Single Application Persistent Data Storage

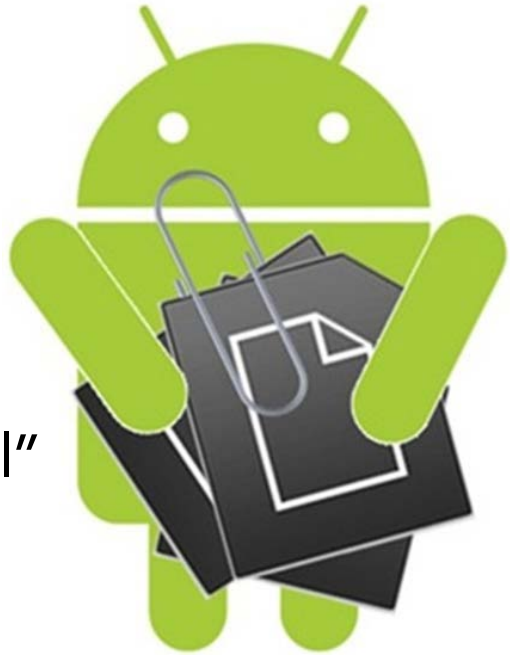
Programming the Android Platform

CS 282
Principles of Operating Systems II
Systems Programming for Android

Some Data Storage Options

- Android offers several ways to store data
  - Files
  - SQLite database
  - SharedPreferences

Android Files



- Android largely uses Java file I/O for local files
  - e.g., File, InputStream, OutputWriter, etc.
- Represents a file system entity identified by a pathname
  - Limitations where you can create, read, & write files
- Storage areas classified as "internal" vs. "external"
  - Internal storage refers to an application's portion of the on-board, always-available flash storage
    - Usually used for application private files
  - External storage can be mounted by the user as a drive in Windows et al.
    - Usually used for public files that are visible to all apps & users
- Files can be saved as a cache or on a more permanent basis
  - Files in a cache directory may be deleted automatically to free up storage for users, whereas files outside the cache must be deleted manually

Android File API

- boolean isDirectory()
  - Return true if this File represents a directory
- boolean mkdir()
  - Creates the directory named by the trailing filename of this file
- boolean exists()
  - Returns a boolean indicating if this file can be found on the underlying file system

public class

File

extends Object
implements Serializable Comparable<T>

java.lang.Object
↳java.io.File

Summary: Fields | Ctors | Methods | Inherited Methods | [Expand All]
Since: API Level 1

Class Overview

An "abstract" representation of a file system entity identified by a pathname. The pathname may be absolute (relative to the root directory of the file system) or relative to the current directory in which the program is running.

The actual file referenced by a `File` may or may not exist. It may also, despite the name `File`, be a directory or other non-regular file.

This class provides limited functionality for getting/setting file permissions, file type, and last modified time.

On Android strings are converted to UTF-8 byte sequences when sending filenames to the operating system, and byte sequences returned by the operating system (from the various `list` methods) are converted to strings by decoding them as UTF-8 byte sequences.

- boolean setReadable(boolean readable)
  - Sets read permission on this File
- String getAbsolutePath()
  - Returns absolute path to this File
- Many others (see documentation)

http://developer.android.com/reference/java/io/File.html

Writing an Internal Memory File

```
// Open file with ContextWrapper.openFileOutput()
FileOutputStream fos = openFileOutput(fileName, MODE_PRIVATE);
PrintWriter pw = new PrintWriter
        (new BufferedWriter(new OutputStreamWriter(fos)));
pw.println(…); // Write to file
  …
pw.close(); // Close file
```

- openFileOutput() (& openFileInput()) do not accept file paths, just simple file names
- Use getFilesDir() & getCacheDir() to return a File object that can create files and subdirectories

Reading an Internal Memory File

```
// Open file with ContextWrapper.openFileInput()
FileInputStream fis = openFileInput(fileName);
BufferedReader fr =
    new BufferedReader(new InputStreamReader(fis));
String line = "";
// Read from file
while (null != (line = fr.readLine())) {
   // process data
 }
// Close file
fr.close();
```

Android External Memory Files

- External memory files are often too big to risk putting in internal storage or should be downloadable off a device at will
- File Context.getExternalFilesDir()
  - Returns a File object pointing to an automatically-created directory on external storage, unique for your application
  - Such files are automatically deleted when an application is uninstalled
- Removable media may appear/disappear without warning
- String Environment.getExternalStorageState()
  - MEDIA_MOUNTED - present & mounted with read/write access
  - MEDIA_MOUNTED_READ_ONLY - present & mounted with read-only access
  - MEDIA_REMOVED - not present
- Need permission to read & write external files
  - <uses-permission android:name= "android.permission.WRITE_EXTERNAL_STORAGE" />
  - <uses-permission android:name= "android.permission.READ_EXTERNAL_STORAGE" />

Android External Memory Files

- External memory files that belong more to a user than an app should use getExternalStoragePublicDirectory()
- File Environment.getExternalStoragePublicDirectory()
  - Returns a File object pointing to an automatically-created directory on external storage set aside for a particular type of file
    - e.g., DIRECTORY_MOVIES, DIRECTORY_MUSIC, DIRECTORY_PICTURES, etc.
  - Such files are *not* deleted when an application is uninstalled
- Also permission to read & write external public files
  - <uses-permission android:name= "android.permission.WRITE_EXTERNAL_STORAGE" />
  - <uses-permission android:name= "android.permission.READ_EXTERNAL_STORAGE" />

Writing an External Memory File

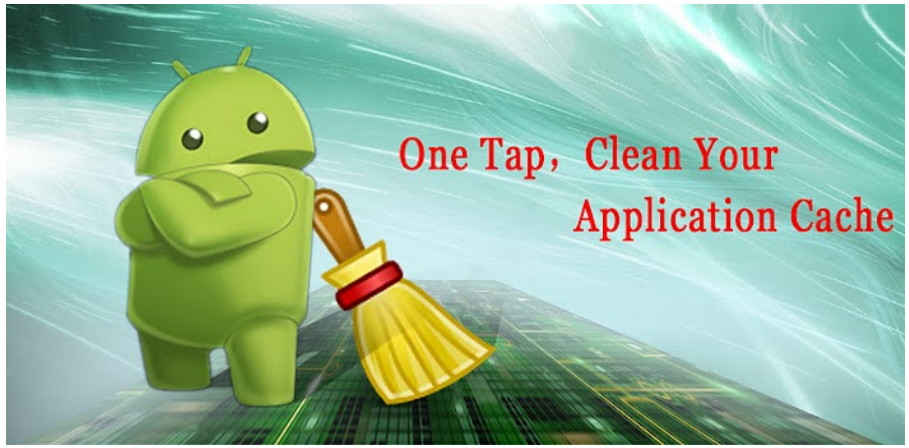
```
public class FileWriteAndReadActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    ...
    if (Environment.MEDIA_MOUNTED.equals(
                                  Environment.getExternalStorageState())) {
     File outFile = new File(getExternalFilesDir(
                                  Environment.DIRECTORY_PICTURES),fileName);
     try {
       BufferedOutputStream os =
            new BufferedOutputStream(new FileOutputStream(outFile));
       BufferedInputStream is =
           new BufferedInputStream(getResources()
                                                .openRawResource(R.drawable.icon));
      copy(is, os);
     } catch (FileNotFoundException e) {}
  }
}
...
```

Writing an External Memory File

```
private void copy(InputStream is, OutputStream os) {
  final byte[] buf = new byte[1024];
  int numBytes;
  try {
    while (-1 != (numBytes = is.read(buf))) {
      os.write(buf, 0, numBytes);
    }
  } catch (IOException e) {…
  } finally {
    try {
      is.close();
      os.close();
    } catch (IOException e) {}
…
```

Android Cache Files

- Cache files are temporary files that may be deleted by the system when storage is low
  - App should continue to functional normally even if these files are deleted from the cache
- File Context.getCacheDir()
  - Returns absolute path to an application-specific directory that can be used for temporary files
- Files deleted automatically when application uninstalled
- Context.getExternalCacheDir() returns a File representing external storage directory for cache files
- These files are also deleted automatically when application uninstalled



Android SQLite

- SQLite provides an in-memory database
- It is typically used in a "relational" fashion
  - i.e., it contains tables (consisting of rows & columns), indexes, etc. that form a "schema"
- Designed to operate within a very small footprint (<300kB) within a single cross-platform disk file
- Implements most of SQL92 & supports ACID transactions
  - ACID = atomic, consistent, isolated & durable
- Access to an SQLite database involves accessing the filesystem
  - This can be slow, so typically perform database operations asynchronously, e.g., inside the AsyncTask class

http://www.vogella.com/articles/AndroidSQLite/article.html

SQLiteDatabase

- SQLiteDatabase is the base class for working with a SQLite database in Android
  - It provides the insert(), update(), & delete() methods
  - It also provides the execSQL() method that can execute an SQL statement directly
- Queries can be created via rawQuery() & query() methods or via the SQLiteQueryBuilder class
- The object ContentValues is used to define key/values
  - The "key" represents the table column identifier & the "value" represents the content for the table record in this column
- ContentValues can be used for inserts & updates of database entries

developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

SQLiteOpenHelper

- Recommended means of using SQLiteDatabase is to subclass the SQLiteOpenHelper class
  - In your constructor call the super() method of SQLiteOpenHelper, specifying the database name & current database version
  - Override onCreate(), which is called by SQLite if the database does not yet exist, e.g., execute CREATE TABLE command
  - Override onUpgrade(), which is called if the database version increases in your app code to allow you to update the database schema
- Use SQLiteOpenHelper methods to open & return underlying database
  - e.g., getReadableData() & getWriteableDatabase() to access an SQLiteDatabase object either in read or write mode

developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

Opening an SQLite Database

```
public class DatabaseHelper extends SQLiteOpenHelper {
  final private static String CREATE_CMD =
          "CREATE TABLE artists (”
                  + “_id”+ “ INTEGER PRIMARY KEY AUTOINCREMENT, ”
                  + “name” + " TEXT NOT NULL)”;

  public DatabaseHelper(Context context) {
    super(context, “artist_db”, null, 1);
  }

  public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_CMD);
  }

  public void onUpgrade(SQLiteDatabase db,
        int oldVersion, int newVersion) { /* … */ }
…
```

- It is best practice to create a separate class per table

Using an SQLite Database

```
public class DatabaseExampleActivity extends ListActivity {
  final static String[] columns ={ "_id", "name"};
  static SQLiteDatabase db = null;
  public void onCreate(Bundle savedInstanceState) {
    …
    DatabaseHelper  dbHelper = new DatabaseHelper
                                  (getActivity().getApplicationContext());
    db = dbHelper.getWritableDatabase();
    insertArtists();
    deleteLadyGaga();
    Cursor c = readArtists();
   displayArtists(c);
}
```

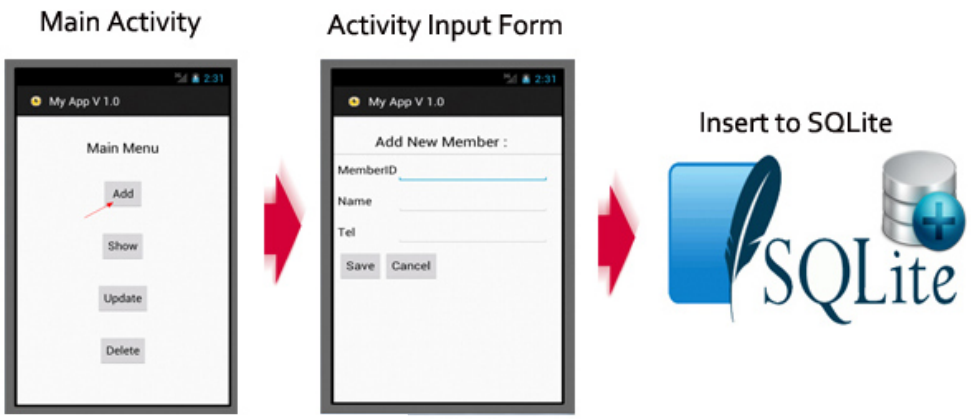
- It's often a good idea to put an SQLiteDatabase into an Application singleton to simplify concurrency & memory management

Inserting into an SQLite Database

public long insert (String table, String nullColHack, ContentValues values)

Method for inserting a row into the database

table	the table to insert the row into
nullCol Hack	optional; may be null.
values	map containing initial col values for row. Keys are col names & values col values




```
private void insertArtists() {
  ContentValues values =
    new ContentValues();
  values.put("name", "Lady Gaga");
  db.insert("artists", null, values);
  values.clear();
  values.put("name", "Johnny Cash");
  db.insert("artists", null, values);
  values.clear();
  values.put("name", "Sting");
  db.insert("artists", null, values);
}
```

http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#insert(java.lang.String, java.lang.String, android.content.ContentValues)

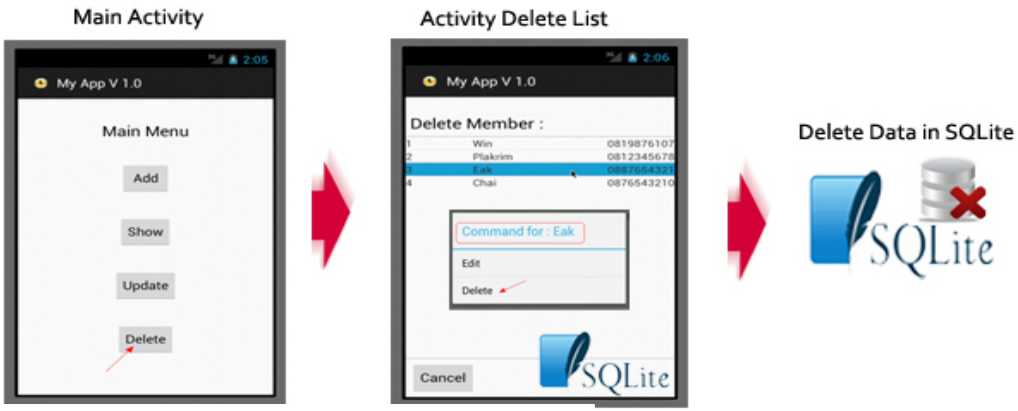
Deleting from an SQLite Database

public int delete(String table, String whereClause, String[] whereArgs)

method for deleting rows in database

table	the table to delete from
where Clause	optional WHERE clause to apply when deleting Passing null deletes all rows

```
private int deleteLadyGaga() {
  return db.delete(
            "artists",
            " name" +"=?",
            new String [] {"Lady Gaga"});
}
```



http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#delete(java.lang.String, java.lang.String, java.lang.String[])

Querying an SQLite Database

- You can use rawQuery() or a query() on an SQLiteDatabase

  public Cursor rawQuery(String sql, String[] selectionArgs)

  Runs the provided SQL and returns a Cursor over the result set.

  Parameters

sql	the SQL query. The SQL string must not be ; terminated
selectionArgs	You may include ?s in where clause in the query, which are replaced by the values from selectionArgs. The values will be bound as Strings.

  Returns

  - A Cursor object, which is positioned before the first entry
  - Cursors are not synchronized, see the documentation for more details.

- **public Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)**
  - query() builds up a SQL SELECT statement from its component parts
    - e.g., name of the table to query, WHERE clause, & positional parameters

Parameters of the query() Method

Parameter	Comment
String dbName	The table name to compile the query against.
int[] columnNames	A list of which table columns to return. Passing "null" will return all columns.
String whereClause	Where-clause, i.e. filter for the selection of data, null will select all data.
String[] selectionArgs	You may include ?s in the "whereClause"". These placeholders will get replaced by the values from the selectionArgs array.
String[] groupBy	A filter declaring how to group rows, null will cause the rows to not be grouped.
String[] having	Filter for the groups, null means no filter.
String[] orderBy	Table columns which will be used to order the data, null means no ordering.

Using Query() vs. rawQuery()

- Using rawQuery() on an SQLiteDatabase

```
private Cursor readArtists() {
  // returns all rows
  return db.rawQuery("SELECT _id, name, FROM artists", null);
}
```

http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#rawQuery(java.lang.String, java.lang.String[])

- Using query() on an SQLiteDatabase

```
private Cursor readArtists() {
  // returns all rows
  return db.query("artists", new String [] {"_id","name"}, null,
                        new String[] {}, null, null, null);
}
```

http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html#query(java.lang.String, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, java.lang.String, java.lang.String)

Cursor Iterators

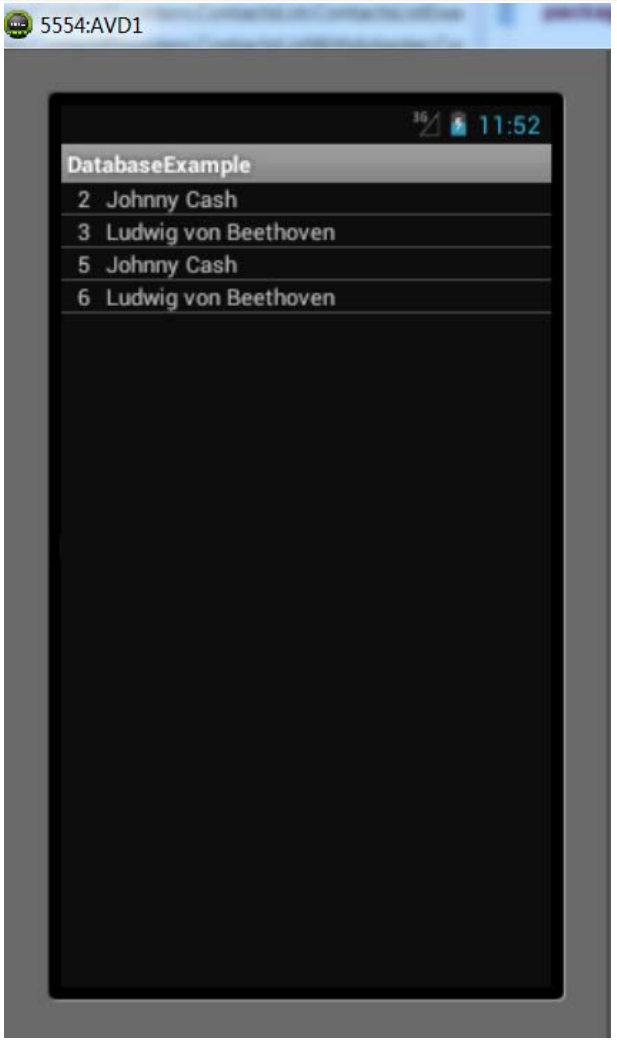
- Query() returns a Cursor Iterator that represents result of a query & points to one row of query result
  - This allows buffering the query results efficiently since it needn't load all data into memory
- getCount() returns # of elements of the resulting query
- moveToFirst() & moveToNext() move between individual data rows
- isAfterLast() checks if the end of the query result has been reached
- Cursor provides typed get*() methods
  - e.g., getLong(columnIndex) & getString(columnIndex) to access column data for current position of result
- Cursor also provides getColumnIndexOrThrow (String) to get column index for a column name of table
- A Cursor must be closed via close()

http://developer.android.com/reference/android/database/Cursor.html

Displaying an SQLite Database

- The SimpleCursorAdapter class maps the columns to the Views based on the Cursor passed to it

```
private void displayArtists (Cursor c) {
  setListAdapter(
      new SimpleCursorAdapter
      ( this,  R.layout.list_layout, c,
        new String [] {"_id", "name"},
        new int[] { R.id._id, R.id.name }));
 }
```



http://developer.android.com/reference/android/widget/SimpleCursorAdapter.html

Examining the Database Remotely

- If your app creates a database it is saved by default in directory

  DATA/data/APP_NAME/databases/FILENAME

- The parts of the above directory are constructed based on the following rules
  - DATA is the path that the Environment. getDataDirectory() method returns
  - APP_NAME is your app name
  - FILENAME is the name you specify in your application code for the database

- Can examine database with sqlite3
  - # adb -s emulator-5554 shell
  - # sqlite3 /data/data/course.examples.DataManagement.DataBaseExample/databases/artist_db

SharedPreferences

- A persistent map that holds key-value pairs of simple data types
  - Automatically managed across application uses
- Often used for long-term storage of customizable application data
  - e.g., user preferences like User ID & Favorite WiFi networks
- Activities can store shared preferences Activity.getPreferences (int mode)
  - Mode: MODE_PRIVATE, MODE_WORLD_READABLE or MODE_WORLD_WRITEABLE
  - Returns SharedPreference for current Activity
- Applications can store shared preferences Context.getSharedPreferences (String name, int mode)
  - name – name of SharedPreference file
  - Mode – MODE_PRIVATE, MODE_WORLD_READABLE, or MODE_WORLD_WRITEABLE
  - Returns named SharedPreference object for this (application) context

Writing/Reading SharedPreferences

- To write a SharedPreference call SharedPreferences.edit()
  - Returns a SharedPreferences.Editor instance
- Add values with SharedPreferences.Editor
- Remove values with SharedPreferences.remove() or SharedPreferences.clear() (removes all)
- Commit values with SharedPreferences.Editor.commit() (synchronous) or SharedPreferences.Editor.apply() (asynchronous -- & preferred)

- To read a SharedPreference use SharedPreferences methods, e.g.,
  - getAll()
  - getBoolean()
  - getString()

SharedPreferences Example

```
public class SharedPreferenceReadWriteActivity extends Activity {
  private static String HIGH_SCORE = "high_score";
  public void onCreate(Bundle savedInstanceState) {
    ...
    final SharedPreferences prefs = getPreferences(MODE_PRIVATE);
    final Button go = ...
    go.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        ...
        int val= ...
        if (val >  prefs.getInt(HIGH_SCORE, 0)) {
          ...
          SharedPreferences.Editor editor = prefs.edit();
          editor.putInt(HIGH_SCORE, val);
          editor.commit();
        }
...
```

SharedPreferences (cont.)

```
public class DataManagementPreferencesActivity extends Activity {
   SharedPreferences prefs;
   final static String USERNAME = "uname";
   public void onCreate(Bundle savedInstanceState) {
      …
    prefs = PreferenceManager.
                getDefaultSharedPreferences(getApplicationContext());
    final Button button  = …;
    button.setOnClickListener(new OnClickListener() {
       public void onClick(View v) {
          startActivity(new Intent
                              (DataManagementPreferencesActivity.this,
                                LoadPreferencesActivity.class));
       }
   });
}
…
```

- getDefaultSharedPreferences(), on PreferenceManager, gets the shared preferences that work in concert with Android's overall preference framework
- Generally the best way to get SharedPreferences

SharedPreferences (cont.)

```
public class LoadPreferencesActivity extends PreferenceActivity {
  protected void onCreate(Bundle savedInstanceState) {
    ...
    final SharedPreferences prefs =
                    PreferenceManager.getDefaultSharedPreferences(this);
    addPreferencesFromResource(R.xml.user_prefs);
    final EditTextPreference uNamePref = (EditTextPreference)
                        getPreferenceScreen().findPreference(USERNAME);
    uNamePref.setSummary(prefs.getString(USERNAME, ""));
    prefs.registerOnSharedPreferenceChangeListener(
     new OnSharedPreferenceChangeListener() {
       public void onSharedPreferenceChanged(
                         SharedPreferences sharedPreferences, String key) {
          uNamePref.setSummary(prefs.getString(USERNAME, ""));
       }
    });
...
```

user_prefs.xml

```
<PreferenceScreen …>
   <EditTextPreference
         android:dialogMessage="Enter Your User Name"
         android:dialogTitle="Change User Name"
         android:positiveButtonText="Submit"
         android:negativeButtonText="Cancel"
         android:title="User Name"
         android:key="uname">
    </EditTextPreference>
</PreferenceScreen>
```

Source Code Examples

- DataManagementFileInternalMemory
- DataManagementFileExternalMemory
- DataManagementSharedPreference
- DataManagementPreferenceActivity
- DataManagementSQL