**Le stockage de données**

Android permet de persister les données des applications via le système de fichiers. Android fournit plusieurs méthodes pour sauvegarder des informations.

En plus des bases de données, il existe deux solutions qui permettent d'enregistrer des données, les préférences partagées et les fichiers :

* Les préférences partagées : permettent d'associer une valeur à un identifiant (clé-valeur). Le couple ainsi créé permet de retenir les différentes options que l'utilisateur veut conserver ou l'état de l'interface graphique. Ces valeurs pourront être partagées entre plusieurs composants.
* Les fichiers : permettent de stocker les données sur le terminal ou sur un périphérique externe.
* Base de données SQLite : des instances de bases de données SQLite sont stockées sur le système de fichiers local.   
  Les fichiers sont enregistrés dans le dossier files et les paramètres de l'application sont enregistrés sous forme de fichiers XML dans le dossier shared\_prefs.

### Préférences partagées

Les préférences partagées, *SharedPreferences*  permettent de stocker et récupérer les données sous forme de paires clé-valeur (préférences) de types primitifs dans le système de fichiers Android.  
La définition de ces préférences peut se faire par une ressource XML.  
  
Pour accéder aux préférences stockées on utilise des méthodes de la classe PreferenceManager.

#### Les données partagées

Le point de départ de la manipulation des préférences partagées est la classe [SharedPreferences](http://developer.android.com/reference/android/content/SharedPreferences.html). Elle possède des méthodes permettant d'enregistrer et récupérer des paires de type clé-valeur.

L'avantage est que ces données restent conservées même si l'application est arrêtée.

Il y a trois méthodes de la classe SharedPreferences :

* PreferenceManager.getDefaultSharedPreferences(Context context),c'est la méthode statique la plus simple.
* getSharedPreferences (String name, int mode) où name est le nom du fichier créé.
* getPreferences(int mode) pour utiliser un fichier standard par activité.

Le paramètre mode peut prendre trois valeurs :

* Context.MODE\_PRIVATE : le fichier créé n'est accessible que par l'application qui l'a créé.
* Context.MODE\_WORLD\_READABLE : le fichier créé peut être lu par n'importe quelle application.
* Context.MODE\_WORLD\_WRITEABLE : le fichier créé peut être lu et **modifié** par n'importe quelle application.

**Remarque**

* Appeler getDefaultSharedPreferences(Context context) revient à appeler getPreferences(MODE\_PRIVATE)
* Utiliser getPreferences(int mode) revient à utiliser

getSharedPreferences (NOM\_PAR\_DEFAUT, mode) avec NOM\_PAR\_DEFAUT un nom généré en fonction du package de l'application.

Ces méthodes retournent une instance de SharedPreference() pointant sur le fichier qui contient les valeurs de préférences.

Pour enregistrer des données (des couples clé-valeur) dans un **SharedPreferences**, il faut utiliser un objet de la classe [SharedPreference.Editor](http://developer.android.com/reference/android/content/SharedPreferences.Editor.html), en utilisant la méthode **edit().**

Pour ajouter des informations on utilise : **putX**(**String key, X value**) avec X le type de l'objet, key la clé et value la valeur associée.

Il faut ensuite valider les changements avec la méthode boolean commit() ou apply().

**SharedPreferences** sh=getSharedPreferences(mypreference, Context.MODE\_PRIVATE);

SharedPreferences.Editor editor = sh.edit();

editor.putString("key", "value");

editor.commit();

SharedPreferences pr = PreferenceManager.getDefaultSharedPreferences(this);

SharedPreferences.Editor editor = pr.edit();

editor.putString("Name","ali");

editor.apply();

Les préférences partagées ne fonctionnent qu'avec les objets de types primitifs : **boolean, float, int, long et String.**

Pour récupérer les données on utilise **getX(String key)**

SharedPreferences prefs = getSharedPreferences(mypreference, MODE\_PRIVATE);

String name = prefs.getString("nom", "nom par defaut");//" nom par defaut "est la valeur par défaut.

int idName = prefs.getInt("idName", 0); //0 est la valeur par defaut.

}

SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(this);

String name = pref.getString("Name", "");

**Exemple**

Pour conserver la couleur préférée de l'utilisateur, il n'est pas possible d'utiliser la classe Color puisque seuls les types de base sont acceptés, alors on pourrait conserver la valeur de la couleur sous la forme d'une chaîne de caractères :

public final static String FAVORITE\_COLOR = "fav\_color";

…

SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);

SharedPreferences.Editor editor = preferences.edit();

editor.putString(FAVORITE\_COLOR, "FFABB4");

editor.commit();

// On veut la chaîne de caractères d'identifiant FAVORITE\_COLOR

// Si on ne trouve pas cette valeur, "FFFFFF" est retournée

String couleur = preferences.getString(FAVORITE\_COLOR, "FFFFFF");

public void onClick(View v) {

String nom = ed1.getText().toString();

String ph = ed2.getText().toString();

String e = ed3.getText().toString();

SharedPreferences.Editor editor = sharedpreferences.edit();

editor.putString(Name, nom);

editor.putString(Phone, ph);

editor.putString(Email, e);

editor.commit();

Toast.makeText(MainActivity.this,"Thanks",Toast.LENGTH\_LONG).show();

}

}

D'autres méthodes de SharedPreferences.Editor :

removeString(String key), clear(), Map<String, ?> getAll().

[**https://www.tutlane.com/tutorial/android/android-shared-preferences-with-examples**](https://www.tutlane.com/tutorial/android/android-shared-preferences-with-examples)

**Des préférences prêtes à l'emploi**

Pour enregistrer les préférences, on propose une activité qui permet d'insérer différents paramètres.

Ce type d'activités s'appelle "PreferenceActivity". Chaque couple clé/valeur est créé automatiquement et sera récupéré automatiquement.

La création se fait en plusieurs étapes :

##### **Étape 1 : créer un fichier XML**

La racine de ce fichier doit être un **PreferenceScreen**.

On le définit souvent dans **res/xml/preference.xml**.

Il est possible de désigner des catégories de préférences. On peut ajouter des préférences avec le nœud PreferenceCategory.

Ce nœud est un layout, il peut donc contenir d'autre vues. Il ne peut prendre qu'un seul attribut, android:title, pour préciser le texte qu'il affichera.

**Exemple**

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

<PreferenceScreen >

<PreferenceCategory android:title="Réseau">

</PreferenceCategory>

<PreferenceCategory android:title="Luminosité">

</PreferenceCategory>

<PreferenceCategory android:title="Couleurs">

</PreferenceCategory>

</PreferenceScreen>

Après les catégories, il faut insérer des préférences :

* android:key : l'identifiant de la préférence partagée. C'est un attribut indispensable, ne l'oubliez jamais.
* android:title : titre principal de la préférence.
* android:summary : description de cette préférence.
* android:dependency : lie une préférence à une autre activité. Il faut y insérer l'identifiant android:key de la préférence dont on dépend.
* android:defaultValue : la valeur par défaut de cette préférence.

Il y a trois types de préférences : sous forme de case à cocher, d'un editText ou d'une liste,

1. **case à cocher**

La première conçue par une case à cocher avec CheckBoxPreference.

À la place du résumé standard, on peut déclarer un résumé qui ne s'affiche que lorsque la case est cochée, android:summaryOn, ou uniquement quand la case est décochée, android:summaryOff.

<CheckBoxPreference

android:key="checkBoxPref"

android:title="Titre"

android:summaryOn="Résumé quand sélectionné"

android:summaryOff="Résumé quand pas sélectionné"

android:defaultValue="true"/>

1. **EditTextPreference**

Le deuxième type de préférences consiste à permettre à l'utilisateur d'insérer du texte avec EditTextPreference avec un EditText

**Attributs :**

android:dialogTitle : permet de définir le texte de la boîte de dialogue,

android:negativeButtonText et android:positiveButtonText permettent respectivement de définir le texte du bouton à droite et celui du bouton à gauche dans la boîte de dialogue.

**Exemple**

<EditTextPreference

android:key="editTextPref"

android:dialogTitle="Titre de la boîte"

android:positiveButtonText="Je valide !"

android:negativeButtonText="Je valide pas !"

android:title="Titre"

android:summary="Résumé"

android:dependency="checkBoxPref" />

1. **ListPreference**

Le troisième type de préférences est un choix dans une liste d'options avec ListPreference. Dans cette préférence, on différencie ce qui est affiché de ce qui est réel.

Elle est pratique pour traduire son application en plusieurs langues.

**Attributs**

* android:dialogTitle, android:negativeButtonText et android:positiveButtonText.
* android:entries : présente les données de la liste que lira l'utilisateur
* android:entryValues : présente les données qui seront enregistrées.

La manière la plus facile pour remplir ces attributs se fait par une ressource de type array,

**Exemple**

<resources>

<array name="liste\_couleurs\_fr">

<item>Bleu</item>

<item>Rouge</item>

<item>Vert</item>

</array>

<array name="liste\_couleurs">

<item>blue</item>

<item>red</item>

<item>green</item>

</array>

</resources>

Pour utiliser cette liste

<ListPreference

android:key="listPref"

android:dialogTitle="Choisissez une couleur"

android:entries="@array/liste\_couleurs\_fr"

android:entryValues="@array/liste\_couleurs"

android:title="Choisir couleur" />

##### **Étape 2 : dans le Manifest**

Pour recevoir cette nouvelle interface graphique, nous avons besoin d'une activité. Il nous faut donc la déclarer dans le Manifest pour pouvoir y accéder avec les intents.

<activity

android:name=".PreferenceActivityExample"

android:label="@string/title\_activity\_preference\_activity\_example" >

</activity>

##### **Étape 3 : en Java**

Cette activité sera en fait de type PreferenceActivity. On peut la traiter comme une activité classique, mais au lieu de setContentView, on utilise addPreferencesFromResource :

**void addPreferencesFromResource(int preferencesResId)**

**Exemple**

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

**addPreferencesFromResource(R.xml.preference);** }

**Lancer cette activité par un intent dans l'activité principale.**

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

startActivity(**new** Intent(**this**, PrefeActivity.**class**));

}

Pour lire les valeurs des préférences à n'importe où dans l'application, on utilise getX(cle, valeur par defaut).

SharedPreferences SP = PreferenceManager.getDefaultSharedPreferences(this);

String strUserName = SP.getString("username", "NA");

boolean bAppUpdates =SP.getBoolean("checkBoxPref",false);

String downloadType =SP.getString("downloadType","1");

### Les fichiers

Il y a deux manières d'utiliser les fichiers : soit sur la mémoire interne du périphérique, soit sur une mémoire externe (par exemple une carte SD (Secure Digital).

Dans tous les cas, on part toujours du Context pour manipuler des fichiers.

#### En mémoire interne

#### Rappels sur l'écriture et la lecture de fichiers

Le cas le plus simple est de manipuler des flux d'octets, ce qui nécessite des objets de type FileInputStream pour lire un fichier et FileOutputStream pour écrire dans un fichier.

La lecture s'effectue avec la méthode int read() et on écrit dans un fichier avec void write(byte[] b).

Pour récupérer un FileOutputStream, utiliser la méthode

**openFileOutput (String name, int mode) avec name** le nom du fichier et mode le mode dans lequel ouvrir le fichier, il peut avoir :

* MODE\_PRIVATE : permet de créer (ou de remplacer) un fichier qui sera utilisé uniquement par l'application.
* MODE\_WORLD\_READABLE : créer un fichier que d'autres applications pourront lire.
* MODE\_WORLD\_WRITABLE : créer un fichier où même d'autres applications pourront lire et écrire.
* MODE\_APPEND : écrire à la fin d'un fichier préexistant, au lieu de créer un fichier.

**Exemple** : écrire mon pseudo dans un fichier

FileOutputStream output = null;

String userName = "ali";

try {

output = openFileOutput("fichier", MODE\_PRIVATE);

output.write(userName.getBytes());

if(output != null)

output.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

Pour lire dans un fichier, on peut utiliser la méthode **openFileInput** (String name).

FileInputStream input = openFileInput(nom\_fich);

int value;

StringBuffer lu = new StringBuffer();

while((value = input.read()) != -1) {

lu.append((char)value);

**D'autres méthodes**

File getFilesDir() : retrouver cet emplacement

boolean deleteFile (String name) : supprimer un fichier

String[] fileList():récupérer une liste des fichiers créés par l'application

##### **Travailler avec le cache**

Les fichiers normaux ne sont supprimés que si quelqu'un le fait.

Maisles fichiers sauvegardés avec le cache peuvent aussi être supprimés par le système d'exploitation afin de libérer de l'espace. C'est un avantage, pour les fichiers qu'on ne veut garder que temporairement.

Pour écrire dans le cache, il suffit d'utiliser la méthode File getCacheDir() pour récupérer le répertoire à manipuler.

#### En mémoire externe

Le problème avec le stockage externe, c'est qu'il n'existe aucune garantie que vos fichiers soient présents.

L'utilisateur peut à tout moment les supprimer ou avoir enlevé le périphérique de son emplacement. Cependant, cette fois la taille disponible de stockage est suffisante.

Pour pouvoir écrire sur un périphérique externe, il faut ajouter la permission WRITE\_EXTERNAL\_STORAGE au Manifest.xml :

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />.

Tout d'abord, pour vérifier que vous avez bien accès à la mémoire externe, on utilise la méthode statique String Environment.getExternalStorageState().

La chaîne de caractères retournée :

* Environment.MEDIA\_MOUNTED
* Environment.MEDIA\_MOUNTED\_READ\_ONLY) :

**Exemple**

if(Environment.MEDIA\_MOUNTED.equals(Environment.getExternalStorageState()))

// Le périphérique est bien monté

else

// Le périphérique n'est pas bien monté ou on ne peut écrire dessus

* Pour obtenir la racine des fichiers du périphérique externe, on utilise la méthode statique File Environment.getExternalStorageDirectory().
* Pour écrire des fichiers à un emplacement précis, utilisez le chemin : /Android/data/<votre\_package>/files/

En effet, les fichiers qui se trouvent à cet emplacement seront automatiquement supprimés dès que l'utilisateur effacera l'application.

La méthode boolean createNewFile() crée le fichier s'il n'existe pas.

**Exemple**

**try** {

File file = **new** File("c:**\\**newfile.txt");

**if** (file.createNewFile()) {

System.out.println("File is created!");

}**else** {

System.out.println("File already exists.");

}

  } **catch** (IOException e) {

e.printStackTrace();

}

**Exercice :** Enregistrer votre pseudo dans deux fichiers : en interne et en externe.

**En interne**

package com.example.exemple4;

public class MainActivity extends Activity {

private String nom\_fich = "fichier1.txt";

private String userName = "moi";

private Button mWrite = null;

private Button mRead = null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

mWrite = (Button) findViewById(R.id.button1);

mWrite.setOnClickListener(new View.OnClickListener() {

public void onClick(View pView) {

try {

FileOutputStream output = openFileOutput(nom\_fich, MODE\_PRIVATE);

output.write(userName.getBytes());

if(output != null)

output.close();

} catch (IOException e) {

e.printStackTrace();

} }

});

mRead = (Button) findViewById(R.id.button2);

mRead.setOnClickListener(new View.OnClickListener() {

public void onClick(View pView) {

try {

FileInputStream input = openFileInput(nom\_fich);

int value;

StringBuffer lu = new StringBuffer();

while((value = input.read()) != -1) {

lu.append((char)value);

}

Toast.makeText(MainActivity.this, "Interne : " + lu.toString(), Toast.LENGTH\_SHORT).show()

if(input != null)

input.close();

} catch (IOException e) {

e.printStackTrace();

} }

});

}}

**En externe**

public class MainActivity extends Activity {

private String PRENOM = "prenom.txt";

private String userName = "Apollidore";

private File mFile = null;

// On crée un fichier qui correspond à l'emplacement extérieur

mFile = new File(Environment.getExternalStorageDirectory().getPath() + "/Android/data/ " + getPackageName() + "/files/" + PRENOM);

public void onClick(View pView) {

try {

// Si le fichier est lisible et qu'on peut écrire dedans

if(Environment.MEDIA\_MOUNTED.equals(Environment.getExternalStorageState())

&& !Environment.MEDIA\_MOUNTED\_READ\_ONLY.equals(Environment.getExternalStorageState())) {

// On crée un nouveau fichier. Si le fichier existe déjà, il ne sera pas créé

mFile.createNewFile();

output = new FileOutputStream(mFile);

output.write(userName.getBytes());

if(output != null)

output.close();

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}}

});

public void onClick(View pView) {

try {

FileInputStream input = openFileInput(PRENOM);

int value;

// On utilise un StringBuffer pour construire la chaîne au fur et à mesure

StringBuffer lu = new StringBuffer();

if(Environment.MEDIA\_MOUNTED.equals(Environment.getExternalStorageState())) {

lu = new StringBuffer();

input = new FileInputStream(mFile);

while((value = input.read()) != -1)

lu.append((char)value);

Toast.makeText(MainActivity.this, "Externe : " + lu.toString(),Toast.LENGTH\_SHORT).show();

if(input != null)

input.close();

}

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} });

}

**public** void writeFileToInternalStorage(String fileName) {

String eol = System.getProperty("line.separator");

BufferedWriter writer = null;

**try** {

writer =

**new** BufferedWriter(**new** OutputStreamWriter(openFileOutput(fileName,

Context.MODE\_PRIVATE)));

writer.write("This is a test1." + eol);

writer.write("This is a test2." + eol);

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (writer != null) {

**try** {

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}}}

**public** void readFileFromInternalStorage(String fileName) {

String eol = System.getProperty("line.separator");

BufferedReader input = null;

**try** {

input = **new** BufferedReader(**new** InputStreamReader(openFileInput(fileName)));

String line;

StringBuffer buffer = **new** StringBuffer();

**while** ((line = input.readLine()) != null) {

buffer.append(line + eol);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (input != null) {

**try** {

input.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}}}