Android Data

Éric Languénou

2013-2014

UNIVERSITÉ DE NANTES

Data in Android Devices

UNIVERSITÉ DE NANTES

Various ways for data storage:

1. Shared Preferences : key-value pair mapping;
2. Internal Storage : storage in the phone internal memory;
3. External Storage : storage in the *external memory*, usually SD card;
4. SQLite database : structured data;
5. Distant Data : storage on a distant server.

Shared Preferences

UNIVERSITÉ DE NANTES

- SharedPreferences class provides us with a frameWork for:
  - persistent storage;
  - mapping access to pairs (key/values) with basics type (`boolean, float, int, long, string`).
- Access is made through the methods :
  - Context.getPreferences(mode):
    - uses default app memory storage
    - mode specifies read/write rights (`MODE_PRIVATE/ MODE_WORLD_READABLE/MODE_WORLD_WRITEABLE`)
- Context.getSharedPreferences(name, mode)
  - specifies the files where the data are stored (this allows multiple files per application)

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Shared Preferences(2)

UNIVERSITÉ DE NANTES

- Access and modification by methods
  `edit()` / `putX()` / `getX()` / `commit()`

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

Internal Storage

UNIVERSITÉ DE NANTES

- uses phone internal memory
- are (by default) private to the application
- deleted when the app is de-installed
- to create and write data in a private file :
  - `Context.openFileOutput(file, mode)` : open a file using access mode, return a `FileOutputStream`
  - `write()` to write data;
  - `close()` to close the file.

Internal Storage(2)

UNIVERSITÉ DE NANTES

- Other usefull methods
  - `getFilesDir()`
  - `getDir()`
  - `deleteFile()`
  - `fileList()`
- advantages :
  - faster
  - more space (compare to sharedPreferences)
  - always avalaible
- Inconvenients
  - limited (structure)

Internal Storage(3)

UNIVERSITÉ DE NANTES

- `Mode` specifies the file access and sharing mode
  - `MODE_PRIVATE`: private to the app; create or destroy if file exists;
  - `MODE_APPEND`: add data to the file;
  - `MODE_WORLD_READABLE`: set the file accessible (read) to any app;
  - `MODE_WORLD_WRITEABLE`: set the file accessible (write) to any app;

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

External Storage (1)

UNIVERSITÉ DE NANTES

- for external devices storage;
- apps can read/write data on this devices (with a prior check for existence);
- data are destroyed whenever the app is uninstalled.

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    //  to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```

External Storage (2)

UNIVERSITÉ DE NANTES

- saving
  - in the SD-card:
    - external memory (removable) can be modified or deleted using usb connexion;
    - internal memory (non removable)
- various directories
  - in the memory:
    - Music/ - Media scanner classifies all media found here as user music.
    - Podcasts/ - Media scanner classifies all media found here as a podcast.
    - Ringtones/ - Media scanner classifies all media found here as a ringtone.
    - Alarms/ - Media scanner classifies all media found here as an alarm sound.
    - Notifications/ - Media scanner classifies all media found here as a notification sound.
    - Pictures/ - All photos (excluding those taken with the camera).
    - Movies/ - All movies (excluding those taken with the camcorder).
    - Download/ - Miscellaneous downloads.

External Storage (3)

UNIVERSITÉ DE NANTES

- data access:
  - API >= 8:
    - `getExternalFilesDir()` : gets the absolute path to external memory
    - a parameter specifies the type of desired media (AUDIO, VIDEO...);
    - this parameter set to null, gets the root path.
  - API <= 7:
    - `getExternalStorageDirectory()` : gets the absolute path to external memory
    - the value is `/Android/data/<package_name>/files`

External Storage : Code Sample API>= 8

UNIVERSITÉ DE NANTES

```
void createExternalStoragePrivateFile() {
    // Create a path where we will place our private file on external
    // storage.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");

    try {
        // Very simple code to copy a picture from the application's
        // resource into the external file.  Note that this code does
        // no error checking, and assumes the picture is small (does not
        // try to copy it in chunks).  Note that if external storage is
        // not currently mounted this will silently fail.
        InputStream is = getResources().openRawResource(R.drawable.balloons);
        OutputStream os = new FileOutputStream(file);
        byte[] data = new byte[is.available()];
        is.read(data);
        os.write(data);
        is.close();
        os.close();
    } catch (IOException e) {
        // Unable to create file, likely because external storage is
        // not currently mounted.
        Log.w("ExternalStorage", "Error writing " + file, e);
    }
}

void deleteExternalStoragePrivateFile() {
    // Get path for the file on external storage.  If external
    // storage is not currently mounted this will fail.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
    if (file != null) {
        file.delete();
    }
}

boolean hasExternalStoragePrivateFile() {
    // Get path for the file on external storage.  If external
    // storage is not currently mounted this will fail.
    File file = new File(getExternalFilesDir(null), "DemoFile.jpg");
    if (file != null) {
        return file.exists();
    }
    return false;
}
```

External Storage : Code S. API>= 8 (cont.)

UNIVERSITÉ DE NANTES

```
void createExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in our own private
    // pictures directory.  Note that we don't really need to place a
    // picture in DIRECTORY_PICTURES, since the media scanner will see
    // all media in these directories; this may be useful with other
    // media types such as DIRECTORY_MUSIC however to help it classify
    // your media for display to the user.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    File file = new File(path, "DemoPicture.jpg");

    try {
        // Very simple code to copy a picture from the application's
        // resource into the external file.  Note that this code does
        // no error checking, and assumes the picture is small (does not
        // try to copy it in chunks).  Note that if external storage is
        // not currently mounted this will silently fail.
        InputStream is = getResources().openRawResource(R.drawable.balloons);
        OutputStream os = new FileOutputStream(file);
        byte[] data = new byte[is.available()];
        is.read(data);
        os.write(data);
        is.close();
        os.close();

        // Tell the media scanner about the new file so that it is
        // immediately available to the user.
        MediaScannerConnection.scanFile(this,
                new String[] { file.toString() }, null,
                new MediaScannerConnection.OnScanCompletedListener() {
            public void onScanCompleted(String path, Uri uri) {
                Log.i("ExternalStorage", "Scanned " + path + ":");
                Log.i("ExternalStorage", "-> uri=" + uri);
            }
        });
    } catch (IOException e) {
        // Unable to create file, likely because external storage is
        // not currently mounted.
        Log.w("ExternalStorage", "Error writing " + file, e);
    }
}
```

External Storage : File Structure; API>= 8

UNIVERSITÉ DE NANTES

- Types de répertoires possibles:
  - `DIRECTORY_MUSIC, DIRECTORY_PODCASTS, DIRECTORY_RINGTONES, DIRECTORY_ALARMS, DIRECTORY_NOTIFICATIONS, DIRECTORY_PICTURES, or DIRECTORY_MOVIES`

```
void deleteExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in the user's
    // public pictures directory and delete the file.  If external
    // storage is not currently mounted this will fail.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    if (path != null) {
        File file = new File(path, "DemoPicture.jpg");
        file.delete();
    }
}

boolean hasExternalStoragePrivatePicture() {
    // Create a path where we will place our picture in the user's
    // public pictures directory and check if the file exists.  If
    // external storage is not currently mounted this will think the
    // picture doesn't exist.
    File path = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
    if (path != null) {
        File file = new File(path, "DemoPicture.jpg");
        return file.exists();
    }
    return false;
}
```

External Storage : File Structure; API< 8

UNIVERSITÉ DE NANTES

- Directories:

`Music/` - Media scanner classifies all media found here as user music.
`Podcasts/` - Media scanner classifies all media found here as a podcast.
`Ringtones/` - Media scanner classifies all media found here as a ringtone.
`Alarms/` - Media scanner classifies all media found here as an alarm sound.
`Notifications/` - Media scanner classifies all media found here as a notification sound.
`Pictures/` - All photos (excluding those taken with the camera).
`Movies/` - All movies (excluding those taken with the camcorder).
`Download/` - Miscellaneous downloads.

SQLite Database (1)

UNIVERSITÉ DE NANTES

- access to SQLite database is restricted to the classes of your app;
- the class `SQLiteOpenHelper` eases the creation of the database.

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

SQLite Database (2)

UNIVERSITÉ DE NANTES

- `SQLiteOpenHelper.getWriteableDatabase()`:
  - write access to the database;
  - return an object `SQLiteData`
- `SQLiteOpenHelper.getReadableDatabase()`:
  - read access to the database;
  - return an object `SQLiteData`
- `SQLiteDatabase.query()`:
  - for request execution;
  - return an object `Query`
  - eases the construction of a `ContentProvider`).
- the `class SQLiteQueryBuilder`
  - for complex requests

SQLite sample (1)

UNIVERSITÉ DE NANTES

Open/create

- manual creation and connexion to the DB;
- data are stored in `/data/data/<application_name>/databases`

```
package higherpass.TestingData;

import android.app.Activity;
import android.os.Bundle;
import android.database.sqlite.SQLiteDatabase;

public class TestingData extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SQLiteDatabase db;
        db = openOrCreateDatabase(
            »»      "TestingData.db"
            »»      , SQLiteDatabase.CREATE_IF_NECESSARY
            »»      , null
          » »       );
    }
}
```

SQLite sample (2)

UNIVERSITÉ DE NANTES

-
-
-

```java
package higherpass.TestingData;

import java.util.Locale;

import android.app.Activity;
import android.os.Bundle;
import android.database.sqlite.SQLiteDatabase;

public class TestingData extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SQLiteDatabase db;

        db = openOrCreateDatabase(
        »       "TestingData.db"
        »       , SQLiteDatabase.CREATE_IF_NECESSARY
        »       , null
        »       );
        db.setVersion(1);
        db.setLocale(Locale.getDefault());
        db.setLockingEnabled(true);

        final String CREATE_TABLE_COUNTRIES =
        »       "CREATE TABLE tbl_countries ("
        »       + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
        »       + "country_name TEXT);";
        final String CREATE_TABLE_STATES =
        »       "CREATE TABLE tbl_states ("
        »       + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
        »       + "state_name TEXT,"
        »       + "country_id INTEGER NOT NULL CONSTRAINT "
        »       + "contry_id REFERENCES tbl_contries(id) "
        »       + "ON DELETE CASCADE);";
        db.execSQL(CREATE_TABLE_COUNTRIES);
        db.execSQL(CREATE_TABLE_STATES);
        final String CREATE_TRIGGER_STATES =
        »       "CREATE TRIGGER fk_insert_state BEFORE "
        »       + "INSERT on tbl_states"
        »       + "FOR EACH ROW "
        »       + "BEGIN "
        »       + "SELECT RAISE(ROLLBACK, 'insert on table "
        »       + ""tbl_states" voilates foreign key constraint "
        »       + ""fk_insert_state"') WHERE (SELECT id FROM "
        »       + "tbl_countries WHERE id = NEW.country_id) IS NULL; "
        »       + "END;";
        db.execSQL(CREATE_TRIGGER_STATES);
    }
}
```

Database creation using `execSQL()`

SQLite sample (3)

UNIVERSITÉ DE NANTES

- Entries definition
  - key/value mapping;
  - `contentValues` fields settings;
- Entries Insertion via method `insert()`

```
ContentValues values = new ContentValues();
values.put("country_name", "US");
long countryId = db.insert("tbl_countries", null, values);
ContentValues stateValues = new ContentValues();
stateValues.put("state_name", "Texas");
stateValues.put("country_id", Long.toString(countryId));
try {
    db.insertOrThrow("tbl_states", null, stateValues);
} catch (Exception e) {
    //catch code
}
```

SQLite sample (4)

UNIVERSITÉ DE NANTES

- Entries update:
  - key/value mapping within `contentValues`
  - using the `update()` method;

```
ContentValues updateCountry = new ContentValues();
updateCountry.put("country_name", "United States");
db.update("tbl_countries", updateCountry, "id=?", new String[] {Long.toString(countryId)});
```

- Entries deletion:
  - using the `delete()` method, together with a parameter specifying the conditions;

```
db.delete("tbl_states", "id=?", new String[] {Long.toString(countryId)});
```

SQLite Access Using Terminal

UNIVERSITÉ DE NANTES

- Android SDK allows access from :
  - emulator;
  - terminal.

sqlite3 from adb shell

```
bash-3.1$ /usr/local/android-sdk-linux/tools/adb devices
List of devices attached
emulator-5554   device
bash-3.1$ /usr/local/android-sdk-linux/tools/adb -s emulator-5554 shell
# ls /data/data/higherpass.TestingData/databases
TestingData.db
# sqlite3 /data/data/higherpass.TestingData/databases/TestingData.db
SQLite version 3.5.9
Enter ".help" for instructions
sqlite> .tables
android_metadata  tbl_countries     tbl_states

bash-3.1$ /usr/local/android-sdk-linux/tools/adb -s emulator-5554 shell
# sqlite3 /data/data/higherpass.TestingData/databases/TestingData.db
SQLite version 3.5.9
Enter ".help" for instructions
sqlite> select * FROM tbl_countries;
1|United States
```

Distant Storage

UNIVERSITÉ DE NANTES

- distant access to any ressource/web
  - using `java.net.* Classes`
  - using `android.net.* Classes`
- data saving using Backup services, using `BackupManager` (which is not a sync service).

ContentProvider (1)

UNIVERSITÉ DE NANTES

the Android mechanism for data sharing among apps;

- Every application (activity, service,...) can access only its own data
  - better security;
  - better modularity;
- the `contentProvider` is the only way to share data between apps.

ContentProvider (2)

UNIVERSITÉ DE NANTES

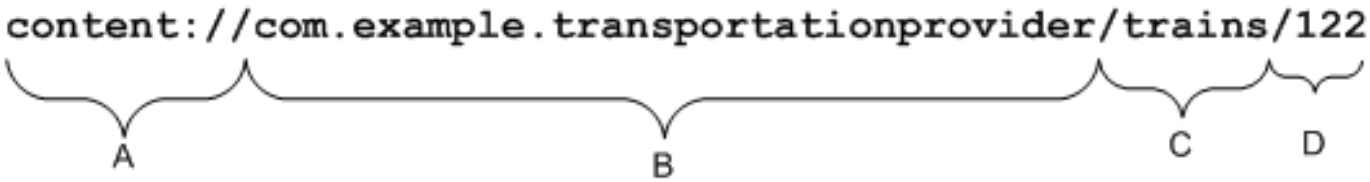
To share data:

- Android offers a vast set of existing `contentProviders`
  - used for specific type of data (audio,video,images...);
  - used with intents using the adapted rights (read/write);
- creation of your own `contentProviders` for specific needs:
  - by derivation of the `contentProviders` Class;
  - by addition of specific data to an existing `contentProviders`.

`contentProviders` Mechanism

UNIVERSITÉ DE NANTES

- Common interface shared by `contentProviders` to:
  - send requests and get results;
  - add/modify/delete data in the `provider`;
- every `contentProvider` has a public URI which specify the exposed data (accessible via an `Intent`)



  - A: standard prefix specifying that the data are controlled by a provider;
  - B: identification of the provider;
  - C: path to the data (e.g the table name);
  - D: identification of the unique entry.

`contentProviders` Mechanism(2)

UNIVERSITÉ DE NANTES

- the coder chooses the storage:
  - XML files, database, web service...
- he/she defines the kind of access and modification rights
- the access is then performed through a `contentResolver`

```
ContentResolver cr = getContentResolver();
```

contentProviders and contentResolvers

UNIVERSITÉ DE NANTES

- Android system :
  - identifies the `contentProvider` which handle the `URI` specified data;
  - runs the processus corresponding to the data
  - creates the objects of `ContentProvider` type.
- Only one instance of every `contentProviders`
- a `contentProvider` can communicate with several `contentResolvers`

Data Model

UNIVERSITÉ DE NANTES

- the data are available to the activities under a `table` type (whatever the internal structure)
  - a unique identifier by entry `_ID`, the table Key
  - a request return an object of type `Cursor` that permits displacement within the table;
    - from entry to entry;
    - from field to field.

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

- a single `contentProvider` may control several tables (one `_ID` per table)

`contentProvider` requests (1)

UNIVERSITÉ DE NANTES

- to build a request, you need
  - the `contentProvider URI`;
  - the field names to be received;
  - the field types to be received;
  - for a specific entry, the corresponding `_ID`
- the access is managed using :
  - `ContentResolver.query()`
  - `ContentResolver.managedQuery()`
- the `managedQuery()` methods handles the life cycle of the `Cursor` returned by the request;
  - freeing the `Cursor` when the activity is on pause;
  - re-requesting when the activity starts again.

`contentProvider` requests (2)

UNIVERSITÉ DE NANTES

Request code sample :

```
public final Cursor query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

- `URI` : the link to the `contentProvider`
- `projection` : a list of columns to be returned;
- `selection` : a filter on the entries (SQL `WHERE`)
- `selectionArgs` : values linked with `jokers` (the '?' char)
- `sortOrder` : sort parameters (SQL `ORDERBY`)
- sends back an object of type `Cursor` or the `null` object

URI format (1)

UNIVERSITÉ DE NANTES

public final Cursor **query** (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

- the URI defines a unique table
- when a `ContentProvider` is created, in order to avoid mistakes, a constant `String` identifying the table should be defined
  - `android.provider.Contacts.People.CONTENT_URI`
  - `android.provider.Contacts.Phones.CONTENT_URI`
  - `android.provider.Contacts.Photos.CONTENT_URI`
  - `android.provider.Contacts.Groups.CONTENT_URI`

URI format (2)

UNIVERSITÉ DE NANTES

public final Cursor **query** (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

- a specific entry can be reached by adding the corresponding `ID` to the `URI`
- methods can be used to construct the `URI`
  - `ContentUris.withAppendedId()`
  - `Uri.withAppendedPath()`

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI for the contact with _ID == 23.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

DataBase Projection

UNIVERSITÉ DE NANTES

```
public final Cursor query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

argument;

- specify the projection to filter the table (column names in a String vector);
- to ease the coding, `ContentProviders` give the Java constants which identify columns.

Constants	
String	AUTHORITY
int	KIND_EMAIL
int	KIND_IM
int	KIND_ORGANIZATION
int	KIND_PHONE
int	KIND_POSTAL

- Java constants which identify columns should be set at the `ContentProvider` creation

Request Sample (1)

UNIVERSITÉ DE NANTES

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
                              People._ID,
                              People._COUNT,
                              People.NAME,
                              People.NUMBER
                            };

// Get the base URI for the People table in the Contacts content provider.
Uri contacts =  People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
                          projection, // Which columns to return
                          null,       // Which rows to return (all rows)
                          null,       // Selection arguments (none)
                          // Put the results in ascending order by name
                          People.NAME + " ASC");
```

- performs a projection of the table `People.CONTENT_URI` in ascendant sort (`ASC`)

Request Sample (2)

UNIVERSITÉ DE NANTES

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

- to get a `Cursor` pointing on the first ID (_ID = 13)

```
Cursor cur = getContentResoler()
.query("content://android.provider
.Contacts.Phones.CONTENT_URI")
```

- to get directly a `Cursor` pointing on the entry with _ID = 44

```
Cursor cur = getContentResoler()
.query("content://android.provider
.Contacts.Phones.CONTENT_URI/44")
```

Request Sample (3)

UNIVERSITÉ DE NANTES

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

- to get a `Cursor` on the table containing only the fields `_ID, NUMBER, LABEL`

```
Cursor cur =
getContentResoler().query("content://android.provider
.Contacts.Phones.CONTENT_URI", new String[]
{Data._ID, Phone.NUMBER, Phone.LABEL)}
```

Request Sample (4)

UNIVERSITÉ DE NANTES

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

- to get a `Cursor` on the table containing only the fields `_ID, NUMBER, LABEL` and the entries which label is equal to `myLabel`

```
Cursor cur =
getContentResoler().query("content://android.provider
.Contacts.Phones.CONTENT_URI", new String[]
Data._ID, Phone.NUMBER, Phone.LABEL,
Phone.LABEL + "=?", new String[]
{String.valueOf(myLabel)})
```

Data Access : Cursor Type

UNIVERSITÉ DE NANTES

- Data access
  - entries
    - `Cusor.moveToFirst()`
    - `Cursor.moveToNext()`
  - columns data
    - `Cursor.getColumnIndex()` return the column ID;
    - `Cursor.getString()`
    - `Cursor.getInt()`
    - `Cursor.getFloat()` return the corresponding data
- sample code
  - API 1.6 and API 2.0 (access are different)
  - access to `ContentProvider` `android.provider.Contacts`
  - display of the identifier, name, phone number, mail address..

V1.6 Content Provider

UNIVERSITÉ DE NANTES

Access to contacts:

- iterate through entries using `Cursor.moveToNext()`

```
package TestContacts;

import android.app.Activity;
import android.content.ContentResolver;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.Contacts;
import android.provider.Contacts.People;

public class TestContacts extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ContentResolver cr = getContentResolver();
        Cursor cur = cr.query(People.CONTENT_URI,
                null, null, null, null);
        if (cur.getCount() > 0) {
            while (cur.moveToNext()) {
                String id = cur.getString(cur.getColumnIndex(People._ID));
                String name = cur.getString(cur.getColumnIndex(People.DISPLAY_NAME));
            }
        }
    }
}
```

V1.6 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts phone number:

- phone numbers are stored as index in the table `Contacts.People.CONTENT_URI (PRIMARY_PHONE_ID)`
- index are pointing to a separated table `(Contacts.Phones.CONTENT_URI)` which could contain several phone entries

```
if (Integer.parseInt(cur.getString(
    cur.getColumnIndex(People.PRIMARY_PHONE_ID))) > 0) {
        Cursor pCur = cr.query(
                Contacts.Phones.CONTENT_URI,
                null,
                Contacts.Phones.PERSON_ID +" = ?",
                new String[]{id}, null);
        int i=0;
        int pCount = pCur.getCount();
        String[] phoneNum = new String[pCount];
        String[] phoneType = new String[pCount];
        while (pCur.moveToNext()) {
            phoneNum[i] = pCur.getString(
                    pCur.getColumnIndex(Contacts.Phones.NUMBER));
            phoneType[i] = pCur.getString(
                    pCur.getColumnIndex(Contacts.Phones.TYPE));
            i++;
        }
}
```

V1.6 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts emails:

- mail addresses are stored as index in the table `Contacts.People.CONTENT_URI`
- index are pointing to a separated table `Contacts.ContactMethods.CONTENT_EMAIL_URI` which could contain several mail entries

```
Cursor emailCur = cr.query(
»       »       Contacts.ContactMethods.CONTENT_EMAIL_URI,
»       »       null,
»       »       Contacts.ContactMethods.PERSON_ID + " = ?",
»       »       new String[]{id}, null);
while (emailCur.moveToNext()) {
    // permet l'accès à tous les emails
}
emailCur.close();
```

V1.6 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts postal addresses:

- a statement should be added to the request `Contact.ContactMethods.KIND = Contacts.ContactMethods.CONTENT_POSTAL_ITEM_TYPE`
- this statement restrains the entries to the postal addresses within the table `Contacts.ContactMethods.CONTENT_URI_URI` which could contain several mail entries

```
String addrWhere = Contacts.ContactMethods.PERSON_ID
        + " = ? AND " + Contacts.ContactMethods.KIND + " = ?";
String[] addrWhereParams = new String[]{id,
        Contacts.ContactMethods.CONTENT_POSTAL_ITEM_TYPE};
Cursor addrCur = cr.query(Contacts.ContactMethods.CONTENT_URI,
        null, addrWhere, addrWhereParams, null);
while(addrCur.moveToNext()) {
        String addr = addrCur.getString(
           addrCur.getColumnIndex(Contacts.ContactMethodsColumns.DATA));
        String type = addrCur.getString(
           addrCur.getColumnIndex(Contacts.ContactMethodsColumns.TYPE));
}
addrCur.close();
```

V1.6 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts organizations:

- data are stored in `Contacts.Organizations.CONTENT_URI`

```
String orgWhere = Contacts.ContactMethods.PERSON_ID + " = ?";
String[] orgWhereParams = new String[]{id};
Cursor orgCur = cr.query(Contacts.Organizations.CONTENT_URI,
      null, orgWhere, orgWhereParams, null);
if (orgCur.moveToFirst()) {
»       String orgName = orgCur.getString(
           orgCur.getColumnIndex(Contacts.Organizations.COMPANY));
»       String title = orgCur.getString(
           orgCur.getColumnIndex(Contacts.Organizations.TITLE));
}
orgCur.close();
```

- add permissions in the `AndroidManifest.xml`

```
<uses-permission
android:name="android.permission.READ_CONTACTS"
/>
```

V2.0 Content Provider

UNIVERSITÉ DE NANTES

Access to contacts:

- iterate through entries using `Cursor.moveToNext()`
- data are stored in `ContactsContract.Contacts.CONTENT_URI`

```
package TestContacts;

import android.app.Activity;
import android.content.ContentResolver;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.ContactsContract;

public class TestContacts extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ContentResolver cr = getContentResolver();
        Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                null, null, null, null);
        if (cur.getCount() > 0) {
            while (cur.moveToNext()) {
                String id = cur.getString(
                        cur.getColumnIndex(ContactsContract.Contacts._ID));
                String name = cur.getString(
                        cur.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                if (Integer.parseInt(cur.getString(cur.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))) > 0) {
                    //Interrogations ultérieures
                }
            }
        }
    }
}
```

V2.0 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts phone number:

- access to a separated table which could contain several phone entries with the contact ID

```
if (Integer.parseInt(cur.getString(
        cur.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))) > 0) {
    Cursor pCur = cr.query(
        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
        null,
        ContactsContract.CommonDataKinds.Phone.CONTACT_ID +" = ?",
        new String[]{id}, null);
    while (pCur.moveToNext()) {
        // Do something with phones
    }
    pCur.close();
}
```

V2.0 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts emails:

- separated table `ContactsContract.CommonDataKinds.Email.CONTENT_U`
- index are pointing to a separated table which could contain several email entries

```
Cursor emailCur = cr.query(
        ContactsContract.CommonDataKinds.Email.CONTENT_URI,
        null,
        ContactsContract.CommonDataKinds.Email.CONTACT_ID + " = ?",
        new String[]{id}, null);
while (emailCur.moveToNext()) {
    // This would allow you get several email addresses
    // if the email addresses were stored in an array
    String email = emailCur.getString(
              emailCur.getColumnIndex(ContactsContract.CommonDataKinds.Email.DATA));
    String emailType = emailCur.getString(
              emailCur.getColumnIndex(ContactsContract.CommonDataKinds.Email.TYPE));
}
emailCur.close();
```

V2.0 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts postal addresses:

- a statement should be added to the request
  `ContactsContract.Data.MIMETYPE = ContactsContract.CommonDataKinds.StructuredPosta .CONTENT_ITEM_TYPE`

```
String addrWhere = ContactsContract.Data.CONTACT_ID + " = ? AND " + ContactsContract.Data.MIMETYPE + " = ?";
String[] addrWhereParams = new String[]{id,
        ContactsContract.CommonDataKinds.StructuredPostal.CONTENT_ITEM_TYPE};
Cursor addrCur = cr.query(ContactsContract.Data.CONTENT_URI,
        null, where, whereParameters, null);
while(addrCur.moveToNext()) {
        String poBox = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.POBOX));
        String street = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.STREET));
        String city = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.CITY));
        String state = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.REGION));
        String postalCode = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.POSTCODE));
        String country = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.COUNTRY));
        String type = addrCur.getString(
                addrCur.getColumnIndex(ContactsContract.CommonDataKinds.StructuredPostal.TYPE));
}
addrCur.close();
```

V2.0 Contact Content Provider

UNIVERSITÉ DE NANTES

Access to contacts organizations:

- data are stored in `ContactsContract.Data.CONTENT_URI`
- add permissions in the `AndroidManifest.xml`
  `<uses-permission android:name="android.permission.READ_CONTACTS" />`

```
String orgWhere = ContactsContract.Data.CONTACT_ID + " = ? AND " + ContactsContract.Data.MIMETYPE + " = ?";
String[] orgWhereParams = new String[]{id,
        ContactsContract.CommonDataKinds.Organization.CONTENT_ITEM_TYPE};
Cursor orgCur = cr.query(ContactsContract.Data.CONTENT_URI,
        null, orgWhere, orgWhereParams, null);
if (orgCur.moveToFirst()) {
        String orgName = orgCur.getString(orgCur.getColumnIndex(ContactsContract.CommonDataKinds.Organization.DATA));
        String title = orgCur.getString(orgCur.getColumnIndex(ContactsContract.CommonDataKinds.Organization.TITLE));
}
orgCur.close();
```

CP : Adding Entries

UNIVERSITÉ DE NANTES

Inserting an entry in a table:

- mapping key/value is used;
- key for column, value for the corresponding data;
- call to the method `ContentResolver.insert()`

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

CP : Deleting Entries

UNIVERSITÉ DE NANTES

Deleting an entry:`ContentResolver.delete()`

```
public final int delete (Uri url, String where, String[] selectionArgs)   Since: API Level 1
```

- through an URI and a specific ID. Only this entry is deleted;
- through an URI and a SQL `where` statement and a `selectionArgs` which specify the entries to be deleted.

MediaStore Example (1)

UNIVERSITÉ DE NANTES

The mediaStore allows the storage of various data relating to Images, Audio and Video

```
import android.provider.MediaStore.Images.Media;
import android.content.ContentValues;
import java.io.OutputStream;

// Save the name and description of an image in a ContentValues map.
ContentValues values = new ContentValues(3);
values.put(Media.DISPLAY_NAME, "road_trip_1");
values.put(Media.DESCRIPTION, "Day 1, trip to Los Angeles");
values.put(Media.MIME_TYPE, "image/jpeg");

// Add a new record without the bitmap, but with the values just set.
// insert() returns the URI of the new record.
Uri uri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, values);

// Now get a handle to the file for that record, and save the data into it.
// Here, sourceBitmap is a Bitmap object representing the file to save to the database.
try {
    OutputStream outStream = getContentResolver().openOutputStream(uri);
    sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
    outStream.close();
} catch (Exception e) {
    Log.e(TAG, "exception while writing image", e);
}
```

ContentProvider HowTo (1)

UNIVERSITÉ DE NANTES

Three steps:

1. create the data storage (file/database/web service)
2. extends the class `ContentProvider` to provide access to the data;
3. declare the `provider` in the file `AndroidManifest.xml`

```
<provider name="com.example.autos.AutoInfoProvider"
          authorities="com.example.autos.autoinfoprovider"
          . . . />
</provider>
```

- `name` for the derived class;
- `authorities` for the identification of the `ContentProvider` towards the `ContentResolver`

ContentProvider HowTo (2)

UNIVERSITÉ DE NANTES

- To extend the `ContentProvider` class, the 6 following abstract methods should be implemented;
  - `query()`: should return a `Cursor`. Some cursors are allready avalaible (MatrixCursor, MergeCursor, SQLiteCursor). For testing purposes, a MockCursor can be used;
  - `insert()`
  - `update()`
  - `delete()`
  - `getType()`
  - `onCreate()`
- because of the possible concurrent accesses, a thread-safe implementation is required
- `ContentResolver.notifyChange()` is used to get notifications of data, if listeners are avalaible.

ContentProvider HowTo (3)

UNIVERSITÉ DE NANTES

Ease the future extensions using :

- class constants to name the tables;
- class constants to name the fields (columns)

```
public static final String PROVIDER_NAME = "com.example.library.libraryprovider";
public static final Uri CONTENT_URI = Uri.parse("content://"+ PROVIDER_NAME + "/books");


public static final String _ID = "_ID";
public static final String BOOK_TITLE = "title";
public static final String BOOK_AUTHOR = "author";
public static final String BOOK_YEAR = "year";
```

Another Sample

UNIVERSITÉ DE NANTES

- `ContentProvider` on a simple example : books
- shows how to declare the `ContentProvider` and the way to access it from another activity;
- for this example, data are created within the code and accessed through a `MatrixCursor`

LibraryContentProvider (1)

UNIVERSITÉ DE NANTES

```
package com.example.library.provider;

import android.content.ContentProvider;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;
import android.provider.BaseColumns;

public class LibraryContentProvider extends ContentProvider {

	public static final String PROVIDER_NAME = "com.example.library.provider.Libraryprovider";
	public static final Uri CONTENT_URI = Uri.parse("content://"+ PROVIDER_NAME + "/books");

	public static final String _ID = "_ID";
	public static final String BOOK_TITLE = "title";
	public static final String BOOK_AUTHOR = "author";
	public static final String BOOK_YEAR = "year";


	// This must be the same as what as specified as the Content Provider authority
	// in the manifest file.
	static final String AUTHORITY = "com.example.library.libraryprovider";

	// sample data to show the ContentProvider principle
	public class Book {
		public String title;
		public String author;
		public String year;
	}
```

LibraryContentProvider (2)

UNIVERSITÉ DE NANTES

```
@Override
public boolean onCreate() {
»       // initialiser des données ici (on simule l'existence d'une BD)
»
»       Book data[] = new Book[3];

»       data[0].title = "Le viel homme et la mer";
»       data[0].author = "E. Hemmingway";
»       data[0].year = "1951";

»       data[1].title = "Les travailleurs de la mer";
»       data[1].author = "V. Hugo";
»       data[1].year = "1866";
»
»       data[2].title = "Moby-Dick";
»       data[2].author = "H. Melville";
»       data[2].year = "1851";


        return true;
}
```

LibraryContentProvider (3)

UNIVERSITÉ DE NANTES

```
@Override
public String getType(Uri uri) {
  // create a new MIME type "com.books" for the values which are returned
  return ContentResolver.CURSOR_DIR_BASE_TYPE + '/' + "com.books";
}

@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {


                MatrixCursor c = new MatrixCursor(new String[] {
                                _ID,
                                BOOK_TITLE,
                                BOOK_AUTHOR,
»       »       »       »       BOOK_YEAR
                });

                int row_index = 0;

                // Add x-axis data
                for (int i=0; i< data.length(); i++) {


                        c.newRow()
                        .add( row_index )
                        .add( data[row_index].title )
                        .add( data[row_index].author )   // Only create data for the first series.
                        .add( data[row_index].year );

                        row_index++;
                }

                return c;
        }
}
```

LibraryContentProvider (4)

UNIVERSITÉ DE NANTES

```java
@Override
public int update(Uri uri, ContentValues contentvalues, String s, String[] as) {
        throw new UnsupportedOperationException(" To be implemented");
}


@Override
public Uri insert(Uri uri, ContentValues contentvalues) {
        throw new UnsupportedOperationException(" To be implemented");
}

@Override
public int delete(Uri uri, String s, String[] as) {
        throw new UnsupportedOperationException(" To be implemented");
}
```

Access to the LibraryContentProvider

UNIVERSITÉ DE NANTES

```
// pour utiliser le Content Provider

// 1) on obtient l'URI définie dans LibraryContentProvider

Uri allBooks = LibraryContentProvider.CONTENT_URI;

// 2) on effectue la requete
Cursor c = getContentResolver().query(allBooks, null, null, null, null );

// 3) et on traite les résultats
if  (c.moveToFirst()) {
      do{

        // we get the data
        String s = c.getString(c.getColumnIndex(
            LibraryContentProvider._ID)) + ", " +
            c.getString(c.getColumnIndex(
               LibraryContentProvider.BOOK_TITLE)) + ", " +
            c.getString(c.getColumnIndex(
               LibraryContentProvider.BOOK_AUTHOR)) + ", " +
            c.getString(c.getColumnIndex(
               LibraryContentProvider.BOOK_YEAR));

      } while (c.moveToNext());
}
```

Data on Android: Conclusion

UNIVERSITÉ DE NANTES

- Structured data;
- easy access to various kind of data
  - own ressources
  - xml
  - SQLite...
- access rights are very clear;
- access control given to the applications.
  - an application manages its data;
  - data sharing are controlled by a `ContentProvider`.