality
SPECIAL PROVIDERS AND CONTRACTS
Calendar Provider and Contact Provider

• Special providers
• Calendar and Contact data considered central to user experience
• Android has built in components to work with Calendar and Contact data
Contacts Provider

• Built to accommodate a wide range of data and manage as much data as possible for each contact
• flexible, powerful, ... complicated
• provides *contract classes* to access and modify Contacts data
Contracts

• Some apps that have content providers provide *Contract classes*

• "help work with Content provider Uris, column names, intent actions, and other features of the content provider"

• Adds a layer of abstraction and encapsulation
Contacts Contract

ContactsContract

extends Object

java.lang.Object

\android.provider.ContactsContract

Class Overview

The contract between the contacts provider and applications. Contains definitions for the supported URIs and columns. These APIs supersede ContactsContract.Contacts.

- Provide information on the tables in the content provider
- ... and some convenience methods for accessing that data
Contacts Contract

• Example of abstraction
• Possible to create cursor and pull out the last time a contact was contacted
Calendar and Contacts Providers

• The Calendar and Contacts data used by many of the apps on the device
• Each have their own APIs to perform CRUD operations
  – create, read, update, delete
• Calendar provider has tables for
  – Calendars, Events, Instances, Attendees, Reminders
• Contact provider manages as much data for each contact as possible leading to a complex organization
  – multiple contract classes for retrieval and modification of contact data
CREATING CONTENT PROVIDERS
Creating ContentProvider

• It is possible to implement a ContentProvider for your app
• You may need / want to provide a ContentProvider if:
  – You want to offer complex data or files to other applications.
  – 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.
• Not normally necessary until you create that million download app
• These are declared in manifest
LOADERS
Loaders

• Alternative approach to getting Cursor from a ContentProvider

• Accessing data from ContentProvider may be a lengthy operation
  – doing this on the UI thread may lead to ANR, unresponsiveness

• Loader interfaces and classes used to do this on a separate thread
  – create their own AsynchTask
CursorLoader

• create a loader
  – API level 11 or greater
  – ListView or ListFragment, but not ListActivity

• implement a class with the proper callback methods for when the CursorLoader is finished and data is ready for access
  – public Loader<Cursor> onCreateLoader(int id, Bundle args)
  – public void onLoadFinished(Loader<Cursor> loader, Cursor data)
  – public void onLoaderReset(Loader<Cursor> loader)