Développement Android (4.3)

Data Access & Persistence

BENJAMIN BILLET

WARNING

Le contenu de cette présentation est basé sur la documentation anglophone officielle d'Android, diffusée sous licence *Creative Commons Attribution 2.5* :

developer.android.com

La plupart des schémas qui composent ce cours proviennent de cette documentation et sont, par conséquent, soumis à cette même licence.

http://creativecommons.org/licenses/by/2.5/

Au Sommaire !

- Content Provider
- Loader
- Storage & Files
- SQLite

CONTENT PROVIDER

Content provider

- Un Content Provider présente les données sous forme de table, comme une petite base de données relationnelle.
- Un Content Provider peut être ouvert aux autres applications.
- Le système embarque un grand nombre de Content Provider (contacts, appels, galerie, dictionnaire, etc.).
- Ceux-ci peuvent être utilisés directement ou au travers d'intents, ce qui évite d'avoir à déclarer les droits correspondants.

Content resolver

- L'accès aux données se fait au travers d'un Content Resolver qui sert d'interface avec le Content Provider (on peut voir ça comme une approche client-serveur).
- Le Content Resolver possède les méthodes CRUD usuelles.
- Les résultats fournis par le Content Resolver sont présentés sous forme de Cursor, une structure de donnée qui peut être parcourue ou fournie à un CursorAdapter pour l'affichage.

Content resolver

```
getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,          // The content URI of the words table
    projectionClause,                          // The columns to return for each row
    selectionClause                            // Selection criteria
    selectionArgs,                             // Selection criteria
    sortOrder                                  // The sort order for the returned rows
);
```

query() argument	SELECT keyword/parameter	
Uri	FROM table_path	De la forme content://[authority]/[table_path] Ex : user_dictionary/words
projection	col,col,col,...	Un tableau précisant les colonnes à récupérer
selection	WHERE col = value	Les critères de sélection
selectionArgs	-	Remplace les ? de la clause de sélection par des valeurs réelles (similaire aux requêtes préparées).
sortOrder	ORDER BY col,col,...	La clause de tri, sous forme de chaîne (identique à un tri SQL).

Remarque : On peut simplement récupérer un élément en ajoutant son ID (s'il en possède) à la fin de l'URI :

```
Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI, 4);
```

CONTENT RESOLVER

```
String word = "something";
String selection = null;
String selectionArgs = new String[] { "" };

if (TextUtils.isEmpty(word) == false)
{
    selection = UserDictionary.Words.WORD + " = ?"; // protecting against SQL injections
    selectionArgs[0] = word;
}

String[] projection = new String[] { UserDictionary.Words.WORD };
String sort = UserDictionary.Words.WORD + " desc";

ContentResolver resolver = getContentResolver();
Cursor cursor = resolver.query(UserDictionary.Words.CONTENT_URI, projection, selection, selectionArgs, sort);

if (null == cursor) // some providers return null if an error occurs, others throw an exception
{
    ...
}
else if (cursor.getCount() < 1) // nothing found
{
    ...
}
else
{
    int index = cursor.getColumnIndex(UserDictionary.Words.WORD);
    while (cursor.moveToNext())
    {
        String myWord = cursor.getString(index);
    }
}
```

Remarque : Ne pas oublier de compléter le manifest !
android.permission.READ_USER_DICTIONARY!

CursorAdapter

```
String[] projection =
{
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};

int[] wordItems = { R.id.word, R.id.locale};

cursorAdapter = new SimpleCursorAdapter(
        getApplicationContext(),  // The application's Context object
        R.layout.wordlistrow,     // A layout in XML for one row in the ListView
        cursor,                   // The result from the query
        projection,               // A string array of column names in the cursor
        wordItems,                // An integer array of view IDs in the row layout
        0                         // Flags (usually none are needed)
);

myListView.setAdapter(cursorAdapter);
```

Insert

```
ContentValues values = new ContentValues();
values.put(UserDictionary.Words.APP_ID, "example.user");
values.put(UserDictionary.Words.LOCALE, "en_US");
values.put(UserDictionary.Words.WORD, "insert");
values.put(UserDictionary.Words.FREQUENCY, "100");

Uri newUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // the user dictionary content URI
    values                           // the values to insert
);

// newUri => content://user_dictionary/words/<id_of_the_new_item>
long id = ContentUris.parseId(newUri);
```

UPDATE

```
ContentValues values = new ContentValues();
values.putNull(UserDictionary.Words.LOCALE);

String selection = UserDictionary.Words.LOCALE + " LIKE ?";
String[] selectionArgs = { "en_%" };

int nbRowsUpdated = 0;
nbRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,    // the user dictionary content URI
    values                               // the columns to update
    selection                            // the column to select on
    selectionArgs                        // the value to compare to
);
```

Delete

```
String selection = UserDictionary.Words.APP_ID + " LIKE ?";
String[] selectionArgs = { "user" };

int nbRowsDeleted = 0;
nbRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,    // the user dictionary content URI
    selection                            // the column to select on
    selectionArgs                        // the value to compare to
);
```

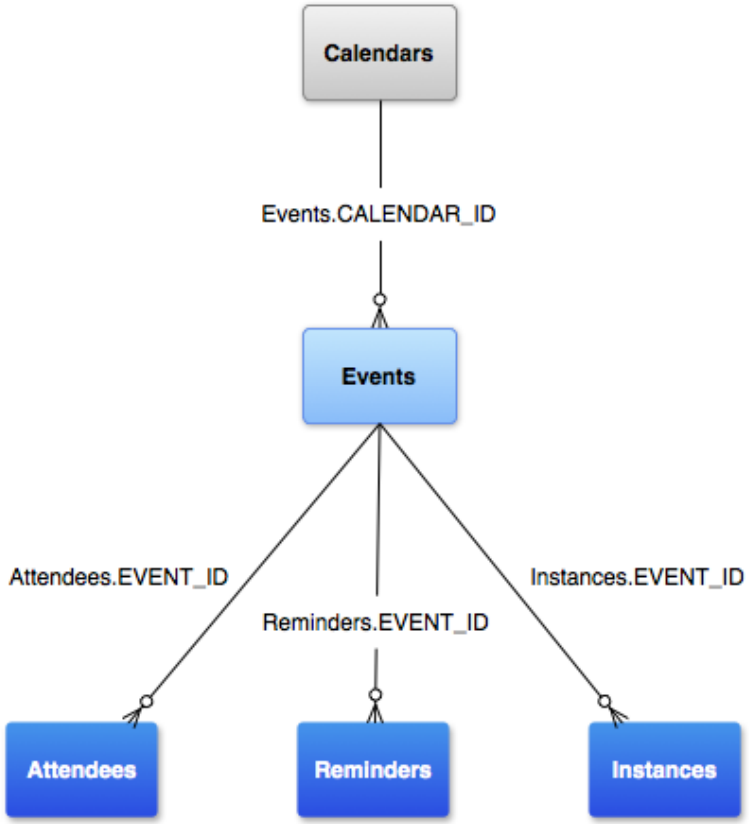
Batch

```
List<ContentProviderOperation> ops = new ArrayList<ContentProviderOperation>();

ops.add(ContentProviderOperation.newInsert(RawContacts.CONTENT_URI)
          .withValue(RawContacts.ACCOUNT_TYPE, accountType)
          .withValue(RawContacts.ACCOUNT_NAME, accountName)
          .build());

ops.add(ContentProviderOperation.newInsert(Data.CONTENT_URI)
          .withValueBackReference(Data.RAW_CONTACT_ID, 0) // id of the item ops[0]
          .withValue(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE)
          .withValue(StructuredName.DISPLAY_NAME, "Mike Sullivan")
          .build());

getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);
```

- withValueBackReference() indique l'index de l'opération qui fournit l'id nécessaire (ici, nous avons besoin de l'id de l'item RawContact pour construire l'item Data).
- A noter qu'il existe aussi bulkInsert() qui effectue des insertions en masse dans une table unique (plus performant).

Calendar Provider

- Calendars : les calendriers
  - nom, couleur.
- Events : les évènements
  - titre, lieu, début/fin.
- Attendees : les participants
  - type, présence.
- Reminders : les alertes
  - type, temps.
- Instances : début et fin de chaque instance d'évènement.




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

Calendar Provider

```
// find calendars
String[] projection = new String[]
{
    Calendars._ID,
    Calendars.ACCOUNT_NAME,
    Calendars.CALENDAR_DISPLAY_NAME,
    Calendars.OWNER_ACCOUNT
};

Uri uri = Calendars.CONTENT_URI;
String selection = "((" + Calendars.ACCOUNT_NAME + " = ?) AND (" // calendars visible by the user
                        + Calendars.ACCOUNT_TYPE + " = ?) AND ("
                        + Calendars.OWNER_ACCOUNT + " = ?))";    // calendars owned by the user

String[] selectionArgs = new String[] { "sampleuser@gmail.com", "com.google", "sampleuser@gmail.com" };
Cursor c = getContentResolver().query(uri, projection, selection, selectionArgs, null);

// update a calendar
ContentValues values = new ContentValues();
values.put(Calendars.CALENDAR_DISPLAY_NAME, "My Calendar");

long id = 2;
Uri updateUri = ContentUris.withAppendedId(Calendars.CONTENT_URI, id);
int nbUpdatedRows = getContentResolver().update(updateUri, values, null, null);
```

- ACCOUNT_TYPE est obligatoire dans le cas de requêtes basées sur l'ACCOUNT_NAME. En effet, plusieurs types de compte peuvent avoir un même nom (e.g., un compte google et un compte hotmail avec la même adresse mail).

Calendar Provider

```
// create event
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 9, 14, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 9, 14, 8, 45);

ContentValues values = new ContentValues();
values.put(Events.DTSTART, beginTime.getTimeInMillis());
values.put(Events.DTEND, endTime.getTimeInMillis());
values.put(Events.TITLE, "My Event");
values.put(Events.DESCRIPTION, "It’s an incredible event !");
values.put(Events.CALENDAR_ID, 42);
values.put(Events.EVENT_TIMEZONE, "France/Paris");
Uri uri = getContentResolver().insert(Events.CONTENT_URI, values);

// update event
ContentValues values = new ContentValues();
values.put(Events.TITLE, "Kickboxing");
Uri updateUri = ContentUris.withAppendedId(Events.CONTENT_URI, 42);
int nbRowsUpdated = getContentResolver().update(updateUri, values, null, null);

// delete event
Uri deleteUri = ContentUris.withAppendedId(Events.CONTENT_URI, 42);
int nbRowsUpdated = getContentResolver().delete(deleteUri, null, null);
```

Calendar Provider

```
long eventID = 42;

// add an attendee
ContentValues values = new ContentValues();
values.put(Attendees.ATTENDEE_NAME, "Trevor");
values.put(Attendees.ATTENDEE_EMAIL, "trevor@example.com");
values.put(Attendees.ATTENDEE_RELATIONSHIP, Attendees.RELATIONSHIP_ATTENDEE);
values.put(Attendees.ATTENDEE_TYPE, Attendees.TYPE_OPTIONAL);
values.put(Attendees.ATTENDEE_STATUS, Attendees.ATTENDEE_STATUS_INVITED);
values.put(Attendees.EVENT_ID, eventID);
Uri uri = getContentResolver().insert(Attendees.CONTENT_URI, values);

// add a reminder
ContentValues values = new ContentValues();
values.put(Reminders.MINUTES, 15);
values.put(Reminders.EVENT_ID, eventID);
values.put(Reminders.METHOD, Reminders.METHOD_ALERT);
Uri uri = getContentResolver().insert(Reminders.CONTENT_URI, values);
```

Calendar Provider

```
// get instances
String[] projection = new String[] { Instances.EVENT_ID, Instances.BEGIN, Instances.TITLE };

Calendar beginTime = Calendar.getInstance();
beginTime.set(2011, 9, 23, 8, 0);
Calendar endTime = Calendar.getInstance();
endTime.set(2011, 10, 24, 8, 0);

String selection = Instances.EVENT_ID + " = ?";
String[] selectionArgs = new String[] { "42" };

Uri.Builder builder = Instances.CONTENT_URI.buildUpon();
ContentUris.appendId(builder, beginTime.getTimeInMillis());
ContentUris.appendId(builder, endTime.getTimeInMillis());

Cursor cur = getContentResolver().query(builder.build(), projection, selection, selectionArgs, null);
```

Calendar Intents

- Simplifie la gestion des calendriers et utilise les activités du système :
  - Pas besoin de déclarer les droits dans le manifest.
  - L'utilisateur n'est pas perturbé par une GUI différente.
- Les paramètres sont transmis en tant qu'extras.

Action	URI	Description
VIEW	content://com.android.calendar/time/<time_ms>	Ouvre le calendrier au temps spécifié
VIEW	content://com.android.calendar/events/<event_id>	Affiche un évènement.
EDIT	content://com.android.calendar/events/<event_id>	Modifie un évènement
EDIT, INSERT	content://com.android.calendar/events	Crée un évènement

Calendar Intents

```
// open the calendar at a date
Uri.Builder builder = CalendarContract.CONTENT_URI.buildUpon();
builder.appendPath("time");
ContentUris.appendId(builder, somethingInMilliseconds);

Intent intent = new Intent(Intent.ACTION_VIEW).setData(builder.build());
startActivity(intent);

// display an event
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, 42);
Intent intent = new Intent(Intent.ACTION_VIEW).setData(uri);
startActivity(intent);

// create an event
Calendar beginTime = Calendar.getInstance();
beginTime.set(2012, 0, 19, 7, 30);
Calendar endTime = Calendar.getInstance();
endTime.set(2012, 0, 19, 8, 30);
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, "Yoga")
        .putExtra(Events.DESCRIPTION, "Group class")
        .putExtra(Events.EVENT_LOCATION, "The gym")
        .putExtra(Events.AVAILABILITY, Events.AVAILABILITY_BUSY)
        .putExtra(Intent.EXTRA_EMAIL, "nicolas@inria.fr,françoise@inria.fr");
startActivity(intent);

// update an event
Uri uri = ContentUris.withAppendedId(Events.CONTENT_URI, 42);
Intent intent = new Intent(Intent.ACTION_EDIT)
        .setData(uri)
        .putExtra(Events.TITLE, "My New Title");
startActivity(intent);
```

Contact Provider

Créer son Content Provider

- Utile pour présenter des données complexes ou des fichiers aux autres applications.
- Utile pour permettre aux utilisateurs de travailler sur ces données depuis d'autres applications.
- Utile pour créer des suggestions de recherche personnalisées.

Créer son Content Provider

- Les tables sont identifiées par une URI, de la forme content://authority/path.
- Les données aussi : content://authority/path/id.
- L'authority correspond au nom donné au provider (généralement le package, pour éviter les conflits).
- Le path correspond à un nom de table, qui n'est pas forcément composé d'un seul élément :
  - content://com.example.app.provider/table1 => table1.
  - content://com.example.app.provider/table2/dataset1 => dataset1.
  - content://com.example.app.provider/table2/dataset2 => dataset2.
  - content://com.example.app.provider/table3 => table3.

Créer son Content Provider

- Toutes les tables doivent avoir une clé primaire, de préférence nommée BaseColumns._ID.
- Pour les données binaires de taille moyenne, il est possible d'utiliser le type BLOB pour les stocker directement dans la table sous forme de tableau d'octets.
- Pour les données plus lourdes (images, musique, vidéo, etc.), privilégier les fichiers.

Créer son Content Provider

- Six méthodes abstraites à implémenter :
  - query() : récupère des données (retourne un Cursor).
  - insert() : insère un tuple (retourne une URI).
  - update() : modifie un tuple (retourne le nombre de lignes mises à jour).
  - delete() : supprime un tuple (retourne le nombre de suppressions).
  - getType() : retourne le type MIMe d'une URI.
  - onCreate() : initialise le provider, invoquée lorsqu'un ContentResolver essayer d'accéder au provider.
- Attention : Un Content Provider est susceptible d'être utilisé de manière concurrentielle !
- Si l'on désire interdire les insertions, les updates ou les deletes, les méthodes respectives doivent simplement retourner 0.

Créer son Content Provider

- Un provider peut aussi manipuler des fichiers, auquel cas getStreamTypes() et openFile() doivent aussi être implémentées.

```
@Override
public ParcelFileDescriptor openFile(Uri uri, String mode)
throws FileNotFoundException
{
    File f=new File(getContext().getFilesDir(), uri.getPath());

    if (f.exists())
        return(ParcelFileDescriptor.open(f, ParcelFileDescriptor.MODE_READ_ONLY));

    throw new FileNotFoundException(uri.getPath());
}
```

Créer son Content Provider

```
<provider
    android:name=".MyProvider"
    android:authorities="fr.inria.my_provider"
    android:exported="true"
    android:initOrder="5"
    android:permission="fr.inria.permissions.PERM"
    android:readPermission="fr.inria.permissions.READ"
    android:writePermission="fr.inria.permissions.WRITE"
/>
```

- android:readPermission et android:writePermission sont prioritaires sur android:permission.
- android:initOrder définit l'ordre de démarrage des providers (les plus élevés démarrent en premier).
- android:authorities identifie le provider au sein du système.
- android:icon et android:label peuvent être utilisés par le système pour afficher un titre et une icône dans *Paramètres > Applications > Toutes*

Créer son Content Provider

- Le matching d'URI peut être effectuée au moyen de la classe utilitaire UriMatcher, qui supporte certains symboles classiques :
  - ***** : une chaîne de caractères valides.
  - **#** : une chaîne de chiffres.

```
UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
matcher.addURI("com.example.app.provider", "table3", 1);
matcher.addURI("com.example.app.provider", "table3/#", 2);
switch (matcher.match(uri))
{
    case 1:
        ...
        break;
    case 2:
        selection = selection + "_ID = " + uri.getLastPathSegment();
        break;
    default:
        ...
}
```

LOADER

Loader

- Utilisable par les activités et les fragments pour charger des données de manière asynchrone.
- Monitore la source de données et présente automatiquement les nouveaux résultats.
- Chaque activité ou fragment possède un LoaderManager qui gère les loaders utilisés.
- CursorLoader est un loader prêt à l'emploi pour charger des données depuis un Content Provider.

LOADER

```
public class MyActivity extends Activity implements LoaderManager.LoaderCallbacks<T>
{
     @Override
     protected void onCreate(Bundle savedInstanceState)
     {
          super.onCreate(savedInstanceState);
          getLoaderManager().initLoader(42, null, this);
     }

     @Override
     public Loader<T> onCreateLoader(int id, Bundle args)
     {
          return ...
     }

     @Override
     public void onLoadFinished(Loader<T> loader, T data)
     {
     }

     @Override
     public void onLoaderReset(Loader<T> loader)
     {
     }
}
```

- Si le loader existe déjà sous cet ID, il est réutilisé.
- Sinon, onCreateLoader() est invoquée pour construire le loader.
- Si l'on désire toutefois forcer le redémarrage du loader :

```
getLoaderManager().restartLoader(0, null, this);
```

LOADER

```
public class MyActivity extends Activity implements LoaderManager.LoaderCallbacks<Cursor>
{
	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		getLoaderManager().initLoader(42, null, this);
	}

	@Override
	public Loader<Cursor> onCreateLoader(int id, Bundle args)
	{
		return new CursorLoader(
				this,		// Parent activity context
				dataUrl,	// Table to query
				projection,	// Projection to return
				null,		// No selection clause
				null,		// No selection arguments
				null		// Default sort order
		);
	}

	@Override
	public void onLoadFinished(Loader<Cursor> loader, Cursor data)
	{
		...
	}

	@Override
	public void onLoaderReset(Loader<Cursor> loader)
	{
		...
	}
}
```

LOADER

```
public class MyActivity extends Activity implements LoaderManager.LoaderCallbacks<Cursor>
{
	private SimpleCursorAdapter adapter;

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

		String[] fromColumns = { "avatar", "pseudo" };
		int[] toFields = { R.id.avatar, R.id.pseudo };

		adapter = new SimpleCursorAdapter(
			this,                // Current context
			R.layout.list_item,  // Layout for a single row
			null,                // No Cursor yet
			fromColumns,         // Cursor columns to use
			toFields,            // Layout fields to use
			0                    // No flags
		);

		ListView listView = (ListView) findViewById(R.id.myList);
		listView.setAdapter(adapter);
		getLoaderManager().initLoader(42, null, this);
	}

	@Override
	public void onLoadFinished(Loader<Cursor> loader, Cursor data)
	{
		adapter.changeCursor(data);
	}

	@Override
	public void onLoaderReset(Loader<Cursor> loader)
	{
		adapter.changeCursor(null);
	}
}
```

STORAGE & FILES

Storages

- Plusieurs types de stockages sont accessibles :
  - Shared Preferences : paires clés valeurs
  - Internal Storage : données privées (mémoire du téléphone).
  - External Storage : données publiques (stockage externe, e.g., carte SD).

SHARED PREFERENCES

- Informations persistentes dans la session utilisateur, sous forme de fichier.

```
SharedPreferences settings = getSharedPreferences("myPreferenceFile", Context.MODE_PRIVATE);
boolean something = settings.getBoolean("something", false);

SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("something", !something);
editor.commit();
```

- Les permissions du fichier peuvent être gérées grâce au mode. Une bonne pratique, cependant, consiste à conserver ses fichiers de préférences privés.
- Si ces informations doivent être transmises à d'autres applications, préférer les Broadcast Receivers, les Services ou les Content Providers.

Internal Storage

- Données persistentes dans la mémoire interne.
- Détruites quand l'application est désinstallée.
- Privées par défaut.

```
String content = "hello world!";
FileOutputStream fos = openFileOutput("my_file", Context.MODE_PRIVATE);
fos.write(content.getBytes());
fos.close();

FileInputStream fis = openFileInput("my_file");
...
```

- Il est possible de stocker des fichiers supplémentaires dans /res/raw pour les intégrer à l'application :

```
Activity.getResources().openRawResource(R.raw.<filename>)
```

Internal Storage

- getCacheDir() : l'emplacement du dossier cache propre à chaque application (supprimé automatiquement lorsque l'espace vient à manquer, mais à vider régulièrement de préférence).
- getFilesDir() : l'emplacement du dossier où l'application stocke ses fichiers.
- getDir() : crée (ou ouvre) un dossier.
- deleteFile() : supprime un fichier.
- fileList() : liste les fichiers stockés par l'application.

External Storage

- Données persistentes dans la mémoire externe (peut ne pas être disponible).
- Accessible en lecture et en écriture par n'importe quelle application.

```
boolean isAvailable = false;
boolean isWriteable = false;
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state))
{
    isAvailable = true;
    isWriteable = true;
}
else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
{
    isAvailable = true;
    isWriteable = false;
}
```

External Storage

```
BroadcastReceiver mExternalStorageReceiver;
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;

void updateExternalStorageState() {
   String state = Environment.getExternalStorageState();
   if (Environment.MEDIA_MOUNTED.equals(state)) {
      mExternalStorageAvailable = mExternalStorageWriteable = true;
   } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
      mExternalStorageAvailable = true;
      mExternalStorageWriteable = false;
   } else {
      mExternalStorageAvailable = mExternalStorageWriteable = false;
   }
   handleExternalStorageState(mExternalStorageAvailable,
         mExternalStorageWriteable);
}

void startWatchingExternalStorage() {
   mExternalStorageReceiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
         Log.i("test", "Storage: " + intent.getData());
         updateExternalStorageState();
      }
   };
   IntentFilter filter = new IntentFilter();
   filter.addAction(Intent.ACTION_MEDIA_MOUNTED);
   filter.addAction(Intent.ACTION_MEDIA_REMOVED);
   registerReceiver(mExternalStorageReceiver, filter);
   updateExternalStorageState();
}

void stopWatchingExternalStorage() {
   unregisterReceiver(mExternalStorageReceiver);
}
```

External Storage

- Une application peut conserver ses propres données dans le stockage externe (détruites à la désinstallation).
- L'application peut créer des dossiers prédéfinis, qui pourront être indexés par le système.
- getExternalCacheDir() = getCacheDir(), mais pour le stockage externe.

```
if (isAvailable && isWriteable)
{
    // access a file from the application folder (if the folder does not exist, it will be created)
    File file = new File(getExternalFilesDir(null), "something.dat");
    OutputStream os = new FileOutputStream(file);
    os.write("something".getBytes());
    os.close();

   // access (or create) a predefined folder for musics
   File file = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), "something.jpg");

   // tell the media scanner about the new file
   MediaScannerConnection.scanFile(this, new String[] { file.toString() }, null,
       new MediaScannerConnection.OnScanCompletedListener()
       {
           public void onScanCompleted(String path, Uri uri)
           {
               ...
           }
       });
}
```

External Storage

- Pour que des données soient conservées même après la désinstallation, deux solutions possibles :
  - getExternalStorageDirectory() pour obtenir le dossier racine du stockage externe.
  - Utiliser les répertoires partagés du système :
    - DIRECTORY_MUSIC (Music/) : musiques de l'utilisateur.
    - DIRECTORY_PODCASTS (Podcasts/) : podcast de l'utilisateur.
    - DIRECTORY_RINGTONES (Ringtones/) : sonneries de l'utilisateur.
    - DIRECTORY_ALARMS (Alarms/) : sons pour les alarmes.
    - DIRECTORY_NOTIFICATIONS (Notifications/) : sons de notifications.
    - DIRECTORY_PICTURES (Pictures/) : images de l'utilisateur (sauf photos).
    - DIRECTORY_MOVIES (Movies/) : vidéos de l'utilisateur (sauf enregistrement caméra).
    - DIRECTORY_DOWNLOADS (Download/) : téléchargements.
    - DIRECTORY_DCIM (DCIM/) : photos et enregistrements caméra.

```
if (isAvailable && isWriteable)
{
     File file = new File(getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), "something.jpg");
     ...
}
```

Credentials Storage

- L'Android Key Store permet de générer et de stocker des clés.
- Android ne peut générer que des couples de clés RSA 2048 bits.

```
// generate a key pair
Context ctx = getContext();
Calendar notBefore = Calendar.getInstance()
Calendar notAfter = Calendar.getInstance();
notAfter.add(1, Calendar.YEAR);
KeyPairGeneratorSpec spec = new KeyPairGeneratorSpec.Builder(ctx)
            .setAlias("key1")
            .setSubject( new X500Principal(String.format("CN=%s, OU=%s", "key1", ctx.getPackageName())))
            .setSerialNumber(BigInteger.ONE).setStartDate(notBefore.getTime())
            .setEndDate(notAfter.getTime()).build();

KeyPairGenerator kpGenerator = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
kpGenerator.initialize(spec);
KeyPair kp = kpGenerator.generateKeyPair();

// in another part of the app, access the keys
KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
KeyStore.PrivateKeyEntry keyEntry = (KeyStore.PrivateKeyEntry)keyStore.getEntry("key1", null);
RSAPublicKey pubKey = (RSAPublicKey)keyEntry.getCertificate().getPublicKey();
RSAPrivateKey privKey = (RSAPrivateKey) keyEntry.getPrivateKey();
```

XML

- L'API SAX et l'API DOM de Java sont disponibles sous Android.
- Android inclut aussi XmlPullParser (www.xmlpull.org), recommandé par Google.
- Libre à vous d'utiliser une librairie Java externe (e.g., Simple XML Serializer).

Serialisation XML

```
File file = new File(getExternalFilesDir(null), "xmlfile.xml");
file.createNewFile();
FileOutputStream fos = new FileOutputStream(file);
XmlSerializer serializer = Xml.newSerializer();
serializer.setOutput(fos, "UTF-8");
serializer.startDocument(null, true);
serializer.setFeature("http://xmlpull.org/v1/doc/features.html#indent-output", true);

serializer.startTag(null, "root");
    serializer.startTag(null, "someChild");
    serializer.endTag(null, "someChild");
    serializer.startTag(null, "anOtherChild");
    serializer.attribute(null, "myAttribute", "myValue");
    serializer.endTag(null, "anOtherChild");
    serializer.startTag(null, "aThirdChild");
    serializer.text("some text inside");
    serializer.endTag(null, "aThirdChild");
serializer.endTag(null, "root");
serializer.endDocument();
serializer.flush();
fos.close();
```

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<root>
    <someChild />
    <anOtherChild myAttribute="myValue" />
    <aThirdChild>some text inside</aThirdChild>
</root>
```

Parsing XML

```
<feed>
<title>My Feed !</title>
    <entry>
        <id>85c93420-3ce5-11e3-aa6e-0800200c9a66</id>
        <title>Wow, that's a good title</title>
        <category name="android" />
        <category name="development" />
        <author>
            <name>bbillet</name>
            <uri>http://benjaminbillet.fr</uri>
        </author>
        <published>2012-02-25T00:30:54Z</published>
    </entry>
    <entry>
    ...
    </entry>
    ...
</feed>
```

Parsing XML

```
File file = new File(getExternalFilesDir(null), "xmlfile.xml");
FileInputStream fis = new FileInputStream(file);

XmlPullParser parser = Xml.newPullParser();
parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
parser.setInput(fis, null);
parser.nextTag();
parser.require(XmlPullParser.START_TAG, null, "feed");
while(parser.next() != XmlPullParser.END_TAG)
{
    if (parser.getEventType() != XmlPullParser.START_TAG)
        continue;

    String name = parser.getName();
    if (name.equals("entry"))
    {
        processEntry(parser);
    }
}
```

Parsing XML

```
public void processEntry(XmlPullParser parser)
{
    parser.require(XmlPullParser.START_TAG, null, "entry");
    while(parser.next() != XmlPullParser.END_TAG)
    {
        if(parser.getEventType() != XmlPullParser.START_TAG)
            continue;

        name = parser.getName();
        if(name.equals("title"))
        {
            parser.require(XmlPullParser.START_TAG, null, "title");
            String title = readText(parser);
            parser.require(XmlPullParser.END_TAG, null, "title");
            ...
        }
        else if(name.equals("category"))
        {
            parser.require(XmlPullParser.START_TAG, null, "category");
            String category = parser.getAttributeValue(null, "name");
            parser.require(XmlPullParser.END_TAG, null, "category");
            ...
        }
        ...
        else
            skip(parser);
    }
}
```

Parsing XML

```
private void skip(XmlPullParser parser)
{
    if(parser.getEventType() != XmlPullParser.START_TAG)
        throw new IllegalStateException();

    int depth = 1;
    while(depth != 0)
    {
        switch(parser.next())
        {
            case XmlPullParser.END_TAG:
                depth--;
                break;
            case XmlPullParser.START_TAG:
                depth++;
                break;
        }
    }
}
```

JSON

- L'API JSON est disponible sous Android (www.json.org).
- Android inclut des classes utilitaires pour simplifier son utilisation.
- Libre à vous d'utiliser une librairie Java externe (e.g., GSon).

SERIALISATION JSON

```
File file = new File(getExternalFilesDir(null), "jsonfile.json");
JsonWriter writer = new JsonWriter(new OutputStreamWriter(file, "UTF-8"));
writer.setIndent("\t");

writer.beginArray();
for (Something something : somethingList)
{
      writer.beginObject();
      writer.name("id").value(something.getId());
      writer.name("text").value(something.getText());
      if (something.getUser() != null)
      {
            writer.name("user");
            writer.beginObject();
            writer.name("pseudo").value(something.getUser().getName());
            writer.name("age").value(something.getUser().getAge());
            writer.endObject();
      }
      else
            writer.name("user").nullValue();

      writer.endObject();
      if (something.getLocation() != null)
      {
            writer.name("location");
            writer.beginArray();
            writer.value(something.getLocation().getX());
            writer.value(something.getLocation().getY());
            writer.endArray();
      }
}
writer.endArray();

writer.close();
```

```
[
    {
        "id": 25467,
        "text": "How do I write JSON on Android ?",
        "user":
        {
            "pseudo": "isabelle",
            "age": 46
        }
    },
    {
        "id": 5987,
        "text": "With android.util.JsonWriter !",
        "user": null,
        "location": [56.942563, -216.436951]
    }
]
```

Parsing JSON

```
File file = new File(getExternalFilesDir(null), "jsonfile.json");
JsonReader reader = new JsonReader(new InputStreamReader(file, "UTF-8"));

List<Something> somethings = new ArrayList<Something>();
reader.beginArray();
while (reader.hasNext())
{
    Something something = new Something();
    reader.beginObject();
    while (reader.hasNext())
    {
        String name = reader.nextName();
        if (name.equals("location"))
        {
            reader.beginArray();
            something.setLocation(reader.nextDouble(), reader.nextDouble());
            reader.endArray();
        }
        else if (name.equals("user") && reader.peek() != JsonToken.NULL)
            readUser(reader);
        else if (name.equals("id")
            something.setId(reader.nextLong());
        else if (name.equals("text"))
            something.setText(reader.nextString());
        else
            reader.skipValue();
    }
    reader.endObject();
    somethings.add(something);
}
reader.endArray();
```

```
private void readUser(JsonReader reader)
{
    reader.beginObject();
    something.setUser(new User());
    while (reader.hasNext())
    {
        if (name.equals("pseudo"))
            something.getUser().setPseudo(reader.nextString());
        else if (name.equals("age"))
            something.getUser().setAge(reader.nextInt());
        else
            reader.skipValue();
    }
    reader.endObject();
}
```

SQLITE

SQLite

- Android fournit un support des bases de données SQLite.
- Les bases sont restreintes aux applications qui les créent (utiliser un Content Provider pour ouvrir vos bases aux autres applications).
- Une base doit contenir une clé primaire _ID pour pouvoir être présentée par un Content Provider.
- SQLiteOpenHelper simplifie la gestion de la base (ouverture uniquement lorsque nécessaire, gestion des versions, etc.).

SQLite

```
public class DatabaseHelper extends SQLiteOpenHelper
{
    public static final String DATABASE_NAME = "mybase.db";
    public static final int DATABASE_VERSION = 2;

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

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL("CREATE TABLE my_table ("
            + BaseColumns._ID + " INTEGER PRIMARY KEY,"
            + "title TEXT,"
            + "content TEXT,"
            + "creationDate INTEGER"
            + ");");
    }

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

- onCreate() est invoquée lorsque la base doit être crée.
- onUpgrade() est invoquée lorsque la base existe déjà, mais d'une version inférieure à celle requise.
- onDowngrade() n'est pas obligatoire. Si elle n'est pas implémenté, l'appelant recevra automatiquement une exception.

SQLite avec un Content Provider

```
public class MyContentProvider extends ContentProvider
{
    private DatabaseHelper dbHelper;
    private UriMatcher matcher;

    public boolean onCreate()
    {
        dbHelper = new DatabaseHelper(getContext());
        matcher = new UriMatcher(UriMatcher.NO_MATCH);
        matcher.addURI("fr.inria.myProvider", "data", 1);
        matcher.addURI("fr.inria.myProvider", "data/#", 2);
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
    {
    }
    @Override
    public Uri insert(Uri uri, ContentValues initialValues)
    {
    }
    @Override
    public int update(Uri uri, ContentValues values, String where, String[] whereArgs)
    {
    }
    @Override
    public int delete(Uri uri, String where, String[] whereArgs)
    {
    }
}
```

```
@Override
public String getType(Uri uri)
{
    switch (matcher.match(uri))
    {
        case 1:
            return "vnd.android.cursor.dir/vnd.inria.data";
        case 2:
            return "vnd.android.cursor.item/vnd.inria.data";
        default:
            throw new IllegalArgumentException(...);
    }
}
```

- Le MIME recommandé pour les données se compose de deux parties :
  - Android-specific : vnd.android.cursor.item (une ligne), vnd.android.cursor.dir (plusieurs lignes).
  - Vendor-specific : vnd.<name>.<type>

SQLite avec un Content Provider

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
{
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    Map<String, String> projectionMap = new HashMap<String, String>();
    projection.add("title", "my_table.title");
    projection.add("content", "my_table.content");

    qb.setTables("my_table");
    qb.setProjectionMap(projectionMap);
    if (matcher.match(uri) == 2)
        qb.appendWhereEscapeString(BaseColumns._ID + " = 42";

    if (TextUtils.isEmpty(sortOrder))
        sortOrder = "creationDate DESC";

    // the db is then effectively open
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    Cursor c = qb.query(
              db,              // the database to query
              projection,      // the columns to return from the query
              selection,       // the columns for the where clause
              selectionArgs,   // the values for the where clause
              null,            // don't group the rows
              null,            // don't filter by row groups
              sortOrder        // the sort order
    );

    // tells the Cursor what URI to watch, so it knows when its source data changes
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
```

SQLite avec un Content Provider

```
@Override
public Uri insert(Uri uri, ContentValues initialValues)
{
    if (matcher.match(uri) != 1)
        throw new IllegalArgumentException("Unknown URI " + uri);

    ContentValues values;
    if (initialValues != null)
        values = new ContentValues(initialValues);
    else
        values = new ContentValues();

    if (values.containsKey("title") == false)
        values.put("title", Resources.getSystem().getString(android.R.string.untitled));

    if (values.containsKey("creationDate") == false)
        values.put("creationDate", System.currentTimeMillis());

    if (values.containsKey("content") == false)
        values.put("content", "");

    SQLiteDatabase db = dbHelper.getWritableDatabase(); // writable db
    long id = db.insert(
              "my_table",
              null,
              values
    );

    if (id > 0)
    {
        Uri rowUri = ContentUris.appendId(uri.buildUpon(), id).build();
        getContext().getContentResolver().notifyChange(rowUri, null); // notify change
        return rowUri;
    }

    throw new SQLException("Failed to insert row into " + uri);
}
```

Remarque : les valeurs du ContentValues doivent être vérifiée, de même que l'URI, avant de valider l'insertion.

SQLite avec un Content Provider

```
@Override
public int update(Uri uri, ContentValues values, String where, String[] whereArgs)
{
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   String finalWhere = where;

   if (matcher.match(uri) == 2)
   {
      finalWhere = BaseColumns._ID + " = " + uri.getLastPathSegment();
      if (where != null)
         finalWhere = finalWhere + " AND " + where;
   }

   int nbUpdatedRows = db.update(
         "my_table",
         values,
         finalWhere,
         whereArgs
   );

   getContext().getContentResolver().notifyChange(uri, null);
   return nbUpdatedRows;
}
```

Remarque : les valeurs du ContentValues doivent être vérifiée, de même que l'URI, avant de valider la modification.

SQLite avec un Content Provider

```
@Override
public int delete(Uri uri, String where, String[] whereArgs)
{
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   String finalWhere = where;

   if (matcher.match(uri) == 2)
   {
      finalWhere = BaseColumns._ID + " = " + uri.getLastPathSegment();
      if (where != null)
         finalWhere = finalWhere + " AND " + where;
   }

   int nbDeletedRows = db.delete(
         "my_table",
         finalWhere,
         whereArgs
      );

   getContext().getContentResolver().notifyChange(uri, null);
   return nbDeletedRows;
}
```