

Des applications localisées sous Flex 2

par Olivier Bugalotto ([Mes articles](#))

Date de publication : 25/03/2008

Ce tutoriel vous expliquera comment réaliser des applications localisées, c'est à dire tenant compte de la langue de l'environnement d'exécution.

- I - Fonctionnement
- II - Des composants localisés
- III - Encore plus de facilité

I - Fonctionnement

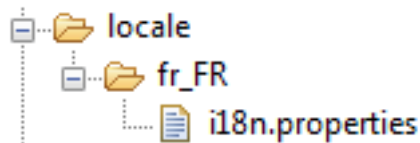
Les ressources permettent de créer des applications localisées à partir du lieu et de la langue de l'environnement d'exécution.

Pour faire une application localisée, il nous faut définir un ou plusieurs fichiers d'extension `.properties` qui contien(nen)t les données à afficher. Ce(s) fichier(s) stocke(nt) les données au format `key = value`, dont voici un exemple :

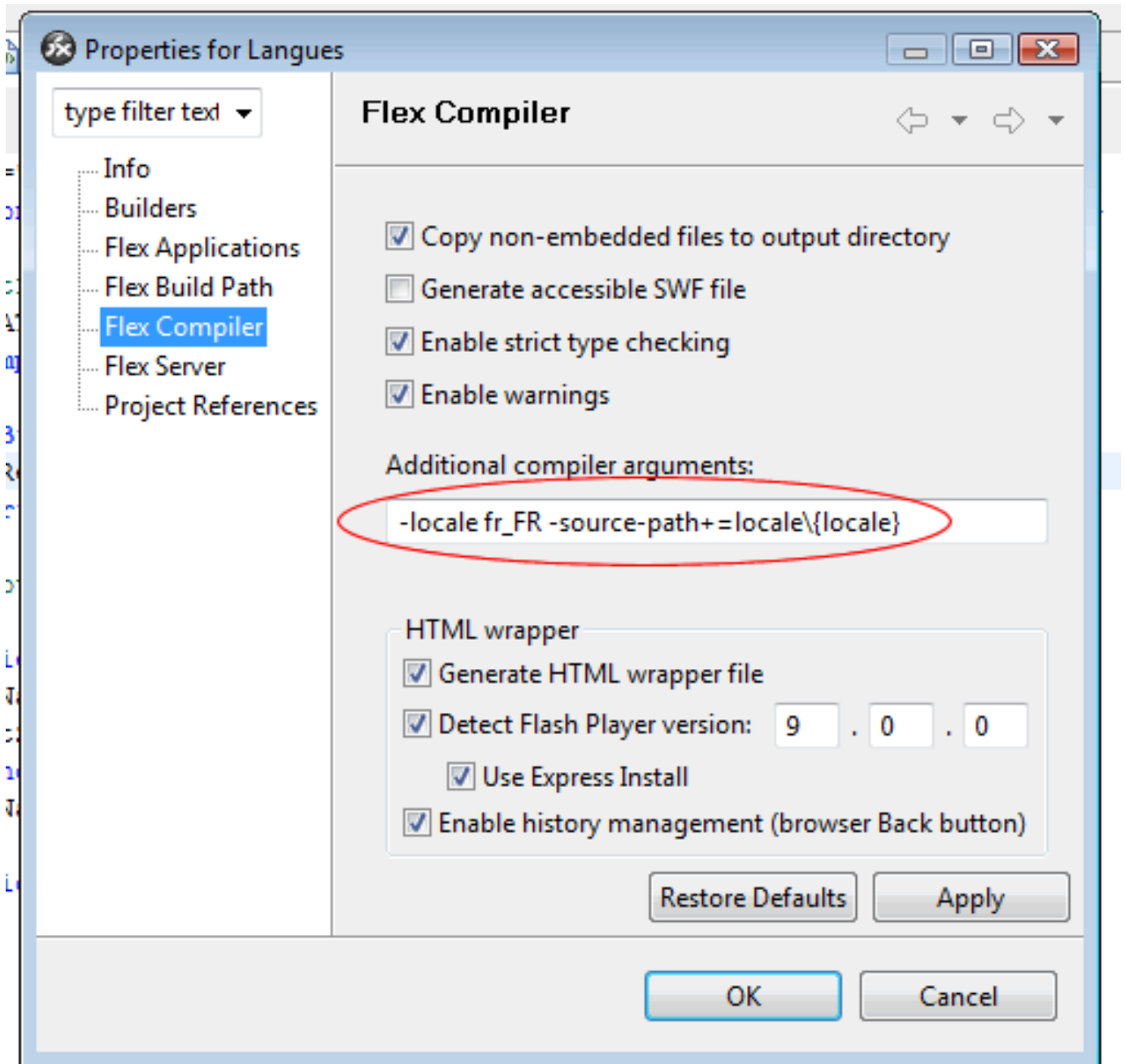
```
// ressources.properties  
titleContact=Fiche contact  
fieldName=Nom  
fieldFName=Prenom  
fieldEmail=Courriel  
labelBtnSave=Enregistrer  
labelBtnCancel=Annuler
```

Ces fichiers seront embarqués dans le fichier SWF résultant de la compilation de votre projet.

Nous placerons ces fichiers ressources dans un dossier nommé *locale* au niveau de chaque projet qui contient lui-même un autre dossier nommé *fr_FR* (langue underscore pays) :



Ce dossier *locale* doit être inclus dans le projet :



Par défaut, il y a un seul argument **-locale en_US**, ici nous modifions cet argument en indiquant le choix de la langue et du pays (**fr_FR**) et nous ajoutons la ligne de commande suivante **-source-path+=locale\{locale}** pour indiquer au compilateur où trouver les fichiers de ressources. Vous aurez compris que la valeur entre bracelet correspond à la valeur du premier argument.

Créons le projet Flex ResourceBundleMXML dans lequel nous allons utiliser le metadata `@Resource` pour accéder aux ressources, exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
```

```

<mx:Style source="styles.css" />

    <!--
bundle correspond au nom du fichier de ressource
key le nom de la cle
-->
<mx:Panel title="@Resource(bundle='i18n',key='titleContact')"
width="314" height="183"
layout="absolute"
titleIcon="@Embed('assets/addressbook2.png')">
<mx:Form styleName="frmContact" left="10" right="10" top="10" bottom="10">
<mx:FormItem label="@Resource(bundle='i18n',key='fieldName')" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
<mx:FormItem label="@Resource(bundle='i18n',key='fieldFName')" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
<mx:FormItem label="@Resource(bundle='i18n',key='fieldEmail')" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
</mx:Form>
<mx:ControlBar height="42" y="128" horizontalAlign="right">
<mx:Button label="@Resource(bundle='i18n',key='labelBtnSave')"/>
<mx:Button label="@Resource(bundle='i18n',key='labelBtnCancel')"/>
</mx:ControlBar>
</mx:Panel>

</mx:Application>
    
```

Nous pouvons réaliser la même chose avec du code Actionscript, en utilisant le metadata *ResourceBundle* et la classe de même nom, ce qui nous donne :

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">

<mx:Style source="styles.css" />

<mx:Script>
<![CDATA[
import mx.resources.ResourceBundle;


[Bindable]
[ResourceBundle("i18n")]
private var i18n:ResourceBundle;

// Execute apres la creation du panel
private function onCreatePanel():void {
    pnlContact.title = i18n.getString("titleContact");
}
]]>
</mx:Script>

<mx:Panel id="pnlContact" width="314" height="183" layout="absolute"
titleIcon="@Embed('assets/addressbook2.png')" creationComplete="onCreatePanel()">
<mx:Form styleName="frmContact" left="10" right="10" top="10" bottom="10">
<mx:FormItem label="{i18n.getString('fieldName')}" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
<mx:FormItem label="{i18n.getString('fieldFName')}" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
<mx:FormItem label="{i18n.getString('fieldEmail')}" width="100%">
<mx:TextInput width="100%" />
</mx:FormItem>
</mx:Form>
<mx:ControlBar height="42" y="128" horizontalAlign="right">
    
```

```
<mx:Button label="{i18n.getString('labelBtnSave')}" />
<mx:Button label="{i18n.getString('labelBtnCancel')}" />
</mx:ControlBar>
</mx:Panel>

</mx:Application>
```

 *i18n est une contraction du mot internationalisation*

II - Des composants localisés

Nous pouvons donc utiliser la classe *ResourceBundle* pour traduire les composants Flex dans la langue que nous souhaitons. Prenons le cas du composant DateChooser (Calendrier). Ce composant propose un ensemble de propriétés nous permettant de modifier les libellés des jours (*dayNames*) et des mois (*monthNames*). Dans un premier temps, créons un fichier de ressources (properties file) :

```
//components.properties
dayNames=D,L,M,M,J,V,S
monthNames=Janvier,Février,Mars,Avril,Mai,Juin,Juillet,Août,Septembre,Octobre,Novembre,Décembre
formatString=DD/MM/YYYY
```

Mais ces propriétés à savoir *dayNames* et *monthNames* attendent un tableau, ainsi la méthode *getStringArray()* de *ResourceBundle* permet de retourner un tableau :

```
// Fichier ComponentResourceBundle.mxml
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="vertical">
  <mx:Script>
    <![CDATA[
      import mx.resources.ResourceBundle;

      [Bindable]
      [ResourceBundle("components")]
      private var i18n:ResourceBundle;
    ]]>
  </mx:Script>
  <mx:DateField dayNames="{i18n.getStringArray('dayNames')}}"
    monthNames="{i18n.getStringArray('monthNames')}}"
    formatString="{i18n.getString('formatString')}" />
  <mx>DateChooser dayNames="{i18n.getStringArray('dayNames')}}"
    monthNames="{i18n.getStringArray('monthNames')}" />
</mx:Application>
```

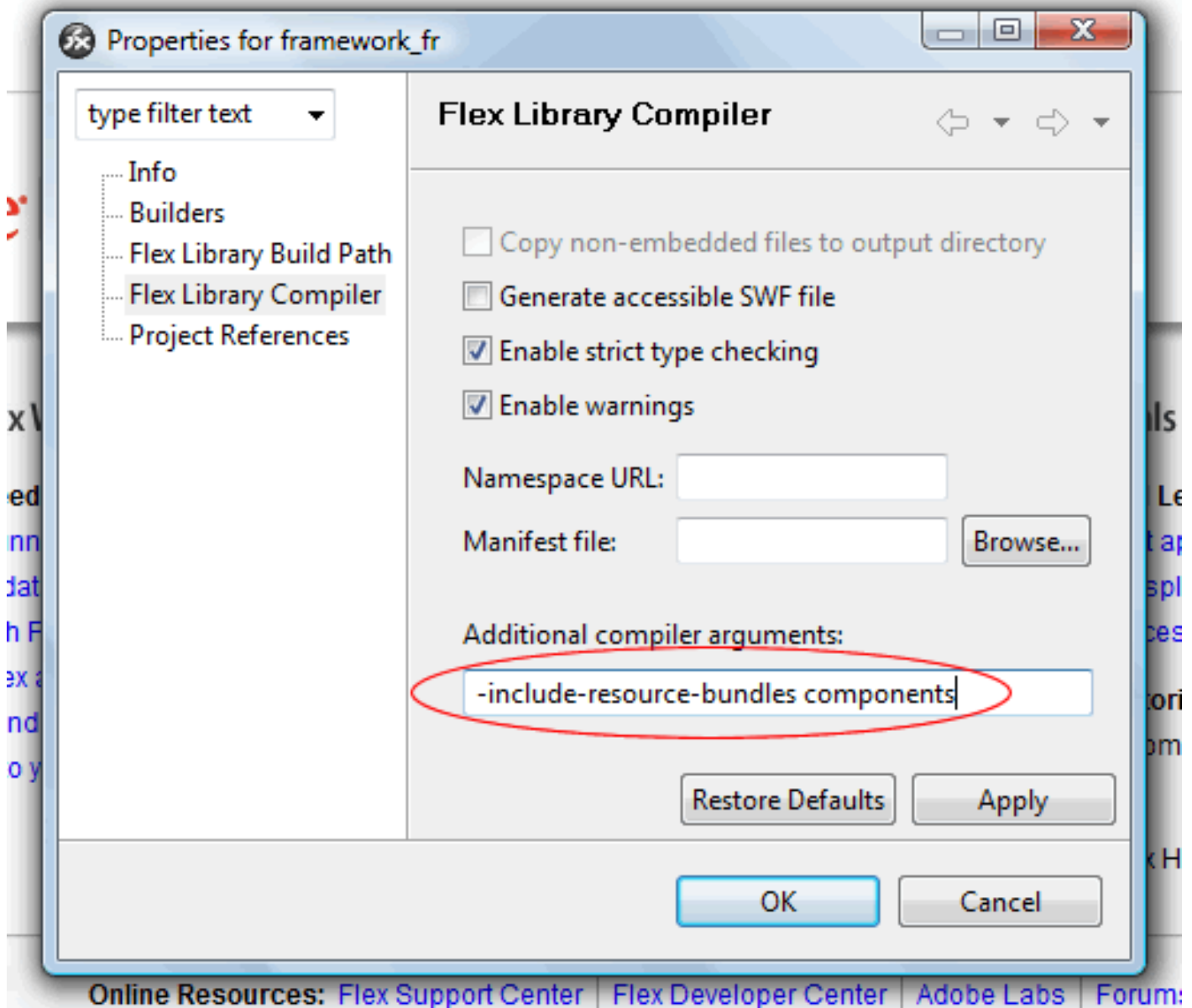
Si vos projets sont le plus souvent destinés, d'un point de vue géographique, à la France, il serait plutôt judicieux de ne pas se préoccuper de traduire à chaque fois les composants. Il suffirait simplement de l'indiquer à travers l'option de commande locale du compilateur en choisissant `fr_FR`.


C'est pour cette raison que nous allons créer un composant compilé (`swc`) qui contiendra tous les fichiers de ressources.

Créons un projet de librairie flex que nous appellerons `framework_rb` et plaçons-y le fichier de ressources précédent. Nous devons indiquer au projet, le(s) fichier(s) de ressources à inclure, ceci avec la ligne de commande suivante :

```
-include-resource-bundles nomDuFichierProperties
```

Dans notre cas :



 *Note: le nom du fichier de ressources sans son extension.*

Nous obtenons le fichier framework_rb.swc dans le dossier bin. Il nous faut maintenant ajouter ce composant à notre projet ComponentResourceBundle.

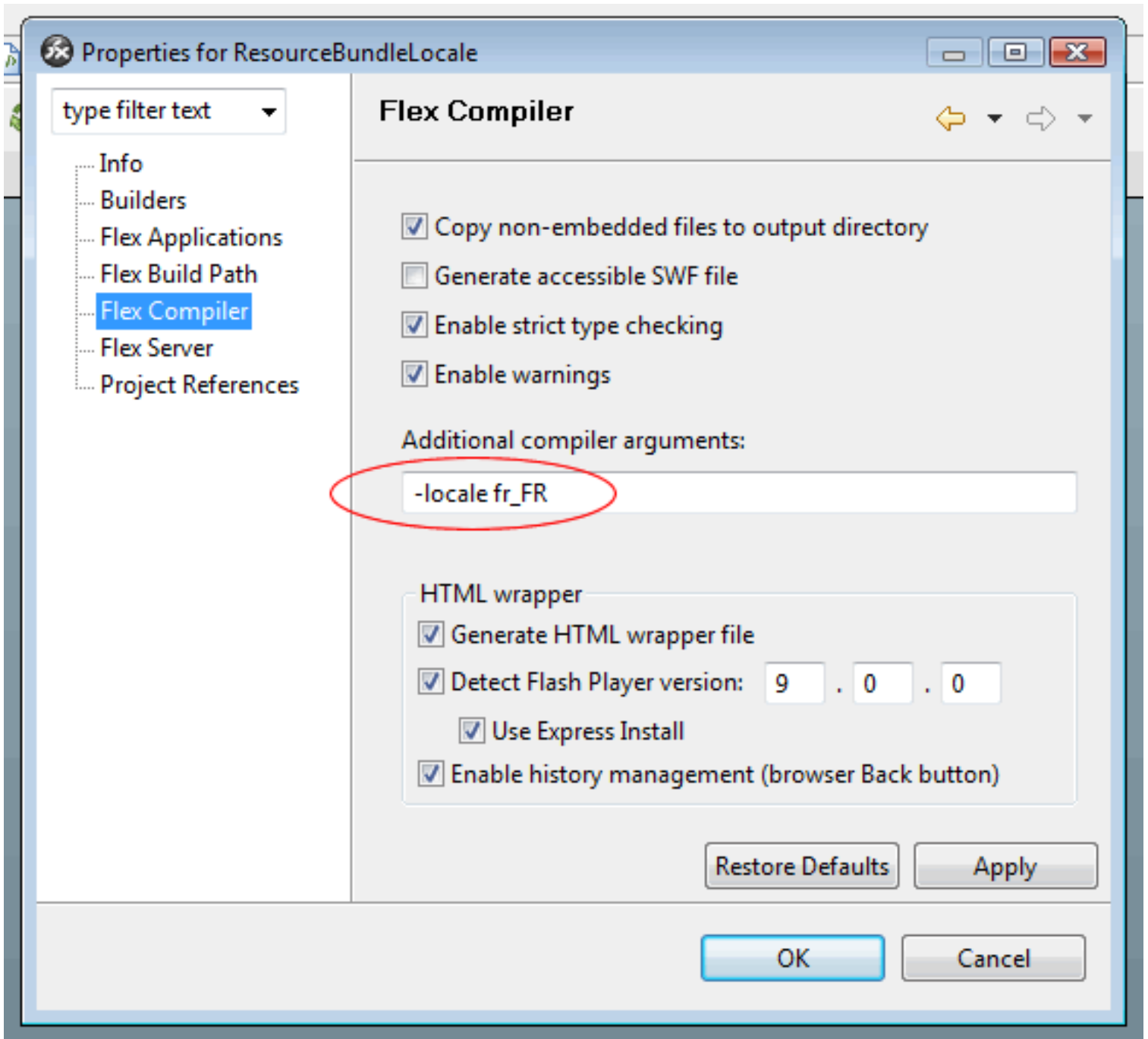
III - Encore plus de facilité

Il existe dans le dossier suivant :

```
C:\Program Files\Adobe\Flex Builder 2\Flex SDK 2\frameworks\locale\en_US
```

plusieurs fichiers de ressources qui contiennent les propriétés par défaut pour tous les composants Flex. Il vous suffit de créer un projet de librairie Flex comme précédemment et d'y inclure ces fichiers traduit dans la langue de votre choix. Après avoir obtenu le fichier compiler (.swc), créer un dossier fr_FR dans le dossier indiqué au-dessus et ajoutez-y le composant compilé (swc).

Maintenant pour tous vos projets à venir, il vous suffira simplement de modifier l'option de commande suivante par le nom du dossier de votre choix de langue :



Dans la version 3 de Flex, nous pouvons travailler avec les ressources de manière dynamique et ainsi permettre à l'utilisateur de choisir la langue de l'application à l'exécution. J'en parlerai dans un prochain tutoriel mais en attendant vous trouverez le code source de ce tutoriel ici : [ResourceBundle.zip](#) ([miroir](#))

