Investir l'écran d'accueil Android avec les AppWidgets

Cet article est rédigé par Android2EE, Consulting, Expertise et Formation Android.

Il est associé à un tutoriel vous montrant comment mettre en place un AppWidget pour votre écran d'accueil. Pour plus d'information (tutoriels, ebooks, formations), une seule adresse :

Android2EE : http://www.android2ee.com.

1 Introduction

L'objectif de cet article est de vous présenter comment concevoir et développer des AppWidgets, ces composants graphiques qui s'installent sur l'écran d'accueil de l'appareil.

Le développement de tels éléments graphiques est profondément différent du développement graphique au sein des activités. Le Design Pattern mis en place n'a rien à voir, il va falloir changer sa philosophie, ses méthodes et ses automatismes en commençant par le design de votre application.

Ce n'est pas les composants graphiques qui changent mais la façon dont vous allez pouvoir interagir avec eux.

Cet article explique dans quel état d'esprit vous devez abordez votre conception, quels sont les Design Patterns à utiliser, comment les mettre en place et bien sûr comment développez tout ça pour fournir un code propre.

Le premier chapitre vous explique les principes fondamentaux, comment et où déclarer les différents éléments d'un AppWidget. À la fin de ce chapitre vous aurez compris comment mettre en place votre projet, comment décrire votre IHM et quels sont les règles graphiques à respecter.

Le second chapitre rentre dans les détails du code Java. Un premier paragraphe vous présente les méthodes du cycle de vie de l'AppWidget, vous expliquant son fonctionnement de manière globale. Le deuxième paragraphe vous montre comment mettre à jour les interfaces graphiques de vos AppWidgets, en particulier comment utiliser les RemoteViews et l'AppWidgetManager. Le troisième paragraphe va vous apprendre à interagir avec l'utilisateur au travers de vos AppWidgets. Toute la communication par Intent entre vos instances d'AppWidget et votre classe Java vous sera expliquée. Enfin, le dernier paragraphe vous montrera comment utiliser l'AlarmManager pour gérer finement la mise à jour de vos AppWidget.

Enfin, cet article est associé à un tutoriel de manière forte. Il est fortement préconisé de télécharger le tutoriel avant de lire l'article de manière à pouvoir le lire tout en allant regarder le tutoriel. Cela vous permet d'avoir une vision appliquée des différents concepts expliqués dans l'article.

Ainsi vous pouvez télécharger le tutoriel ici : AppWidgetTuto sur Android2EE.

Et celui-ci met en place le widget suivant :

2 Vision globale et principes fondamentaux

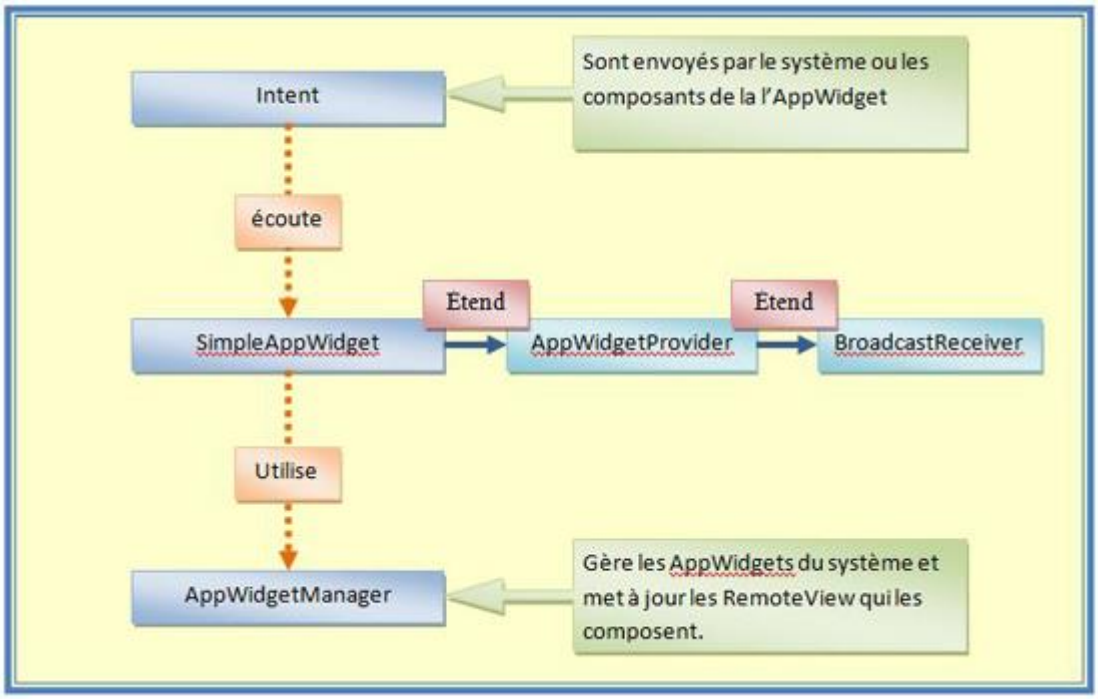
Bien qu'étant une interface graphique, l'AppWidget doit être considéré comme une image ne pouvant interagir avec l'utilisateur de manière usuelle ; vous ne pouvez pas récupérer de pointeur vers un élément graphique du widget. C'est un type de programmation complètement différent qui doit être mis en place pour interagir avec votre widget, que ce soit une interaction utilisateur ou une mise à jour programmatique.

En fait un AppWidget est un IntentReceiver particulier, pas une activité ! Il n'a pas de cycle de vie, est créé à la réception d'un Intent et est détruit dès que l'Intent est consommé. Il ne peut donc pas stocker de valeurs ; les attributs sont inutiles ou doivent toujours être considérés comme *null* à la réception d'un nouvel Intent.

Un AppWidget contient un RemoteView qui n'est autre que l'ensemble des composants graphiques affichés par l'AppWidget. On peut placer sur chaque élément graphique de la RemoteView un écouteur de clic qui lance un Intent quand il est déclenché.

Un AppWidget peut posséder plusieurs instances car il peut être installé à plusieurs endroits dans l'écran d'accueil. Chaque installation pointe vers une unique instance de l'AppWidget.

Le schéma suivant donne une bonne vision de ce qu'est un AppWidget et de son fonctionnement.



Tout d'abord, il hérite d'AppWidgetProvider, qui lui-même hérite de BroadCastReceiver. C'est donc un BroadCastReceiver que les équipes d'Android ont customisé pour nous faciliter leur utilisation dans le cadre des AppWidgets.

Ensuite, un AppWidget n'interagit qu'avec des Intents, ceux-ci peuvent être déclenchés par le système ou par les composants graphiques de l'AppWidget eux-mêmes.

Pour mettre à jour les composants graphiques de l'AppWidget, l'objet AppWidgetManager est utilisé. On ne met pas à jour les composants de l'AppWidget, on les recrée.

3 Déclaration d'un AppWidget

L'AppWidget se déclare dans le manifeste de votre application, le fichier AndroidManifest.xml. Il se définit au moyen d'un fichier xml dans le dossier res\xml\. Son interface graphique initiale est décrite dans un fichier de layout classique (res\layout\widget_layout.xml).

3.1 Déclaration dans le manifeste

La déclaration d'un AppWidget dans le AndroidManifest.xml est quasi-identique à celle d'un BoradcastReceiver :

```
<!-- Broadcast Receiver that will process AppWidget updates -->
      <receiver
         android:label="@string/widget_name"
         android:name=".SimpleWidget" >
         <intent-filter >
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
         </intent-filter>
```

```
        <intent-filter >
            <action android:name="com.android.tuto.widget.simple.sample1.click1" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/widget_properties_definition" />
    </receiver>
```

Où :

- le label est le nom de votre widget qui sera affiché à l'utilisateur ;
- son name est la classe qui gère ce widget ;
- les intent-filter sont les intentions qui seront écoutées par votre widget et consommées par la classe définie par la balise name ;
- le meta-data, lui, définit deux choses :
    - le name qui est toujours le même pour vos AppWidgets,
    - la resource qui pointe vers le fichier xml décrivant votre AppWidget.

Maintenant, votre application déclare posséder un AppWidget pour l'écran d'accueil.

3.2 Définition dans un fichier xml

Vous devez définir les attributs de votre AppWidget dans un fichier xml de la manière suivante :

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="250dp"
    android:minHeight="110dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical">
</appwidget-provider>
```

Où :

- minWidth et minHeight sont les tailles minimales de votre widget, nous reviendrons sur cette notion de dimension ;
- updatePeriodMillis définit la période de rafraichissement de votre widget. Elle ne peut être inférieure à 30 minutes. Gardez en tête qu'une mise à jour trop fréquente épuisera la batterie, il est préconisé de le faire au plus une fois par heure. Un chapitre expliquera comment mettre en place des mises à jour plus efficaces ;
- previewImage permet de définir l'icône à afficher à l'utilisateur lorsque celui-ci ajoute un widget à l'écran d'accueil (introduit par Android 3.0) ;
- initialLayout décrit le layout qui sera utilisé pour construire le widget ;
- configure vous permet de définir une activité qui sera lancée à l'installation du Widget pour permettre à l'utilisateur d'initialiser des variables ou des préférences ;
- resizeMode permet de définir comment le widget peut être redimensionné (introduit par Android 3.0).

3.3 Description de l'interface graphique avec un fichier de layouts

Avant de vous lancer dans une interface graphique ultra-complexe en pensant que tout fonctionne à l'identique, vous devez savoir que seuls les composants suivants sont utilisables dans une AppWidget :

Vous pouvez utiliser les layouts suivants :

- FrameLayout ;
- LinearLayout ;
- RelativeLayout.

Et les composants graphiques suivants:

- AnalogClock ;
- Button ;
- Chronometer ;
- ImageButton ;
- ImageView ;
- ProgressBar ;
- TextView ;
- ViewFlipper ;
- ListView ;
- GridView ;
- StackView ;
- AdapterViewFlipper.

Attention : vous ne pouvez pas utiliser les classes filles de ces composants.

À cette restriction près, vous construisez votre composant graphique de manière usuelle, comme pour une activité (ne mettez pas de listeners dans le xml, cela ne sert à rien, ils ne seront pas appelés).

Un exemple de fichier de layout :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical"
   android:background="#FF00FF00">

   <TextView
      android:id="@+id/textView"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/app_name" />

   <ImageButton
      android:id="@+id/imageButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/ic_android2ee_violet" >
   </ImageButton>

   <ImageView
      android:id="@+id/imageView"
      android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:src="@drawable/ic_android2ee_blc" />

</LinearLayout>
```

3.4 Règles graphiques des AppWidgets

Le document de référence concernant les règles graphiques à utiliser pour les AppWidgets se trouve ici : http://developer.android.com/guide/practices/ui_guidelines/widget_design.html.

3.4.1 Taille

Tout d'abord, il faut savoir que l'écran d'accueil est divisé en une grille de cellules. Pour les smartphones, cette grille est majoritairement de quatre lignes par quatre colonnes et sur les tablettes de huit colonnes par sept lignes. Quand votre widget va être ajouté à l'écran d'accueil, il sera étiré pour occuper le nombre de cellules minimal (horizontalement et verticalement) de manière à respecter l'espace défini par les variables MinWidth et MinHeight que vous avez données dans sa définition.

Ainsi, il est extrêmement important de penser en nombre de cellules requises pour afficher votre widget, puis de convertir ce nombre en MinWidth et MinHeight en utilisant la règle suivante :

Si n est le nombre de cellules à afficher, alors la taille associée en dp est (70×n)-30. Autrement dit, le tableau suivant s'applique

Nombre de cellules	Taille associée (dp)
1	40
2	110
3	180
4	250
n	(70×n)-30

Donc un widget nécessitant au moins trois cellules horizontalement et quatre verticalement définira sa taille minimale ainsi :

```
android:minWidth="180dp"
android:minHeight="250dp"
```

La documentation officielle donne aussi comme autre forme de calcul de donner la vraie taille minimale de votre widget. Pour cela, il faut désigner votre widget dans sa taille minimale et calculer cette taille. Il sera automatiquement étiré pour remplir le nombre de cellules. Qui dit étirement dit image de type Nine-Patch (soit dit en passant).

Pour GingerBread, il était couramment admis que les cellules avaient une dimension de 74*74 dp (density independant pixel) et 2 de marge… Mais ça c'était avant.

Et puis une autre remarque, depuis HoneyComb, les Widgets sont redimensionnables (en largeur/hauteur). Du coup, MinHeight et MinWidth deviennent en fait les tailles par défaut. Pour définir des tailles minimales qui soient respectées, il faut utiliser MinResizeHeight et MinResizeWidth. Bien sûr, cela ne marche pas pour un SDK inférieur à Honey Comb…

3.4.2 Marge

Il faut définir des marges pour votre widget et, mauvaise nouvelle, cela dépend de la version du SDK de l'appareil sur lequel va être ajouté votre widget. En effet depuis ICS, les marges sont automatiquement ajoutées.

Pour pallier ce problème, voilà comment il faut faire (toujours d'après le site Android) :

1. Définissez votre balise sdk-target à 14 (ou plus) dans le AndroidManifest.xml. Attention, cela implique beaucoup de choses pour votre projet, réfléchissez-y bien ;
2. Dans le fichier de layout de votre widget définissez comme marge pour votre layout parent **android:padding="@dimen/widget_margin" ;**
3. Ensuite, créer deux fichiers dans votre dossier de ressource values :
   - **res/values/dimens.xml** qui contient la ligne suivante :

```
<dimen name="widget_margin">8dp</dimen>
```

   - **res/values-v14/dimens.xml** qui contient la ligne suivante :

```
<dimen name="widget_margin">0dp</dimen>
```

3.4.3 Background

Comme je vous l'ai dit, votre widget sera étiré pour remplir l'espace qui lui est alloué. Le corollaire direct est que votre image de fond doit être une image de type nine-patch (possédant des zones définies comme étant à étirer).

Enfin sachez que Google met à votre disposition son templatePack graphique, qui vous fournit les briques graphiques élémentaires pour de designer votre widget (si vous êtes graphiste). Vous le trouverez ici : http://developer.android.com/shareables/app_widget_templates-v4.0.zip.

4 Code Java de l'AppWidget

Passons aux choses sérieuses et voyons comment fonctionne ce composant.

Il faut toujours garder en tête que :

- votre AppWidget n'est pas une Activité mais un BroadcastReceiver ;
- votre AppWidget étend AppWidgetProvider ;
- votre AppWidget peut être installée plusieurs fois dans l'écran d'accueil, on parle donc d'instance de votre AppWidget.

4.1 Les méthodes natives de l'AppWidgetProvider

L'AppWidget possède les méthodes de réception d'Intents suivantes :

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {}
public void onDeleted(Context context, int[] appWidgetIds) {}
public void onDisabled(Context context) {}
public void onEnabled(Context context) {}
public void onReceive(Context context, Intent intent) {}
```

Chacune de ces méthodes est appelée à la réception d'un Intent spécifique.

4.1.1 onUpdate

La méthode onUpdate est appelée lors de la réception de l'Intent `ACTION_APPWIDGET_UPDATE`. Cet Intent est envoyé par le système dans les cas suivants :

- sortie de veille/redémarrage de l'appareil ;
- première installation d'une instance de votre AppWidget ;
- périodiquement en fonction du paramètre `android:updatePeriodMillis` que vous avez défini pour votre widget.

Une exception, toutefois, si vous avez associé à votre AppWidget une activité d'initialisation (AppWidget Configuration Activity), la méthode onUpdate ne sera pas appelée lors de l'installation de l'appareil, c'est l'activité qui sera exécutée.

Il faut donc dans cette méthode initialiser votre widget. Le mot update est mal choisi.

4.1.2 onDeleted

Cette méthode est appelée lors de la réception de l'Intent `ACTION_APPWIDGET_DELETED`. Cet Intent est envoyé par le système lors de la suppression de l'une de vos instances de votre AppWidget de l'écran d'accueil.

4.1.3 onDisable et onEnable

La méthode onEnable est appelée la première fois qu'une instance de votre AppWidget est installée. Si l'utilisateur ajoute de nouvelles instances, cette méthode ne sera pas rappelée. Si vous avez besoin d'installer une base de données (ou ce genre de choses), c'est dans cette méthode qu'il faut le faire. Elle est appelée lors de la réception de l'Intent `ACTION_APPWIDGET_ENABLED`.

La méthode onDisable est appelée lorsque la dernière instance de votre AppWidget est désinstallée. Si vous avez installé une base de données, c'est ici qu'il faut la détruire (ou demander à l'utilisateur s'il souhaite cette destruction). Elle est appelée lors de la réception de l'Intent `ACTION_APPWIDGET_DISABLED`.

4.1.4 onReceive

La méthode onReceive est la méthode native de récupération des Intents. Les Intents précédents sont filtrés et reventilés vers les méthodes ci-dessus, vous ne les recevrez pas dans la méthode onReceive.

La méthode onReceive va vous être utile pour écouter les Intents que vous vous envoyez, typiquement ceux liés à l'écoute des clics utilisateurs.

4.2 Manipulation du AppWidgetManager et des RemoteViews

Toute l'interaction avec votre widget s'effectuera au travers de l'AppWidgetManager et de la RemoteView qui affiche (s'il s'agit de "l'AppWidgetManager et de la RemoteView" ensemble alors "qui affichent", sinon il faut revoir la phrase) vos composants graphiques.

4.2.1 RemoteViews

Les RemoteViews sont des objets qui permettent de décrire une vue qui sera hébergée par le processus d'une autre application. La description xml de la vue est utilisée par la RemoteViews pour reconstruire une View (via un LayoutInflater) et la RemoteViews sert d'interface pour manipuler ces composants graphiques. Remarquez que la RemoteViews n'hérite que de la classe Objet, en d'autres termes, ce n'est pas une Views. D'autre part, ne vous amusez pas à contourner la RemoteViews pour manipuler directement vos interfaces graphiques, à part si

vous avez compris toutes les problématiques associées à faire tourner votre code dans un autre processus (notion de droit, de fuite mémoire, de programmation concurrentes…). Soit vous travaillez chez Google sur le code source du SDK, soit pas. Si vous n'y travaillez pas, utilisez la RemoteViews pour modifier le comportement des composants graphiques qui la composent.

La création d'une RemoteView s'effectue de la manière suivante :

```
RemoteViews views;
views = new RemoteViews(context.getPackageName(), R.layout.widget_layout_two);
```

Le constructeur prend en paramètre le nom du package qui contient votre classe AppWidgetProvider qui gère votre widget et le layout qui contient la description de l'IHM à construire.

Pour manipuler les composants graphiques au sein de votre RemoteViews vous ne pouvez pas utiliser la méthode findViewsById, en fait vous ne devez pas récupérer de pointeur vers ces composants. Il faut utiliser les méthodes de l'objet RemoteViews qui vous permettent d'effectuer ces opérations. Quelques exemples :

```
//Affecte un texte à un TextView (équivalent de TextView.setText)
views.setTextViewText(R.id.textView, context.getString(R.string.app_name) + " : " + i);
//Appelle la méthode passée en paramètre (setClickable) sur le composant en lui passant le paramètre true
views.setBoolean(R.id.checkBox, "setClickable", true);
//Appelle la méthode passée en paramètre (setClickable) sur le composant en lui passant le paramètre true
views.setString(R.id.imageButton, "setText", "La string a afficher");
//Définit la couleur du Texte
views.setTextColor(R.id.textView, Color.GREEN);
//Affiche ou cache un composant
views.setViewVisibility(R.id.imageButton, true);
```

Comme on le voit dans ces exemples, on peut faire beaucoup de choses sur les composants, mais pas non plus tant que ça. Le pattern est toujours le même, on utilise les méthodes de la RemoteViews pour modifier le comportement des composants graphiques qui la composent.

4.2.2 AppWidgetManager

L'AppWidgetManager, quant à lui, nous permet de récupérer la liste des identifiants des instances de notre AppWidget installées sur l'écran d'accueil et de les mettre à jour.

```
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
ComponentName componentName = new ComponentName(context, SimpleWidget.class);
int[] appWidgetIds = appWidgetManager.getAppWidgetIds(componentName);
final int N = appWidgetIds.length;
…
// Mise à jour de la vue de l'AppWidget dont l'identifiant est appWidgetId
appWidgetManager.updateAppWidget(appWidgetId, views);
```

J'attire votre attention sur AppWidgetManager.*getInstance*(context) qui implique que si vous souhaitez utiliser l'AppWidgetManager dans une méthode, vous devez passer à cette méthode le contexte, sinon vous ne retrouverez jamais votre AppWidgetManager.

4.2.3 Exemple d'utilisation

Ce paragraphe va vous permettre de comprendre l'utilisation de la RemoteViews et de l'AppWidgetManager dans le contexte d'un AppWidgetProvider.

Tout d'abord une implémentation de la méthode onUpdate. Cette implémentation est primordiale à comprendre pour mettre en place un AppWidget. Elle vous explique comment créer et mettre à jour vos AppWidgets. Elle est quasiment générique et se retrouve dans pratiquement tous les AppWidget créées. En effet, il faut dans tous les cas, retrouver le nombre d'instances de vos AppWidget installées, puis pour chaque instance vous devez la recréer puis la mettre à jour en utilisant l'AppWidgetManager. Donc prenez un temps particulier pour bien la comprendre car vous allez la réutiliser.

Voici le code :

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
//Nombre d'instances installées
final int N = appWidgetIds.length;
//L'objet RemoteViews
RemoteViews views;
// identifiant de l'instance courante
int appWidgetId

// Mise à jour de toutes les instances du widget installées (donc une boucle for)
for (int i = 0; i < N; i++) {
        // identifiant de l'instance courante
        appWidgetId = appWidgetIds[i];
        // Création de la vue
        views = new RemoteViews(context.getPackageName(), R.layout.widget_layout_two);
        //Il n'est pas possible de récupérer un élément de la RemoteViews pour le modifier
        //Il faut utiliser les méthodes de la RemoteViews
        //Affecte une image à un composant (équivalent à ImageView.setImageResource)
        views.setImageViewResource(R.id.imageView, R.drawable.ic_android2ee_bleu_inv);
        //Affecte un texte à un TextView (équivalent de TextView.setText)
        views.setTextViewText(R.id.textView, context.getString(R.string.app_name) + " : " + i);
        // Et surtout demande de mettre à jour l'instance de l'AppWidget dont l'identifiant est appWidgetId avec
cette vue
        appWidgetManager.updateAppWidget(appWidgetId, views);
}
```

Le deuxième exemple est celui d'une méthode de notre vue qui est appelée par la méthode onReceive et qui effectue la mise à jour du widget dont l'identifiant est passé en paramètre. Cette méthode est très utile car elle factorise la mise à jour / création de vos instances d'AppWidget. En effet, vos instances seront mises à jour soit dans la méthode onUpdate, soit à la réception d'un Intent. Il est ainsi très utile de factoriser cette mise à jour/ construction. J'attire votre attention sur le paramètre widgetId de cette méthode ; celui-ci peut être l'identifiant d'une instance d'un widget (dans le cas où l'on vient de recevoir un Intent de cet instance) qu'il faut mettre à jour, ou il peut-être une constante qui demande de reconstruire toutes les instances de l'AppWidget, ici ALL_WIDGET (dans le cas où la méthode est appelée par onUpdate qui a à charge la reconstruction de toutes les instances). Enfin cet exemple est très utile car il vous montre comment récupérer une instance de l'AppWidgetManager et de la liste des identifiants de vos instances d'AppWidget à partir de l'objet Context.

Le code de la méthode est très similaire à celui de onUpdate ci-dessus, le voici :

```
/*** The constant to ask to rebuild all the instance of the widget */
private static final Integer ALL_WIDGETS = -1;

private void updateWidget(Context context, int widgetId) {
//Retrouver l'instance courante du AppWidgetManager associé à notre context
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
```

```
//Instancier le ComponentName qui n'est autre que le nom de la classe relative à notre contexte
ComponentName componentName = new ComponentName(context, SimpleWidget.class);
//Retrouver les identifiants des instances de notre AppWidget installées
int[] appWidgetIds = appWidgetManager.getAppWidgetIds(componentName);
//Calculer le nombre de ces instances
final int N = appWidgetIds.length;
// Déclarer l'objet RemoteView
RemoteViews views;
//Trouver l'identifiant de l'image qui mettra à jour l'image dans le widget
int drawableId= R.drawable.ic_android2ee_blc;
//identifiant de l'instance courante
int appWidgetId ;
// Parcourir toutes les instances et les mettre à jour si l'identifiant correspond
for (int i = 0; i < N; i++) {
        //Mise à jour de l'identifiant de l'instance
        appWidgetId = appWidgetIds[i];
        //Si cet identifiant est celui passé en paramètre ou si l'identifiant passé en paramètre est la constante pour
        // tous les widgets on met à jour
        if ((widgetId == ALL_WIDGETS) || (widgetId == appWidgetId)) {
                //Instanciation de la RemoteView
                views = new RemoteViews(context.getPackageName(), R.layout.widget_layout_two);
                // Reconstruction de cette RemoteViews
                views.setImageViewResource(R.id.imageView, drawableId);
                views.setTextViewText(R.id.textView, context.getString(R.string.app_name) + " : " + i);
                // Et surtout demande de mettre à jour l'instance de l'instance  avec cette vue
                appWidgetManager.updateAppWidget(appWidgetId, views);
        }
}
}
```

4.3 Ajouter des listeners de clics à vos composants graphiques

Pour interagir avec votre AppWidget vous ne pouvez utiliser que les listeners de clics.

4.3.1 Déclaration de l'Intent

Comme toujours, pour que votre AppWidget puisse écouter un Intent, il faut le déclarer dans le fichier AndroidManifest.xml de votre application. Pour cela il suffit de rajouter un nouveau bloc intent-filter dans la déclaration de votre AppWidget :

```
<!-- Broadcast Receiver that will process AppWidget updates -->
 <receiver
    android:label="@string/widget_name"
    android:name=".SimpleWidget" >
    <intent-filter >
       <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <intent-filter >
       <action android:name="com.android.tuto.widget.simple.sample1.click1" />
    </intent-filter>

    <meta-data
       android:name="android.appwidget.provider"
       android:resource="@xml/widget_properties_definition" />
```

```
</receiver>
```

Ensuite, il faudra, dans votre classe AppWidget qui étend AppWidgetProvider, déclarer la constante qui correspond à cet Intent pour pouvoir l'utiliser :

```
private static final String INTENT_CLICK1 = "com.android.tuto.widget.simple.sample1.click1";
```

4.3.2 Mise en place de l'envoi de l'Intent

Cela s'effectue au moment où vous construisez votre RemoteViews, vous allez décider quels composants vous faites réagir aux clics utilisateurs et leur adjoindre une PendingIntent. L'exemple suivant montre comment le faire dans la méthode onUpdate. La partie de code qui nous intéresse est mise en avant ;

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
//Nombre d'instances installées
final int N = appWidgetIds.length;
//L'objet RemoteViews
RemoteViews views;
// identifiant de l'instance courante
int appWidgetId

// Mise à jour de toutes les instances du widget installées (donc une boucle for)
for (int i = 0; i < N; i++) {
        // identifiant de l'instance courante
        appWidgetId = appWidgetIds[i];
        // Création de la vue
        views = new RemoteViews(context.getPackageName(), R.layout.widget_layout_two);
        //Il n'est pas possible de récupérer un élément de la RemoteViews pour le modifier
        //Il faut utiliser les méthodes de la RemoteViews
        //Affecte une image à un composant (equivalent à ImageView.setImageResource)
        views.setImageViewResource(R.id.imageView, R.drawable.ic_android2ee_bleu_inv);
        //Affecte un texte à un TextView (équivalent de TextView.setText)
        views.setTextViewText(R.id.textView, context.getString(R.string.app_name) + " : " + i);


// ->Début de la partie qui nous intéresse :
//-------------------------------------------------
        // Ajout d'un listener de click
        // Déclaration de l'Intent où INTENT_CLICK1 est défini comme suit
        //private static final String INTENT_CLICK1 = "com.android.tuto.widget.simple.sample1.click1";
        Intent intentClick1 = new Intent(INTENT_CLICK1);
        // Ajout de l'identifiant de l'appWidget pour savoir quelle instance envoie cet Intent
        intentClick1.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
        // Création du PendingIntent qui sera envoyé. J'attire votre attention sur le deuxième paramètre
        //qui permet d'associer un unique PendingIntent à votre composant, sinon ils seront identiques
        // pour toutes les instances de vos widgets et vous ne retrouverez jamais votre paramètre extra de
        // l'Intent initial
        PendingIntent pendingIntent = PendingIntent.getBroadcast(context, appWidgetId, intentClick1, 0);
        // Affectation de ce PendingIntent à l'action du clic sur le composant souhaité
        views.setOnClickPendingIntent(R.id.imageView, pendingIntent);

// <-Fin de la partie qui nous intéresse.
//-----------------------------------------------

        // Et surtout demande de mettre à jour l'instance de l'AppWidget dont l'identifiant est appWidgetId avec
```

```
        // cette vue
        appWidgetManager.updateAppWidget(appWidgetId, views);
}
```

4.3.3 Récupération des Intents envoyés

Cela s'effectue dans votre méthode onReceive qui va recevoir l'Intent que vous venez d'envoyer et qui ventilera l'action à effectuer sur une méthode spécifique. Dans l'exemple ci-dessous cela met à jour l'IHM, on reconstruit donc la RemoteViews, mais vous pouvez effectuer toute autre action :

```
public void onReceive(Context context, Intent intent) {
super.onReceive(context, intent);
// ici on traite les intents:
if (INTENT_CLICK1.equals(intent.getAction())) {
        //Récupération de l'identifiant de l'instance de l'AppWidget qui a envoyé l'Intent
        int appWidgetId = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,ALL_WIDGETS);
        //Appel de la méthode de traitement en lui passant le contexte si elle doit mettre à jour l'IHM
        updateWidget(context,appWidgetId);
}
}

private void updateWidget(Context context, int widgetId) {
//Retrouver l'instance courante du AppWidgetManager associé à notre context
AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
//Instancier le ComponentName qui n'est autre que le nom de la classe relative à notre contexte
ComponentName componentName = new ComponentName(context, SimpleWidget.class);
//Retrouver les identifiants des instances de notre AppWidget installées
int[] appWidgetIds = appWidgetManager.getAppWidgetIds(componentName);
//Calculer le nombre de ces instances
final int N = appWidgetIds.length;
// Déclarer l'objet RemoteView
RemoteViews views;
//Trouver l'identifiant de l'image qui mettra à jour l'image dans le widget
int drawableId= R.drawable.ic_android2ee_blc;
//identifiant de l'instance courante
int appWidgetId ;
// Parcourir toutes les instances et les mettre à jour si l'identifiant correspond
for (int i = 0; i < N; i++) {
        //Mise à jour de l'identifiant de l'instance
        appWidgetId = appWidgetIds[i];
        //Si cet identifiant est celui passé en paramètre ou si l'identifiant passé en paramètre est la constante pour
        // tous les widgets on met à jour
        if ((widgetId == ALL_WIDGETS) || (widgetId == appWidgetId)) {
                //Instanciation de la RemoteView
                views = new RemoteViews(context.getPackageName(), R.layout.widget_layout_two);
                // Reconstruction de cette RemoteViews
                views.setImageViewResource(R.id.imageView, drawableId);
                views.setTextViewText(R.id.textView, context.getString(R.string.app_name) + " : " + i);
                // Ajout d'un listener de click
                // Déclaration de l'Intent où INTENT_CLICK1 est défini comme suit
                //private static final String INTENT_CLICK1 = "com.android.tuto.widget.simple.sample1.click1";
                Intent intentClick1 = new Intent(INTENT_CLICK1);
                // Ajout de l'identifiant de l'appWidget pour savoir quelle instance envoie cet Intent
                intentClick1.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId);
```

```
                // Création du PendingIntent qui sera envoyé. J'attire votre attention sur le deuxième paramètre
                //qui permet d'associer un unique PendingIntent à votre composant, sinon ils seront identiques
                // pour toutes les instances de vos widgets et vous ne retrouverez jamais votre paramètre extra de
                // l'Intent initial
                PendingIntent pendingIntent = PendingIntent.
                        getBroadcast(context, appWidgetId, intentClick1, 0);
                // Affectation de ce PendingIntent à l'action du clic sur le composant souhaité
                views.setOnClickPendingIntent(R.id.imageView, pendingIntent);

                // Et surtout demande de mettre à jour l'instance de l'instance  avec cette vue
                appWidgetManager.updateAppWidget(appWidgetId, views);
        }
}
```

4.3.4 Factorisation des noms des Intents dans le fichier string.xml

Dans l'exemple décrit dans ce paragraphe, la chaîne "com.android.tuto.widget.simple.sample1.click1" est dupliquée et écrite en dur dans le code. D'un point de vue pédagogique c'est plus clair. Par contre dans un vrai projet, il faut toujours factoriser vos chaînes de caractères. Pour cela, vous allez les déclarer dans votre fichier values/string.xml puis en faire référence dans votre Manifest et votre classe Java.

Ce qui donne dans votre fichier string.xml:

```
<resources>

   ...
   <string name="INTENT_CLICK1">com.android2ee.tuto.appwidget.sample1.click1</string>
   <string name="INTENT_UPDATE">com.android2ee.tuto.appwidget.sample1.update</string>
   ...

</resources>
```

Dans votre AndroidManifest.xml :

```
<receiver
        android:label="@string/widget_name"
        android:name=".AppWidgetTuto" >
        <intent-filter >
           <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <intent-filter >
           <action android:name="@string/INTENT_CLICK1" />
        </intent-filter>

        <intent-filter >
           <action android:name="@string/INTENT_UPDATE" />
        </intent-filter>

        <meta-data
           android:name="android.appwidget.provider"
           android:resource="@xml/widget_properties_definition" />
     </receiver>
```

Et enfin dans votre code Java :

```
// Define its Intent
Intent intent = new Intent(context.getResources().getString(R.string.INTENT_UPDATE));
…
// Manage the Intent
if (context.getResources().getString(R.string.INTENT_CLICK1).equals(intent.getAction())) {
...
```

4.4 Utilisation de l'alarme pour la mise à jour du Widget

Vous pouvez lancer une Thread dans la méthode onUpdate ou dans vos autres méthodes pour mettre à jour votre AppWidget, mais c'est vraiment une très mauvaise idée. Je dirai même que c'est la porte ouverte aux fuites mémoires. Donc si vous pensez utiliser cette méthode, oubliez ça de suite.

Pour gérer la mise à jour de votre AppWidget vous pouvez utiliser le paramètre updatePeriodMillis déclaré dans le fichier de définition de votre AppWidget, mais ce n'est pas très souple et de nombreuses restrictions s'imposerons à vous.

La meilleure façon d'effectuer ces mises à jour automatiques est d'utiliser l'alarme, l'AlarmManager. Cet objet permet d'envoyer périodiquement un Intent dans le système et ainsi de déclencher dans votre code une mise à jour. Le principe est identique à celui de la mise en place d'écouteurs de clics ; je définis un Intent et je le récupère dans la méthode onReceive pour le traiter.

Ainsi, il faut définir dans votre fichier manifeste que votre AppWidget écoute ce nouvel Intent

```
<!-- Broadcast Receiver that will process AppWidget updates -->
 <receiver
    android:label="@string/widget_name"
    android:name=".SimpleWidget" >
    <intent-filter >
       <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <intent-filter >
       <action android:name="com.android.tuto.widget.simple.sample1.click1" />
    </intent-filter>

     <intent-filter >
       <action android:name="com.android.tuto.widget.simple.sample1.update" />
    </intent-filter>
    <meta-data
       android:name="android.appwidget.provider"
       android:resource="@xml/widget_properties_definition" />
 </receiver>
```

Définir dans votre classe AppWidget, cette nouvelle variable

```
private static final String INTENT_UPDATE = "com.android.tuto.widget.simple.sample1.update";
```

Puis définir dans votre méthode onReceive l'action associée à la récupération de cet Intent :

```
public void onReceive(Context context, Intent intent) {
super.onReceive(context, intent);
// ici on traite les intents:
if (INTENT_CLICK1.equals(intent.getAction())) {
        //Récupération de l'identifiant de l'instance de l'AppWidget qui a envoyé l'Intent
        int appWidgetId = intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,ALL_WIDGETS);
        //Appel de la méthode de traitement en lui passant le contexte si elle doit mettre à jour l'IHM
        updateWidget(context,appWidgetId);
}  else if(INTENT_UPDATE.equals(intent.getAction())) {
        //On vient de recevoir l'Intent envoyé par l'alarme, on fait quelque chose
        forceUpdate(context);
}
}
```

Ensuite, on déclare l'alarme et on la lance dans la méthode onEnable (celle qui est appelée à la première installation de la première instance de votre widget) :

```
public void onEnabled(Context context) {
//récupération de l'AlarmManager
AlarmManager alarmManager = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
//Définition de l'intent
Intent intent = new Intent(INTENT_UPDATE);
//définition du pendingIntent envoyé
PendingIntent pendingIntentUpdate = PendingIntent.getBroadcast(context,0,intent,0);
//Mise en place de l'alarme : mode où le temps de veille ne compte pas
//dont la première alarme est déclenchée dans un demi-jour (12h)
//se répétera toutes les douze heures
//et enverra le pendingIntent
alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
        AlarmManager.INTERVAL_HALF_DAY,
        AlarmManager.INTERVAL_HALF_DAY,
        pendingIntentUpdate);
…
```

Et enfin, on n'oublie pas de la désactiver quand l'utilisateur désinstalle la dernière instance de votre widget :

```
public void onDisabled(Context context) {
        super.onDisabled(context);
        // récupération de l'AlarmManager
        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);
        // Définition de l'intent
        Intent intent = new Intent(INTENT_UPDATE);
        // définition du pendingIntent envoyé
        PendingIntent pendingIntentUpdate = PendingIntent.getBroadcast(context, 0, intent, 0);
        // Suppression de l'alarme
        alarmManager.cancel(pendingIntentUpdate);
}
```

Vous avez remarqué que nous avons déclaré deux fois le même PendingIntent, une méthode factorisant ce code serait la bienvenue :

```
/** * The constant action to set the alarm */
private final int SET_ALARM = 1;
/** * The constant action to cancel the alarm */
```

```
private final int CANCEL_ALARM = 2;

/** * La méthode pour mettre en place ou désactiver l'alarme des mises à jour */
private void manageAlarm (Context context, int action) {
        // récupération de l'AlarmManager
        AlarmManager alarmManager = (AlarmManager)
                context.getSystemService(Context.ALARM_SERVICE);
        // Définition de l'intent
        Intent intent = new Intent(INTENT_UPDATE);
        // définition du pendingIntent envoyé
        PendingIntent pendingIntentUpdate = PendingIntent.getBroadcast(context, 0, intent, 0);
        // Mise en place de l'action
        if (CANCEL_ALARM == action) {
                // Annulation de l'alarme associée au PendingIntent
                alarmManager.cancel(pendingIntentUpdate);
        } else if (SET_ALARM == action) {
                //Mise en place de l'alarme: mode où le temps de veille ne compte pas
                //dont la première alarme est déclenchée dans un demi-jour (12h)
                //se répétera toutes les douze heures
                //et enverra le pendingIntent
                alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
                AlarmManager.INTERVAL_HALF_DAY,
                AlarmManager.INTERVAL_HALF_DAY, pendingIntentUpdate);
        }
}

public void onEnable(Context context) {
                // mise en place de l'alarme de l'AlarmManager
                manageAlarm(context, SET_ALARM);
…
}
public void onDisabled(Context context) {
                super.onDisabled(context);
                // Suppression de l'alarme
                manageAlarm(context, CANCEL_ALARM);
        }
```

4.4.1 Digression sur le PendingIntent

Un PendingIntent décrit un Intent et une action à effectuer avec lui. Il est remis à d'autres applications afin que celles-ci puissent effectuer l'action demandée de votre part dans le futur.

En donnant un PendingIntent à une autre application vous lui accordez « vos droits » le tant d'effectuer cette action. Elle agit en tant que représentant de votre application. C'est pour cela que le nom de l'Intent passé au PendingIntent est le nom de votre classe (path inclus).

Ainsi, dans notre exemple, l'AlarmManager enverra l'Intent comme s'il était vous dans le système.

4.5 Utilisation du SharedPereference

Il est parfois utile voir nécessaire de faire persister des paramètres associés aux différentes instances des AppWidgets installées.

Pour cela l'utilisation de l'objet SharedPreference est préconisée. Deux manières d'effectuer ce traitement sont possibles :

- soit vous créez autant de fichiers SharedPreference que d'instances de vos AppWidget ;
- soit vous créez au sein de votre SharedPreference autant de clefs suffixées par l'identifiant de ces instances.

Personnellement, je trouve la première solution plus propre car les clefs restent identiques, et le fichier qui stocke ces données est associé à l'instance de votre AppWidget. La seconde solution est moins structurée, moins propre, enfin de mon point de vue. D'un autre côté, la première solution oblige à la destruction de fichiers sur le terminal, la seconde non. Il faut donc peser le pour et le contre de ces deux solutions. Moi, j'opte pour la première.

Seule la première solution sera présentée.

Important : il ne faudra pas oublier de supprimer les données quand le widget sera détruit, sinon vous polluez la mémoire de l'appareil pour rien et c'est le mal.

```
/***  */
private static final String SIMPLE_WIDGET_NAME = "SimpleWidget_";
/** * */
private static final String ON_UPDATE_CALLED_COUNTER = "onUpdateCalledCounter";

@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
// mise en place de l'alarme de l'AlarmManager
manageAlarm(context, SET_ALARM);
//Le nom du fichier de préférence
String myPreferences;
//L'objet SharedPreference
SharedPreferences preferences;
//Son éditeur
SharedPreferences.Editor editor;
//une variable pipo
int onUpdateCalledCounter;
//Le nombre de widgets
final int N = appWidgetIds.length;
//La remote views
RemoteViews views;
// Perform this loop procedure for each App Widget that belongs to this provider
for (int i = 0; i < N; i++) {
        int appWidgetId = appWidgetIds[i];
        //Gestion des preferences
        // Le nom du fichier de préférence
        myPreferences = SIMPLE_WIDGET_NAME+appWidgetId;
        //Récupération de l'objet SharedPreference associé
        preferences = context.getSharedPreferences(myPreferences, Activity.MODE_PRIVATE);
        //Récupération de la valeur associée à la clef ON_UPDATE_CALLED_COUNTER
        onUpdateCalledCounter=preferences.getInt(ON_UPDATE_CALLED_COUNTER, 0);
        //on l'incrémente
        onUpdateCalledCounter++;
        //On ouvre l'éditeur
        editor=preferences.edit();
```

```
        //On met à jour la variable
        editor.putInt(ON_UPDATE_CALLED_COUNTER, onUpdateCalledCounter);
        //Et on oublie pas le commit
        editor.commit();
        //Et on continue la méthode
…
}

@Override
public void onDeleted(Context context, int[] appWidgetIds) {
super.onDeleted(context, appWidgetIds);
//Le nom du fichier de préférence
String myPreferences;
//Le fichier SharedPreference en tant que File et son fichier backup
File sharedPreferencefile, sharedPreferenceBackupfile;
//Le nombre de widgets
final int N = appWidgetIds.length;

//Destruction des fichiers SharedPreference associés à ces instances
for (int i = 0; i < N; i++) {
        int appWidgetId = appWidgetIds[i];
        //Tout d'abord il faut vider les valeurs de l'objet
        // Le nom du fichier de préférence
        myPreferences = SIMPLE_WIDGET_NAME+appWidgetId;
        //Vidage du fichier
        context.getSharedPreferences(myPreferences, Activity.MODE_PRIVATE).edit().clear().commit();
        //ensuite il faut détruire le fichier lui-même :
        //on les récupère
        sharedPreferencefile= new File("/data/data/.../shared_prefs/"+myPreferences+".xml");
        sharedPreferencefile= new File("/data/data/.../shared_prefs/"+myPreferences+".bak");
        //on les détruit
        sharedPreferencefile.delete();
        sharedPreferenceBackupfile.delete();
}
}
```

4.6 Lancement d'une activité à l'installation d'une instance du widget

Enfin un dernier mot concernant le lancement d'une activité à l'installation d'une instance du Widget.

Parfois, il est utile de lancer une activité quand une instance de votre AppWidget est installée. Cela permet d'interagir avec l'utilisateur pour lui demander par exemple de mettre en place les préférences associées à cet AppWidget (taux de rafraîchissement…).

C'est assez simple à mettre en place. Cela s'effectue en trois étapes.

Tout d'abord, il faut déclarer l'activité dans le Manifest et définir qu'elle se lance par réception de l'action APPWIDGET_CONFIGURE :

```
<activity
        android:name=".ConfigureAppWidget"
        android:label="@string/app_name" >
        <intent-filter>
```

```
            <action android:name="android.apwidget.action.APPWIDGET_CONFIGURE" />
        </intent-filter>
</activity>
```

Ensuite dans votre fichier de définition de votre AppWidget, il suffit de rajouter la balise de lancement de l'activité de configuration :

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
        android:label="@string/hello"
        android:minWidth="180dip"
        android:minHeight="40dip"
        android:initialLayout="@layout/widget_layout"
        android:updatePeriodMillis="1800000"
        android:configure="com.android2ee.tuto.appwidget.sample1.ConfigureAppWidget"
        >
```

Enfin, votre activité doit respecter certaines règles :

- récupérer l'identifiant de l'instance de l'AppWidget qui a appelé l'activité, en analysant l'Intent reçu ;
- renvoyer un Intent portant les données de votre configuration à l'AppWidget (si nécessaire).

Si vous souhaitez mettre à jour l'instance de votre AppWidget au sein de votre activité, il vous suffit de faire comme d'habitude, en utilisant l'AppWidgetManager que vous récupérez par le Context de votre activité.

La récupération de l'identifiant de l'instance se retrouve ainsi :

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
        mAppWidgetId = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID,
                AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

Ensuite, si vous souhaitez renvoyer des informations à votre AppWidget et terminer votre activité :

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

Remarquez que si vous ne lancez pas cet Intent à la fin de votre activité, votre AppWidget le recevra de manière automatique mais son résultat sera à RESULT_CANCEL. Ce mécanisme permet de notifier à l'APPWidget que l'utilisateur est sorti de l'activité via le bouton « back ». Ainsi, si vous souhaitez savoir si l'utilisateur a bien terminé son activité, utilisez ce mécanisme.