DEVELOPPEMENT MOBILE

1 Gestion de flux RSS

1.1 Objectif

L'objectif principal de cet exercice est de comprendre les principes permettant la réalisation d'une application Android exploitant les données fournit par un flux RSS pour les afficher directement dans une vue à partir des possibilités offertes par le SDK d'Android.
Afin de pouvoir réaliser le chargement des données XML à partir d'un flux RSS, nous exploiterons un parseur XML de type SAX. De plus, Android n'autorisant plus le chargement de données provenant d'Internet à partir du thread principal de l'interface graphique, nous exploiterons la classe **AsyncTask** pour réaliser ce chargement en tache de fond.

Exploitation d'un flux RSS

Nous allons exploiter un flux RSS fournit par la Nasa et dont l'adresse est la suivante:
https://www.nasa.gov/rss/image_of_the_day.rss.
Ce flux fournit l'image du jour proposée par la Nasa (***NASA Image of the Day***). Un flux RSS correspond à un dialecte XML dont les tags sont normalisés. Dans notre cas, nous allons nous intéresser aux contenus des <item>, en particulier aux tags : <title>, <description>, <pubDate> pour les contenus textuels et au tag <enclosure> pour l'url de l'image.
Voici un extrait du flux RSS correspondant:

```
<item>
        <title>Pine Island Glacier 2013: Nov. 10</title>
        <link>http://www.nasa.gov/content/goddard/pine-island-glacier-2013-nov-10 </link>
        <description>This MODIS image taken by NASA's Aqua satellite on Nov. 10, 2013, shows an
        ...
        track it and try to predict its path using satellite data.
        Image credit: NASA
        </description>
        <enclosure
        url="http://www.nasa.gov/sites/default/files/styles/946xvariable_height/public/pig_2013.1
        1.10.jpg?itok=MExtuRfV" length="482052" type="image/jpeg" />
        <guid isPermaLink="false">http://www.nasa.gov/content/goddard/pine-island-glacier-2013-nov-
        10</guid>
        <pubDate>Thu, 14 Nov 2013 12:00:00 EDT</pubDate>
        <source url="http://www.nasa.gov/rss/dyn/image_of_the_day.rss">NASA Image of the Day
        </source>
</item>
```

Pour afficher les données extraites nous prévoyons une vue de conception minimaliste. La description XML étant la suivante (remplacez le contenu du fichier main_activity.xml):

Code XML 1

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="fill_parent"
        android:layout_height ="fill_parent"
        android:id = "@+id/scrollView"
        android:tag ="MyScrollView"
        tools:context = ".MainActivity" >

    <LinearLayout
        android:layout_width ="match_parent"
        android:layout_height= "wrap_content"
        android:id ="@+id/linearLayout"
        android:tag = "MyLinearLayout"
        android:orientation="vertical">

        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:weightSum="1">

            <Button
                style="?android:attr/buttonStyleSmall"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/image_precedente"
                android:id="@+id/button"
                android:layout_weight="0.5" />

            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/image_suivante"
                android:id="@+id/button2"
                android:layout_weight="0.5" />
        </LinearLayout>

        <TextView
            android:id = "@+id/imageTitle"
            android:layout_width= "wrap_content"
            android:layout_height= "wrap_content"
            android:layout_marginBottom = "10dp"
            android:tag = "MyImageTitle"
            android:text="@string/test_image_title" />
```

```
    <TextView
        android:id="@+id/imageDate"
        android:layout_width ="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dp"
        android:tag="MyImageDate"
        android:text= "@string/test_image_date" />

    <ImageView
        android:id= "@+id/imageDisplay"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:contentDescription= "@string/description_image"
        android:adjustViewBounds = "true"
        android:tag = "MyImageDisplay"
        android:src ="@mipmap/ic_launcher"/>

    <TextView
        android:id="@+id/imageDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:tag="MyImageDescription"
        android:text= "@string/test_image_description"/>
  </LinearLayout >
</ScrollView>
```

Trois ***TextView*** pour le titre, la date de parution et la description et une ***ImageView*** pour l'image. Nous englobons le tout dans un ***ScrollView*** afin que si la quantité de données affichées dépasse la taille de l'écran, il soit possible de faire scroller cet affichage pour en voir la totalité. On peut remarquer que les différents composants de type View ont été taggés pour la suite (android:tag ="...").
Nous créons ensuite une classe MyRSSsaxHandler qui hérite de la classe DefaultHandler. Cette classe fait partie du paquetage classique org.xml.sax. Un ensemble d'attributs sont définis pour cette classe afin de savoir où on est situé dans le flux RSS et sauvegarder les données lues:

```
private String url = null ;// l'URL du flux RSS à parser
// Ensemble de drapeau permettant de savoir où en est le parseur dans le flux XML
private boolean inTitle = false ;
private boolean inDescription = false ;
private boolean inItem = false ;
private boolean inDate = false ;
// L'image référencée par l'attribut url du tag <enclosure>
private Bitmap image = null ;
private String imageURL = null ;
// Le titre, la description et la date extraits du flux RSS
private StringBuffer title = new StringBuffer();
private StringBuffer description = new StringBuffer();
private StringBuffer date = new StringBuffer();
private int numItem = 0; // Le numéro de l'item à extraire du flux RSS
private int numItemMax = - 1; // Le nombre total d'items dans le flux RSS
```

La méthode **setUrl** permet de fixer l'URL à parser.

```
public void setUrl(String url){
   this.url= url;
}
```

La méthode processFeed initialise le parser XML, ouvre un flux d'entrée à partir de l'adresse Internet du flux RSS (`new URL(url).openStream();`), réalise sa lecture (`reader.parse(...)`), puis charge l'image correspondante (`getBitmap`).

```
public void processFeed(){
   try {
      numItem = 0; //A modifier pour visualiser un autre item

      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser parser = factory.newSAXParser();
      XMLReader reader = parser.getXMLReader();
      reader.setContentHandler(this);
      InputStream inputStream = new URL(url).openStream();
      reader.parse(new InputSource(inputStream));
      image = getBitmap(imageURL);
      numItemMax = numItem;
   }catch(Exception e) {
      Log.e("smb116rssview", "processFeed Exception" + e.getMessage());
   }
}
```

Pour extraire les informations du flux XML (i.e. RSS) nous utilisons la méthode **startElement** qui est appelée à chaque rencontre de tag XML ouvrant et la méthode **characters** qui permet d'extraire les données textuelles entre deux tags XML.

Donnez le code correspondant à l'extraction des données du flux RSS pour les éléments : **title**, **description**, **pubDate** et **enclosure** (pour ce dernier, il faut réaliser l'extraction de l'URL de l'image, si disponible) pour un item donné (correspondant à numItem). Pour vérifier le bon fonctionnement de l'extraction, vous afficherez les données extraites dans le LogCat(voir suite pour la problématique d'un chargement asynchrone). Le prototype de startElement étant le suivant:

```
public void startElement(String uri, String localName, String qName, Attributes
        attributes) throws SAXException {

    //. . .
}
```

Vous devrez également définir la méthode **characters** pour récupérer les contenus textuels correspondant à un tag:

```
public void characters(char ch[], int start, int length){
   String chars = new String(ch).substring(start, start+length);
        // . . .
}
```

La lecture d'une image à partir de son URL sera réalisée en exploitant la classe **HttpURLConnection**.
L'extraction d'un bitmap à partir d'un stream sera réalisée à partir de la méthode **BitmapFactory.decodeStream().** Compléter le code de la méthode suivante pour réaliser ce travail:

```
public Bitmap getBitmap(String imageURL){
{
        //. . .
}
```

Afin de pouvoir télécharger des données à partir d'Internet, il faut donner à l'application la permission de réaliser ce type de tâche. Pour cela, il faut modifier le fichier AndroidManifest.xml de la façon suivante:

```
<uses-sdk  android:minSdkVersion ="8"   android:targetSdkVersion  = "18" />
<uses-permission android:name="android.permission.INTERNET"/>
```

En ajoutant la permission INTERNET.
Nous ne pouvons cependant pas ajouter dans la méthode **onCreate** de l'activité de notre application le chargement direct des données à partir du flux RSS de la façon suivante:

```
MyRSSsaxHandler handler = new MyRSSsaxHandler();
handler.setUrl("https://www.nasa.gov/rss/image_of_the_day.rss");
handler.processFeed(); //ne pas faire cela ici !!!
```

Vérifiez que le code ci-dessus produit bien une exception!

La raison est que depuis la version 3.0 d'Android, il n'est plus possible de télécharger des données à partir du thread principal (UI thread Android) d'une activité. Cette restriction a pour raison d'éviter de geler l'interface graphique de l'application Android pendant l'attente de la réception des données par le réseau qui peut aller de quelques millisecondes à plusieurs secondes. Pour éviter ce problème, la solution est de réaliser le chargement des données par le réseau de façon asynchrone.

Le SDK d'Android propose une classe **AsyncTask** pour faciliter cette gestion asynchrone en réalisant la gestion des threads correspondants (vous pouvez vous baser sur l'exemple du cours). Il suffit de définir les besoins spécifiques à l'application en créant votre propre classe héritant de celle-ci. Je vous donne directement ci-dessous le code correspondant dans notre cas.

```
public class DownloadRssTask extends AsyncTask<MyRSSsaxHandler, Void, MyRSSsaxHandler> {
private MainActivity monActivity;
public DownloadRssTask(MainActivity monActivity){
      this.monActivity = monActivity;
   }
protected MyRSSsaxHandler doInBackground(MyRSSsaxHandler... handler){
      handler[0].processFeed();
      return handler[0];
   }
/** The system calls this to perform work in the UI thread and delivers
 * the result from doInBackground() */
protected  void  onPostExecute(MyRSSsaxHandler handler)
{
// Mise à jour des données de notre vue à partir
// du titre, date, image et description lus
monActivity.resetDisplay(handler.getTitle(),handler.getDate(),handler.getImage(),
handler.getDescription().toString());
}
}
```

Et son exploitation dans la méthode **onCreate** de l'activité.

```
handler = new MyRSSsaxHandler();
handler.setUrl("https://www.nasa.gov/rss/image_of_the_day.rss");
Toast.makeText(this,"chargement image :"+handler.getNumber(), Toast.LENGTH_LONG).show();
new DownloadRssTask(this).execute(handler);
```

La classe générique **AsynTask** a trois types génériques comme paramétrage. Ils ont dans l'ordre la signification suivante:

1. Le type de donnée passé à la méthode réalisant le travail en tâche de fond.
2. Le type de donnée exploité pour publier l'avancement du travail[1] en tâche de fond.
3. Le type de résultat produit par le travail en tâche de fond et permettant l'actualisation des données affichées par la vue.

Si une de ces données n'est pas utile, il suffit d'exploiter la classe vide **Void**.

Il existe 4 étapes lors de l'exploitation de la classe **AsyncTask** (voir la documentation pour les détails), mais nous n'en utiliserons que deux, car actuellement nous n'affichons pas de progression du chargement. Le premier **doInBackground**,comme son nom l'indique, réalise le traitement en tâche de fond à partir de données du premier type générique passé par l'appel de la méthode execute dans le thread UI. La méthode execute pouvant être appelée plusieurs fois, les données passées à la méthode doInBackground sont alors accessibles dans un tableau afin de pouvoir prendre en compte l'ensemble des appels à la méthode execute. La seconde **onPostExecute** est appelée à la fin de **doInBackGround** et est exécuté dans le thread UI, ce qui lui permet d'agir sur l'affichage de la vue comme le fait notre méthode **resetDisplay** à partir de la donnée générique n°3.
Dans notre cas elle est de même type et c'est le même objet handler qui est partagé par les deux threads, mais sans aucun problème de synchronisation dû aux particularités de fonctionnement des objets de la classe **AsyncTask**.
La méthode **onPostExecute** ne pouvant être appelée qu'après la fin de la méthode **doInBackground**, bien que n'étant pas traitée dans le même thread.
La méthode **resetDisplay** doit être ajoutée à la classe **MainActivity**. Elle affichera les informations lues en modifiant le text (**setText**) ou l'image (**setImageBitmap**) des éléments de type View correspondant au layout XML de l'activité. Vous devrez exploiter la méthode **findViewById** afin de pouvoir réaliser ces modifications. Complétez le prototype ci-dessous

```
public void resetDisplay(String title, String date, Bitmap image, String description){
// . . .
}
```

La **ScrollView** vous permet de vous déplacer verticalement dans la vue.

Quelques exercices:

1. Ajoutez deux boutons en début de vue, alignés horizontalement, permettant le passage d'une image à la suivante (ou précédente). Les boutons feront parties de la **ScrollView**. Ils pourront donc disparaître lorsque l'on fait scroller le contenu du flux affiché vers le bas.

2. On souhaite que les deux boutons soit toujours visibles en haut de l'écran, tout en pouvant scroller sur les données affichées pour un item du flux RSS. Proposez une solution permettant d'avoir ces deux modalités.
3. Adaptez et testez votre programme avec un autre flux RSS (utilisez http://www.leparisien.fr/services/rss/ ou http://ctrlq.org/rss/ pour trouver un flux RSS vous convenant).

[1] Utile pour afficher une barre de progression par exemple.

4. Actuellement le passage d'un item à un autre, nécessite le rechargement du flux XML afin d'extraire le contenu de l'item n°i. Proposez une nouvelle version dans laquelle l'ensemble des données est chargé en mémoire lors du premier parcours du flux RSS. Le changement d'item en appuyant sur un bouton ne nécessitera alors plus de faire de rechargement. Le flux RSS pouvant avoir été modifié lors d'une longue utilisation de l'application, vous ajouterez au menu la possibilité de faire un rafraîchissement du contenu sauvegarder en mémoire.

5. Pensez à stocker les données dans une base de données qui sera utiliée lorsque la connexion internet sera coupée.