

# Créer un nouveau format d'entrée pour LogParser en VBScript

par Baptiste Wicht ([home](#))

Date de publication : Le 5 Décembre 2007

Cet article va vous apprendre à développer votre propre format d'entrée pour LogParser en Visual Basic Script.

1 - Introduction.....	3
1.1 - LogParser, c'est quoi ?.....	3
1.2 - WSC, c'est quoi ?.....	3
2 - Créer le nouveau format d'entrée.....	4
3 - Implémentation des méthodes.....	6
3.1 - OpenInput.....	6
3.2 - GetFieldCount.....	6
3.3 - GetFieldName.....	6
3.4 - GetFieldType.....	7
3.5 - ReadRecord.....	7
3.6 - GetValue.....	7
3.7 - CloseInput.....	7
4 - Utilisation.....	8
5 - Conclusion.....	10
5.1 - Le code complet.....	10

## 1 - Introduction

Dans cet article, nous allons apprendre comment développer un plugin WSC pour LogParser.

### 1.1 - LogParser, c'est quoi ?

LogParser est un outil gratuit de Microsoft permettant de faire des recherches dans des fichiers de logs. Mais il permet également bien plus que cela; il peut lire dans les journaux d'événements système, dans différents fichiers, dans l'Active Directory ou encore dans des flux RSS.

Il permet ensuite de récupérer ces informations dans différents modes de sortie, des fichiers logs, lignes de commande, des graphiques Excel, dans une base de données, ...

LogParser utilise un langage de requêtage très proche du SQL, par exemple, pour rechercher toutes les dates et les adresses ip des serveurs concernés dans un fichier de log IIS, il suffirait de faire :

```
LogParser "SELECT date, time, s-ip FROM iis.log" -i:IIS
```

Il permet encore quelques autres petites choses, mais comme ce n'est pas le sujet de cet article, je ne vais pas m'étendre sur ce sujet. Si vous voulez plus d'informations sur LogParser, je vous invite à lire [cette page](#) chez Microsoft.

Vous pouvez le télécharger [ici](#).

### 1.2 - WSC, c'est quoi ?

Windows Script Components, c'est une façon de créer des composants COM réutilisables et puissants avec des langages de script. On peut créer des WSC avec du VBS, du JavaScript, du PerlScript, du PScript ou encore du Python.

Ces composants s'utilisent de manière quasi similaire à des DLL, on peut les inscrire ou les désinscrire de l'ordinateur. Une fois inscrits, les autres programmes peuvent les utiliser.

## 2 - Créer le nouveau format d'entrée

Comme vous avez pu le lire dans l'introduction, LogParser supporte de nombreux formats de logs. Mais comment faire pour gérer son propre format de log ? LogParser propose l'utilisation de composants COM pour gérer le format d'entrée qu'on veut.

Dans notre cas, admettons, qu'on ait un simple format de log qui comprend la date, l'heure, l'adresse ip source, l'adresse IP destination et une action. Tous ces champs sont séparés par des espaces et les champs vides sont représentés par des -. Chaque fichier log a également un entête. Voici donc un exemple de fichier de log dans notre format :

```
#Software: Nom du logiciel
#Version: 1.0
#Date:2007-10-31 00:00:06
#Fields: date time c-ip s-ip action
2007-10-31 00:00:06 10.16.18.125 172.16.3.89 /actions/bonjour.do
2007-10-31 00:00:16 10.16.18.125 172.16.25.56 /actions/bonjour.do
2007-10-31 00:01:06 10.16.18.34 172.16.25.78 /actions/hello.do
2007-10-31 00:10:26 10.16.18.56 172.16.25.12 /actions/print.do
2007-10-31 00:20:06 10.16.18.89 172.16.25.98 /actions/bonjour.do
2007-10-31 03:02:06 10.16.18.12 172.16.25.44 /actions/edit.do
2007-11-01 10:02:16 10.16.18.190 172.16.25.33 /actions/aurevoir.do
```

Ce n'est pas un format connu de LogParser, il ne pourra donc pas y faire de requêtes. Mais on peut y remédier en développant un "plugin" pour LogParser qui va se charger de la partie récupération de données.

On va donc créer notre fichier WSC. Un fichier WSC est en fait un fichier XML qui contient une partie sur l'enregistrement du composant dans le système, une partie présentant les méthodes publiques et une partie avec le script qui implémente les méthodes publiques.

Tout d'abord, il faut bien sûr un entête XML et le composant root se nommera component :

```
<?xml version="1.0"?>
<component>
```

Ensuite, on va devoir créer la partie "registration" qui contient les informations sur notre composant :

```
<registration
  description="Notre format"
  progid="input.WSC"
  version="1.00"
  classid="{c47fg5f7-1520-4984-9848-6a745fg7fd84}"
>
</registration>
```

Voilà ce que représentent ces champs :

- **description** : C'est simplement la description de notre composant
- **progid** : C'est l'identifiant de notre composant. Je préconise d'utiliser le nom du fichier
- **version** : C'est la version de notre composant
- **classid** : C'est un numéro d'identification qui peut être utilisé à la place du progid

Donc il n'y a rien de compliqué de ce côté-là ;)

On va ensuite passer à la description des méthodes publiques. Pour cela, on va créer des éléments method, dans un élément parent public. Une méthode se présente sous la forme suivante :


```
<method name="Nom de la méthode">
  <PARAMETER name="Nom du paramètre 1"/>
```

```

<PARAMETER name="Nom du paramètre 2"/>
</method>

```

Je crois que l'exemple parle de lui-même.

 *On peut aussi ajouter des paramètres globaux qui seront mis dans la ligne de commande. Pour cela, on utilise un élément property avec en attribut name son nom et on ajoute un élément put vide pour indiquer qu'on va créer une méthode putNomDeLaPropriété.*

Les méthodes qu'il faut implémenter pour pouvoir gérer un format d'entrée personnel sont les suivantes :

- *OpenInput* : Ouvre la source
- *GetFieldCount* : Retourne le nombre de champs du format
- *GetFieldName* : Retourne le nom d'un champ
- *GetFieldType* : Retourne le type d'un champ
- *GetValue* : Retourne la valeur pour un champ précis
- *ReadRecord* : Lit un élément de la source
- *CloseInput* : Ferme la source

Voici donc comment on va les déclarer :

```

<public>
<method name="OpenInput">
  <PARAMETER name="fileName"/>
</method>
<method name="GetFieldCount">
</method>
<method name="GetFieldName">
  <PARAMETER name="index"/>
</method>
<method name="GetFieldType">
  <PARAMETER name="index"/>
</method>
<method name="GetValue">
  <PARAMETER name="index"/>
</method>
<method name="ReadRecord">
</method>
<method name="CloseInput">
  <PARAMETER name="bAbort"/>
</method>
</public>

```

Donc de coté-là, rien de spécial. On va maintenant passer à l'implémentation de chacune des méthodes.

### 3 - Implémentation des méthodes

L'implémentation des différentes méthodes se fait dans la balise script :

```
<script language="VBScript">
<![CDATA[

]]>
</script>
```

Tout le code se trouvera dans cette balise.

#### 3.1 - OpenInput

Cette méthode permet d'ouvrir le fichier log. On devra aussi passer les 4 lignes d'entête pour ne pas qu'elles soient prises en compte dans la recherche :

```
Dim m_file
Dim m_objFso

Function OpenInput(fileName)
    Set m_objFso = CreateObject("Scripting.FileSystemObject")
    Set m_file = m_objFso.OpenTextFile(fileName, 1, false)

    'On passe outre le header
    m_file.ReadLine
    m_file.ReadLine
    m_file.ReadLine
    m_file.ReadLine
End Function
```

On ouvre donc le fichier de log et ensuite on lit les 4 premières lignes d'entête.

#### 3.2 - GetFieldCount

Cette méthode doit juste retourner le nombre de champs du format. Donc 5 dans notre cas :

```
Function GetFieldCount()
    GetFieldCount = 5
End Function
```

#### 3.3 - GetFieldName

On doit retourner cette fois le nom du champ en fonction de son index :

```
Function GetFieldName(index)
    Select Case index
        Case 0
            GetFieldName = "date"
        Case 1
            GetFieldName = "time"
        Case 2
            GetFieldName = "c-ip"
        Case 3
            GetFieldName = "s-ip"
        Case 4
            GetFieldName = "action"
    End Select
End Function
```

### 3.4 - GetFieldType

Dans notre cas, tous les champs sont des chaînes de caractères, on va donc retourner 3 qui indique un String :

```
function GetFieldType(index)
    'Tous les champs sont de type String
    GetFieldType = 3
end function
```

Voici tous les types possibles :

- integer : 1
- real : 2
- string : 3
- timestamp : 4
- null : 5

### 3.5 - ReadRecord

Cette méthode va permettre de lire l'enregistrement suivant. Dans notre cas, on va aussi séparer la ligne de log à chaque espace pour trouver les valeurs, de plus, il faut retourner False si le fichier est à la fin :

```
Dim m_tokens

function ReadRecord()
    Dim currentLine

    If m_file.AtEndOfStream Then
        ReadRecord = False
    Else
        currentLine = m_file.ReadLine
        m_tokens = Split(currentLine)

        ReadRecord = True
    End If
end function
```

Rien de bien compliqué donc.

### 3.6 - GetValue

Cette méthode permet de récupérer la valeur d'un champ dans l'élément en cours :

```
function GetValue(index)
    GetValue = m_tokens(index)
end function
```

### 3.7 - CloseInput

Et enfin, cette méthode permet de fermer le fichier d'entrée :

```
function CloseInput(bAbort)
    m_file.Close
    Set m_file = Nothing
    Set m_objFso = Nothing
end function
```

## 4 - Utilisation

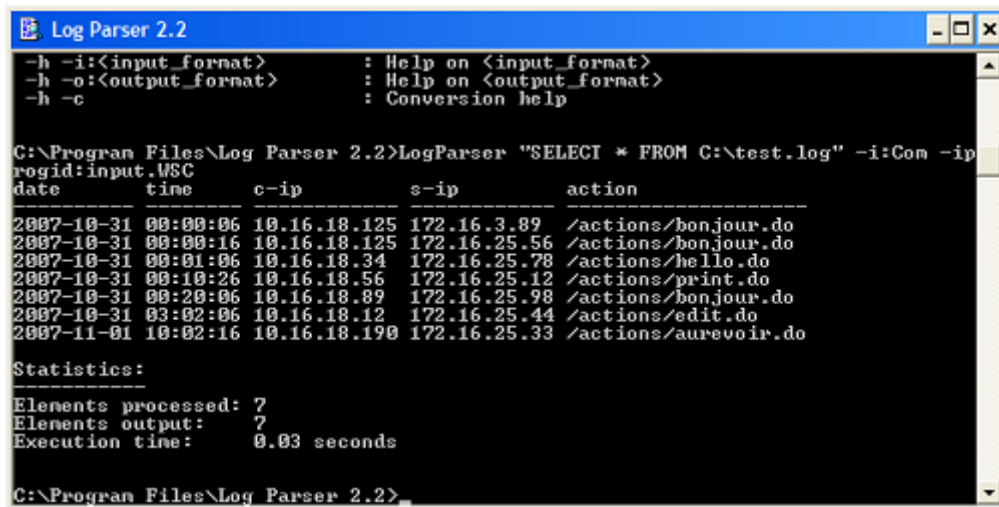
Voilà, nous avons maintenant codé notre fichier WSC, il ne nous reste plus qu'à l'utiliser.

La première chose à faire est d'enregistrer votre fichier .WSC. Pour cela, il suffit d'effectuer un clic droit dessus et d'utiliser le bouton register ou alors vous pouvez utiliser la commande regsvr32 dans l'invite de commandes.

Ensuite, on va pouvoir exécuter une requête LogParser avec notre format perso :

```
LogParser "SELECT * FROM C:\test.log" -i:Com -iprogid:input.WSC
```

Ce qui nous donnera dans la console :



```

Log Parser 2.2
-h -i:<input_format>      : Help on <input_format>
-h -o:<output_format>    : Help on <output_format>
-h -c                    : Conversion help

C:\Program Files\Log Parser 2.2>LogParser "SELECT * FROM C:\test.log" -i:Com -iprogid:input.WSC
date      time      c-ip      s-ip      action
-----
2007-10-31 00:00:06 10.16.18.125 172.16.3.89 /actions/bonjour.do
2007-10-31 00:00:16 10.16.18.125 172.16.25.56 /actions/bonjour.do
2007-10-31 00:01:06 10.16.18.34 172.16.25.78 /actions/hello.do
2007-10-31 00:10:26 10.16.18.56 172.16.25.12 /actions/print.do
2007-10-31 00:20:06 10.16.18.89 172.16.25.98 /actions/bonjour.do
2007-10-31 03:02:06 10.16.18.12 172.16.25.44 /actions/edit.do
2007-11-01 10:02:16 10.16.18.190 172.16.25.33 /actions/aurevoir.do

Statistics:
Elements processed: 7
Elements output: 7
Execution time: 0.03 seconds

C:\Program Files\Log Parser 2.2>

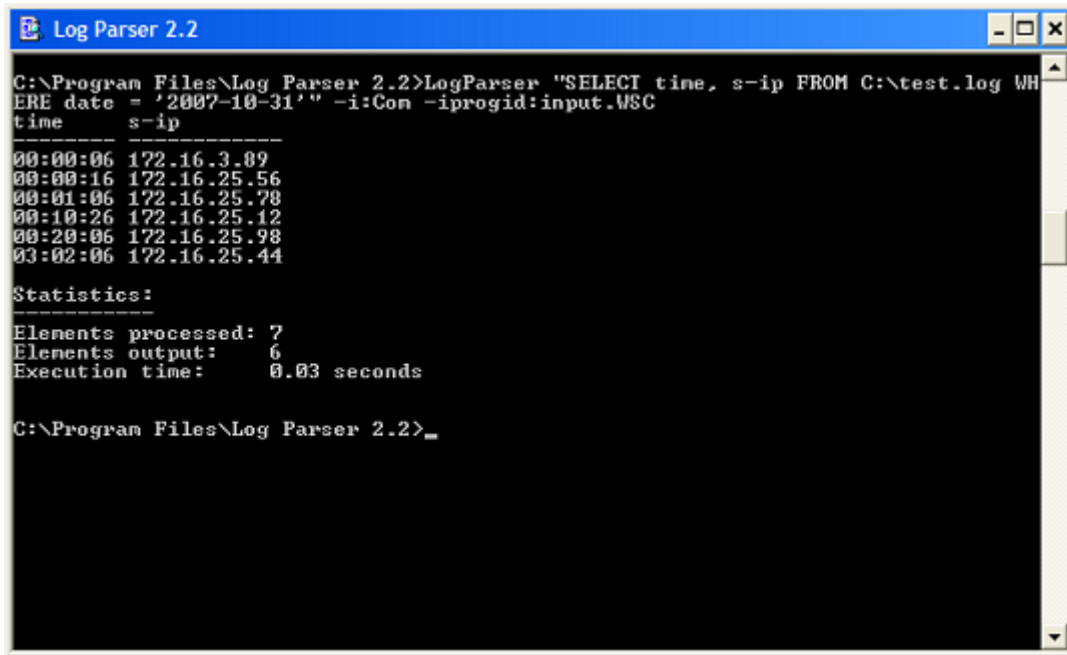
```

*Exécution LogParser*

On peut donc bien voir qu'il voit bien nos champs. Pour le tester vraiment, on pourrait essayer une autre requête :

```
LogParser "SELECT time, s-ip FROM C:\test.log WHERE date = '2007-10-31'" -i:Com -iprogid:input.WSC
```

Qui nous donnerait :



```
C:\Program Files\Log Parser 2.2>LogParser "SELECI time, s-ip FROM C:\test.log WHERE date = '2007-10-31'" -i:Con -iprogid:input.WSC
time      s-ip
-----
00:00:06 172.16.3.89
00:00:16 172.16.25.56
00:01:06 172.16.25.78
00:10:26 172.16.25.12
00:20:06 172.16.25.98
03:02:06 172.16.25.44

Statistics:
-----
Elements processed: 7
Elements output:    6
Execution time:     0.03 seconds

C:\Program Files\Log Parser 2.2>_
```

*Exécution LogParser*

Cela nous montre bien que notre format d'entrée a bien été pris en compte par LogParser.

## 5 - Conclusion

Nous voilà maintenant arrivés à la fin de cet article. Vous savez maintenant comment développer un "plugin" pour LogParser pour vos formats de fichier spéciaux.

Si vous voulez en savoir plus sur LogParser, je vous invite à le télécharger et à l'essayer. Une documentation très complète est livrée avec.

Un grand merci à **LineLe** pour ses corrections.

### 5.1 - Le code complet

Voici le code complet de notre fichier WSC :

```
input.WSC
<?xml version="1.0"?>
<component>

<registration
  description="Notre format"
  progid="input.WSC"
  version="1.00"
  classid="{c47fg5f7-1520-4984-9848-6a745fg7fd84}"
>
</registration>

<public>
  <method name="OpenInput">
    <PARAMETER name="fileName"/>
  </method>
  <method name="GetFieldCount">
  </method>
  <method name="GetFieldName">
    <PARAMETER name="index"/>
  </method>
  <method name="GetFieldType">
    <PARAMETER name="index"/>
  </method>
  <method name="GetValue">
    <PARAMETER name="index"/>
  </method>
  <method name="ReadRecord">
  </method>
  <method name="CloseInput">
    <PARAMETER name="bAbort"/>
  </method>
</public>

<script language="VBScript">
<![CDATA[

Dim m_file
Dim m_tokens
Dim m_objFso

function OpenInput(fileName)
  Set m_objFso = CreateObject("Scripting.FileSystemObject")
  Set m_file = m_objFso.OpenTextFile(fileName, 1, false)

  'On passe outre le header
  m_file.ReadLine
  m_file.ReadLine
  m_file.ReadLine
  m_file.ReadLine
End Function
```

**input.WSC**

```
function GetFieldCount()
  GetFieldCount = 5
End Function

Function GetFieldName(index)
  Select Case index
    Case 0
      GetFieldName = "date"
    Case 1
      GetFieldName = "time"
    Case 2
      GetFieldName = "c-ip"
    Case 3
      GetFieldName = "s-ip"
    Case 4
      GetFieldName = "action"
  End Select
End Function

Function GetFieldType(index)
  'Tous les champs sont de type String
  GetFieldType = 3
end Function

Function GetValue(index)
  GetValue = m_tokens(index)
end Function

Dim m_tokens

function ReadRecord()
  Dim currentLine

  If m_file.AtEndOfStream Then
    ReadRecord = False
  Else
    currentLine = m_file.ReadLine
    m_tokens = Split(currentLine)

    ReadRecord = True
  End If
end function

function CloseInput(bAbort)
  m_file.Close
  Set m_file = Nothing
  Set m_objFso = Nothing
end function

]]>
</script>

</component>
```