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

Clciker

- Which are you more likely to interact with directly in your apps?

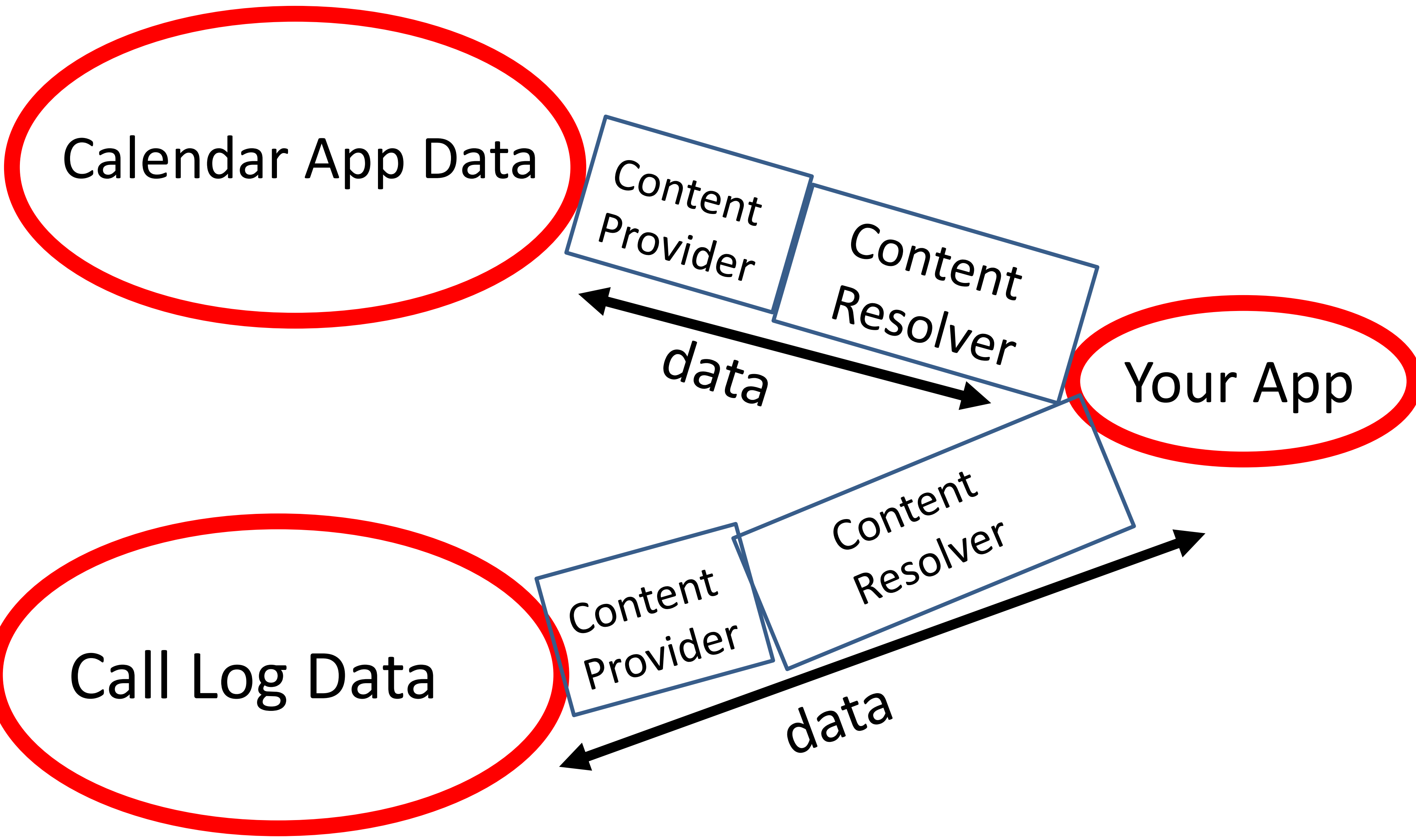
A. A Content Resolver

B. A Content Provider

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



Content Providers

- Many of the built in applications on devices have ***content providers*** to allow other apps to access data
- Examples of apps with content providers
  - AlarmClock
  - CalendarContract (API level 14)
  - CallLog (sent and received calls)
  - ContactsContract
  - MediaStore (audio / visual data)
  - UserDictionary
  - VoicemailContract
  - many, many more
- http://developer.android.com/reference/android/provider/package-summary.html

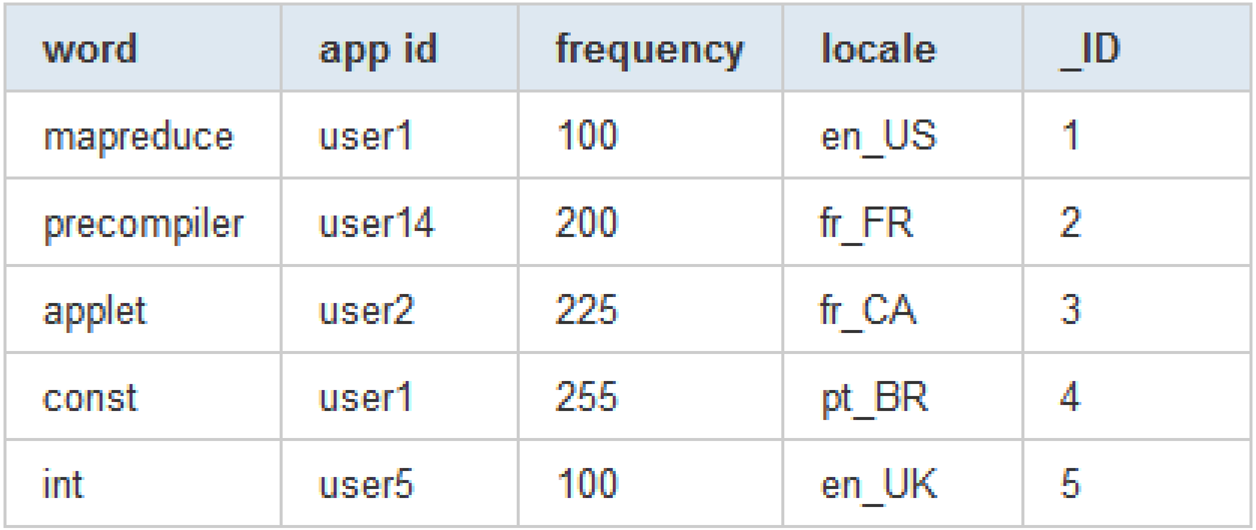
Content Providers

- 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.



word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

- primary key optional
- _ID column required to bind data from provider to a ListView

ConentProvider 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 data base, flat file, server accessible via network
- content://contacts/people

Listing Content Providers

- Simple to list all ContentProviders available on Device / System
  - Varies from device to device

```
private void showContentProviders() {
    int count = 0;
    for (PackageInfo pack
            : getPackageManager()
            .getInstalledPackages(PackageManager.GET_PROVIDERS)) {
        ProviderInfo[] providers = pack.providers;
        if (providers != null) {

            count += providers.length;
            for (ProviderInfo provider : providers) {
                Log.d(TAG, "provider: " + provider.authority);
            }
        }
    }
```

USING CONTENT PROVIDERS AND CONTENT RESOLVERS

Accessing Data

- Use a ContentResolver client object in your app
- ContentResolver communicates with ContentProvider
  - provides "CRUD" functionality, **C**reate, **R**etrieve, **U**pdate, **D**elete
- 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:

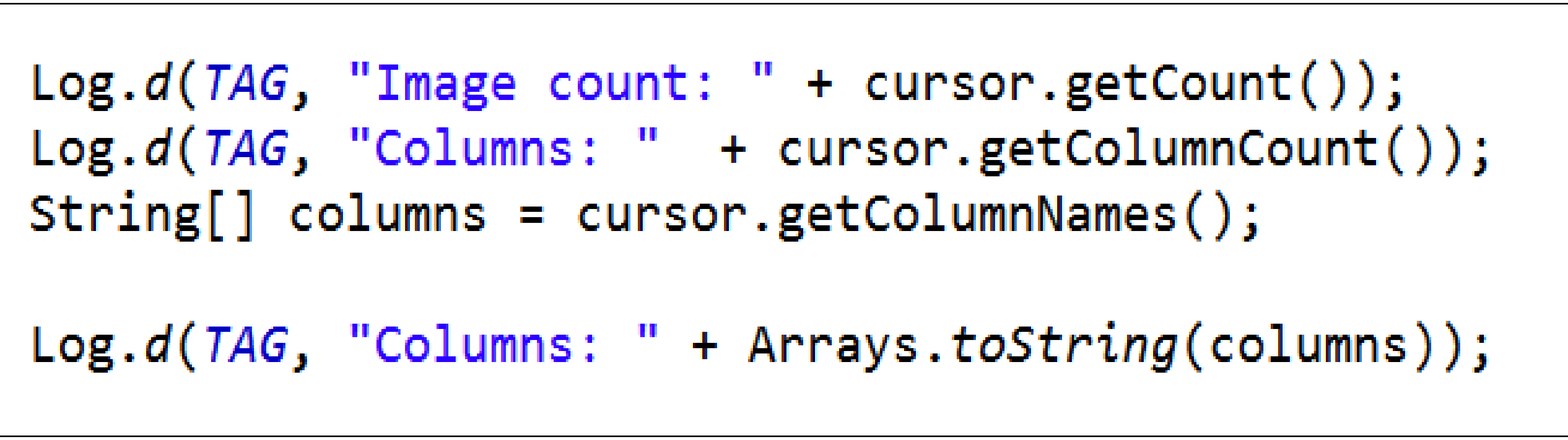
```
Cursor cursor = getContentResolver().query(
        /* The content URI of the image table*/
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        /* String[] projection, The columns to return for each row
         * if null, get them all*/
        null,
        /*  String selection criteria, 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

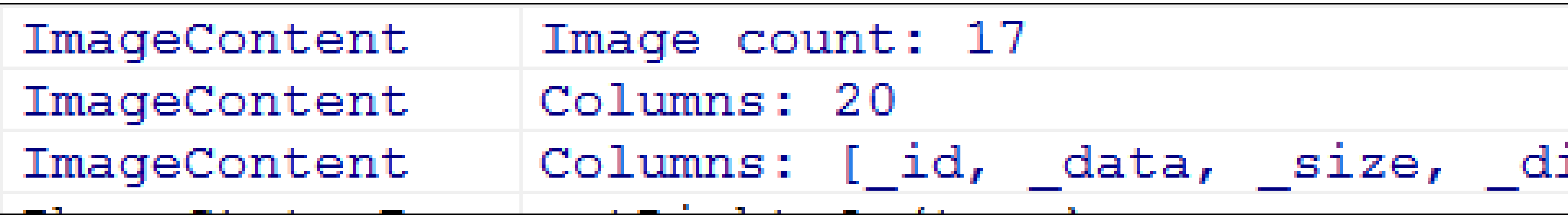
Accessing Content via Provider

- After obtaining cursor:



```
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:



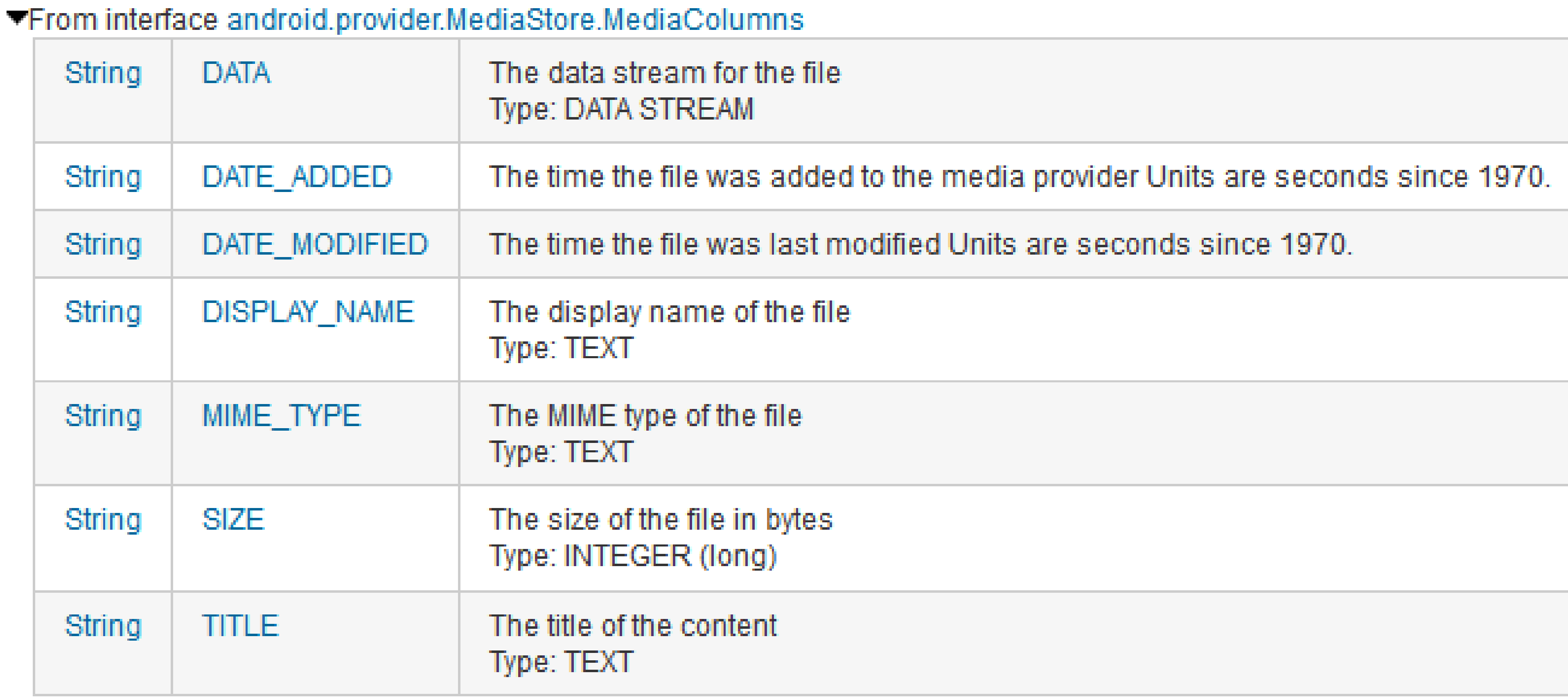
```
ImageContent    Image count: 17
ImageContent    Columns: 20
ImageContent    Columns: [_id, _data, _size, _di
[illegible]
```

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]

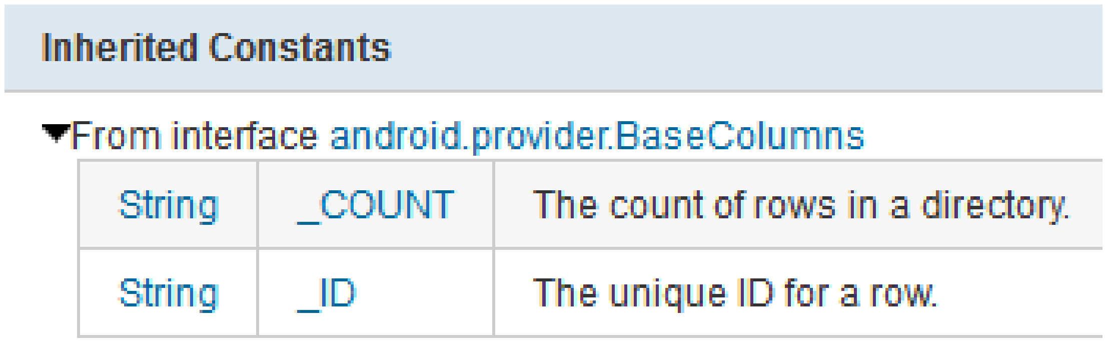
- Columns documented in ContentProvider classes and interfaces



▼From interface android.provider.MediaStore.MediaColumns

String	DATA	The data stream for the file Type: DATA STREAM
String	DATE_ADDED	The time the file was added to the media provider Units are seconds since 1970.
String	DATE_MODIFIED	The time the file was last modified Units are seconds since 1970.
String	DISPLAY_NAME	The display name of the file Type: TEXT
String	MIME_TYPE	The MIME type of the file Type: TEXT
String	SIZE	The size of the file in bytes Type: INTEGER (long)
String	TITLE	The title of the content Type: TEXT

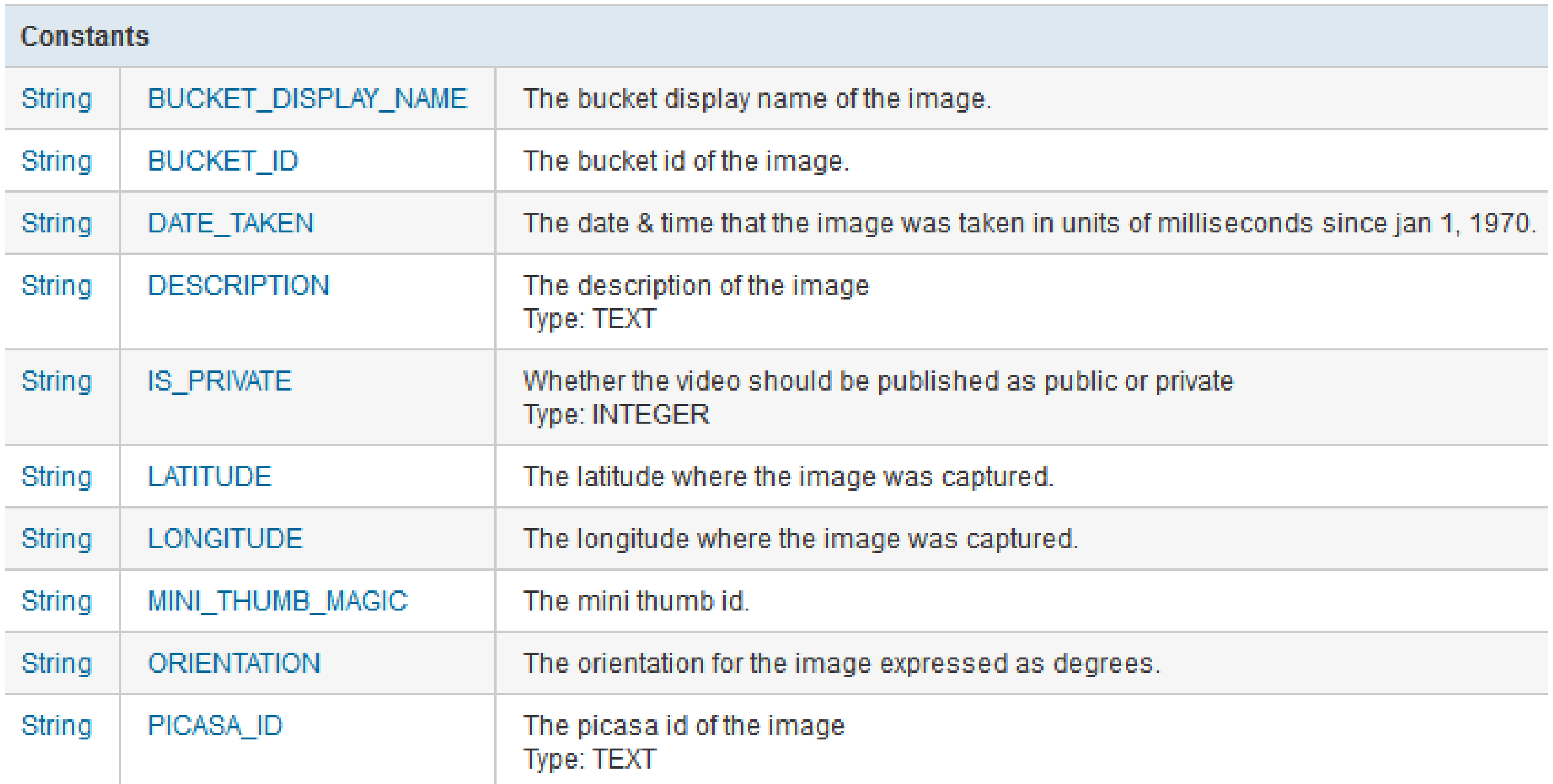
MediaStore.Images.Media Columns



Inherited Constants

▼From interface android.provider.BaseColumns

String	_COUNT	The count of rows in a directory.
String	_ID	The unique ID for a row.



Constants		
String	BUCKET_DISPLAY_NAME	The bucket display name of the image.
String	BUCKET_ID	The bucket id of the image.
String	DATE_TAKEN	The date & time that the image was taken in units of milliseconds since jan 1, 1970.
String	DESCRIPTION	The description of the image Type: TEXT
String	IS_PRIVATE	Whether the video should be published as public or private Type: INTEGER
String	LATITUDE	The latitude where the image was captured.
String	LONGITUDE	The longitude where the image was captured.
String	MINI_THUMB_MAGIC	The mini thumb id.
String	ORIENTATION	The orientation for the image expressed as degrees.
String	PICASA_ID	The picasa id of the image Type: TEXT

Selection Columns

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

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

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

Showing Data in Logcat

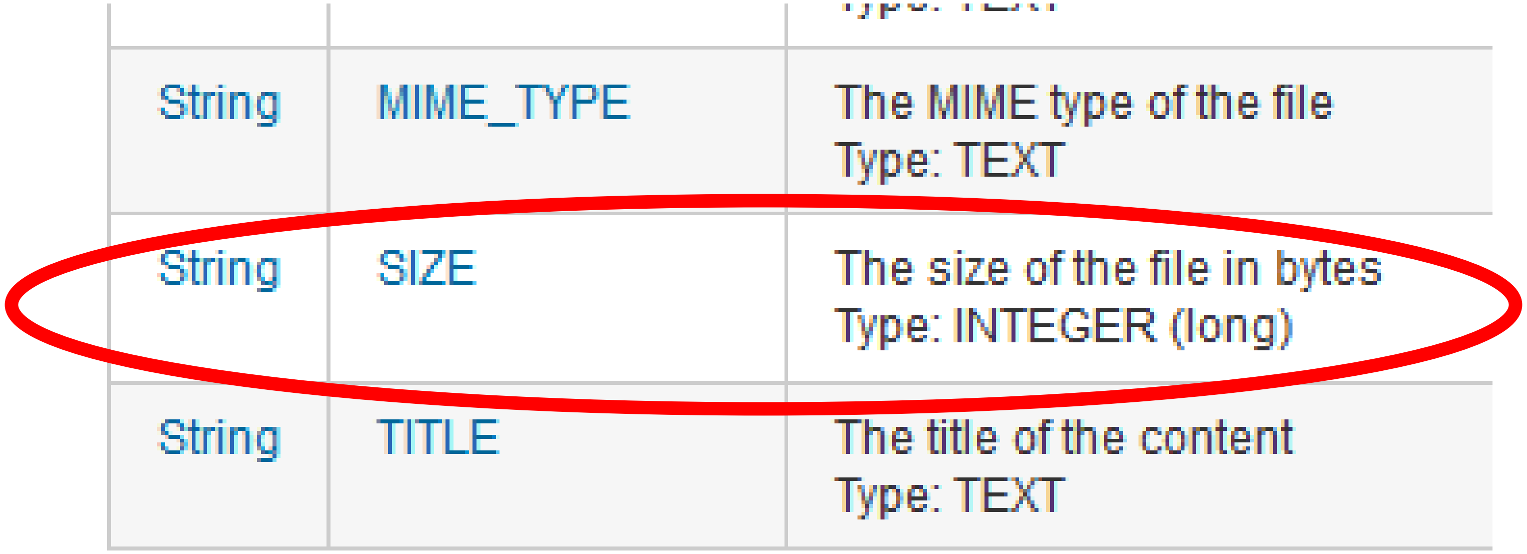
```
// get column indices
int size
    = cursor.getColumnIndex(MediaStore.Images.Media.SIZE);
int dateTaken
    = cursor.getColumnIndex(MediaStore.Images.Media.DATE_TAKEN);
int orientation
    = cursor.getColumnIndex(MediaStore.Images.Media.ORIENTATION);

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



		Type: TEXT
String	MIME_TYPE	The MIME type of the file Type: TEXT
String	SIZE	The size of the file in bytes Type: INTEGER (long)
String	TITLE	The title of the content Type: TEXT

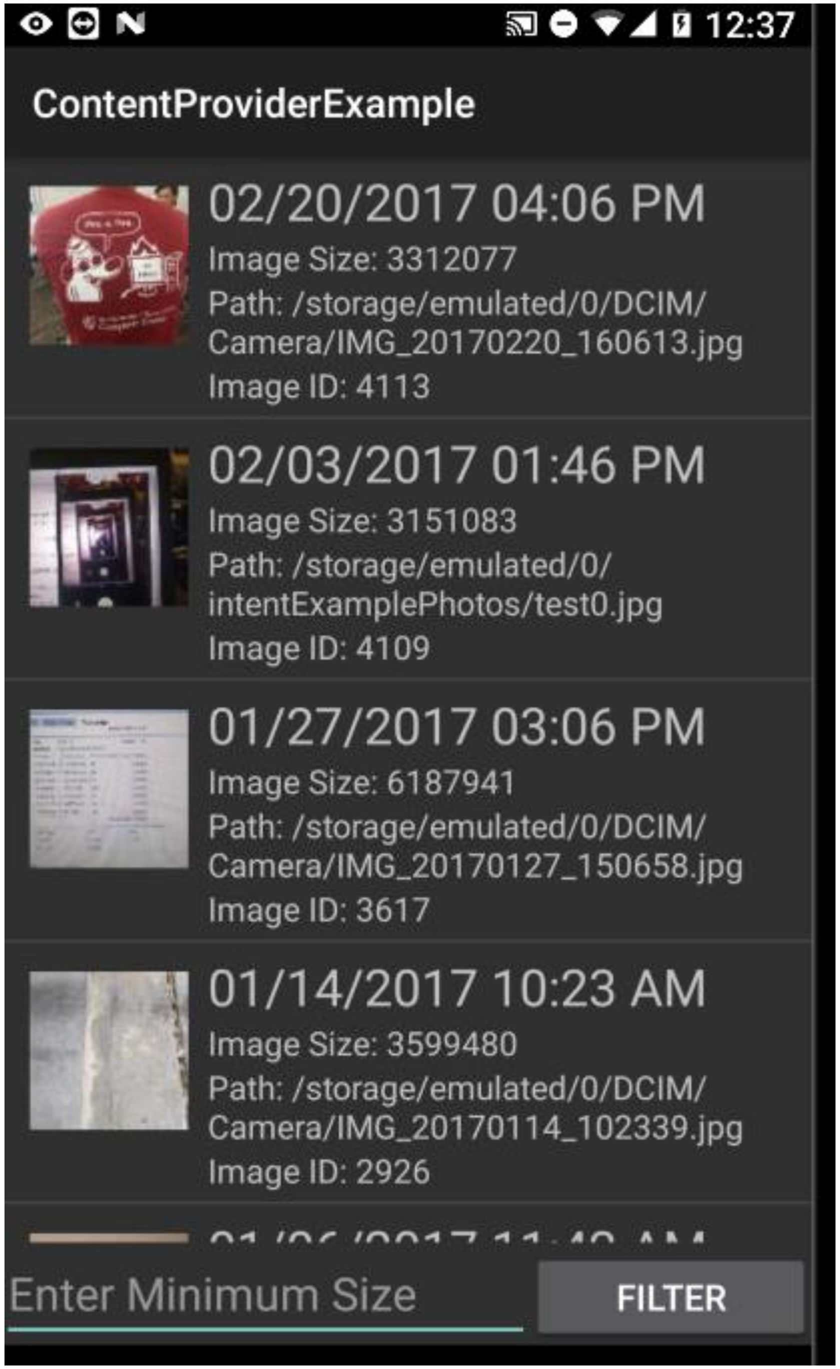
Using Selection Criteria

- 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

```
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);
```


12:37

ContentProviderExample

02/20/2017 04:06 PM
Image Size: 3312077
Path: /storage/emulated/0/DCIM/Camera/IMG_20170220_160613.jpg
Image ID: 4113

02/03/2017 01:46 PM
Image Size: 3151083
Path: /storage/emulated/0/intentExamplePhotos/test0.jpg
Image ID: 4109

01/27/2017 03:06 PM
Image Size: 6187941
Path: /storage/emulated/0/DCIM/Camera/IMG_20170127_150658.jpg
Image ID: 3617

01/14/2017 10:23 AM
Image Size: 3599480
Path: /storage/emulated/0/DCIM/Camera/IMG_20170114_102339.jpg
Image ID: 2926

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

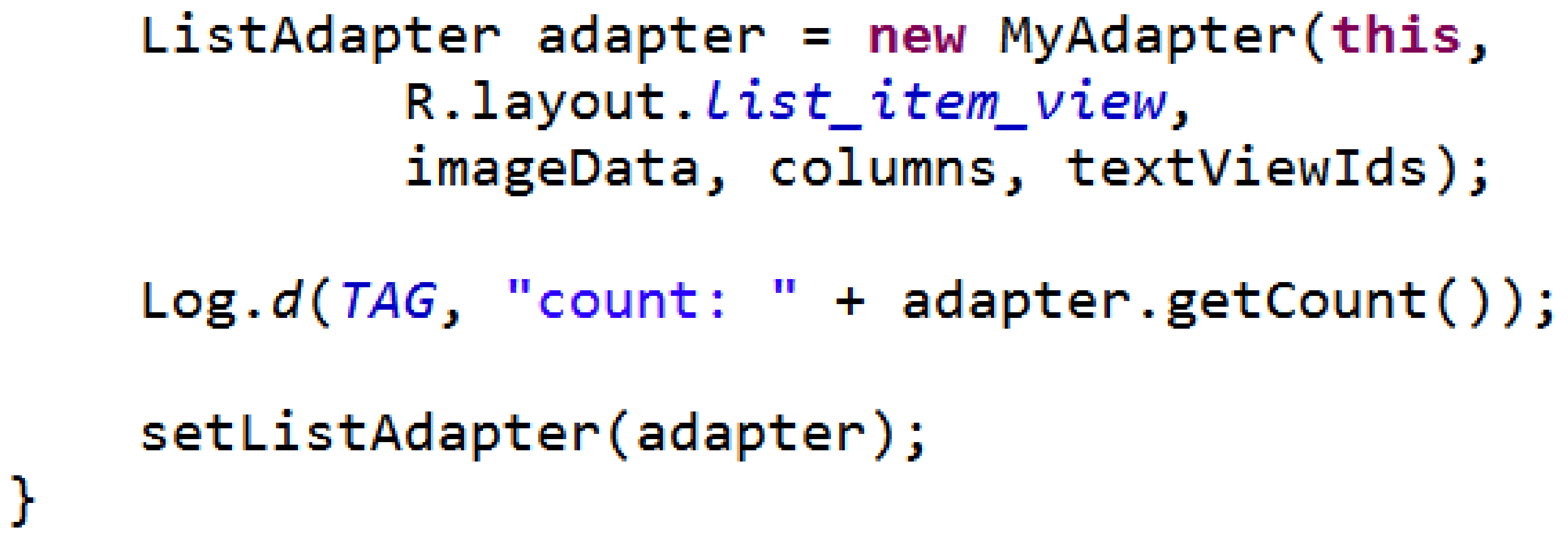
- Specify columns to get from ContentProvider
- Create view that will hold data
  - one row
- Obtain cursor from ContentProvider
- Use ListAdapter to convert data from Cursor to Views
- Sub class adapter to format text

Display Data from ContentProvider

```
private void populateLisView() {
    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

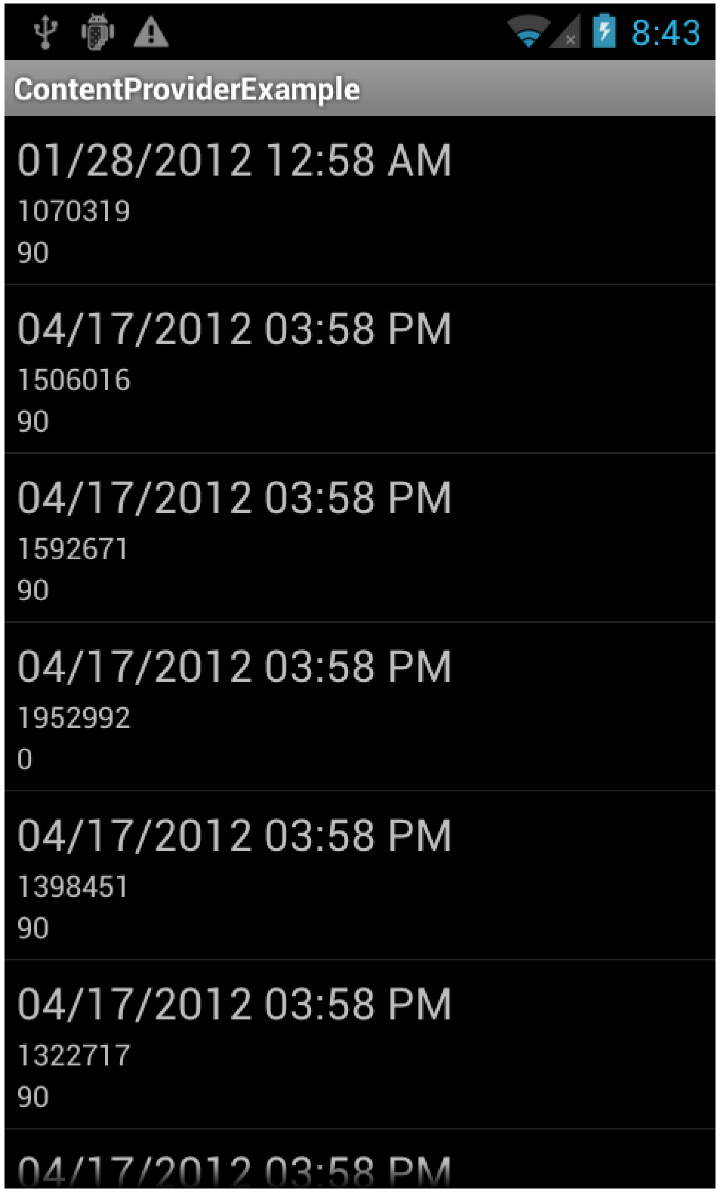


```
    ListAdapter adapter = new MyAdapter(this,
            R.layout.list_item_view,
            imageData, columns, textViewIds);

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

    setListAdapter(adapter);
}
```

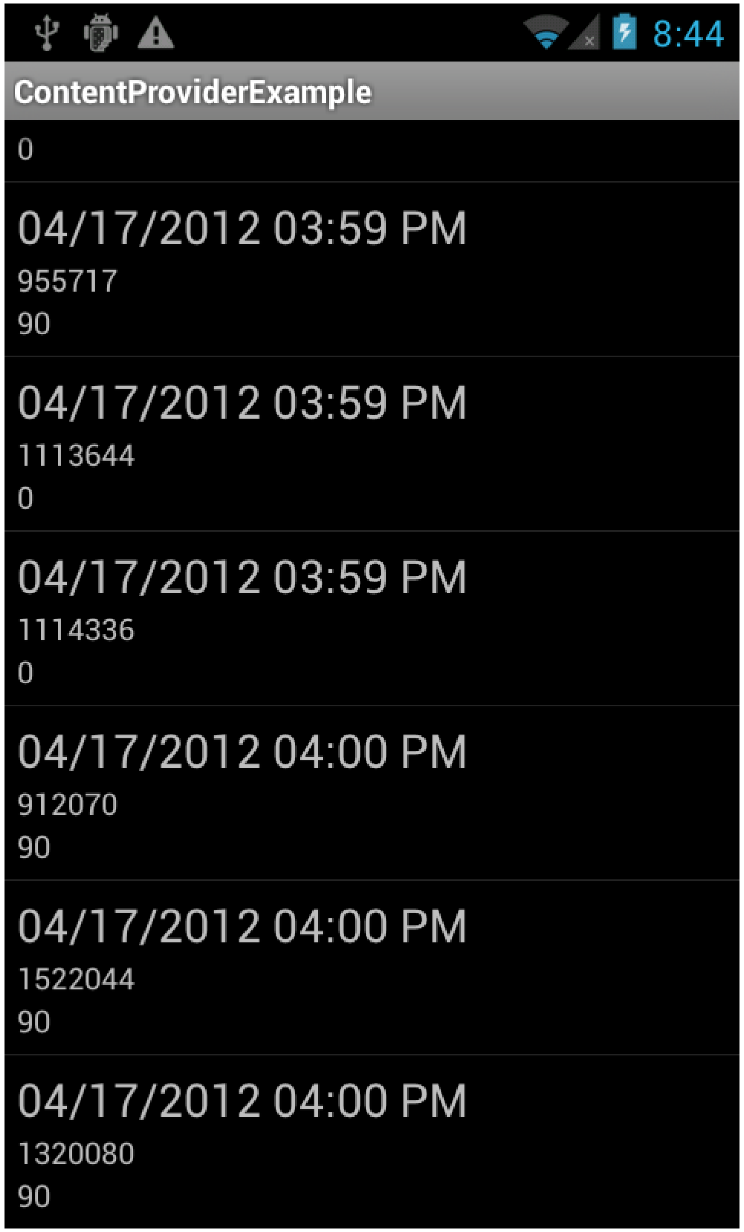
Subclass Adapter

```
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



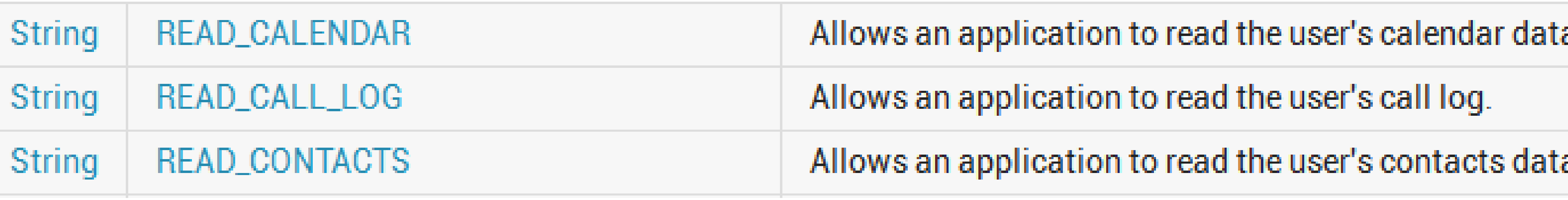
```
ContentProviderExample
01/28/2012 12:58 AM
1070319
90
04/17/2012 03:58 PM
1506016
90
04/17/2012 03:58 PM
1592671
90
04/17/2012 03:58 PM
1952992
0
04/17/2012 03:58 PM
1398451
90
04/17/2012 03:58 PM
1322717
90
04/17/2012 03:58 PM
```



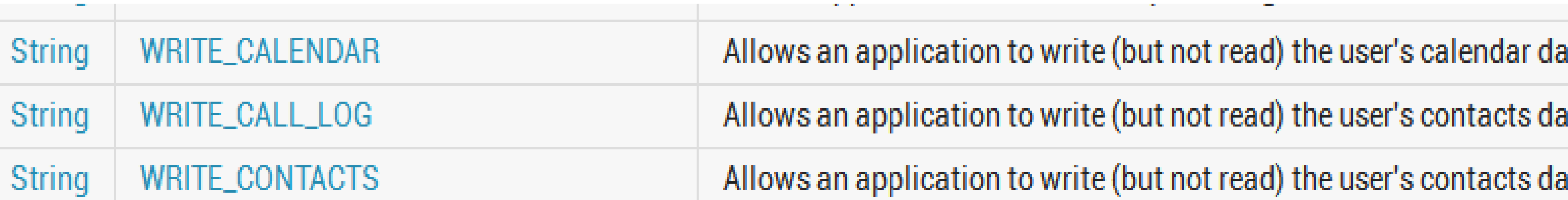
```
ContentProviderExample
0
04/17/2012 03:59 PM
955717
90
04/17/2012 03:59 PM
1113644
0
04/17/2012 03:59 PM
1114336
0
04/17/2012 04:00 PM
912070
90
04/17/2012 04:00 PM
1522044
90
04/17/2012 04:00 PM
1320080
90
```

Permissions

- Using certain content providers require an application request permission
  - Calendar and Contact permissions are catergorized as Dangerous Permissions
  - Must request at runtime



String	READ_CALENDAR	Allows an application to read the user's calendar dat
String	READ_CALL_LOG	Allows an application to read the user's call log.
String	READ_CONTACTS	Allows an application to read the user's contacts dat



String	WRITE_CALENDAR	Allows an application to write (but not read) the user's calendar da
String	WRITE_CALL_LOG	Allows an application to write (but not read) the user's contacts da
String	WRITE_CONTACTS	Allows an application to write (but not read) the user's contacts da

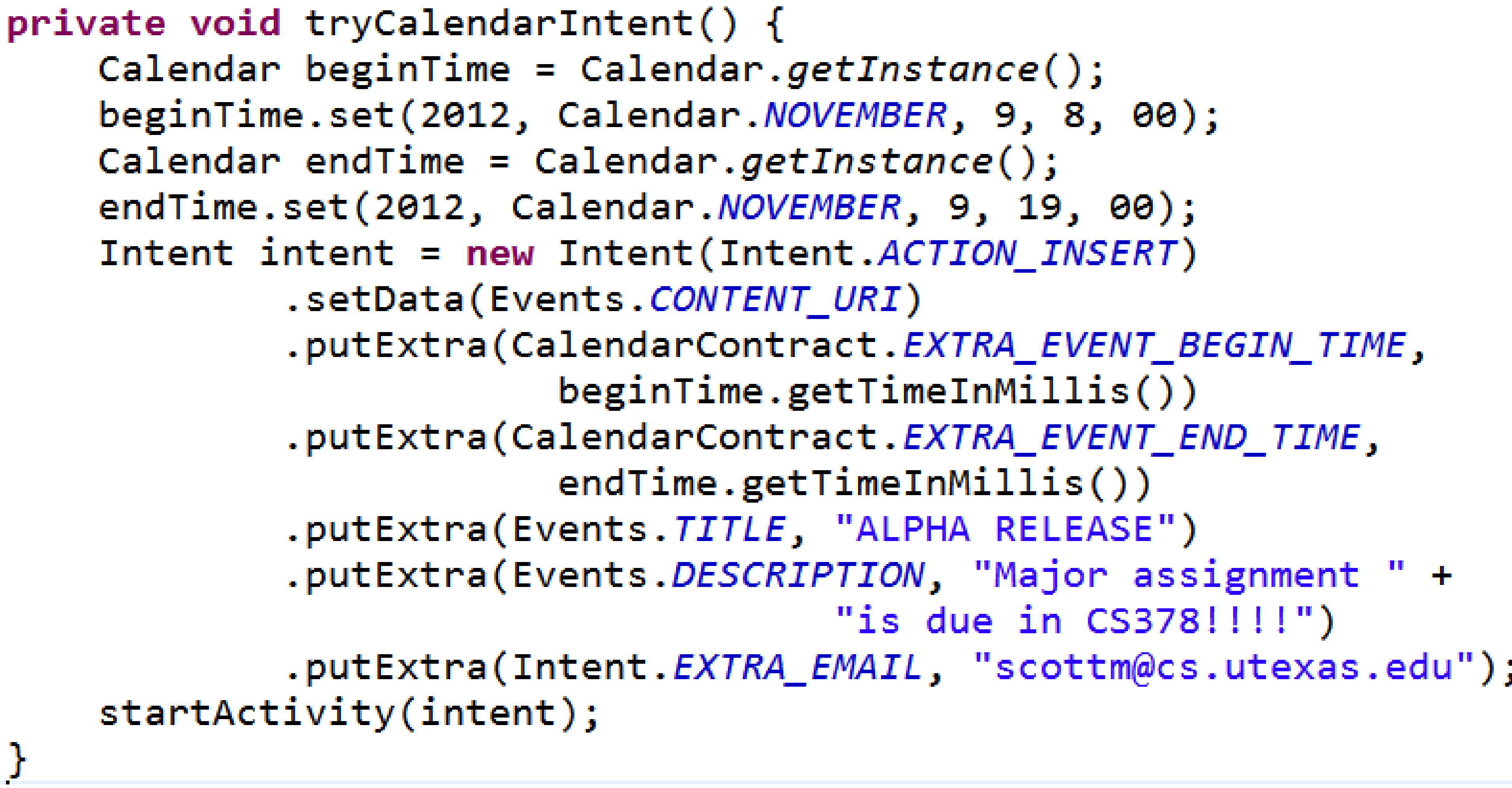
Content Provider Capabilities

- 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

Calendar Event Add



```
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

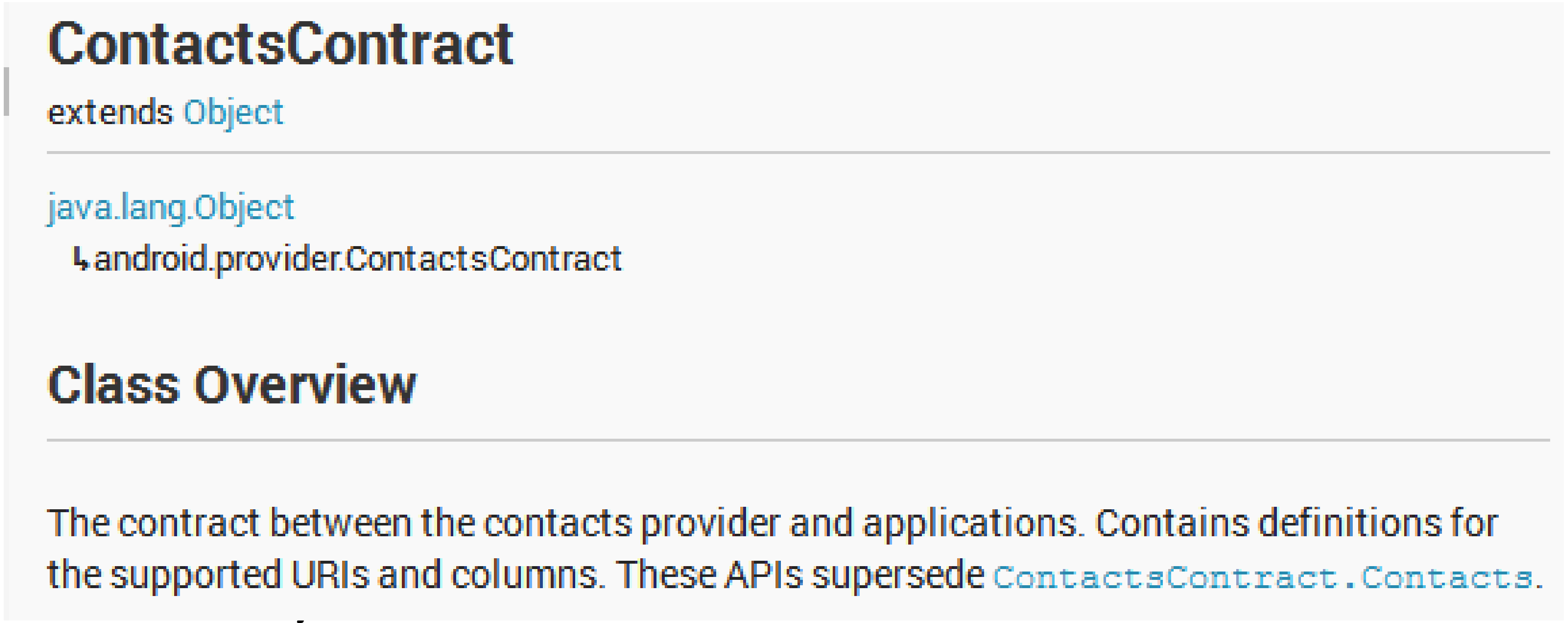
Contacts Provider

- 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

- 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.
  - Want the ability to change how you store data without changing how your own app accesses the data

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)