CAI-IP01 SMB116

Les **Ressources**

Les Ressources (1)

- Les ressources sont tous les éléments utilisés par l'application qui ne sont pas du code.
- Selon leur type, ces fichiers de ressources sont répartis dans des dossiers différents et leur contenu est codé différemment.

Dossier	Type de ressource	Type de contenu
res/anim	Animations	XML
res/drawable	Eléments graphiques	Bmp, Png, Gif, XML
res/color	Couleurs selon état (ColorStateList)	XML
res/layout	Disposition des éléments dans les vues	XML
res/menu	Menus	XML
res/values	Couleurs, Labels, Styles	XML
res/font	Fontes	XML
res/raw	Tous les autres types de ressources	indéfini

Les Ressources (2)

- **Utilité des Ressources:**
  - Description des interfaces graphiques en XML : Les **« Layout »**
  - Externalisation des labels par l'utilisation des **« Values »**. (utilisé pour l'internationalisation)
  - Référencement des couleurs et des styles sous forme de **« Values »**.
  - Référencement pour leur utilisation d'éléments graphiques : les **« Drawable »**
  - Exploitation de fichiers et contenus embarqués dans l'application : les **« Raw »**

Les Ressources (3)

- **Utilisation de ses Ressources par une application :**
  - Toutes les ressources d'une application sont packagées et un ID unique leur est affecté
  - Une classe nommée **R** est générée et référence tous ces ID (Integer) sous forme de constantes
  - Ces actions sont réalisées automatiquement à la compilation par **aapt (Adroid Asset Packaging Tool)**
  - Chaque constante est codée sous la forme:
    - Type de ressource: string, drawable, etc.
    - Nom de la ressource
    - Dans le code l'accès à une ressource s'effectue à partir de l'intance de `Resources` de l'application obtenue

    par la méthode: `getResources()`

    Exemple d' ID: R.string.hello, R.drawable.icone_application

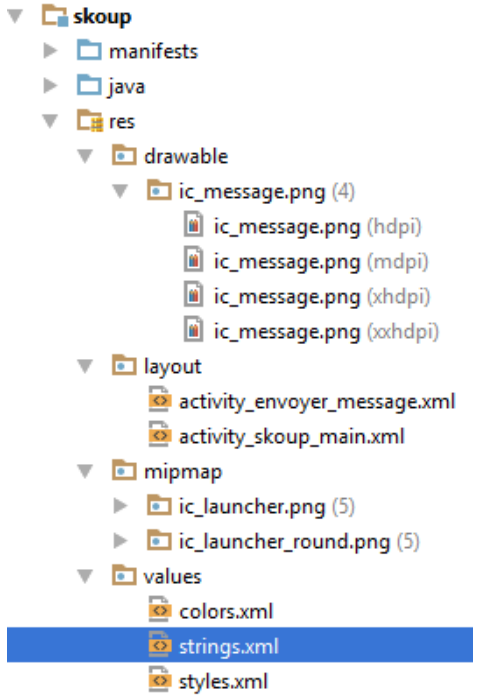
Les Layouts

- Une ressource de type Layout définit la présentation de l'UI d'une activité
- C'est un fichier au format XML comportant la description structurée des composants de l'UI

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="edu.cai.skoup.SkoupMain">

    <LinearLayout
        android:layout_width="368dp" android:layout_height="495dp"
        android:orientation="vertical«  app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent« app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent">

        <Button
            android:id="@+id/btnEnvoyerMessage«  android:layout_width="wrap_content«  android:layout_height="wrap_content"
            android:layout_gravity="center«  android:drawableTop="@drawable/ic_message«  android:onClick="onBtnEnvoyerMessage"
            android:text="@string/envoyer_message«  android:textAllCaps="false" />

    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

Les chaines externalisées (1)

- Les chaines externalisées sont stockées dans le fichier de ressource « string.xml » de type « values »




```xml
<resources>
    <!-- Chaine simple -->
    <string name="app_name">Skoup</string>
    <string name="envoyer_message">Envoyer un message</string>

    <!-- Message avec paramètres ($s pour chaine, $d pour décimal-->
    <string name="message_envoye">Le message pour %1$s a été envoyé en %2$d secondes</string>

    <!-- Tableau de chaines -->
    <string-array name="statuts">
        <item>Disponible</item>
        <item>Occupé</item>
        <item>Hors-ligne</item>
    </string-array>

    <!-- chaine singulier / plufriel -->
    <plurals name="message-sigulier_pluriel">
        <!-- quantity = ["zero" | "one" | "two" | "few" | "many" | "other"] -->
        <item quantity="one">message</item>
        <item quantity="other">messages</item>
    </plurals>
</resources>
```

Les chaines externalisées (2)

- Utilisation des chaines externalisées dans les layouts:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    . . .
          <Button   android:id="@+id/btnEnvoyerMessage"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:drawableTop="@drawable/ic_message"
                    android:onClick="onBtnEnvoyerMessage"
                    android:text="@string/envoyer_message"   ← Fait référence à la chaine externalisée
                    android:textAllCaps="false"
          />

    . . .
</android.support.constraint.ConstraintLayout>
```

L'utilisation dans les layouts est limitée aux chaines simples

Les chaines externalisées (3)

- **Utilisation des chaines externalisées dans le code :**
  - Utilisation des chaines simples:

```
String laChaineSimple = getResources().getString(R.string.envoyer_message);
```

  - Utilisation de chaines avec paramètres:

```
<string name="message_envoye">Le message pour %1$s a été envoyé en %2$d secondes</string>
```

    - Un formateur et des valeurs pour les paramètres sont nécessaires:

```
String laChaineParametree = String.format(getString(R.string.message_envoye), "toto", 2);
```

Les chaines externalisées (4)

- **Utilisation des chaines externalisées dans le code :**
  - Utilisation des tableaux de chaines :

```
String[] status = getResources().getStringArray(R.array.statuts);
```

  - Utilisation des chaines « Plurielles » : déclinaison d'une même chaine selon son nombre

```
<!-- chaine singulier / pluriel -->
<plurals name="message_singulier_pluriel">
    <!-- quantity = ["zero" | "one" | "two" | "few" | "many" | "other"] -->
    <item quantity="one">message</item>
    <item quantity="other">messages</item>
</plurals>
```

```
int nbMessage = 2;
String lblMessages = getResources().getQuantityString(R.plurals.message_singulier_pluriel, nbMessage);
```

Les « Drawable » (1)

- **Les « Drawable » sont toutes les ressources qui peuvent être dessinées (Icones, Images, etc.):**
- **Bitmap File :** fichier image (.png, .jpg ou gif)
- **Nine-Patch File :** fichier png avec des régions étirables pour permettre le redimensionnement de l'image
- **Layer List :** tableau d'image (des « Drawable ») dessiné les unes sur les autres (le plus grand index dessiné au-dessus)
- **State List :** liste de « Drawable » à afficher lorsqu'un composant est dans un état particulier (ex: bouton pressé)
- **Level List :** liste de « Drawable » à afficher en fonction d'une fourchette de valeurs numériques le « Level »
- **Transition Drawable :** Transition entre deux « Drawable »
- **Inset Drawable :** un « Drawable » qui contient un autre « Drawable » à l'intérieur avec une bordure spécifiée
- **Clip Drawable :** un « Drawable » qui contient un autre « Drawable » coupé si il dépasse.
- **Scale Drawable :** un « Drawable » qui contient un autre « Drawable » dont il adapte la taille.

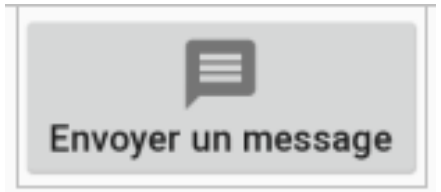
Les « Drawable » (2)

- Pour plus d'informations sur la liste de tous les types de « Drawable » :

https://developer.android.com/guide/topics/resources/drawable-resource.html

- Les « Drawable » de type image: 
- Utilisation d'un Drawable par un « Layout » ou une ressource XML :

```
<Button
    android:id="@+id/btnEnvoyerMessage"
    android:drawableTop="@drawable/ic_message"
    android:onClick="onBtnEnvoyerMessage"
    android:text="@string/envoyer_message"
 />
```



- Utilisation d'un Drawable dans le code :

```
Drawable imageEnvoyer = getResources().getDrawable(R.drawable.ic_message);
```

Les « Drawable » (2)

- Un « Drawable » particulier : La **« State List »** est définie sous la forme d'un XML définissant les « Drawable » à afficher en fonction de l'état d'un composant :

Exemple d'une « State List » pour un bouton:

- Ajout des images (les « Drawable ») : « ic_down », « ic_focus », « ic_message » 

- Définition du XML de la State List dans un fichier « bouton_state.xml »:

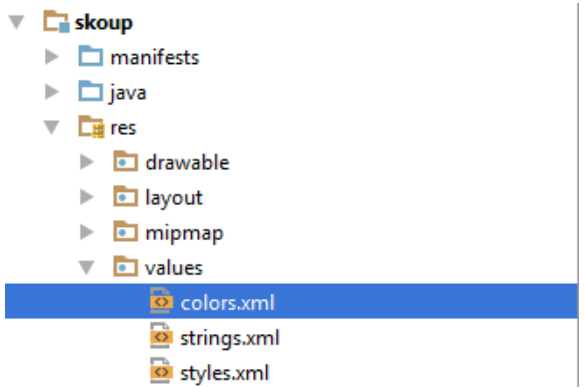
```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!- Image a afficher quand le bouton a le focus  -->
    <item android:state_focused="true" android:drawable="@drawable/ic_focus"/>
    <!- Image a afficher quand le bouton est pressé -->
    <item android:state_pressed="true" android:drawable="@drawable/ic_down"/>
    <!- autres cas -->
    <item android:drawable="@drawable/ic_message" />
</selector>
```

- Application de la « State List » au bouton par affectation de l'ID de la « State List » en tant que drawable:

```
android:drawableTop="@drawable/bouton_state"
```

Les Couleurs

- Les couleurs sont stockées dans le fichier de ressource « colors.xml » de type « values »




```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3f51b5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="color_skoup_background">#a8cc760d</color>
</resources>
```

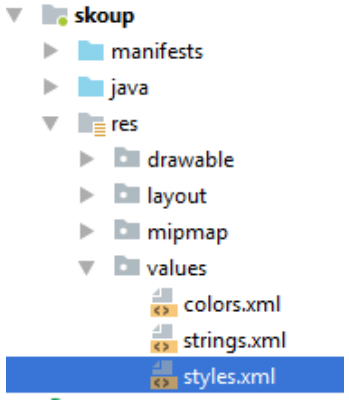
- Utilisation de la couleur: affectation de son ID à une propriété de type couleur

```
android:background="@color/color_skoup_background"
```

```
int couleurFond = getResources().getColor(R.color.color_skoup_background);

Drawable drawableCouleurFond = getResources().getDrawable(R.color.color_skoup_background, null);
((ConstraintLayout)findViewById(R.id.skoup_main_layout)).setBackground(drawableCouleurFond);
```

Les Styles et les Thèmes

- Les styles sont stockées dans le fichier de ressource « styles.xml » de type « values »
- Un style peut être créé à partir de rien ou bien hérité d'un style existant




```xml
<resources>
    <!-- Style hérité du style Theme.AppCompat.Light.DarkActionBar -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <!-- Style créé à partir de rien -->
    <style name="SkoupTheme" >
        <item name="colorPrimary">@color/color_skoup_background</item>
    </style>
</resources>
```

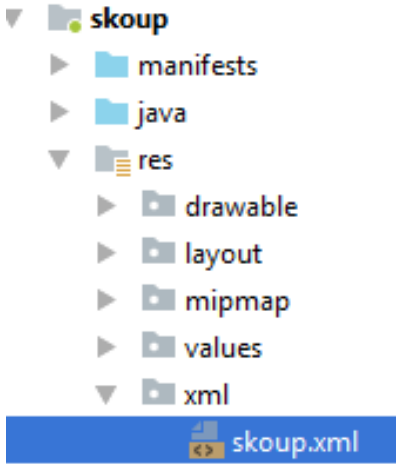
- Un style peut être référencé par l'application en utilisant son ID du type: **@style/nomStyle**

```xml
<TextView android:id="@+id/lblPour" android:text="Pour :" style="@style/SkoupTheme" />
```

https://developer.android.com/guide/topics/resources/style-resource.html

Les Fichiers XML (1)

- Les fichiers XML sont stockées dans le dossier de ressources « res/xml »
- Ils permettent de stocker tous les types d'informations sous forme textuelle




```xml
<?xml version="1.0" encoding="utf-8"?>
<skou-config xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- dernier utilisateur -->
    <skoup-username>toto</skoup-username>
</skou-config>
```

- Pour accéder à une ressource XML et la lire:

```
XmlPullParser xpp = getResources().getXml(R.xml.skoup);
```

Les Fichiers XML (2)

- Lire une ressource XML: (http://vogella.developpez.com/tutoriels/android/traitement-xml-xmlpullparser/)
  - XmlPullParser: simple parseur, lecture par élément
    - A chaque lecture d'un élément: le type est accessible par **`getEventType()`**
    - L'évènement '**END_DOCUMENT**' signifie la fin de l'exploration du document

```
try {
    String lastSkoupUser = "";
    XmlPullParser xpp = getResources().getXml(R.xml.skoup);
    // Tq non fin de document
    while (xpp.getEventType() != XmlPullParser.END_DOCUMENT) {
        // Si début d'élément
        if (xpp.getEventType() == XmlPullParser.START_TAG) {
            // Si élément "skoup-username"
            if (xpp.getName().equals("skoup-username")) {
                lastSkoupUser = xpp.nextText();
            }
        }
        xpp.next();
    }
    Log.i("XML",lastSkoupUser);
} catch (Throwable t) {
   Toast.makeText(this, "Echec lecture skoup.xml", 4000).show();
}
```

- Pour créer des fichiers XML (non ressource) on peut utiliser la classe **`XmlSerializer`**

Le choix des Ressources par Android (1)

- Selon la **« locale »** d'un équipement Android, les libellés d'une application devront être traduits
- En fonction de l'**orientation** de l'écran: un Layout particulier pourrait être plus adapté
- Selon la **taille** de l'écran, sa **densité** en pixel, son **rapport Hauteur/Largeur** etc. l'affichage d'une image particulière peut être plus adaptée.
- **Android** met à disposition un mécanisme permettant d'automatiser ces choix:
  - Les ressources par défaut sont stockées dans les dossiers comme: res/values, res/layout, res/drawable etc.
  - Pour définir des ressources alternatives selon la Locale, la taille ou l'orientation de l'écran etc. : il faut créer des dossiers dont le nom est suffixée par des « **Qualifieurs** » décrivant les conditions d'utilisation.

Le choix des Ressources par Android (2)

- Les types de qualifieurs sont: (https://android.developpez.com/cours/specialisation-ressources/)
  - Langue et région code langue (cf: ISO 639-1) + « _ » + code région (ISO 3166-1-alpha-2) : *fr, en, de, fr_CH, fr_FR etc.*
  - Code réseaux de télécommunication nationaux et commerciaux
  - Taille de l'écran : *small, medium, large et xlarge*
  - Format d'écran en terme de ratio (Screen Aspect): *long, notlong*
  - Orientation de l'écran: *port* pour portrait, *land* pour landscape ou *square* pour carré
  - Densité « pixellique » de l'écran: *ldpi* pour low dpi, *mdpi* pour medium dpi et *hdpi* pour high dpi
  - Et tant d'autres: contexte d'utilisation (*car* / *desk*), situation temporelle (*night/notnight)*, type d'écran tactile (stylus / finger / notouch), état du clavier (*keysoft / keyexposed / keyshidden*), type de clavier (*nokeys, qwerty, 12key*)

Le choix des Ressources par Android (3)