Localisation des applications Flex

par Jim_Nastiq (autres articles)

Date de publication : 25/10/2008

Cet article a pour but de vous présenter la gestion du multilinguisme dans une application Flex. L'application détecte automatiquement la langue de l'utilisateur pour en adapter, si la langue est gérée, son contenu. La langue par défaut est appliquée si l'application ne l'a pas détectée. Elle permet également le changement à la volée de la langue pour toute l'application en cours d'exécution.

I - Pré-requis

Avant de commencer, il faut tout d'abord, obtenir une copie locale du framework. En effet, le framework est proposé en anglais(En-US) il faut donc le copier pour créer un framework français (fr-FR).
Pour cela on ouvre une invite de commande et on se place dans le répertoire "bin" du SDK Flex (par défaut C:\Program Files\Adobe\Flex Builder 3\sdks\3.0.0\bin):

déplacement vers le répertoire d'installation

```
cd C:\Program Files\Adobe\Flex Builder 3\sdks\3.0.0\bin
```

Puis, on copie le framework de base pour pouvoir l'utiliser dans une autre langue:

copie du framework

```
copylocale.exe en_US fr_FR
```

Si une erreur du type "unable to find JDK" ou "unable to find JVM" apparait, il faut ajouter une variable d'environnement JAVA_HOME (pour pointer vers le JDK):

- Panneau de configuration->Système, onglet Avancés, puis Variables d'environnements
- dans la zone "variables système" : cliquer sur Nouveau
- Nom de la variable : JAVA_HOME
- Valeur : répertoire vers le jdk (par exemple : C:\java\j2sdk1.4.2_17)
- si besoin redémarrer l'invit' de commande et recommencer l'étape précédente.

II - Créer ses dictionnaires

Maintenant, nous pouvons commencer un nouveau projet que je nommerai localization. Dans les propriétés de ce projet, sur la partie Flex Compiler, nous rajoutons:

variables de compilation

```
-locale=fr_FR,en_US -source-path=../locale/{locale}
```

Ainsi, la langue par défaut est le français et à défaut de définition pour un composant du framework en français, il sera en anglais. De plus la seconde partie indique le chemin vers, ce que j'appelle, les dictionnaires. Il faut donc à présent créer ces dictionnaires. Nous créons donc un dossier "locale" à la racine du projet (au même niveau que src), puis dans ce dossier nouvellement créé on ajoute les dossiers en_US et fr_FR avec, à l'intérieur de chacun, un fichier i18n.properties (i18n étant l'abréviation de internationalisation, mais libre à vous de le nommer comme bon vous semble). Voici, le contenu de mon fichier i18n.properties (version FR) :

contenu d'un dictionnaire

```
#Commentaires bla bla
home_title=Bonjour!
title_window=Titre de la fenêtre
welcome_text=Bienvenue sur le composant 1
combo_text=C'est ici que vous pouvez changer la langue
button_text=Clique sur moi!
```

Vous l'avez remarqué, rien de compliqué, il suffit de copier-coller ce code dans le fichier anglais et de traduire... Pour les caractères non ASCII (é, è , ...), il faut sauvegarder le fichier français au format UTF-8, pour cela clic droit sur le fichier->properties et choisir Other dans Text field encoding, puis UTF-8.

III - Utilisation des dictionnaires dans un composant MXML

Maintenant, que les dictionnaires sont disponibles, comment y accéder ? C'est très simple, par défaut (référence à la variable de compilation) c'est le français qui est chargé, mais avec une ComboBox, par exemple, on peut passer d'une langue à l'autre pour cela, on utilise le resourceManager et sa propriété localeChain(Array) qui propage l'information à toute l'application. Sur l'évènement CHANGE de la ComboBox on a donc:

```
resourceManager.localeChain = [myComboBox.selectedItem];
```

Il faut, pour que la modification soit effective, que les text et label de l'application récupère les valeurs depuis le resourceManager évidemment, voici comment opérer :

```
<!-- Dire au compilateur quelle ressource utiliser -->
<mx:Metadata>
  [ResourceBundle("i18n")]
</mx:Metadata>

<!-- Depuis un fichier MXML -->
<mx:Text text="{resourceManager.getString('i18n', 'welcome_text')}"/>
```

```
//Depuis un composant AS (sous classe de UIComponent)
this.mybutton.label = resourceManager.getString('i18n', 'button_text');

//Code AS n'étant pas une sous classe de UIComponent
this.mybutton.label = ResourceManager.getInstance().getString('i18n', 'button_text');
```

IV - Utilisation des dictionnaires dans un composant AS

A présent, nous avons sans problème le changement à la volée pour les composants MXML mais pour les composants AS, il y a une petite subtilité. En effet, l'affectation du text ou du label doit se faire dans la méthode resourcesChanged du UIComponent. Voici un exemple de composant AS:

```
package
{
 import mx.containers.Canvas;
 import mx.controls.Button;

 public class myCompAS extends Canvas
 {
  public var mybutton:Button;

  public function myCompAS(){

  }

  protected function init():void{
   this.resourcesChanged();
  }

  override protected function resourcesChanged():void{
   if( this.mybutton != null ){
    this.mybutton.label = resourceManager.getString('i18n', 'button_text');
   }
  }
 }
}
```

Voici donc une démonstration d'une animation flash localisable à la volée, les sources sont disponibles avec le menu contextuel (clic droit):