Data Storage

CS 2046
Mobile Application Development
Fall 2010
Announcements

• HW1 due at midnight **tonight**!
  – See newsgroup for ListView/Adapter clarification.
  – Office hours after class.
  – Don’t forget to submit to CMS!

• HW2 will be released shortly, due Monday, 11/8.
  – Part 1 based off today’s lecture
  – Part 2 based off Wednesday’s lecture
Recap

• Simple data storage methods
  – Key/Value pairs: SharedPreferences
  – Read-only files: res/raw, openRawResource()
  – Read/write files:
    • Internal storage: openInputStream(), openOutputStream()
    • Limited in space, but private to application
Other Internal Storage Methods

- `getFilesDir()`
  - Gets path in filesystem where internal files are stored
- `getDir()`
  - Creates (or opens) directory in internal storage space
- `deleteFile()`
- `fileList()`
External Storage

• Supported on almost every Android device
  – Doesn’t mean active and present at all times
    • User can remove SD card

• All files are world-readable/world-writable
  – Can be modified/removed by other applications
  – Can be modified/removed by the user
Prerequisites

• In Android > 1.5, must declare permission (for writes) in AndroidManifest:

```xml
<manifest>
  <uses-permission
     android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  ...
</manifest>
```

• Get state with
  Environment.getExternalStorageState()
  – One of MEDIA_MOUNTED, MEDIA_MOUNTED_READ_ONLY, or other.
Accessing External Files

• **Application specific:**
  
  – File f = new File(getExternalStorageDirectory(), 
    “/Android/data/<package_name>/files/”)
  
  – Automatically deleted on API 8+

• **Shared files:**

  – new File(getExternalStorageDirectory(), “Dir/”, where Dir is one of:
    
    • Music/, Podcasts/, Ringtones/, Alarms/, Notifications/, Pictures/, Movies/, Download/
    
    • Media scanner will automatically scan files and add to appropriate library.
SQLite Databases

• Tabular data

• Don’t worry if you don’t know SQL!
  – You’ll only ever write one full SQL command in this class – creating the initial database.
  – Plenty of sample code if you’re confused

• Easiest method of creation:
  – Extend SQLiteOpenHelper
  – Call getWritableDatabase() and getReadableDatabase() for an SQLiteDatabase object, which can be queried.
SQLite Datatypes

• Four possibilities
  – INTEGER, REAL, TEXT, BLOB
  – Essentially enough to store anything.
    • Int, long $\rightarrow$ INTEGER, float, double $\rightarrow$ REAL

• Most SQL tables have a special ID column to uniquely identify each row
  – Type: INTEGER PRIMARY KEY

• Create tables with the CREATE command:
  – CREATE TABLE table_name (_id INTEGER PRIMARY KEY, col1 TEXT, col2 INTEGER, col3 BLOB);
Other SQL Snippets

• WHERE clauses
  – Specify the rows returned or modified in a query
  – WHERE happy = 1 AND greeting = “hello” OR age > 5
    • In Android, we leave out the “WHERE ”
    • Plenty more operators, but this is the basic idea.

• Sort orders
  – Order in which results should be returned
  – String formatted like “age ASC happy DESC”
    • ASC goes low to high, DESC high to low
    • Can have multiple columns – sorts by first, then second if first is equal.
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + NOTES_TABLE_NAME + " (" + Notes._ID + " INTEGER PRIMARY KEY," + Notes.TITLE + " TEXT," + Notes.MODIFIED_DATE + " INTEGER" + ");");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + NOTES_TABLE_NAME);
        onCreate(db);
    }
}
ContentProvider

• Could just use raw commands of SQLiteDatabase class to access this database.

• Cons:
  – Accessing raw SQL database isn’t simple
  – Other applications won’t be able to access data

• So, we usually wrap the database in a ContentProvider to make access smoother.
Accessing a ContentProvider

- Use Context.getContentResolver() to get a ContentResolver object.

- Has query, insert, update, delete methods.

- Takes a content URI as input:
ContentProvider Queries

• query() method takes these arguments:
  – Content URI for the content to return
  – Projection – which columns to return
    • String[] array of column names
  – Where clause – which rows to return
  – WhereArgs – don’t worry for now
  – Sort order – order in which to return results
Cursors

- Query returns a Cursor object containing the rows.
- Suppose we made a query with the projection, 
  `new String[] { "Col1", "Col2" }`
  - `Col1` = INTEGER, `Col2` = TEXT

```java
while (cursor.moveToNext()) {
    int col1 = cursor.getInt(0);
    String col2 = cursor.getString(1);
    ...
}
```
Inserts

• ContentValues: object containing key-value pairs
  – Key: column name in table
  – Value: value to insert or update

• Insert: Call ContentResolver.insert() on the content URI for the table to insert into, and values to insert
Updates, Deletes

• Updates and deletes are essentially identical
  – Only difference – update includes the values being updated, delete doesn’t since rows are just being removed.

• Where clause
  – If blank, modify all records at URI
    • Either entire table or specific ID if present in URI
  – If not blank, only modifies record matching clause.
Creating a ContentProvider

• Somewhat boilerplate
  – This sample code is fairly close to the solution to Part 1 of Assignment 2.

• But ContentProviders are very useful for storing application information.
  – Moreover, can have special queries
  – e.g. content://com.example.notes/last_updated – could return note which was last updated for an App Widget.
Initial Setup - Constants

- Useful to define a class of constants
  - Establishes the interface used to access provider

```java
public final class Notes implements BaseColumns {
    public static final String AUTHORITY = "com.google.provider.NotePad";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/notes");
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.google.note";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.google.note";
    public static final String DEFAULT_SORT_ORDER = "modified DESC";
    public static final String TITLE = "title";
    public static final String MODIFIED_DATE = "modified";
}
```
Initial Setup - Provider

- DatabaseHelper class for accessing SQLite DB
- Static constants:

```java
private static final String DATABASE_NAME = "note_pad.db";
private static final int DATABASE_VERSION = 1;
private static final String NOTES_TABLE_NAME = "notes";

private static final int NOTES = 1;
private static final int NOTE_ID = 2;

private static final UriMatcher sUriMatcher;

static {
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(Notes.AUTHORITY, "notes", NOTES);
    sUriMatcher.addURI(Notes.AUTHORITY, "notes/#", NOTE_ID);
}
```
@Override
public boolean onCreate() {
    mOpenHelper = new DatabaseHelper(getContext());
    return true;
}

@Override
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
        case NOTES:
            return Notes.CONTENT_TYPE;
        case NOTE_ID:
            return Notes.CONTENT_ITEM_TYPE;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
    }
}
public Cursor query(Uri uri, String[] projection, String where,
    String[] whereArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(NOTES_TABLE_NAME);

    switch (sUriMatcher.match(uri)) {
    case NOTE:
        break;
    case NOTE_ID:
        qb.appendWhere(Notes._ID + "=" +
            uri.getPathSegments().get(1));
        break;
    default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    SQLiteDatabase db = mOpenHelper.getReadableDatabase();
    Cursor c = qb.query(db, projection, where, whereArgs,
        null, null, sortOrder);
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
ContentProvider - insert

```java
public Uri insert(Uri uri, ContentValues values) {
    // Validate the requested uri
    if (sUriMatcher.match(uri) != NOTES) {
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    long rowId = db.insert(NOTES_TABLE_NAME, Notes.NOTE, values);

    if (rowId > 0) {
        Uri noteUri = ContentUris.withAppendedId(Notes.CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(noteUri, null);
        return noteUri;
    }

    throw new SQLException("Failed to insert row into " + uri);
}
```
ContentProvider – update/delete

- Delete = update except for values

```java
public int update(Uri uri, ContentValues values, String where,
    String[] whereArgs) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri)) {
        case NOTES:
            count = db.update(NOTES_TABLE_NAME, values, where,
                whereArgs);
            break;
        case NOTE_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.update(NOTES_TABLE_NAME, values, Notes._ID +
                "=" + noteId + (!TextUtils.isEmpty(where) ? " AND" +
                " + where + ") : ""), whereArgs);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI "+ uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```
ContentProviders - Recap

• That’s all of the code necessary for a basic SQL-backed ContentProvider.
  – Don’t forget to add entry to AndroidManifest file!

• For more features, you can modify the basic framework.
  – Example – Assignment 2, automatic maintenance of LAST_MODIFIED column.

• Could also use a non-SQL backend if desired.
## Storage Methods Summary

<table>
<thead>
<tr>
<th>Data</th>
<th>Visibility</th>
<th>Size</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value Pairs</td>
<td>Private</td>
<td>Small</td>
<td>SharedPreferences</td>
</tr>
<tr>
<td>File</td>
<td>Private, Read-only</td>
<td>Small</td>
<td>res/raw</td>
</tr>
<tr>
<td>File</td>
<td>Private, Read/Write</td>
<td>Small-Medium</td>
<td>Internal Storage</td>
</tr>
<tr>
<td>File</td>
<td>Public, Read/Write</td>
<td>Any</td>
<td>External Storage</td>
</tr>
<tr>
<td>Table</td>
<td>Private</td>
<td>Any</td>
<td>SQLite</td>
</tr>
<tr>
<td>Table</td>
<td>Public</td>
<td>Any</td>
<td>ContentProvider</td>
</tr>
</tbody>
</table>
Intro of the Day - Live Folders

• Displays application data on the Home Screen outside of your application.

• Data source: ContentProvider
  – Display will automatically update if a change is made in background

• From article at:
  http://developer.android.com/resources/articles/live-folders.html
Live Folder Activity

• First step: Have an Activity which tells Android what to do
  – Can just reuse existing Activity
  – Specify an Intent Filter with:
    • action=android.intent.action.CREATE_LIVE_FOLDER
    • category=android.intent.category.DEFAULT
if(LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(getIntent().getAction())) {
    Intent i = new Intent();
    i.setData(CONTENT_URI);
    i.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "Example");
    i.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON, Intent.ShortcutIconResource.fromContext(this, R.drawable.<live_folder_icon>));
    i.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE, LiveFolders.DISPLAY_MODE_LIST);
    setResult(RESULT_OK, i);
    finish();
}
Changes to ContentProvider

• Provider must obey certain rules for the URI passed by Activity:
  – Must return at least two columns, LiveFolders._ID and LiveFolders.NAME
    • Optional columns for things like icon, description, Intent to run when item is clicked

• Add URI for LiveFolder queries

• Use a projection map to “rename” columns
private static final HashMap<String, String> LIVE_FOLDER_PROJECTION_MAP;

static {
    LIVE_FOLDER_PROJECTION_MAP = new HashMap<String, String>();
    LIVE_FOLDER_PROJECTION_MAP.put(LiveFolders._ID, Notes._ID + " AS " + LiveFolders._ID);
    LIVE_FOLDER_PROJECTION_MAP.put(LiveFolders.NAME, Notes.TITLE + " AS " + LiveFolders.NAME);
}

In query:

SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
switch(sUriMatcher.match(uri)) {
    case LIVE_FOLDER:
        qb.setProjectionMap(LIVE_FOLDER_PROJECTION_MAP);
        ...
}