Automatisation de Microsoft Excel pour Mac OS X en utilisant Java et AppleScript

Table des matières :

I Introduction

Au cours de ce tutoriel vous allez pouvoir voir comment on peut utiliser Applescript et Java afin d'automatiser l'envoi de données vers l'application Microsoft Excel pour Mac OS X. Avec Java 6 il est maintenant plus aisé de réaliser cette tâche. Ce tutoriel a été réalisé en utilisant Netbeans, mais ceci fonctionne naturellement avec n'importe quel éditeur de code Java. Voyons ce qui est nécessaire afin de réaliser cette tâche dans de bonnes conditions…

II Contexte de cet article

Nous allons aborder ce sujet en construisant une classe que vous pourrez ensuite réutiliser dans vos travaux futurs ou pourquoi pas pour l'améliorer. Cette classe va vous permettre de définir la structure de votre feuille de calcul. Il va vous permettre ensuite de définir l'exportation des données automatique vers votre feuille de calcul que vous aurez préalablement définie.

III utilisation du jdk 6

Pour réaliser ces opérations, il va vous falloir préalablement utiliser dans votre projet le jdk 6 de java dans la page des propriétés de votre projet, dans la catégorie Sources en bas de la page.

IV Ecriture de la classe

1. Création de la nouvelle classe

Nous allons maintenant entrer dans le vif du sujet. Vous allez donc créer une nouvelle classe Java, appelé le nouveau fichier objExcel par exemple et le fichier est créé avec le corps de la nouvelle classe comme ceci :

```
public class objExcel {

}
```

Nous allons maintenant définir tous les aspects nécessaires de la classe en commençant par les différentes méthodes utiles pour gérer les scripts de type AppleScript, et ensuite nous verrons les méthodes permettant de concevoir une feuille de calculs sous Microsoft Excel et d'envoyer des données vers cette feuille.

2. Définition du corps de la classe

a. Définition de la méthode permettant de gérer des scripts sans argument

Avec java pour réaliser cette opération nous devons utiliser une méthode qui va permettre de créer un moteur de script AppleScript, ceci est réalisable en utilisant le type d'objet ScriptEngineManager qui va permettre de créer un gestionnaire de moteur de script. Une fois ceci effectué il suffit de lui associer un moteur de script AppleScript en créant une instance d'objet de type ScriptEngine en lui associant le type de script utiliser sous forme d'une chaine de caractères.

```
ScriptEngineManager em = new ScriptEngineManager();
ScriptEngine appleScript = em.getEngineByName("AppleScript");
```

On utilise une gestion d'exceptions pour prévenir tout plantage éventuel au cas où l'évaluation du script par le moteur ne pourrait pas se faire. Cette méthode prend en argument une chaine de caractères qui n'est rien d'autre que le script AppleScript qui sera passée en arguments à notre moteur de script (et initialement à notre méthode que l'on vient de construire). L'évaluation du script se fait grâce à la méthode eval de notre instance appleScript.

```
try {
        appleScript.eval(Script);
    }
    catch(ScriptException e) {
        e.printStackTrace();
    }
  }
```

Voici la méthode dans son intégralité :

```
public void RunScriptWithNoArgs(String Script) {
    ScriptEngineManager em = new ScriptEngineManager();
     ScriptEngine appleScript = em.getEngineByName("AppleScript");
try {
        appleScript.eval(Script);
    }
    catch(ScriptException e) {
        e.printStackTrace();
    }
  }
```

b. Définition de la méthode permettant de gérer des scripts avec arguments

La méthode précédente n'est utile que pour la gestion de scripts n'ayant pas d'arguments en paramètre. Nous allons voir maintenant la méthode qui nous permet de gérer des arguments. Le cœur de la méthode est sensiblement identique à celle conçue précédemment. La grosse nouveauté va être le passage des paramètres au script.
Le début et la fin de la méthode sont identiques à ce que l'on vient de définir précédemment. Donc nous allons nous intéresser à la partie permettant de définir la définition des paramètres. Pour réaliser cette opération nous avons besoin de définit un contexte de script qui est défini à partir de l'instance de type ScriptEngine que nous venons de définir précédemment.

```
ScriptContext context = appleScript.getContext();
```

Une fois ce contexte définit nous allons lui associer une instance d'objet « Bindings » (mappage clé/valeur), en utilisant la méthode getBindings du contexte que l'on vient de créer.

```
Bindings bind = context.getBindings(ScriptContext.ENGINE_SCOPE);
```

Une fois le l'instance Bindings créée on effectue une suppression de toutes les paires clés/valeurs pouvant resté en mémoire. Par l'intermédiaire de la méthode clear().

```
bind.clear();
```

Une fois cette opération effectuée il est alors possible de passer les nouvelles valeurs correspondant aux paramètres que l'on veut passer à notre script.
Comme premier paramètre à passer en argument il faut créer une première paire clé/valeur référençant le nom de la méthode AppleScript (celle que l'on définit comme on « nom_fonction » au début de chaque script. Ceci est effectué grâce à la fonction put de notre instance de type Bindings, comme ceci :

```
bind.put("javax_script_function", functionName);
```

functionName étant un paramètre de notre méthode de classe.

Il suffit ensuite d'associer les paires de clé/valeurs qui permettent de référencer les différents paramètres qui seront utilisés pour le script.

```
bind.put(ScriptEngine.ARGV, args);
```

Les arguments sont passés à notre méthode de classe sous forme de collection.

Voici donc la méthode complète :

```
public void RunScriptWithArgs(String Script, String functionName, List args) {
```

```
ScriptEngineManager em = new ScriptEngineManager();
 ScriptEngine appleScript = em.getEngineByName("AppleScript");

 ScriptContext context = appleScript.getContext();



 Bindings bind = context.getBindings(ScriptContext.ENGINE_SCOPE);
 bind.clear();
 bind.put("javax_script_function", functionName);

 bind.put(ScriptEngine.ARGV, args);



 try {
    appleScript.eval(Script, bind);
 }
 catch(ScriptException e) {
    e.printStackTrace();
 }
}
```

c. Définition du constructeur de la classe

Nous allons poursuivre par le constructeur. Nous ne l'avons pas expliciter avant parce que le constructeur s'appuie sur une des méthodes créées précédemment. En effet , dans cette partie nous allons demander l'ouverture de l'application Microsoft Excel et ensuite créer une feuille de calcul. Dans cet exemple nous allons créer une feuille de calcul par défaut nommée « Feuil1 », donc nous n'avons pas besoin de créer de commandes Applescript qui gèrent des arguments.
Il faut dans un premier temps définir le script sous forme d'une chaine de caractère que l'on va créer grâce à la propriété appleScriptCommand que l'on a défini précédemment.
Voici le script :

```
appleScriptCommand =  "tell application \"Microsoft Excel\"\n" +
                                    "activate\n" +
                                    "open \"Feuil1\" \n" +

                                   "end tell\n";
```

Si l'on regarde d'un peu plus près ce script, on voit que l'on demande le lancement de l'application Excel par la commande tell, on active ensuite l'application par l'intermédiaire de la commande « activate ». Ensuite on demande l'ouverture d'une feuille de calcul qui se nomme par défaut Feuil1. Et on termine le script par la commande « end tell ». A noter que les caractères « \n » sont nécessaire pour marquer un retour à la ligne.
Il suffit d'appeler ensuite la fonction RunScriptWithNoArgs définie précédemment :

```
public objExcel() {
    appleScriptCommand =  "tell application \"Microsoft Excel\"\n" +
                          "activate\n" +
                          "open \"Feuil1\" \n" +
                          "end tell\n";
    RunScriptWithNoArgs(appleScriptCommand);
  }
```

d. Création des entêtes

Nous allons créer la méthode de création des entêtes. Nous allons gérer ce type de données en utilisant un tableau de type redimensionnable c'est-à-dire de type ArrayList. Chaque nom d'entête sera donc ajouté à cette liste dynamique. Cette liste est passée en paramètre de la méthode de création des entêtes.
Voyons maintenant le script AppleScript qui va être utilisé pour créer nos entêtes de colonnes dans Excel.

```
String appleScriptCommand =   "on InsertHeader(numCol, valeur)\n" +
                              "tell application \"Microsoft Excel\"\n" +
                               "activate application \"Microsoft Excel\"\n" +

                               "activate object worksheet \"Feuil1\" \n" +

                               "set value of cell 1 of column numCol of active
                               sheet to valeur \n" +

                              "end tell\n" +
                              "end InsertHeader";
```

Si l'on regarde le script de plus près, on s'aperçoit que les paramètres sont passés en argument entre parenthèse lors de l'appel à la commande de déclaration de la méthode « on InsertHeader ». On réactive ensuite l'application Excel ainsi que sa feuille de style que l'on a créé dans le constructeur. L'ajout de valeur dans une cellule Excel se fait par l'intermédiaire de cette ligne :

```
set value of cell 1 of column numCol of active sheet to valeur
```

Les entêtes de colonne sont créées sur la première ligne. Seul le numéro de colonne change, cette notification est prise en compte par le paramètre numCol que l'on a associé à la méthode. Le paramètre « valeur » contient le nom de l'entête. Il nous reste maintenant à définir comment ces paramètres sont envoyés à notre méthode en appelant la fonction « RunScriptWithArgs ». La méthode de classe que l'on a créé attend qu'on lui fournisse comme paramètre une liste d'headers (List headers).
Cette liste d'headers va être traité par l'intermédiaire de cette portion de code :

```
for(int cpt=0; cpt < headers.size(); cpt++) {
        List<Object> args = new ArrayList<Object>();
        args.add(cpt+1);
```

```
            args.add(headers.get(cpt).toString());
            RunScriptWithArgs(appleScriptCommand, "InsertHeader", args);
        }
```

On va compter le nombre d'éléments contenu dans notre liste. Cette information va nous fournir le numéro de colonne que l'on associe en premier argument dans notre liste. La seconde valeur étant le nom de l 'entête. Ces informations sont donc ajoutées à une liste dynamique qui va être fournie en paramètre à notre méthode de classe RunScriptWithArgs, en même temps que le nom de la méthode, et du script que l'on vient de créer.

Voici la méthode dans son intégralité :

```
    public void CreateHeaders(List headers) {
        String applescriptCommand =   "on InsertHeader(numCol, valeur)\n" +
                                      "tell application \"Microsoft Excel\"\n" +
                                      "activate application \"Microsoft Excel\"\n" +

                                      "activate object worksheet \"Feuil1\" \n" +

                                        "set value of cell 1 of column numCol of active sheet to
                                        valeur \n" +

                                      "end tell\n" +
                                      "end InsertHeader";




        for(int cpt=0; cpt < headers.size(); cpt++) {
            List<Object> args = new ArrayList<Object>();
            args.add(cpt+1);
            args.add(headers.get(cpt).toString());
            RunScriptWithArgs(applescriptCommand, "InsertHeader", args);
        }

    }
```

e. Méthode d'insertion de données

La définition de cette méthode est sensiblement identique à celle que nous venons de voir dans la partie précédente à ceci près que dans ce cas précis nous avons un paramètre de plus à définir en effet la ligne est un paramètre de plus qui va entrer en jeu…

Voici le script dans son intégralité

```
String applescriptCommand =   "on InsertData(numCol, numLig, valeur)\n" +
                        "tell application \"Microsoft Excel\"\n" +
                        "activate application \"Microsoft Excel\"\n" +
                        "activate object worksheet \"Feuil1\" \n" +
                         "set value of cell NumLig of column numCol of active sheet to valeur
                \n" +
                        "end tell\n" +
                        "end InsertData";
```

Comme nous pouvons le constater seule la partie qui permet d'ajouter une valeur à une cellule donnée change.
L'envoi des paramètres au script se fait de la même manière que précédemment.
Imaginons que nous avons par exemple une feuille de calcul dans laquelle on veut insérer des données concernant des pronostics de pari sur une série de match. La méthode peut s 'écrire comme ceci

```
public void InsertData(String[] MatchName, float[] Prono1, float[] PronoN, float[]
Prono2) {
    String appleScriptCommand =   "on InsertData(numCol, numLig, valeur)\n" +
                        "tell application \"Microsoft Excel\"\n" +
                        "activate application \"Microsoft Excel\"\n" +

                        "activate object worksheet \"Feuil1\" \n" +

                        "set value of cell NumLig of column numCol of active sheet to valeur
\n" +

                        "end tell\n" +
                        "end InsertData";


    for(int col=1; col <= 4; col++) {
      for(int NumLig=0; NumLig < MatchName.length; NumLig++) {


      List<Object> args = new ArrayList<Object>();
      args.add(col);
      args.add(NumLig+2);
      if (col == 1) {
        if (!MatchName[NumLig].equals("")) {
          args.add(MatchName[NumLig]);

        }
        else {
          break;
        }

      }
      if (col == 2) {
```

```
            if (Prono1[NumLig] != 0) {
                args.add(Prono1[NumLig]);
            }
            else {
                break;
            }

        }
        if (col == 3) {
            if (PronoN[NumLig] != 0){
                args.add(PronoN[NumLig]);
            }
            else {
                break;
            }
        }
        if (col == 4) {
            if (Prono2[NumLig] != 0) {
                args.add(Prono2[NumLig]);
            }
            else {
                break;
            }

        }


        RunScriptWithArgs(applescriptCommand, "InsertData", args);


        }
    }
```

Cette méthode est un exemple mais vous permet de voir comment introduire des données dans un tableau Excel, à vous de personnaliser cette méthode pour qu'elle puisse traiter des informations de façon plus générale. Comme vous pouvez le constater les données sont intégrées dans la liste « args » par colonne. Le script est lancé pour chaque colonne.

V Conclusion

Ce tutoriel vous a permis de voir qu'il était possible d'automatiser l'utilisation d'Excel à partir d'une application écrite avec le langage Java. Et ce en utilisant le moteur de script AppleScript d'Apple. Donc Java dans sa version 6 est une bonne alternative pour créer des petites applications d'automatisation de tâches, même si Apple semble renier l'univers Java.