
Atelier 6

Persistance des Données :

Le partage des données et Content Provider

Walid ZEDDINI
www.zeddini.com

Présentation du ContentProvider

Une base de données SQLite est privée à l'application qui l'a créée. Si vous voulez partager des données avec d'autres applications, vous pouvez utiliser un ContentProvider .

Un ContentProvider permet aux applications d'accéder aux données. Dans la plupart des cas, ces données sont stockées dans une base de données SQLite.

Un ContentProvider peut être utilisé en interne dans une application pour accéder aux données. Si les données doivent être partagées avec une autre application un ContentProvider le permet.

L'accès à un ContentProvider se fait via un URI. La base de l'URI est définie par la déclaration de le ContentProvider dans le fichier *AndroidManifest.xml* via l'attribut `android:authorities`.

Plusieurs sources de données Android, par exemple les contacts, sont accessibles via ContentProviders . Typiquement, les classes d'implémentation d'un ContentProvider fournissent des constantes publiques pour l'URI.

Mon Propre ContentProvider

Pour créer votre propre ContentProvider vous devez définir une classe qui étend (hérite) `android.content.ContentProvider` . Vous devez également déclarer votre ContentProvider dans le fichier *AndroidManifest.xml* . Cette entrée doit spécifier l'attribut `android:authorities` qui permet d'identifier le ContentProvider. Cette autorité est la base de l'URI pour accéder aux données et elle doit être unique.

```
<provider
    android:name=".contentprovider.MyProduitContentProvider"
    android:authorities="tn.rnu.isi.contentproviderdemo.contentprovider" >
</provider>
```

Votre ContentProvider doit implémenter plusieurs méthodes, par exemple `query ()` , `insert ()` , `update ()` , `delete ()` , `getType ()` et `onCreate ()` . Dans le cas où vous ne comptez pas implémenter certaines méthodes, vous devez prévoir d'ajouter un `UnsupportedOperationException ()` .

La méthode `query ()` doit retourner un objet `Cursor`.

Sécurité et ContentProvider

Par défaut, un ContentProvider sera disponible pour d'autres programmes. Si vous souhaitez utiliser votre ContentProvider seulement à l'interne, vous pouvez utiliser l'attribut `android:exported=false` dans la déclaration de votre ContentProvider dans le fichier *AndroidManifest.xml* .

Sécurité des threads

Si vous travaillez directement avec les bases de données et vous disposez de plusieurs « writers » de différents threads vous pouvez rencontrer des problèmes de concurrence.

Le `ContentProvider` peut être consulté à partir de plusieurs programmes en même temps, donc il faut mettre en œuvre un accès thread-safe. Le plus simple est d'utiliser le mot clé `synchronized` en face de toutes les méthodes de la `ContentProvider`, de sorte qu'un seul thread peut accéder à ces méthodes en même temps.

Si vous ne souhaitez pas que Android synchronise l'accès aux données au `ContentProvider`, réglez l'attribut `android:multiprocess=true` dans la définition de votre `<provider>` dans le fichier `AndroidManifest.xml`. Cela permet la création d'une instance du provider dans chaque processus client, ce qui élimine la nécessité d'effectuer une communication inter-processus (IPC).

Loader

Les loaders ont été introduits avec la version Android 3.0 et font partie de la couche de compatibilité pour les anciennes versions Android (à partir de Android 1.6). Ils sont disponibles dans la classe `Activity` et `Fragment`.

Les loaders permettent de charger des données en mode **asynchrone**, et peuvent surveiller la source des données et fournir de nouveaux résultats lors des changements de contenu. Ils persistent également entre les changements de configuration.

Vous pouvez utiliser la classe abstraite `AsyncTaskLoader` comme base d'implémentation de votre classe `Loader`. Pour une `ContentProvider` basée sur une base de données `SQLite`, on utilise généralement la classe `CursorLoader`. Ce `Loader` exécute la requête du curseur dans un thread d'arrière-plan afin que l'application ne soit pas bloquée. Les loaders remplacent les *Activity-managed cursors* qui sont obsolètes maintenant.

Il est de bonne pratique que l'*activity* qui utilise un *loader* qu'elle implémente directement l'interface `LoaderManager.LoaderCallbacks`.

La création d'un `Loader` s'effectue via l'appel de méthode `getLoaderManager().initLoader(0, null, this)`.

Le troisième paramètre de `initLoader()` est la classe qui est appelé une fois l'initialisation a été lancée (classe de rappel ou callback). Typiquement, l'*activity* est utilisée comme classe callback. Le premier paramètre est un ID (identifiant unique) qui peut être utilisé par la classe callback pour identifier les *loaders* plus tard. Le deuxième paramètre est un ensemble de paramètres qui peut être donnée à la classe callback pour plus d'informations.

En fait, le `loader` n'est pas directement créé par l'appel de la méthode `getLoaderManager().initLoader()`, mais doit être créé par la méthode de la classe callback `onCreateLoader()`.

Une fois que le `loader` a terminé la lecture de données en mode asynchrone, la méthode `onLoadFinished()` de la classe callback est appelée. Ici vous pouvez mettre à jour votre interface utilisateur.

Si le curseur devient invalide, la méthode `onLoaderReset()` est appelée sur la classe callback.

Curseurs et Loaders

Un des défis avec l'accès à la base de données est la lenteur de cet accès. L'autre défi est que l'application doit prendre en considération le cycle de vie des composants correctement, par exemple, ouvrir et fermer le curseur si un changement de configuration aura lieu.

Pour gérer le cycle de vie, vous pouvez utiliser la méthode `ManagedQuery ()` de la classe `activity` avant Android 3.0. Comme d'Android 3.0 cette méthode est obsolète et vous devez utiliser le cadre `loader` pour accéder au `ContentProvider`.

Le `SimpleCursorAdapter` classe, qui peut être utilisé avec `ListView`, par la méthode `swapCursor ()`. Votre `loader` peut utiliser cette méthode pour mettre à jour le curseur dans sa méthode `onLoadFinished ()`. La classe `CursorLoader` rebranche le curseur après un changement de configuration.

Activité 1 : ContentProvider

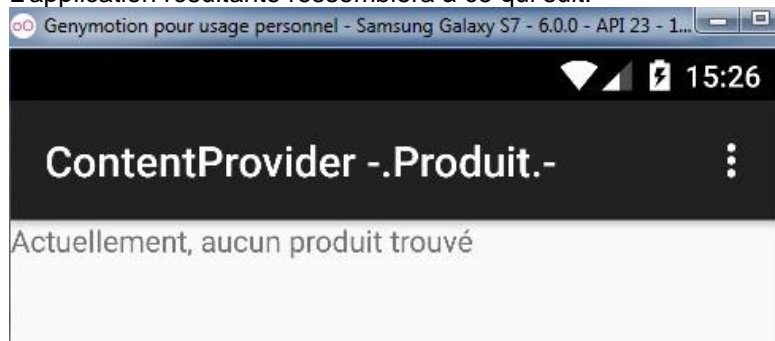
Nous allons créer une application " **ContentProviderDemo**" qui permet à l'utilisateur d'entrer des produits. Ces articles seront stockées dans la base de données SQLite et accessibles via un `ContentProvider`.

Les articles produits sont appelées «produit» ou «produit s» dans ce TP.

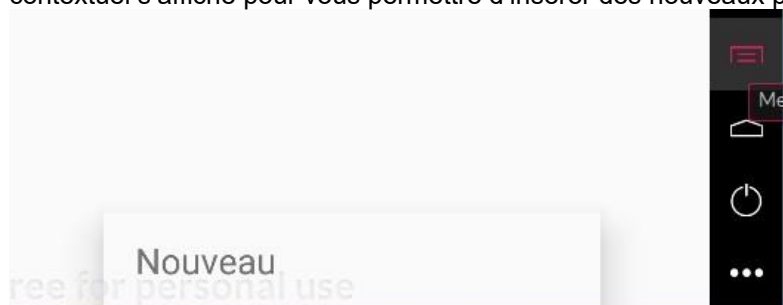
L'application se compose de deux *activités*, l'une pour voir une liste de tous les articles produit et l'autre pour la création et la modification d'un élément produit spécifique. Ces deux *activités* vont communiquer via *intentions*.

Pour charger de manière asynchrone et gérer le curseur de la principale classe *activité* on va utiliser un `loader` (chargeur).

L'application résultante ressemblera à ce qui suit.



Cliquer sur les boutons du clavier « CTL +M ou F2 + Page down » ou bien sur le bouton Menu de l'émulateur, le menu contextuel s'affiche pour vous permettre d'insérer des nouveaux produits :



En cliquant sur le bouton « Nouveau », l'écran suivant s'affiche.

ContentProvider -.Produit.-

HAUT DE GAMME ▾


Galaxy's


Samsung


CONFIRMER


Insérer les produits un à un, la liste des produits sera alimentée au fur et à mesure :


ContentProvider Nouveau


Samsung

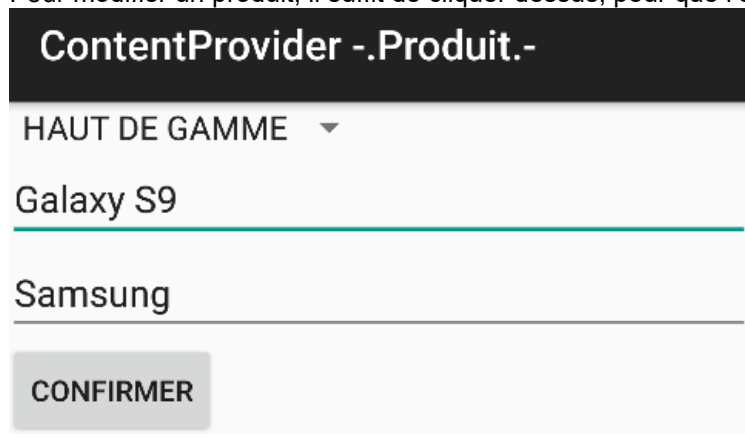

Huawei


IPhone

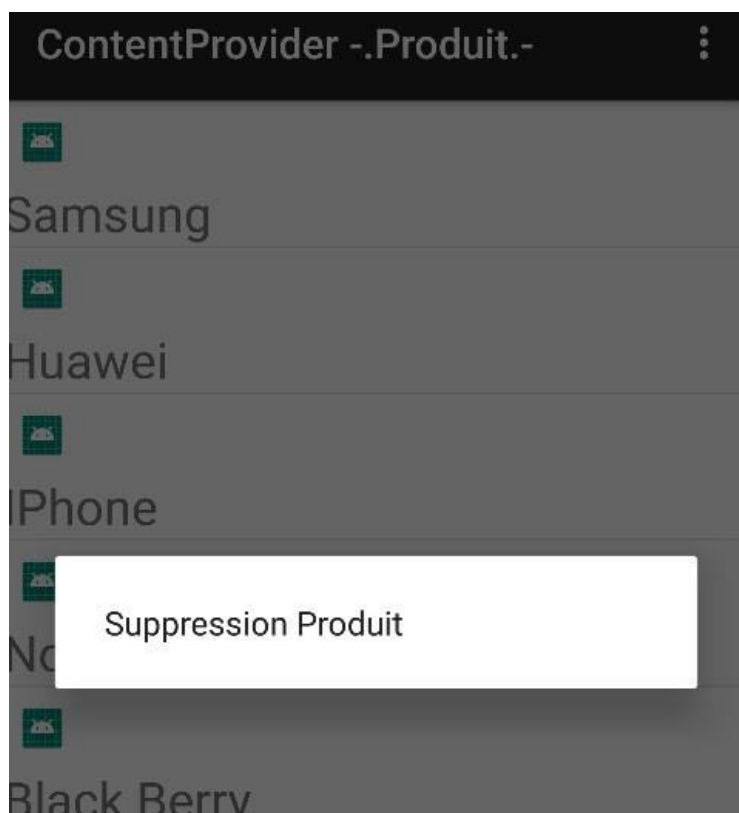

Nokia


Black Berry

Pour modifier un produit, il suffit de cliquer dessus, pour que l'écran de modification s'affiche :



Pour supprimer un item (produit) de la liste, maintenez enfoncé le clic sur l'item :



Créer un projet Android «ContentProviderDemo» implémentant l'interface utilisateur ci-dessus:

Pour se faire, nous allons créer un projet qui s'appellera **ContentProviderDemo**, avec la **version** si PC performant **8.0** ou **7.0** sinon **6.0** du SDK et une résolution **800-1280**.

- **Nom de l'application** : ContentProviderDemo
- **Package** : tn.rnu.isi.contentproviderdemo
- **Activité** : MainActivity

Créer une autre activité appelée ProduitDetailActivity .

Les classes de base de données

Créez le package **tn.rnu.isi.contentproviderdemo.database** . Ce paquet va stocker les classes nécessaires à la manipulation de la base de données.

Une bonne pratique est d'envisager d'avoir une classe séparée par table. Même si nous n'avons qu'une seule table dans ce TP, nous allons suivre cette pratique. De cette façon, nous serons prêts, au cas où notre schéma de base de données change en ajoutant d'autre table. Créer la classe **ProduitTable**. Elle contient des constantes portant les noms de la table et des colonnes.

```
package tn.rnu.isi.contentproviderdemo.database;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class ProduitTable {

    // Database table
    public static final String TABLE_PRODUIT = "Produit";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_CATEGORY = "category";
    public static final String COLUMN_MARQUE = "marque";
    public static final String COLUMN_MODEL = "model";

    // Database creation SQL statement
    private static final String DATABASE_CREATE = "CREATE TABLE "
        + TABLE_PRODUIT
        + "("
        + COLUMN_ID + " integer primary key autoincrement, "
        + COLUMN_CATEGORY + " text not null, "
        + COLUMN_MARQUE + " text not null, "
        + COLUMN_MODEL
        + " text not null"
        + ");";

    public static void onCreate(SQLiteDatabase database) {
        database.execSQL(DATABASE_CREATE);
    }

    public static void onUpgrade(SQLiteDatabase database, int oldVersion,
                                int newVersion) {
        Log.w(ProduitTable.class.getName(), "Upgrading database from version "
            + oldVersion + " to " + newVersion
            + ", which will destroy all old data");
        database.execSQL("DROP TABLE IF EXISTS " + TABLE_PRODUIT);
        onCreate(database);
    }
}
```

Créez la classe suivante **ProduitDatabaseHelper** . Cette classe hérite de **SQLiteOpenHelper** et appelle les méthodes statiques de la classe helper **ProduitTable**.

```
package tn.rnu.isi.contentproviderdemo.database;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
```

```

import android.database.sqlite.SQLiteOpenHelper;

public class ProduitDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "produittable.db";
    private static final int DATABASE_VERSION = 1;

    public ProduitDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    // Method is called during creation of the database
    @Override
    public void onCreate(SQLiteDatabase database) {
        ProduitTable.onCreate(database);
    }
    // Method is called during an upgrade of the database,
    // e.g. if you increase the database version
    @Override
    public void onUpgrade(SQLiteDatabase database, int oldVersion,
        int newVersion) {
        ProduitTable.onUpgrade(database, oldVersion, newVersion);
    }
}

```

Nous allons utiliser un **ContentProvider** pour accéder à la base de données, nous n'allons pas écrire un objet d'accès aux données (DAO) comme nous l'avons fait dans les exemples précédents de la persistance **SQLite**.

Créer ContentProvider

Créez le package **tn.rnu.isi.contentproviderdemo.contentprovider**. Puis, créez la classe **MyProduitContentProvider** suivante qui hérite de **ContentProvider**.

```

package tn.rnu.isi.contentproviderdemo.contentprovider;

import java.util.Arrays;
import java.util.HashSet;

import tn.rnu.isi.contentproviderdemo.database.ProduitDatabaseHelper;
import tn.rnu.isi.contentproviderdemo.database.ProduitTable;

import android.content.ContentProvider;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

public class MyProduitContentProvider extends ContentProvider {

    // database
    private ProduitDatabaseHelper database;

    // Used for the UriMacher
    private static final int PRODUITS = 10;
    private static final int PRODUIT_ID = 20;

```



```

private static final String AUTHORITY =
"tn.rnu.isi.contentproviderdemo.contentprovider";

private static final String BASE_PATH = "produits";
public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY
+ "/" + BASE_PATH);

public static final String CONTENT_TYPE = ContentResolver.CURSOR_DIR_BASE_TYPE
+ "/produits";
public static final String CONTENT_ITEM_TYPE = ContentResolver.CURSOR_ITEM_BASE_TYPE
+ "/produit";

private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);
static {
    sURIMatcher.addURI(AUTHORITY, BASE_PATH, PRODUITS);
    sURIMatcher.addURI(AUTHORITY, BASE_PATH + "/#", PRODUIT_ID);
}

@Override
public boolean onCreate() {
    database = new ProduitDatabaseHelper(getContext());
    return false;
}

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

    // Using SQLiteQueryBuilder instead of query() method
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();

    // Check if the caller has requested a column which does not exists
    checkColumns(projection);

    // Set the table
    queryBuilder.setTables(ProduitTable.TABLE_PRODUIT);

    int uriType = sURIMatcher.match(uri);
    switch (uriType) {
        case PRODUITS:
            break;
        case PRODUIT_ID:
            // Adding the ID to the original query
            queryBuilder.appendWhere(ProduitTable.COLUMN_ID + "="
+ uri.getLastPathSegment());
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }

    SQLiteDatabase db = database.getWritableDatabase();
    Cursor cursor = queryBuilder.query(db, projection, selection,
selectionArgs, null, null, sortOrder);
    // Make sure that potential listeners are getting notified
    cursor.setNotificationUri(getContext().getContentResolver(), uri);

    return cursor;
}

@Override
public String getType(Uri uri) {

```

```

        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        int uriType = surimatcher.match(uri);
        SQLiteDatabase sqlDB = database.getWritableDatabase();
        long id = 0;
        switch (uriType) {
            case PRODUITS:
                id = sqlDB.insert(ProduitTable.TABLE_PRODUIT, null, values);
                break;
            default:
                throw new IllegalArgumentException("Unknown URI: " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return Uri.parse(BASE_PATH + "/" + id);
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int uriType = surimatcher.match(uri);
        SQLiteDatabase sqlDB = database.getWritableDatabase();
        int rowsDeleted = 0;
        switch (uriType) {
            case PRODUITS:
                rowsDeleted = sqlDB.delete(ProduitTable.TABLE_PRODUIT, selection,
                    selectionArgs);
                break;
            case PRODUIT_ID:
                String id = uri.getLastPathSegment();
                if (TextUtils.isEmpty(selection)) {
                    rowsDeleted = sqlDB.delete(ProduitTable.TABLE_PRODUIT,
                        ProduitTable.COLUMN_ID + "=" + id,
                        null);
                } else {
                    rowsDeleted = sqlDB.delete(ProduitTable.TABLE_PRODUIT,
                        ProduitTable.COLUMN_ID + "=" + id
                            + " and " + selection,
                        selectionArgs);
                }
                break;
            default:
                throw new IllegalArgumentException("Unknown URI: " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return rowsDeleted;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {

        int uriType = surimatcher.match(uri);
        SQLiteDatabase sqlDB = database.getWritableDatabase();
        int rowsUpdated = 0;
        switch (uriType) {
            case PRODUITS:
                rowsUpdated = sqlDB.update(ProduitTable.TABLE_PRODUIT,
                    values,
                    selection,

```

```

        selectionArgs);
        break;
    case PRODUIT_ID:
        String id = uri.getLastPathSegment();
        if (TextUtils.isEmpty(selection)) {
            rowsUpdated = sqlDB.update(ProduitTable.TABLE_PRODUIT,
                values,
                ProduitTable.COLUMN_ID + "=" + id,
                null);
        } else {
            rowsUpdated = sqlDB.update(ProduitTable.TABLE_PRODUIT,
                values,
                ProduitTable.COLUMN_ID + "=" + id
                    + " and "
                    + selection,
                selectionArgs);
        }
        break;
    default:
        throw new IllegalArgumentException("Unknown URI: " + uri);
}
getContext().getContentResolver().notifyChange(uri, null);
return rowsUpdated;
}

private void checkColumns(String[] projection) {
    String[] available = { ProduitTable.COLUMN_CATEGORY,
        ProduitTable.COLUMN_MARQUE, ProduitTable.COLUMN_MODEL,
        ProduitTable.COLUMN_ID };
    if (projection != null) {
        HashSet<String> requestedColumns = new
HashSet<String>(Arrays.asList(projection));
        HashSet<String> availableColumns = new
HashSet<String>(Arrays.asList(available));
        // Check if all columns which are requested are available
        if (!availableColumns.containsAll(requestedColumns)) {
            throw new IllegalArgumentException("Unknown columns in projection");
        }
    }
}
}
}

```

MyProduitContentProvider implémente les méthodes : `update()`, `insert()`, `delete()` et `query()`. Ces méthodes mappe plus ou moins directement l'interface **SQLiteDatabase**. Elle a également la méthode `checkColumns()` pour valider une requête et demande uniquement si les colonnes sont valides.

Enregistrez votre **ContentProvider** dans votre fichier **AndroidManifest.xml**.

```

...
<provider
    android:name=".contentprovider.MyProduitContentProvider"
    android:authorities="tn.rnu.isi.contentproviderdemo.contentprovider" >
</provider>

</application>

```

Ressources

Notre application nécessite plusieurs ressources. D'abord définir un menu **listmenu.xml** dans le dossier **res/menu** . Si vous utilisez l'assistant de ressource Android pour créer le fichier "listmenu.xml", le dossier sera créé pour vous, si vous créez le fichier manuellement, vous devez également créer manuellement le dossier **res/menu** s'il n'existe pas.

Ce fichier XML sera utilisé pour définir le menu d'options dans notre application. L'attribut `android:showAsAction="always"` veillera à ce que cette entrée de menu est affiché dans la *ActionBar* de notre application.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/insert"
        android:title="Nouveau">
    </item>
</menu>
```

L'utilisateur sera en mesure de sélectionner la priorité pour les articles produits (HAUT DE GAMME ou BAS DE GAMME). Pour les priorités que nous créons un tableau de chaînes. Créez le fichier suivant **priority.xml** dans le dossier **res/values**.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string-array name="priorities">
        <item>HAUT DE GAMME</item>
        <item>BAS DE GAMME</item>
    </string-array>

</resources>
```

Définissez également des chaînes supplémentaires pour l'application. Modifier **strings.xml** sous **res/values** .

```
<resources>
    <string name="app_name">ContentProvider -.Produit.-</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="hello">Hello World</string>
    <string name="no_produits">Actuellement, aucun produit trouvé</string>
    <string name="menu_insert">Ajout Produit</string>
    <string name="menu_delete">Suppression Produit</string>
    <string name="produit_category">Catégorie</string>
    <string name="produit_marque">Supprimer Produit</string>
    <string name="produit_edit_model">Modèle</string>
    <string name="produit_edit_marque">Marque</string>
    <string name="produit_edit_confirm">Confirmer</string>
</resources>
```

Définissez également des fichiers supplémentaires pour l'application. Créer **dimens.xml** sous **res/values** .

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
```

```
<dimen name="activity_vertical_margin">16dp</dimen>

</resources>
```

Définissez également des fichiers supplémentaires pour l'application. Créer **main.xml** sous **res/menu** .

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"/>
</menu>
```

Layouts

Nous allons définir trois modèles. L'un sera utilisé pour l'affichage d'une ligne dans la liste, les autres seront utilisées par nos *activités* .

Créez le fichier de configuration **produit_row.xml** dans le dossier **res/layout** .

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="30dp"
        android:layout_height="24dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="8dp"
        android:src="@mipmap/ic_launcher" >
    </ImageView>

    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="6dp"
        android:lines="1"
        android:text="@+id/TextView01"
        android:textSize="24dp"
    >
    </TextView>
</LinearLayout>
```

Créez le fichier de configuration **produit_list.xml**. Cette layout définit la façon dont la liste va ressembler.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ListView>

    <TextView
        android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no_produits" />
</LinearLayout>

```

Créez le fichier de configuration **produit_edit.xml** . Ce layout sera utilisée pour afficher et modifier un élément produit dans l' *activité* **ProduitDetailActivity** .

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Spinner
        android:id="@+id/category"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/priorities" >
    </Spinner>

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <EditText
            android:id="@+id/produit_edit_model"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="@string/produit_edit_model"
            android:imeOptions="actionNext" >
        </EditText>
    </LinearLayout>

    <EditText
        android:id="@+id/produit_edit_marque"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="top"
        android:hint="@string/produit_edit_marque"
        android:imeOptions="actionNext" >
    </EditText>

```

```

<Button
    android:id="@+id/produit_edit_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/produit_edit_confirm" >
</Button>
</LinearLayout>

```

Créez le fichier de configuration **activity_main.xml**

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>

```

Activités

Modifier le code de vos activités comme suit. La Première **MainActivity.java** :

```

package tn.rnu.isi.contentproviderdemo;

import tn.rnu.isi.contentproviderdemo.contentprovider.MyProduitContentProvider;
import tn.rnu.isi.contentproviderdemo.database.ProduitTable;
import android.annotation.SuppressLint;
import android.app.ListActivity;
import android.app.LoaderManager;
import android.content.CursorLoader;
import android.content.Intent;
import android.content.Loader;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

/*
 * ProduitsOverviewActivity displays the existing produit items
 * in a list
 *
 * You can create new ones via the ActionBar entry "Insert"
 */

```

```
* You can delete existing ones via a long press on the item  
*/
```

```
public class MainActivity extends ListActivity implements  
    LoaderManager.LoaderCallbacks<Cursor> {  
    private static final int ACTIVITY_CREATE = 0;  
    private static final int ACTIVITY_EDIT = 1;  
    private static final int DELETE_ID = Menu.FIRST + 1;  
    // private Cursor cursor;  
    private SimpleCursorAdapter adapter;  
  
    /** Called when the activity is first created. */  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.produit_list);  
        this.getListView().setDividerHeight(2);  
        fillData();  
        registerForContextMenu(getListView());  
    }  
  
    // Create the menu based on the XML defintion  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        MenuInflater inflater = getMenuInflater();  
        inflater.inflate(R.menu.listmenu, menu);  
        return true;  
    }  
  
    // Reaction to the menu selection  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        switch (item.getItemId()) {  
            case R.id.insert:  
                createProduit();  
                return true;  
        }  
        return super.onOptionsItemSelected(item);  
    }  
  
    @Override  
    public boolean onContextItemSelected(MenuItem item) {  
        switch (item.getItemId()) {  
            case DELETE_ID:  
                AdapterContextMenuInfo info = (AdapterContextMenuInfo) item  
                    .getMenuInfo();  
                Uri uri = Uri.parse(MyProduitContentProvider.CONTENT_URI + "/"  
                    + info.id);  
                getContentResolver().delete(uri, null, null);  
                fillData();  
                return true;  
        }  
        return super.onContextItemSelected(item);  
    }  
  
    private void createProduit() {  
        Intent i = new Intent(this, ProduitDetailActivity.class);  
        startActivity(i);  
    }  
}
```



```

// Opens the second activity if an entry is clicked
@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);
    Intent i = new Intent(this, ProduitDetailActivity.class);
    Uri produitUri = Uri.parse(MyProduitContentProvider.CONTENT_URI + "/"
        + id);
    i.putExtra(MyProduitContentProvider.CONTENT_ITEM_TYPE, produitUri);

    startActivity(i);
}

@SuppressLint("NewApi")
private void fillData() {

    // Fields from the database (projection)
    // Must include the _id column for the adapter to work
    String[] from = new String[] { ProduitTable.COLUMN_MARQUE };
    // Fields on the UI to which we map
    int[] to = new int[] { R.id.label };

    getLoaderManager().initLoader(0, null, this);
    adapter = new SimpleCursorAdapter(this, R.layout.produit_row, null,
        from, to, 0);

    setListAdapter(adapter);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.add(0, DELETE_ID, 0, R.string.menu_delete);
}

// Creates a new loader after the initLoader () call
@SuppressLint("NewApi")
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String[] projection = { ProduitTable.COLUMN_ID,
        ProduitTable.COLUMN_MARQUE };
    CursorLoader cursorLoader = new CursorLoader(this,
        MyProduitContentProvider.CONTENT_URI, projection, null, null,
        null);
    return cursorLoader;
}

@SuppressLint("NewApi")
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    adapter.swapCursor(data);
}

@SuppressLint("NewApi")
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // data is not available anymore, delete reference
    adapter.swapCursor(null);
}
}

```

Et ProduitDetailActivity.java

```
package tn.rnu.isi.contentproviderdemo;

import tn.rnu.isi.contentproviderdemo.contentprovider.MyProduitContentProvider;
import tn.rnu.isi.contentproviderdemo.database.ProduitTable;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;

/*
 * ProduitDetailActivity allows to enter a new produit item
 * or to change an existing
 */
public class ProduitDetailActivity extends Activity {
    private Spinner mCategory;
    private EditText mMarqueText;
    private EditText mModelText;

    private Uri produitUri;

    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.produit_edit);

        mCategory = (Spinner) findViewById(R.id.category);
        mMarqueText = (EditText) findViewById(R.id.produit_edit_marque);
        mModelText = (EditText) findViewById(R.id.produit_edit_model);
        Button confirmButton = (Button) findViewById(R.id.produit_edit_button);

        Bundle extras = getIntent().getExtras();

        // Check from the saved Instance
        produitUri = (bundle == null) ? null : (Uri) bundle
            .getParcelable(MyProduitContentProvider.CONTENT_ITEM_TYPE);

        // Or passed from the other activity
        if (extras != null) {
            produitUri = extras
                .getParcelable(MyProduitContentProvider.CONTENT_ITEM_TYPE);

            fillData(produitUri);
        }

        confirmButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                if (TextUtils.isEmpty(mMarqueText.getText().toString())) {
                    makeToast();
                } else {
                    setResult(RESULT_OK);
                }
            }
        });
    }
}
```

```

        finish();
    }
}

});
}

private void fillData(Uri uri) {
    String[] projection = { ProduitTable.COLUMN_CATEGORY,
        ProduitTable.COLUMN_MARQUE, ProduitTable.COLUMN_MODEL };
    Cursor cursor = getContentResolver().query(uri, projection, null, null,
        null);
    if (cursor != null) {
        cursor.moveToFirst();
        String category = cursor.getString(cursor
            .getColumnIndexOrThrow(ProduitTable.COLUMN_CATEGORY));

        for (int i = 0; i < mCategory.getCount(); i++) {

            String s = (String) mCategory.getItemAtPosition(i);
            if (s.equalsIgnoreCase(category)) {
                mCategory.setSelection(i);
            }

            mMarqueText.setText(cursor.getString(cursor
                .getColumnIndexOrThrow(ProduitTable.COLUMN_MARQUE)));
            mModelText.setText(cursor.getString(cursor
                .getColumnIndexOrThrow(ProduitTable.COLUMN_MODEL)));

            // Always close the cursor
            cursor.close();
        }
    }

    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        saveState();
        outState.putParcelable(MyProduitContentProvider.CONTENT_ITEM_TYPE, produitUri);
    }

    @Override
    protected void onPause() {
        super.onPause();
        saveState();
    }

    private void saveState() {
        String category = (String) mCategory.getSelectedItemAtPosition(0);
        String marque = mMarqueText.getText().toString();
        String model = mModelText.getText().toString();

        // Only save if either marque or model
        // is available

        if (model.length() == 0 && marque.length() == 0) {
            return;
        }
        ContentValues values = new ContentValues();
        values.put(ProduitTable.COLUMN_CATEGORY, category);
        values.put(ProduitTable.COLUMN_MARQUE, marque);
    }
}

```

```

        values.put(ProduitTable.COLUMN_MODEL, model);

        if (produitUri == null) {
            // New produit
            produitUri =
getContentResolver().insert(MyProduitContentProvider.CONTENT_URI, values);
        } else {
            // Update produit
            getContentResolver().update(produitUri, values, null, null);
        }
    }

    private void makeToast() {
        Toast.makeText(ProduitDetailActivity.this, "Indiquer une marque SVP",
            Toast.LENGTH_LONG).show();
    }
}

```

Le résultat **AndroidManifest.xml** ressemble à ce qui suit.

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="tn.rnu.isi.contentproviderdemo">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ProduitDetailActivity"
            android:windowSoftInputMode="stateVisible|adjustResize" >
        </activity>

        <provider
            android:name=".contentprovider.MyProduitContentProvider"
            android:authorities="tn.rnu.isi.contentproviderdemo.contentprovider" >
        </provider>

    </application>
</manifest>

```

Il est à noter que `android:windowSoftInputMode="stateVisible|adjustResize"` est défini pour le **PrduitDetailActivity** . Cela rend le clavier mieux harmonisé avec les widgets, mais il n'est pas requis pour ce TP.

CODE SOURCE

Le code source du projet **ContentProviderDemo.rar** est téléchargeable à partir du lien:

<https://drive.google.com/open?id=17y1JfvEJKPJAr7PdXmwoINtkVfGkBby>