

L'internationalisation dans Mbinkus Farm

Le problème

Quand on développe une application, on ne sait généralement pas tous les utilisateurs qui l'utiliseront. De ce fait, on ne connaît pas leur langue, est-ce le Français ou l'Anglais ou ... ?

Alors, comment faire ?

On peut développer l'application pour chaque langue. Dans ce cas, il faudra refaire toute l'application dans chaque langue et cela demande beaucoup de temps et d'énergie. Imaginons que notre application doit être utilisée dans plus de dix pays qui ont chacun leur langue. On va très vite en besogne.

Avons-nous une autre solution ?

Eh bien oui !

Si notre application s'adaptait à son environnement d'exécution (la langue, la monnaie, ...) ? ce serait préférable que d'avoir plusieurs applications. Imaginons qu'un bug soit détecté sur l'une d'entre elles, on devrait le résoudre dans toutes les autres (Elles font toutes les mêmes choses, c'est juste la langue qui change).

C'est ce qu'on appelle **l'Internationalisation**.

Notre pays étant bilingue (Français et Anglais), nous avons voulu l'implémenter sur notre application « Script Farm ». A ce jour l'implémentation n'est effective que sur la page de Login, c'est de celle-ci que sera inspirée ce document. Nous utiliserons des ResourceBundle et des fichiers .properties. Jamais entendu parler ?

(Pour ceux qui ont répondu oui, Il faut cravacher ! En tout cas dans la suite nous allons codifier ...)

A vos IDEs, on commence.

L'internationalisation en Java

Avant toute chose, pourquoi en Java ?

Notons d'abord que L'internationalisation est possible dans la majorité des langages de programmation.

Maintenant que c'est fait, pour répondre à la question, nous développons « Script Farm » en Java. J'espère que tout est clair maintenant ? 😊

Pas vraiment, comment on s'y prend ?

Tout doux, on y arrive. Mais avant, parlons de l'objet « Locale ».

Qu'est ce que c'est ?

Un Objet de type « Locale » identifie une langue et un Pays.

Comment le construit-on ?

Son constructeur attend deux paramètres :

-Le code langue (deux caractères minuscules conformes à la norme ISO-639 : exemple "de" pour l'allemand, "en" pour l'anglais, "fr" pour le français, ...).

-Le second est le code pays (deux caractères majuscules conformes à la norme ISO-3166 : exemple : "DE" pour l'Allemagne, "FR" pour la France, "US" pour les Etats Unis, etc ...). Ce paramètre est obligatoire : si le pays n'a pas besoin d'être précisé, il faut fournir une chaîne vide.

```
Locale locale_US = new Locale("en","US");
```

```
Locale locale_en = new Locale("en","");
```

On peut aussi fournir un 3^e paramètre pour préciser d'avantage la localisation par exemple la plate-forme d'exécution (il ne respecte aucun standard car il ne sera défini que dans l'application qui l'utilise) :

```
Locale locale_windows = new Locale("fr","FR","WINDOWS");
```

Il y a un problème !!! Devons-nous mémoriser tout ça ?

Malheureusement, la classe Locale définit des constantes pour certaines langues et pays.

En ce qui concerne les pays, on peut avoir :

```
Locale locale_japon = Locale.JAPAN ;
```

Ce qui est équivalent à :

```
Locale locale_japon = new Locale("ja","JP");
```

On peut aussi vouloir spécifier les langues seulement

```
Locale locale_japon = Locale.JAPANESE ;
```

Ce qui est équivalent à :

```
Locale locale_japon = new Locale("ja","");
```

Un cadeau pour la route 😊

Comment connaître le Locale par défaut :

```
Locale de = Locale.getDefault();
```

Comment connaître la liste des Locales disponibles :

```
Locale liste[] = DateFormat.getAvailableLocales();  
for (int i = 0; i < liste.length; i++)  
{  
    System.out.println(liste[i].toString());  
    // toString retourne le code langue et le code pays séparé d'un souligné  
}
```

Merci. Mais jusqu'ici je ne vois toujours pas comment l'internationalisation se fait !

Si vous vous rappelez, on a dit qu'on utilisera des ResourceBundle et des fichiers .properties

On va arrêter de vous perdre du temps 😊 Entrons maintenant dans le vif du sujet.

Les fichiers .properties

Pour que notre application s'adapte aux langues, il faut traduire tous les mots et phrases qu'on affiche. Je suis désolé, mais si vous voulez d'une telle application, c'est le prix à payer. Sinon, vous pouvez la redévelopper pour chaque langue. Et même là, vous aurez toujours besoin de traduire chaque mot et phrase qui sera affiché.

En traduisant, nous allons créer des fichiers .properties : Un pour chaque langue et un qui sera choisi par défaut.

Les lignes d'un fichier .properties sont soit :

- Un commentaire. Dans ce cas, la ligne commence par un « # ».
- Une paire clés/valeur séparées par un « = » Ex : appName = Script Farm
- vides.

```
# and open the template in the editor.

appName = Script Farm
appDescription = Manage and control your farm activities easily
appSettings = App Settings
or = Or
username = Username
passwordButton = Forgot Password ?
passwordField = Password
signButton = Sign Up
signAdminButton = Sign Up As Administrator
exitButton = Exit
reportGood=Server is up: Go to do
authentificationFailed=Authentification Failed. Verify your informations
loadDashboardFailed=Load Dashboard failed
```

Exemple du contenu d'un fichier .properties

Il doit avoir au moins autant de fichier .properties que de langue : Chaque langue a son .properties et il y a le .properties par défaut. Il y a une façon particulière de nommer ces fichiers. Nous y reviendrons.

Les ResourceBundle

Une ResourceBundle est une classe java rattachée à un fichier .properties. En fait plusieurs :

- Le fichier qui sera utilisé par défaut. Ex : login.properties
- Ceux qui seront utilisés pour d'autres langues. Ex : login_en_EN.properties, login_fr_FR.properties, login_fr.properties.

NB : Ces fichiers doivent être dans le même répertoire.

Comme vous pouvez le constater, les autres fichiers ont des noms particuliers.

Il y a-t-il une logique ?

Forcement !

Pour ceux qui ne l'on pas remarquée, la voici :

- Il commence exactement comme le nom du fichier par défaut.
- Ensuite on ajoute le code la langue et/ou du pays séparés par un espace.

Ex : Le fichier qui sera utilisé pour traduire l'application en Allemand sera : login_de.properties.

Un ResourceBundle nous permet selon la langue courante de sélectionner le fichier qui sera utilisé pour la traduction.

Il suffit de faire

ResourceBuilder.getBundl(" « chemin_d'accès »/ « Nom_fichier_par_defaut »", « Locale ») ;

« Locale » désigne l'instance de la classe Locale correspondante à la langue qu'on veut utiliser.

```
ResourceBundle titres;  
Locale fr = new Locale("fr");  
titres = ResourceBundle.getBundl("languages/login", fr);
```

Initialisation d'un ResourceBundle

Naturellement, il faudra importer toutes les packages nécessaires. Comme chacun de nous utilise un IDE, nous ne nous attarderons pas dessus ; l'IDE fera son job en vous disant quels packages importer.

Sachez que java.util contient tout le nécessaire.

L'utilisation d'un ResourceBundle

A présent que nous avons initialiser notre ResourceBundle, comment l'utiliser ?

On avait dit que les .properties contient des couples clés/valeur.

Oui et alors ?

Les clés devront être identiques dans toutes les fichiers .properties rattaché au fichier .properties par défaut.

Pour implémenter l'internalisation, il suffira pour chaque texte à afficher de l'associer à une clés et de donner la signification de ce texte dans les différentes langues(dans chaque fichier ,properties). Au lieu d'écrire le texte, on utilisera la méthode getString() et on lui donnera la clés en paramètre.

titres.getString(«appName ») ;

Si la clé fournie en paramètre n'est pas disponible, une MissingResourceException est levée.

Ces fameuses lignes rouges que nous connaissons crès bien. Cette image vous parle bien sûr !

```

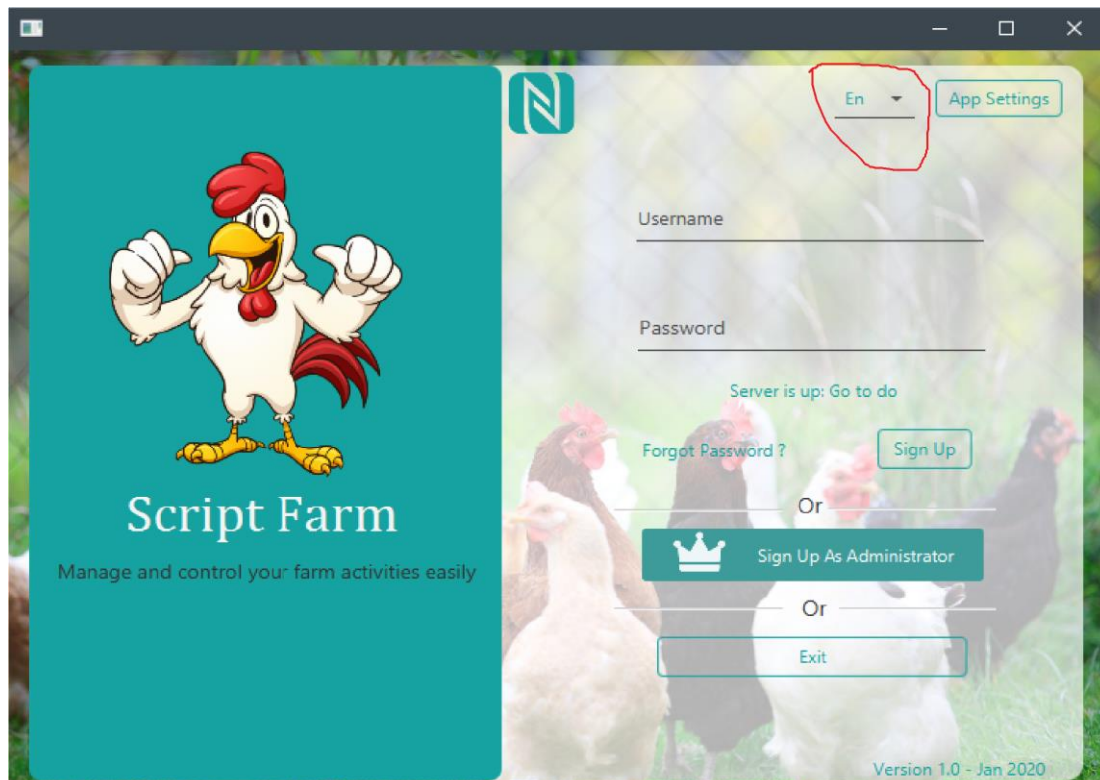
... 1 more
Caused by: java.util.MissingResourceException: Can't find resource for bundle java.util.PropertyResourceBundle, key appname
    at java.util.ResourceBundle.getObject(ResourceBundle.java:450)
    at java.util.ResourceBundle.getObject(ResourceBundle.java:444)
    at java.util.ResourceBundle.getString(ResourceBundle.java:407)
    at controller.LoginController.setResource(LoginController.java:157)
    at controller.LoginController.loadResource(LoginController.java:152)

```

MissingResourceException

L'internationalisation et Script Farm

Dans la vue du login, on a ajouté un comboBox qui nous permet de sélectionner la langue :



login

Dans le contrôleur, on initialise notre ResourceBundle et nos variables Locales qui vont identifier nos langues (Français et Anglais) et une variable String qui nous permettra de connaître la langue utilisée à tout instant.

```

@FXML
private JFXComboBox<String> language;

ResourceBundle titres;

Locale en = new Locale("en");

Locale fr = new Locale("fr");

String lang = "En";

```

Pour gérer l'internationalisation, on a créé trois méthodes :

- Un listener sur la comboBox

```
@FXML
void languageChanged(ActionEvent event){
    Tools.print("Language Changed");
    String v = language.getValue();
    if(!v.equalsIgnoreCase(lang)){
        loadResource(v);
        lang = v;
    }
}
```

- Une méthode qui charge la ressource correspondante en fonction de la langue qui lui est envoyée

```
@FXML
public void initReport() { ...15 lines }

void loadResource(String language){
    if(language.equalsIgnoreCase("Fr"))
    {   Tools.print("Lang = FR");
        titres = ResourceBundle.getBundle("languages/login", fr);
    }
    else{
        titres = ResourceBundle.getBundle("languages/login", en);
        Tools.print("Lang = En");
    }
    setResource();
}
```

- Une méthode qui applique la ressource sur toute l'interface

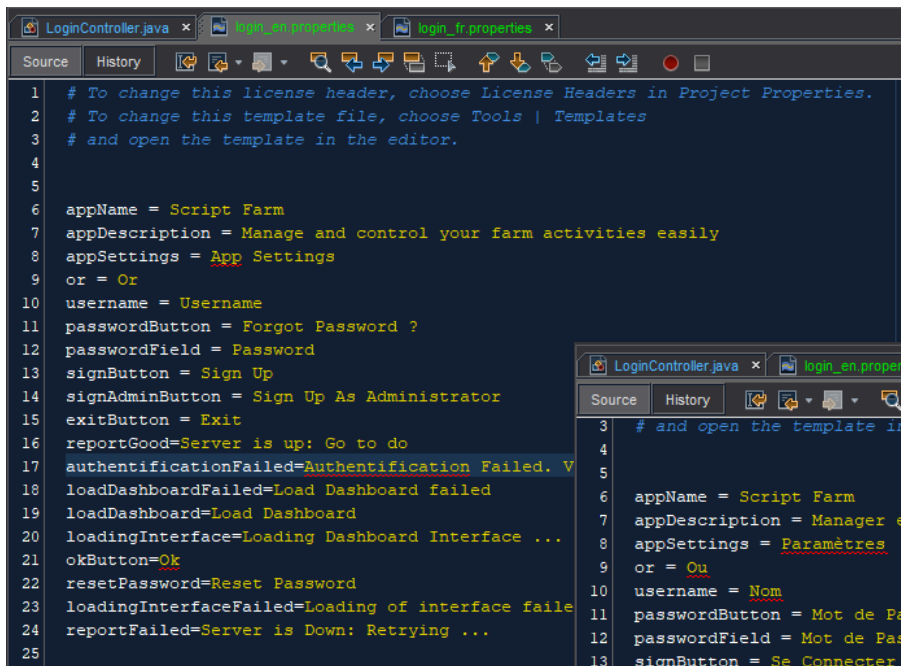
```
void setResource(){
    appName.setText(titres.getString("appName"));
    appDescription.setText(titres.getString("appDescription"));
    appSettings.setText(titres.getString("appSettings"));
    username.setPromptText(titres.getString("username"));
    passwordField.setPromptText(titres.getString("passwordField"));
    passwordButton.setText(titres.getString("passwordButton"));
    signButton.setText(titres.getString("signButton"));
    exitButton.setText(titres.getString("exitButton"));
    signAdminButton.setText(titres.getString("signAdminButton"));
    if(report.getId().equalsIgnoreCase("report"))
        report.setText(titres.getString("reportGood"));
    else if(report.getId().equalsIgnoreCase("reportFailed"))
        report.setText(titres.getString("authenticationFailed"));
    else{}
    or.setText(titres.getString("or"));
    or2.setText(titres.getString("or"));
}
```

Dans la méthode initialize du controller, on initialise le comboBox , et on charge la resource

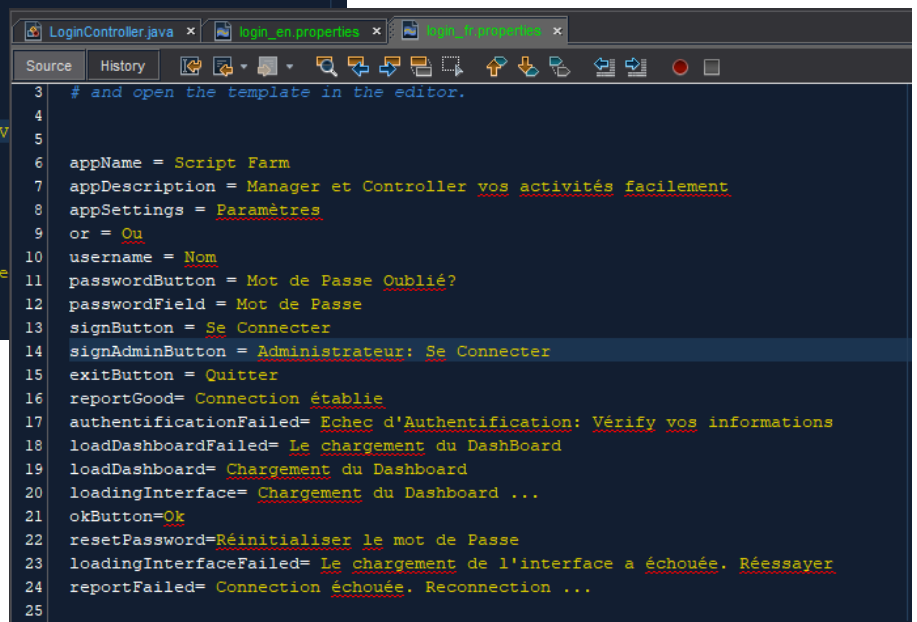
```
@Override
public void initialize(URL url, ResourceBundle rb){
    language.getItems().addAll("En","Fr");
    language.setValue("En");
    loadResource("En");
}
```

Mais où sont les fichiers .properties ?

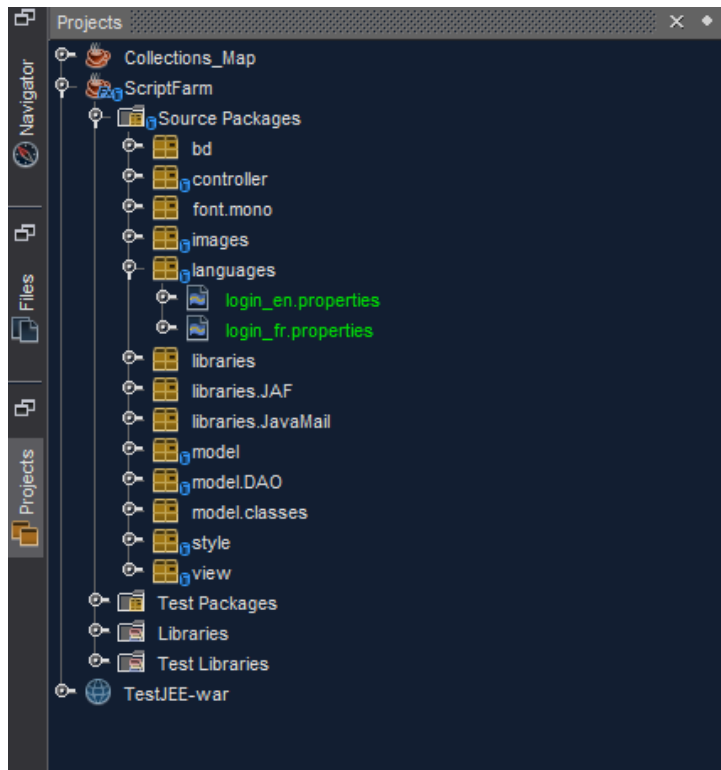
Ils sont là Bro. Juste en dessous.



```
1 # To change this license header, choose License Headers in Project Properties.
2 # To change this template file, choose Tools | Templates
3 # and open the template in the editor.
4
5
6 appName = Script Farm
7 appDescription = Manage and control your farm activities easily
8 appSettings = App Settings
9 or = Or
10 username = Username
11 passwordButton = Forgot Password ?
12 passwordField = Password
13 signButton = Sign Up
14 signAdminButton = Sign Up As Administrator
15 exitButton = Exit
16 reportGood=Server is up: Go to do
17 authenticationFailed=Authentication Failed. V
18 loadDashboardFailed=Load Dashboard failed
19 loadDashboard=Load Dashboard
20 loadingInterface=Loading Dashboard Interface ...
21 okButton=Ok
22 resetPassword=Reset Password
23 loadingInterfaceFailed=Loading of interface failed
24 reportFailed=Server is Down: Retrying ...
25
```



```
3 # and open the template in the editor.
4
5
6 appName = Script Farm
7 appDescription = Manager et Controller vos activités facilement
8 appSettings = Paramètres
9 or = Ou
10 username = Nom
11 passwordButton = Mot de Passe Oublié?
12 passwordField = Mot de Passe
13 signButton = Se Connecter
14 signAdminButton = Administrateur: Se Connecter
15 exitButton = Quitter
16 reportGood= Connection établie
17 authenticationFailed= Echec d'Authentification: Vérify vos informations
18 loadDashboardFailed= Le chargement du DashBoard
19 loadDashboard= Chargement du Dashboard
20 loadingInterface= Chargement du Dashboard ...
21 okButton=Ok
22 resetPassword=Réinitialiser le mot de Passe
23 loadingInterfaceFailed= Le chargement de l'interface a échouée. Réessayer
24 reportFailed= Connection échouée. Reconnection ...
25
```



NDEMA @The Script Farm Team