3.1 Internationalisation

L'internationalisation est native dans *esup-commons*. L'intérêt n'est pas seulement de fournir une application en plusieurs langages... L'externalisation de toutes les chaînes de caractères et la possibilité d'utiliser simultanément plusieurs fichiers de chaînes (les *bundles*) permettent de simplifier la personnalisation des applications par les administrateurs.

- Principes
  - Configuration
  - Implémentations disponibles
- Préconisations d'usage
  - Nommage des bundles
  - Modification des fichiers de messages
- Externalisation des chaînes sans internationalisation
- Utilisation
  - Dans une vue
    - JSF
      - Ajout d'un langage
      - Accès depuis un contrôleur (Java)
      - Changement des messages d'erreur par défaut de JSF
    - Spring MVC

Principes

Configuration

L'internationalisation est définie dans le fichier de configuration **/properties/i18n/i18n.xml**. On y trouvera par exemple :

```
<bean id="i18nService"
  class="org.esupportail.commons.services.i18n.ResourceBundleMessageSourceI18nServiceImpl">
  <property name="messageSource" ref="msgs" />
</bean>

<bean id="msgs"
  class="org.esupportail.commons.services.i18n.ReloadableResourceBundleMessageSource">
  <!-- org.esupportail.commons.services.i18n.ReloadableResourceBundleMessageSource since 0.3.3
       org.springframework.context.support.ReloadableResourceBundleMessageSource before -->
  <property name="basenames">
    <list>
      <value>classpath:properties/i18n/bundles/Custom</value>
      <value>classpath:properties/i18n/bundles/Messages</value>
      <value>classpath:properties/i18n/bundles/Commons</value>
    </list>
  </property>
  <property name="cacheSeconds" value="60" />
</bean>
```

i18nService

- La propriété **messageSource** fait référence à l'implémentation Spring du son mécanisme de ResourceBundleMessageSource

msgs

- Le nom du bean lui même est important car il est notamment utilisé dans les vues JSF
- La propriété **basenames** donne la liste des fichiers de messages à utiliser. Chaque valeur de la liste correspond au chemin du fichier de message dans le *classpath*. Par exemple, on cherchera pour la valeur **Custom** les fichiers **Custom_<locale>.properties** ou à défaut **Custom.properties**. Par convention, on groupera tous les fichiers de messages dans le répertoire **/properties/i18n/bundles**.
- La propriété cacheSeconds donne le délai en entre cache relecture des fichiers de messages

⚠ Depuis la version 0.3.3 de EC2 la classe est de type **org.esupportail.commons.services.i18n.ReloadableResourceBundleMessageSource** et plus **org.springframework.context.support.ReloadableResourceBundleMessageSource** afin de supporter la méthode **getStrings()** du service i18n.

Implémentations disponibles

esup-commons offre plusieurs implémentations afin de gérer l'internationalisation des applications :

- **ResourceBundleMessageSourceI18nServiceImpl** utilise l'implémentation Spring du son mécanisme de ResourceBundleMessageSource (depuis ESUP-Commons 2 - 0.2.7)

Les autres implémentations sont conservées pour des raisons de compatibilité mais devrait disparaître à terme :

- **BundleI18nServiceImpl** lit les chaînes de caractères depuis un unique fichier de messages.
- **BundleCachingI18nServiceImpl** étend **BundleI18nServiceImpl**. Elle permet la mise en cache des chaînes de caractères lues depuis le fichier de messages, pour des raisons de performance.
- **BundlesI18nServiceImpl** lit les chaînes de caractères depuis plusieurs fichiers de messages.
- **BundlesCachingI18nServiceImpl** étend **BundlesI18nServiceImpl** en apportant des fonctionnalités de cache, pour les mêmes raisons de performance.

Préconisations d'usage

Nommage des bundles

Chaque entrée d'un fichier de message peut être surchargée par la même entrée définie dans le fichier précédent. En effet, c'est la première définition d'une entrée qui compte.

Les fichiers Commons_**.properties** font partie du projet *esup-commons*, ils ne doivent pas être modifiés par les développeurs d'applications.

Si le développeur a besoin de surcharger une entrée de ce fichier, il peut le faire grâce aux fichiers **Messages**_.properties. C'est aussi dans ce fichier qu'il mettra tous les messages relatifs à son application.

Les fichiers **Custom**_.poperties* permettent à l'exploitant de surcharger les messages livrés avec l'application (qu'il s'agisse de messages de *esup-commons* ou de l'application).

Exercice : Surcharger un bundle Afficher l'énoncé

Modification des fichiers de messages

Pour modifier les fichiers de messages il est recommandé d'utiliser *Ressource Bundle Editor* (*RBE*). Il permet notamment d'éditer plusieurs fichiers, correspondants à plusieurs langues, en même temps. Les messages enregistrés dans ces fichiers peuvent contenir des paramètres qui seront dynamiquement renseignés lors de l'exécution. Ces paramètres apparaissent dans les messages sous cette forme :

```
{n}
```

avec **n** commençant à 0. Exemple :

```
L''utilisateur {0} est maintenant gestionnaire du service {1}.
```

⚠ Noter ci-dessus l'échappement des apostrophes lorsque la chaîne contient un ou plusieurs paramètres.

```
L''utilisateur {0} est maintenant gestionnaire du service {1,choice,0#annuaire|1#helpdesk}.
```

⚠ Dans l'exemple ci-dessus, le paramètre 1 ne peut avoir que 2 valeurs : soit annuaire, soit helpdesk.

Externalisation des chaînes sans internationalisation

Comme vu en introduction de cette partie, externaliser les chaînes sans internationaliser conserve un intérêt pour la distribution des applications : faciliter la personnalisation locale des applications là où elles sont portées.

Dans ce cas, on pourra simplement utiliser un seul fichier de ressources, correspondant au langage déclaré.

Utilisation

Dans une vue

JSF

Dans le fichier faces-config.xml il faut ajouter à la définition de l'application le resolver d'expression suivant :

```
<el-resolver>org.esupportail.commons.jsf.ResourceBundleFacesELResolver</el-resolver>
```

Le bean **msgs** défini dans le fichier **/properties/i18n/i18n.xml** sera alors accessible dans les pages JSF. Exemple :

```
<h:outputText value="#{msgs['DEEPLINKING_DEMO.GENERATE_URL.PARAM_NAME']}" />
```

Ajout d'un langage

L'ajout d'un langage se fait dans le fichier de configuration **src/main/webapps/WEB-INF/jsf/faces-config.xml**. Exemple :

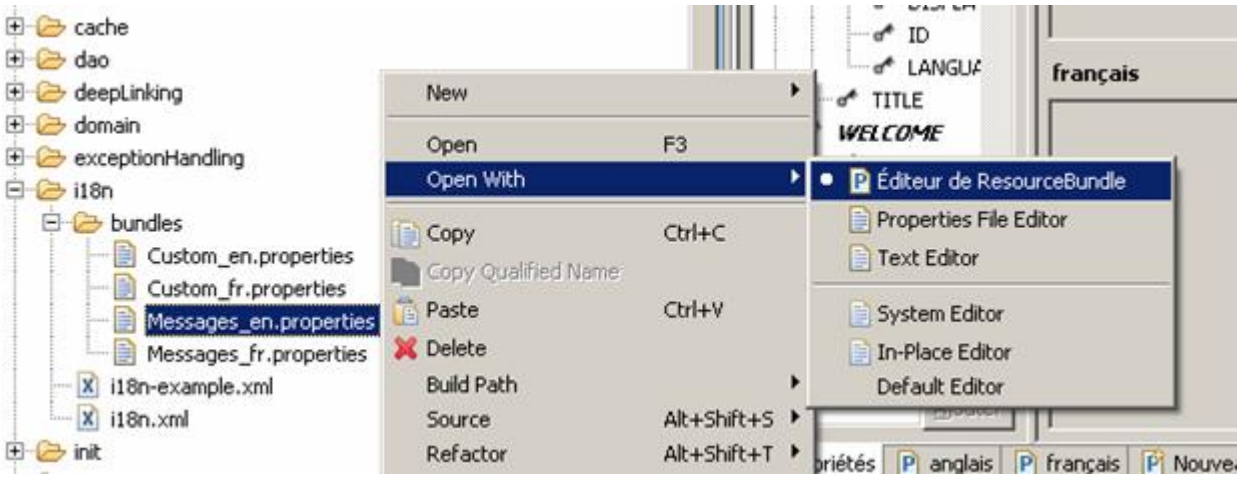
```
<locale-config>
  <default-locale>fr</default-locale>
  <supported-locale>fr</supported-locale>
  <supported-locale>en</supported-locale>
</locale-config>
```

Il suffit ensuite d'écrire le *bundle* correspondant.

Exercice : Ajouter un langage Afficher l'énoncé

Accès depuis un contrôleur (Java)

Par héritage, toute classe (métier ou contrôleur) qui étend **AbstractApplicationAwareBean** bénéficie des services de **AbstractI18nAwareBean** et peut appeler ses méthodes d'internationalisation.



On utilise cela en particulier pour les messages (d'information ou d'erreur) lancés par les contrôleurs et affichés sur les vues *JSF* à l'aide de la balise **<h:messages>**. On utilisera par exemple :

```
addErrorMessage(null, "MANAGERS.MESSAGE.USER_NOT_FOUND", userId);
```

On utilisera également cette fonctionnalité à toute autre fin en accédant directement aux fichiers de messages de cette manière :

```
String msg = getString("MY.MESSAGE");
```

On peut également passer des paramètres à la méthode **getString()**, ainsi qu'une *locale* (si l'on souhaite une locale différente de la locale courante), qui est donnée par ma méthode **getLocale()**.

Changement des messages d'erreur par défaut de JSF

Pour modifier les messages d'erreurs par défaut ou les internationaliser, il faut écrire un fichier **JsfMessage_fr.properties** dans le répertoire **/properties /i18n/bundles** et ajouter :

```
<message-bundle>properties/i18n/bundles/JsfMessages</message-bundle>
```

au fichier **/properties/jsf/application.xml** pour que *JSF* prenne en charge ce nouveau *bundle.*

Par exemple lorsque l'on utilise les validations implicites avec l'attribut **required="true"** au lieu d'avoir le message « *une donnée est requise* », on pourra avoir un message personnalisé en indiquant les messages suivants dans le fichier **JsfMessages.properties** :

```
javax.faces.component.UIInput.REQUIRED = Erreur de validation
javax.faces.component.UIInput.REQUIRED_detail = "et {0} alors !!!
```

Spring MVC

Spring MVC nécessite que le bean implémentant **MessageSource** porte l'id **messageSource**. Dans l'exemple ci-dessus vous pouvez changer **msgs** en **m essageSource** ou définir un alias :

```
<alias name="msgs" alias="messageSource"/>
```

Ensuite dans la vue vous accédez aux messages avec ce type de code :

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<h1><spring:message code="title"/></h1>
<spring:message code="messageWithParam" arguments="${param1},${param2}"/>
```