Les Expressions Rationnelles appliquées en VBA Access

par Charles A.

Date de publication : 28/02/2006

Dernière mise à jour : 28/02/2006

Thèmes abordés : - présentation de la bibliothèque Microsoft VBScript Regular Expressions - introduction à l'écriture d'expressions rationnelles - application à un cas pratique : la coloration syntaxique Niveau requis : moyen

I - Introduction

I-A - Mais de quoi parle-t-on ?

Le sujet de ce tutoriel est l'**implémentation dans Access des expressions rationnelles** également nommées expressions régulières de l'anglais *Regular Expressions* qu'on abrège communément en **RegEx** ou **RegExp**.

C'est un système de notation extrêmement puissant issu du monde UNIX qui a été conçu pour parcourir des chaînes, d'y trouver des sous-chaînes définies par des motifs et éventuellement de procéder à des remplacements.

I-B - Mais à quoi ça sert ?

Suite à une question posée sur le forum Access.

Bonsoir tout le monde !!

J'ai dans un logiciel de vidéothèque une table dont le titre est stocké de la façon suivante

titrefilm (article)

Exemple

12 TRAVAUX D'ASTÉRIX (LES)

Comment faire pour récupérer le titre sans l'article dans une requete ??

Quand l'auteur de la question dit : "titrefilm (article)" il définit ce qu'on appelle un motif et il ne reste plus qu'à le formaliser en langage **RegExp**.

On pourrait très bien proposer une fonction VBA avec un **InstrRev()** sur la parenthèse ouvrante, mais la solution **RegExp** reste de très loin plus élégante et puissante.

Dans cet article, le cas pratique retenu est la **coloration syntaxique**.

En effet, si nous souhaitons effectuer un export d'un module de code en y mettant une "intelligence" par la coloration syntaxique, nous prenons conscience de la difficulté de la tâche.

Par exemple : repérer des sous-chaînes entre double-quotes (") exige soit d'effectuer plusieurs séries de **Split()** soit de parcourir la chaîne, caractère par caractère.

L'utilisation d'expressions rationnelles nous soulage de cette complexité.

I-C - Les contributions de Developpez.com sur les RegExp

N'hésitez pas à consulter les articles de mes confrères sur le même sujet dans leur langage respectif.

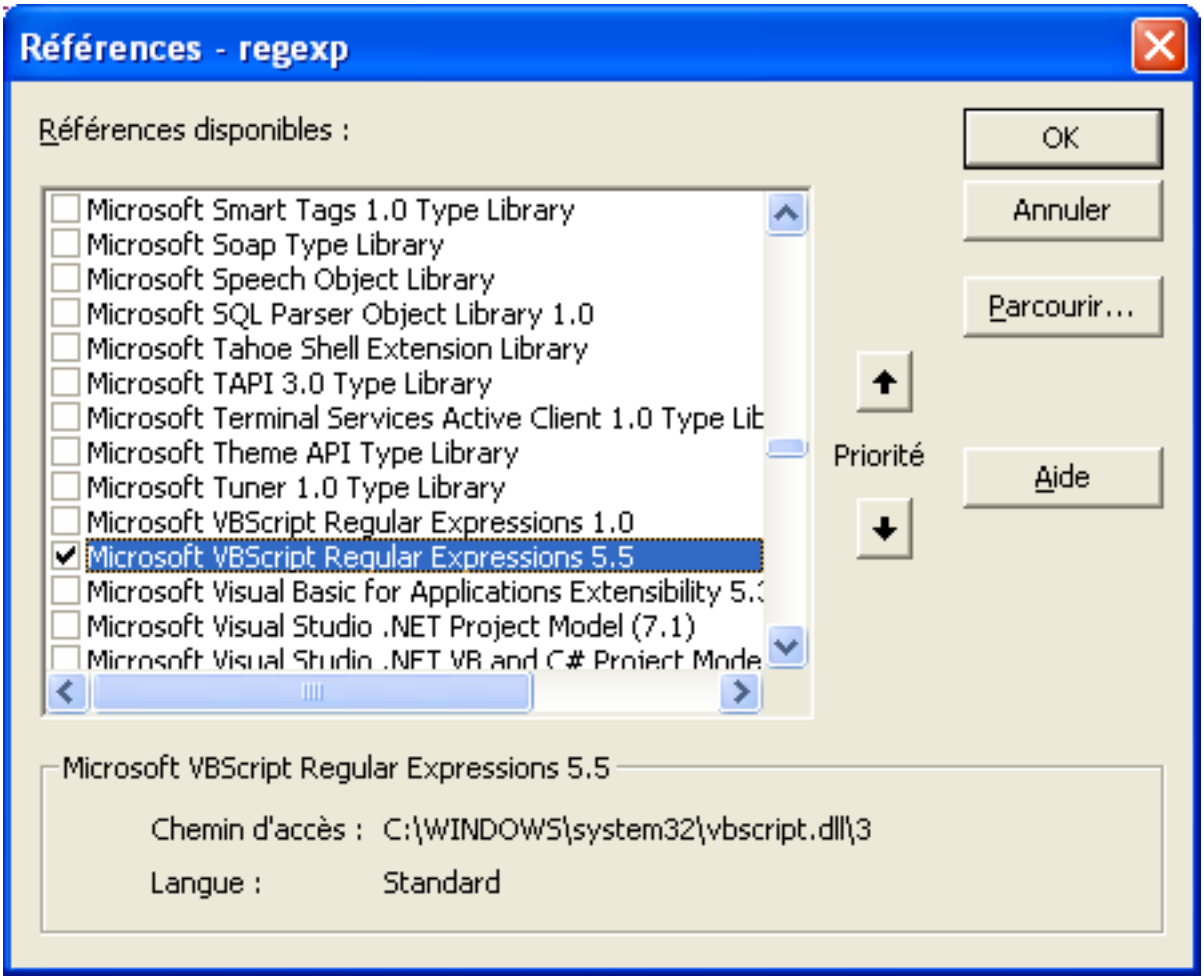
auteur	langage	article
Louis-Guillaume Morand	**.Net**	Utilisation des expressions régulières en .Net
Nicolas Joseph	**C**	les expressions régulières en C
Hugo Etievant	**PHP**	Les expressions régulières et les fonctions standard de traitement des chaînes de caractères
Michel Deriaz	**JAVA**	RegexSR - Maniement d'expressions régulières
Hugo Etievant	**JAVA**	Les expressions régulières avec l'API Regex de JAVA

II - La bibliothèque Microsoft VBScript Regular Expressions

Microsoft Access n'inclue pas les regexp de manière native puisque nous avons vu qu'elles viennent de l'univers Unix.

Cependant pour parer à cette lacune, Microsoft a mis à disposition une bibliothèque de code : **Microsoft VBScript Regular Expressions**.

La première étape consiste à l'ajouter à vos références de projet, dans l'éditeur : Menu **Outils > Références ...**



Ajout de la référence VBScript Regular Expressions

II-A - Présentation théorique

L'expression rationnelle vise à repérer une ou plusieurs structures identifiées au sein d'une chaîne de caractères.

Ce repérage se fait au moyen de la définition d'un **motif** (*Pattern*).

Les fonctions possibles sont ensuite la **validation**, l'**exploration** et les **remplacements**.

II-A-1 - Les propriétés

II-A-1-a - Pattern (chaîne)

C'est la propriété qui définit le motif sur lequel se fait la recherche.

Il vous faut bien évidement la définir avant d'exploiter les recherches, la validation ou les remplacements.

Pattern
```
   reg.Pattern = "(\x22[^\x22\n]*\x22)"
```

II-A-1-b - Multiline (booléen)

Active ou non la recherche sur plusieurs lignes à la fois.

La propriété est mise sur **False par défaut**.

Il vous faut bien évidement la définir avant d'exploiter les recherches, la validation ou les remplacements.

Multiline
```
   reg.Multiline = False
```

II-A-1-c - IgnoreCase (booléen)

Précise si la recherche est sensible ou non à la casse (majuscules/minuscules).

La propriété est mise sur **False par défaut**.

Il vous faut bien évidement la définir avant d'exploiter les recherches, la validation ou les remplacements.

IgnoreCase
```
reg.IgnoreCase = False
```

II-A-1-d - Global (booléen)

Précise si la recherche porte sur la première occurence ou sur toutes.

La propriété est mise sur **False par défaut**.

Il vous faut bien évidement la définir avant d'exploiter les recherches, la validation ou les remplacements.

Global
```
reg.Global = True
```

II-A-2 - Les méthodes

II-A-2-a - Test (validation)

C'est une méthode qui renvoie un **booléen**.

.Test renvoie **True** si le motif défini en *Pattern* est trouvé dans la chaîne.

.Test
```
reg.Pattern = "a"
MsgBox reg.Test("papa")
```

La boîte de dialogue renverra **True** puisque "a" est bien contenu dans la chaîne "papa".

II-A-2-b - Execute (exploration)

Cette méthode permet d'explorer les occurences qui vérifient le **Pattern**.

L'argument de la méthode est la chaîne sur laquelle nous travaillons.

Nous pouvons ensuite parcourir :

. la collection des correspondances (MatchCollection)

. les sous-correspondances (SubMatches) comme dans l'exemple ci-dessous.

.Execute
```
       Dim reg As VBScript_RegExp_55.regexp
       Dim Match As VBScript_RegExp_55.Match
       Dim Matches As VBScript_RegExp_55.MatchCollection

       ' instanciation
       Set reg = New VBScript_RegExp_55.regexp

       reg.Pattern = "(\w)(a)"
       Set Matches = reg.Execute("capacité")
       For Each Match In Matches
           Debug.Print "source >>", Match.Value
           For i = 0 To Match.SubMatches.Count - 1
               Debug.Print "[$" & i + 1 & "]", Match.SubMatches(i)
           Next i
       Next Match
```

Ce qui donnera :

Exécution
```
   ================================
         Travaux liés à .Execute
   ================================
   source >>     ca
   [$1]          c
   [$2]          a
   source >>     pa
   [$1]          p
   [$2]          a
```

II-A-2-c - Replace (remplacement)

C'est une méthode qui renvoie une **chaîne**.

Les arguments de la méthode sont :

- la chaîne sur laquelle nous travaillons

- la chaîne qui définit le remplacement avec les éventuelles variables de sous-motifs.

Les regexp autorisent la définition de sous-motifs entre parenthèses, ils sont matérialisés par des variables de type Perl $x : xième sous-motif.

.Replace
```
        reg.Pattern = "([l|d|a]\w{0,2})( )(.*)"
        Debug.Print reg.Replace("les douze travaux d'Astérix", "$3 ($1)")
```

Le code renverra la chaîne : "douze travaux d'Astérix (les)".

II-B - Présentation par la pratique

présentation complète de VBScript RegExp
```
   Function AllRegExp(ByVal strSource As String, _
                      ByVal strPattern As String, _
                      ByVal strReplace As String)

        ' déclarations
        Dim i As Integer
        Dim reg As VBScript_RegExp_55.RegExp
        Dim Match As VBScript_RegExp_55.Match
        Dim Matches As VBScript_RegExp_55.MatchCollection

        ' instanciation
        Set reg = New VBScript_RegExp_55.regexp

        ' attribution des propriétés
        ' ''' motif / pattern
        reg.Pattern = strPattern
        ' ''' application sur toutes les occurences
        reg.Global = True
        ' ''' on ignore la casse
        reg.IgnoreCase = True
        ' ''' le motif ne porte pas sur plus d'une ligne à la fois
        reg.Multiline = False

        ' Utilisation de .Execute
        '   pour parcourir la liste des motifs correspondants
        '                .Matches
        '                .SubMatches
        Debug.Print "================================="
        Debug.Print "      Travaux liés à .Execute"
        Debug.Print "================================="
        Set Matches = reg.Execute(strSource)
        For Each Match In Matches
            Debug.Print "source >>", Match.Value
            For i = 0 To Match.SubMatches.Count - 1
                Debug.Print "[$" & i + 1 & "]", Match.SubMatches(i)
            Next i
        Next Match

        ' Utilisation de .Test
        '   pour savoir si le motif a été trouvé sur la chaîne source
        Debug.Print "================================="
        Debug.Print "        Travaux liés à .Test"
        Debug.Print "================================="
        Debug.Print reg.Test(strSource)

        ' Utilisation de .Replace
        '   pour remplacer les motifs par blocs, grâce aux parenthèses capturantes
        Debug.Print "================================="
        Debug.Print "      Travaux liés à .Replace"
        Debug.Print "================================="
        Debug.Print reg.Replace(strSource, strReplace)

        ' libération d'objets
        Set Matches = Nothing
        Set Match = Nothing
```

présentation complète de VBScript RegExp
```
      Set reg = Nothing

   End Function
```

Ce qui donne en mode exécution :

Exécution
```
   AllRegexp "douze travaux d'Astérix (les)","(.*) \((.*)\)","$2 $1"
   ================================
         Travaux liés à .Execute
   ================================
   source >>     douze travaux d'Astérix (les)
   [$1]          douze travaux d'Astérix
   [$2]          les
   ================================
           Travaux liés à .Test
   ================================
   Vrai
   ================================
         Travaux liés à .Replace
   ================================
   les douze travaux d'Astérix
```

Nous utiliserons cette fonction **AllRegexp()** dans la partie suivante pour tester nos regexp.

III - L'écriture d'expressions rationnelles

Si les **RegExp** sont un outil vraiment puissant, il faut se rendre à l'évidence que l'écriture des **motifs** (*patterns*) peut s'avérer rebutante pour le néophyte.

Pas de panique cependant, il s'agit d'un langage obéissant à des règles.

Je vous propose de vous y initier au moyen de deux exemples précis :

- une réécriture de titre de films

- une validation d'adresse ip.

III-A - Apprentissage par la pratique : les titres de film

Vous vous souvenez de la question posée sur le forum, rappelée en introduction de l'article ?

Comment transformer de manière automatique ceci :

douze travaux d'Astérix (les)

en

les douze travaux d'Astérix

III-A-1 - Analysons un peu ...

Schématisons l'expression à transformer de la manière la plus générale possible?

Nous avons le titre sur plusieurs mots, puis un espace, puis un article entre parenthèses.

Soit 3 blocs que nous allons isoler dans des parenthèses (choix stratégique que vous allez comprendre après).

(titre sur plusieurs mots)(caractère espace)((article entre parenthèses))

les parenthèses ( ) servent à découper un motif par blocs

Etant donné que les parenthèses sont des caractères réservés, pour repérer ces caractères, nous devons les échapper en plaçant devant un antislash \

***les recherches portant sur les caractères réservés doivent être « échappées » avec un antislash : ***

Nous corrigeons notre expression comme suit :

(titre sur plusieurs mots)(caractère espace)(\(un ou plusieurs mots\))

III-A-2 - Capturer du texte

Pour récupérer un ou plusieurs mots / caractères nous utilisons : **.***

- le point "." est un identificateur : signifie n'importe quel caractère excepté \n (retour à la ligne)

- l'astérisque "*" est un quantifieur : indique un nombre d'occurences supérieur ou égal à 0

Reformulons notre notation compte tenu de ces principes.

(.*)( )(\(.*\))

Appliquons notre notation avec la fonction **AllRegex()** définie en partie II-B.

Exécution

```
all<b>RegExp</b> "douze travaux d'Astérix (les)","(.*)( )(\(.*\))",""
================================
      Travaux liés à .Execute
================================
source >>     douze travaux d'Astérix (les)
[$1]          douze travaux d'Astérix
[$2]
[$3]          (les)
================================
        Travaux liés à .Test
================================
Vrai
```

Le motif est valide : indiqué par Vrai sur la méthode **.Test**

Les variables sont **$1**, **$2** et **$3** sont conformes à nos attentes formulées lors de l'analyse.

Les sous-motifs capturés entre parenthèses sont mis dans des variables de type Perl $1, $2 ... $n

III-A-3 - Optimisation

Dans ce cas précis, nous sommes parvenus à détacher en une seule instruction : le titre et l'article ; malheureusement l'article reste entre parenthèses.

Du fait que nous souhaitons l'isoler, nous allons donc sortir les parenthèses de notre capture en définissant le nouveau motif comme suit :

(.*) \((.*)\)

Nous excluons les caractères inutiles de notre recherche en les sortant des parenthèses capturantes, à savoir l'espace et les parenthèses.

Exécution

```
allregexp "douze travaux d'Astérix (les)","(.*) \((.*)\)",""
================================
      Travaux liés à .Execute
================================
source >>     douze travaux d'Astérix (les)
[$1]          douze travaux d'Astérix
[$2]          les
================================
        Travaux liés à .Test
================================
Vrai
```

III-A-4 - Remplacement

Notre chaîne est correctement découpée, il ne nous reste plus qu'à la reconstituer.

.Execute nous permet d'identifier les sous-motifs avec leurs noms de variable Perl.

$1 = "douze travaux d'Astérix" (en abstration le titre)

$2 = "les" (en abstraction l'article)

Le reconstitution se fait par la méthode **.Replace**

Il suffit de préciser ce que nous souhaitons en sortie sous forme d'une chaîne incluant les variables, soit :

"$2 $1"

Ce qui donne :

Exécution
```
allregexp "douze travaux d'Astérix (les)","(.*) \((.*)\)","$2 $1"
==================================
      Travaux liés à .Execute
==================================
source >>     douze travaux d'Astérix (les)
[$1]          douze travaux d'Astérix
[$2]          les
==================================
        Travaux liés à .Test
==================================
Vrai
==================================
      Travaux liés à .Replace
==================================
les douze travaux d'Astérix
```

Reconditionnement des titres
```
Function SupprParentheses(ByVal strExp As String) As String

Dim reg As New VBScript_RegExp_55.RegExp

reg.Pattern = "(.*) \((.*)\)"
SupprParentheses = reg.Replace(strExp, "$2 $1")

Set reg = Nothing

End Function
```

Mission accomplie !

III-B - Apprentissage par la pratique : une adresse ip valide

Voici maintenant une réponse posée à une autre question du forum.

Comment puis-je m'assurer que l'adresse ip fournie est valide ?

III-B-1 - Analysons un peu ...

Comme lors du cas précédent, la réponse à notre problème exige un peu d'analyse.

Qu'est-ce qu'une adresse ip ?

Quelques exemples : **122.123.1.250**, **251.110.255.255** ...

Il s'agit de 4 entiers inférieurs ou égaux à 255 séparés par des points.

Schématisons :

(entier <= 255).(entier <= 255).(entier <= 255).(entier <= 255)

III-B-2 - Quantifions

Il est aisé de voir que la structure d'une ip est un motif répétitif.

Les RegExp sont parfaitement adaptées à ce cas puisqu'elles permettent de simplifier l'écriture, la factoriser.

Reprenons :

((entier <= 255).)3 fois (entier <= 255)

Cette quantification se fait grâce aux accolades {} et se note donc :

((entier <= 255).){3}(entier <= 255)

> ***Pour préciser combien de fois se répète un motif, il suffit de préciser le nombre d'occurences entre des accolades.***
>
> ***(ab){3} correspond avec ababab ou xxxababababxxx***
>
> ***Nous pouvons également préciser un nombre minimal et maximal***
>
> ***(ab){1,3} correspond avec ab, abab, ababab ...***

III-B-3 - la Classe des RegExp

Il nous faut maintenant formuler ce qu'est un **entier <=255**

Les RegExp travaillent sur des chaînes et non pas sur des numériques, nous ne pourrons donc pas tester une valeur.

En revanche un **entier <=255** peut couvrir les types de chaînes suivantes :

- un chiffre (digit) : entre 0 et 9

- deux chiffres (digit) : entre 0 et 9 et entre 0 et 9, **soit les valeurs 0 -> 99**

- trois chiffres (digit) :

  - si le premier chiffre est un 1 : les suivants peuvent se situer entre 0 et 9, **soient les valeurs 100 -> 199**

  - si le premier chiffre est un 2 : le second devra être compris entre 0 et 5

    - si le second est entre 0 et 4 : le troisième pourra se situer entre 0 et 9, **soient les valeurs 200 -> 249**

    - si le second est un 5 : le troisième devra se situer entre 0 et 5, **soient les valeurs 250 -> 255**

Nous couvrons à présent tout l'éventail des possibilités.

En RegExp pour exprimer qu'un caractère se situe dans une plage nous définissons une classe.

[a-z] couvre tous les caractères situés entre a et z

[a-d] couvre a ou b ou c ou d

Par conséquent, un chiffre entre 0 et 9 peut s'écrire **[0-9]**

Et un chiffre entre 0 et 5 : **[0-5]**

Les RegExp fournissent des classes prêtes à l'emploi :

\d (comme le d de Digit) est équivalent à [0-9]

III-B-4 - les alternatives

Nous avons décomposé les chaînes possibles d'un entier <= 255

le point d'interrogation est un quantifieur qui indique entre 0 et 1 occurence

1?2 correspond à 12 ou 2

On pourrait également l'écrire 1{0,1}2

entre 0 et 9 s'écrit **\d** (classe des *Digit*)

entre 0 et 99 s'écrit **\d?\d**

entre 000 et 199 s'écrit **1?\d?\d**

entre 200 et 249 s'écrit **2[0-4]\d**

entre 250 et 255 s'écrit **25[0-5]**

Il faut maintenant indiquer que nous voulons ces trois cas.

L'alternative se fait au moyen du caractère pipe |

(ab)|(cd)|(ef) correspond à ab et cd et ef

Réécrivons notre entier : **(25[0-5]|2[0-4]\d|1?\d?\d)**

III-B-5 - Dénouement

Attention notre formalisation n'est pas suffisante :

(25[0-5]|2[0-4]\d|1?\d?\d)\.) fontionne avec 225 mais aussi 2259 car 225 est contenu.

Nous utilisons ^ : début de chaîne et $: fin de chaîne pour limiter notre recherche

^(25[0-5]|2[0-4]\d|1?\d?\d)\.)$ fontionnera avec 225 mais pas avec 72259

Nous sommes parvenus à formaliser notre entier.

Dernière information le . est un idenficateur, nous devons donc « *l'échapper* » avec l'antislash.

Ce qui donne : **^((25[0-5]|2[0-4]\d|1?\d?\d)\.){3}(25[0-5]|2[0-4]\d|1?\d?\d)$**

Mettons le à l'épreuve :

Exécution
```
allregexp "255.199.9.99","^((25[0-5]|2[0-4]\d|1?\d?\d)\.){3}(25[0-5]|2[0-4]\d|1?\d?\d)$",""
================================
        Travaux liés à .Test
================================
Vrai
```

Validation d'IP
```
Function ValidIp(ByVal strIP As String) As Boolean

Dim reg As New VBScript_RegExp_55.RegExp

reg.Pattern = "^((25[0-5]|2[0-4]\d|1?\d?\d)\.){3}(25[0-5]|2[0-4]\d|1?\d?\d)$"
ValidIp = reg.Test(strIP)

Set reg = Nothing

End Function
```

Mission accomplie.

III-C - Compléments théoriques

C'est la partie un petit peu plus aride du tutoriel, mais il est utile de synthétiser les outils de notation mis à disposition par la bibliothèque RegExp.

Bon courage !

III-C-1 - Positions

Symbole	Utilisation	Exemple
^	Indique un début de chaîne	**^A** fonctionne avec "**Albert**" mais pas avec "**C'est Albert**"
$	Indique une fin de chaîne	**tard$** fontionne avec "**couche-tard**" mais pas avec "**mieux vaut tard que jamais**"
\b	Indique une fin de mot	**tard\b** fontionne avec "**motard**" mais pas avec "**attardé**"
\B	Indique une non fin de mot	**tard\B** fontionne avec "**attardé**" mais pas avec "**motard**"

III-C-2 - Symboles

Symbole	Correspondance
un caractère Alphanumérique	la caractère en question
\n	saut à la ligne
\f	saut de page
\r	retour charriot
\t	tabulation horizontale
\v	tabulation verticale
\?	le caractère "?"
*	le caractère "*"
\.	le caractère "."
|	le caractère "|"
\{	le caractère "{"
\}	le caractère "}"
\[	le caractère "["
\]	le caractère "]"
\(	le caractère "("
\)	le caractère ")"
\+	le caractère "+"
\\	le caractère "\"
\ooo	le caractère dont le code octal est ooo
\xhh	le caractère dont le code hexa est hhh
\uxxxx	le caractère ASCII dont l'UNICODE est xxxx

III-C-3 - Classes

Classe	Correspondance
[abcdef]	un des caractères en entre crochets. Correspond avec "Albert" sur le "b" Ne correspond pas avec "zut"
[a-f]	un des caractères dans la plage comprise entre a et f Correspond avec "Albert" sur le "b" Ne correspond pas avec "zut"
[^abcdef] ou [^a-f]	un des caractères de l'ensemble complémentaire à [abcdef] Ne correspond pas avec "baba" Correspond avec "zut" sur le z
.	n'importe quel caractère sauf le \n (saut de ligne)
\w	de w (word) correspond à un mot (non accentué) équivalent de la classe [a-zA-Z_0-9]
\W	ensemble complémentaire du précédent équivalent de la classe [^a-zA-Z_0-9]
\d	de d (digit) correspond à un chiffre équivalent de la classe [0-9]
\D	ensemble complémentaire du précédent équivalent de

Classe	Correspondance
	la classe [^0-9]
\s	de s (space) correspond à un caractère d'espacement équivalent de la classe [\t\r\n\v\f]
\S	ensemble complémentaire du précédent équivalent de la classe [^ \t\r\n\v\f]

III-C-4 - Quantifieurs

Quantifieur	Correspondance
{x}	correspond si ce qui précède se répète x fois. ex : \d{5} est un nombre à 5 chiffres
{x,y}	correspond si ce qui précède se répète entre x et y fois. \d{1,5} est un nombre entre 1 et 5 chiffres \d{3,} est un nombre d'au moins 3 chiffres
?	correspond si ce qui précède se répète entre 0 et 1 fois **(a)?battre** par ailleurs équivalent de **(a){0,1}battre** correspond avec "abattre" et "battre"
*	correspond si ce qui précède se répète 0 fois ou un nombre illimité de fois équivalent de {0,} **(a)*battre** par ailleurs équivalent de **(a){0,}battre** correspond avec "abattre", "battre" et "aaaaabattre"
+	correspond si ce qui précède se répète 1 fois ou un nombre illimité de fois équivalent de {1,} **(a)+battre** par ailleurs équivalent de **(a){1,}battre** correspond avec "abattre", "aaaaabattre" mais pas avec "battre"

III-C-5 - Alternatives et Groupements

Symbole	Utilisation
()	permet de regrouper une notation et de l'affecter à un sous-motif ex : **(pa)?(ci)** correspond à "**pacifique**" et "**citron**"
|	n'importe lequel des groupes séparés par | vérifie le motif ex : **^((dé)|(ef)|(coef))ficient** fonctionne avec **déficient**, **efficient**, **coefficient**

III-C-6 - Références

Symbole	Utilisation
()\n	permet de faire référence à un groupe capturé entre parenthèses numéro n ex : **(\w)\s+\1** : \1 fait référence à groupe \w soit un même mot, idéal pour repérer les répétitions. correspond à "**Celui qui est là, n'est pas ailleurs.**" est valide puisque "**est**" est répété.

IV - Cas pratique : la coloration syntaxique

Nous allons appliquer notre savoir-faire fraichement acquis dans un cas pratique : **la coloration syntaxique**.

Attention : le niveau s'élève d'un cran ici, l'assimilation des bases théoriques vues en point II et III s'avère nécessaire.

IV-A - Cahier des charges

Notre projet a pour but de pouvoir documenter notre code sur le net, pour cela nous souhaitons pouvoir exporter des modules de code au format HTML.

Il s'avère qu'un code n'est rien d'autre qu'un fichier texte.

On pourrait se contenter d'un simple export, mais il faut avouer que le rendu n'est pas très intéressant.

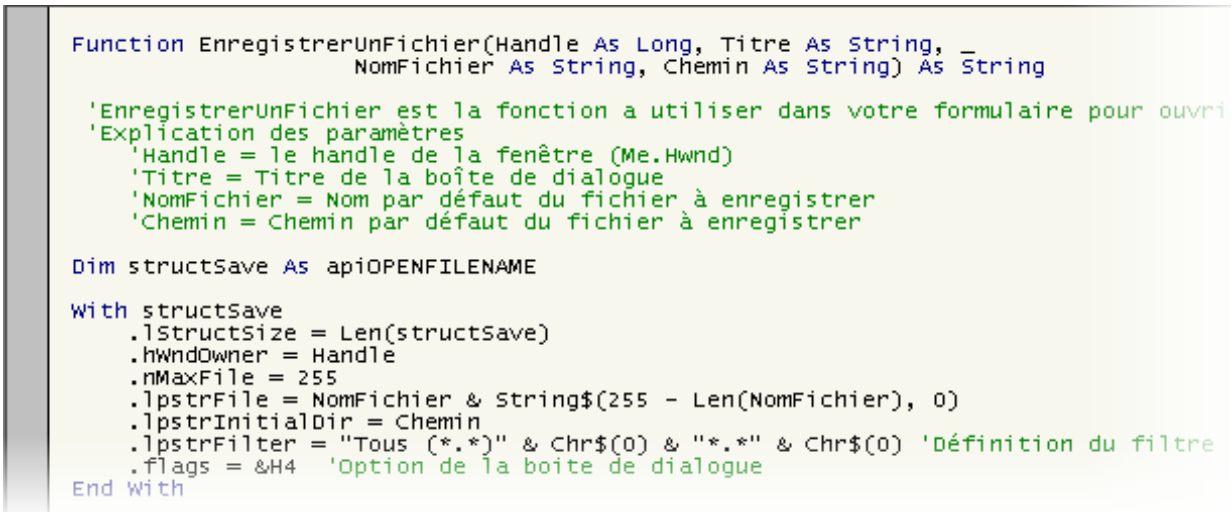
Fichier Edition Format ?

```
Function EnregistrerUnFichier(Handle As Long, Titre As String, _
                    NomFichier As String, Chemin As String) As String

 'EnregistrerUnFichier est la fonction a utiliser dans votre formulaire pou
 'Explication des paramètres
    'Handle = le handle de la fenêtre (Me.Hwnd)
    'Titre = Titre de la boîte de dialogue
    'NomFichier = Nom par défaut du fichier à enregistrer
    'Chemin = Chemin par défaut du fichier à enregistrer

Dim structSave As apiOPENFILENAME

With structSave
    .lStructSize = Len(structSave)
    .hwndOwner = Handle
    .nMaxFile = 255
    .lpstrFile = NomFichier & String$(255 - Len(NomFichier), 0)
    .lpstrInitialDir = Chemin
    .lpstrFilter = "Tous (*.*)" & Chr$(0) & "*.*" & Chr$(0) 'Définition du
    .flags = &H4  'Option de la boite de dialogue
End With
```

Export simple d'un module de code

Nous souhaitons donc avoir un rendu visuel plus proche de celui de l'éditeur VBA qui utilise la coloration syntaxique.

C'est à dire qu'un mot clé est reconnu et prend une couleur particulière.



```
Function EnregistrerUnFichier(Handle As Long, Titre As String, _
                    NomFichier As String, Chemin As String) As String

 'EnregistrerUnFichier est la fonction a utiliser dans votre formulaire pour ouvri
 'Explication des paramètres
    'Handle = le handle de la fenêtre (Me.Hwnd)
    'Titre = Titre de la boîte de dialogue
    'NomFichier = Nom par défaut du fichier à enregistrer
    'Chemin = Chemin par défaut du fichier à enregistrer

Dim structSave As apiOPENFILENAME

With structSave
    .lStructSize = Len(structSave)
    .hWndOwner = Handle
    .nMaxFile = 255
    .lpstrFile = NomFichier & String$(255 - Len(NomFichier), 0)
    .lpstrInitialDir = Chemin
    .lpstrFilter = "Tous (*.*)" & Chr$(0) & "*.*" & Chr$(0) 'Définition du filtre
    .flags = &H4  'Option de la boite de dialogue
End With
```

L'éditeur VBA et la coloration syntaxique

IV-B - Eléments à colorer

L'IDE de programmation VBA comprend par défaut 3 niveaux de coloration :

- **Le mot clé de langage VBA** : nativement en bleu (ex : Then)

- **Le commentaire** signalé par un quote (') hors chaîne : nativement en vert

- **Le reste du langage** : nativement en noir.

Nous allons améliorer ce process et ajouter deux nouvelles catégories :

- **Les chaînes** : tout texte compris entre deux double quotes (")

- **Les types de données** : exemple *Boolean*, *Byte* ou *Integer*.

IV-C - Analyse des éléments

IV-C-1 - Les mots clés

Un mot clé appartient à une liste prédéfinie.

Nous remarquons au passage que tous les mots clés VBA ne sont pas, loin de là, affectés par une coloration

syntaxique.

Par exemple, **Left()** fonction de chaîne ô combien connue n'est pas colorée.

Il nous faut donc parcourir notre chaîne pour trouver les occurences des mots en questions.

Ces mots peuvent être :

- en début de ligne

- en fin de ligne

- délimités par les caractères suivants : " .()"

[*espace*][*point*][*parenthèse ouvrante*][*parenthèse fermante*]

Nous définissons donc le tableau des mots-clés à colorer et pour chacun nous allons appliquer un remplacement RegExp.

Le pattern sera :

Pattern
```
reg.Pattern = "(\W|^)(" & KeyWords(i) & ")(\W|$)"
```

Explications :

Nous découpons le regexp en 3 parties :

- **(\W|^)** : Soit un caractère qui n'est pas de la classe Word **[a-zA-Z_0-9]**

Soit un début de chaîne

- **"(" & KeyWords(i) & ")"** : parcours du tableau de mots-clés

- **(\W|$)** : Soit un caractère qui n'est pas de la classe Word **[a-zA-Z_0-9]**

Soit une fin de chaîne

Remplacement
```
strBuff = reg.Replace(strBuff, "$1<span class=key>$2</span>$3")
```

Ce qui aboutit à mettre entre nos balises de style **span** le mot clé.

IV-C-2 - Les commentaires

Un commentaire en VB / VBA est simplement matérialisé par une **apostrophe (')** ou un **REM** en début de ligne.

Il faut bien sûr que cette apostrophe ne soit pas comprise à l'intérieur de deux double-quotes (") sinon elle est inclue à la chaîne.

L'éditeur VBA colore jusqu'à la fin de ligne.

Le commentaire peut prendre toute la ligne ou bien la fin seulement.

Pattern
```
reg.Pattern = "(\n)(([^\x22\n]*\x22[^\x22\n]*\x22)*)([^\x22\n']*)('.*)"
```

Explications :

Nous découpons cette fois en 4 parties principales (dont une imbriquée) :

- **(\n)** : nous recherchons un retour à la ligne, par conséquent un début de ligne

- **(([^\x22\n]*\x22[^\x22\n]*\x22)*)** : - pour mémoire **\xZZ** indique un caractère de code hexa ZZ

dans notre cas 22 en hexa = 34 en décimal, soit le caractère double-quote (")

- la section **[^\x22\n]*\x22*[^\x22\n]*\x22** autorise qu'une partie à gauche de l'apostrophe est une chaîne VB comprise entre double-quotes (")

- **([^\x22\n']*)** : nous utilisons une classe pour exclure certains caractères grâce à ^ : les guillemets ("), le retour à la ligne et l'apostrophe.

Nous sommes donc certains de ne pas nous situer dans une chaîne entre double-quotes et d'obtenir la première apostrophe de l'expression.

- **('.*)** : nous capturons toute chaîne précédée de l'apostrophe (')

Ce qui aboutit au remplacement suivant :

Remplacement
```
strBuff = reg.Replace(strBuff, "$1$2$4<span class=commentaire>$5</span>")
```

Il nous reste à gérer les commentaires précédés du mot-clé **REM**.

Pour la réperer, il s'agit d'une ligne entamée par **REM** suivi d'un espace puis le commentaire, le tout précédé ou non par des espaces ou tabulations.

- la classe des caractères de type "espace" est **\s** ;

- pour quantifier "aucun ou plusieurs" nous utilisons ***** ;

- pour capturer ce qui suit REM nous cherchons n'importe quel caractère (le point : .) répété 0 ou plusieurs fois (l'astérisque : *) : **.***

Nous obtenons le motif suivant : **(\s*)(REM .*)**

Remplacement
```
strBuff = reg.Replace(strBuff, "$1<span class=commentaire>$2</span>")
```

IV-C-3 - Les types

Cette distinction n'est pas native dans VB / VBA, nous allons mettre en gras les types de données.

La méthode est rigoureusement identique à celle des mots clés.

Nous utilisons cette fois un autre tableau **Types()**.

Pattern et Remplacement
```
reg.Pattern = "(\W|^)(" & Types(i) & ")(\W|$)"
strBuff = reg.Replace(strBuff, "$1<span class=type>$2</span>$3")
```

IV-C-4 - Les chaînes

Là encore, la coloration syntaxique n'est pas native dans VB / VBA, mais cela rend le code beaucoup plus lisible.

Il nous faut récupérer simplement une chaîne comprise entre deux double-quotes (") et qui ne contient pas elle-même de double-quote.

Pattern
```
reg.Pattern = "(\x22[^\x22\n]*\x22)"
```

Nous récupérons un seul bloc délimité par les double-quotes (") et qui ne contient pas lui même ce caractère ou un retour à la ligne.

Et voici le remplacement simplissime :

Remplacement
```
strBuff = reg.Replace(strBuff, "<span class=chaine>$1</span>")
```

IV-C-5 - HTML & Nettoyages divers

Avant d'ajouter du code HTML nous évitons de nous faire polluer le balisage en remplaçant **<** par **<**

tags HTML
```
' empêcher les ouvertures de tag HTML
strBuff = Replace(strBuff, "<", "&lt;")
```

Les retours chariot VB ne s'appliquent pas en HTML.

Nous devons les matérialiser en repérant le retour chariot (\n) auquel nous ajoutons

retours chariots
```
   ' les retours chariot
   reg.Pattern = "(\n)"
   reg.Global = True
   reg.IgnoreCase = True
   strBuff = reg.Replace(strBuff, "$1<br />")
```

Autre nettoyage, les tags imbriqués.

En effet notre code risque d'engendrer des colorations syntaxiques de mot-clés dans une chaîne ou dans un commentaire.

Nous allons y remédier :

- le motif recherché sera situé entre les balises **span**

  ()(.*)().

- au lieu de **.*** nous utilisons **\w{6,11}** et indiquer un mot comprenant entre 6 et 11 caractères ce qui évite de récupérer le style '**key**' et le style '**type**'.

- nous capturons un autre tag de style (**\w{3,11}**) avec cette fois entre 3 et 11 caractères pour pouvoir neutraliser tous les styles, y compris **'key'** et **'type'**

colorations imbriquées
```
       ' Highlight dans un Highlight
       reg.Pattern = "(<span class=\w{6,11}>)(.*)(<span class=\w{3,11}>)(.*)(</span>)(.*</span>)"
       reg.Multiline = False
       reg.Global = True
       reg.IgnoreCase = True
       Do While reg.Test(strBuff)
           strBuff = reg.Replace(strBuff, "$1$2$4$6")
       Loop
```

Nous passons une boucle sur le remplacement pour nettoyer toutes les occurences.

Dernière modification : **les espaces**.

Un simple remplacement de deux espaces contigus par un double caractère 160 suffit.

les espaces
```
        ' les espaces
        strBuff = Replace(strBuff, "  ", "  ")
```

IV-D - La fonction d'export avec coloration syntaxique

La fonction d'export
```
    Option Explicit

    Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
        (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
        ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

    Function xPortCode(ByVal modName As String, ByVal sizeFont As Integer, ByVal MeHwnd As Long)

        Dim i As Long
        Dim t0 As Single, t1 As Single
        Dim Fic As Integer
        Dim strBuff As String
        Dim reg As VBScript_RegExp_55.RegExp
        Dim KeyWords() As String, KeyWordsList As String
        Dim Types() As String, TypesList As String
        t0 = Timer

        Set reg = New VBScript_RegExp_55.regexp
        Fic = 1

        Reset

        ' ouverture du fichier en écriture
        Open "d:\temp\export " & modName & " (" & Format(Now, "yy-mm-dd") & ").html" For Output As #Fic

        ' écriture des en-têtes HTML et style
        Print #Fic, "<HTML>"
        Print #Fic, "<HEAD><TITLE>Export au format HTML du module : " & modName & "</TITLE>"
        Print #Fic, "<meta http-equiv=""Content-Type"" content=""text/html; charset=ISO-8859-1"" />"
        Print #Fic, "<style type='Text/css'>"
        Print #Fic, "<!--"
        Print #Fic, "BODY {"
        Print #Fic, "margin-top:0; margin-left:10; margin-right:0;"
        Print #Fic, "font-family: Lucida Console, Tahoma, Verdana, Arial, Helvetica, sans-serif;"
        Print #Fic, "font-size: " & sizeFont & "px;" ' la variable argument sizeFont passe dans la
    définition du style
        Print #Fic, "}"
        Print #Fic, ".commentaire {"
        Print #Fic, "color: #669933;"
        Print #Fic, "}"
        Print #Fic, ".chaine {"
        Print #Fic, "color: #993399;"
        Print #Fic, "}"
        Print #Fic, ".key {"
        Print #Fic, "color: #0033BB;"
        Print #Fic, "}"
        Print #Fic, ".type {"
        Print #Fic, "font-weight: bold;"
        Print #Fic, "color: #3366CC;"
        Print #Fic, "}"
        Print #Fic, "-->"
        Print #Fic, "</style>"
        Print #Fic, "</HEAD>"
        Print #Fic, "<BODY>"


        ' ouverture du module
        DoCmd.OpenModule modName

        ' récupération du texte du module
        strBuff = Application.Modules(modName).Lines(1, Application.Modules(modName).CountOfLines)

        ' empêcher les ouvertures de tag HTML
        strBuff = Replace(strBuff, "<", "&lt;")

        ' les retours chariot
```

La fonction d'export

```
        reg.Pattern = "(\n)"
        reg.Global = True
        reg.IgnoreCase = True
        strBuff = reg.Replace(strBuff, "$1<br />")


        ' 1- les mots-clé
        KeyWordsList = "AddressOf©Alias©And©As©ByRef©ByVal©Call©Case©Close©CBool©CByte©CCur©" & _
                "CDate©CDec©CDbl©CInt©CLng©CSng©CStr©CVar©Const©Compare©Database©Declare©Debug©Default©"
    & _
                "Dim©Do©Each©Else©ElseIf©End©Enum©Erase©Error©Explicit©Event©Exit©False©For©" & _
                "Friend©Function©Get©GoTo©Handles©If©Implements©Imports©In©Inherits©" & _
                "Interface©Is©Let©Lib©Like©Loop©Me©Mod©New©Next©Not©Nothing©" & _
                "On©Open©Option©Optional©Or©ParamArray©Preserve©Print©Private©Property©Protected©" & _
                "Public©RaiseEvent©ReadOnly©Redim©REM©Resume©Return©Select©Set©Shared©Static©" & _
                "Step©Stop©Sub©Then© To ©True©Type©TypeOf
    ©Until©UBound©When©Wend©While©With©WithEvents©WriteOnly©Xor"

        KeyWords = Split(KeyWordsList, "©")
        For i = 0 To UBound(KeyWords)
            reg.Pattern = "(\W|^)(" & KeyWords(i) & ")(\W|$)"
            reg.Multiline = False
            reg.Global = True
            reg.IgnoreCase = True
            strBuff = reg.Replace(strBuff, "$1<span class=key>$2</span>$3")
        Next i

        ' 2- les commentaires
        '  les REM
        reg.Pattern = "(\s)(rem .*)"
        reg.Multiline = False
        reg.Global = True
        reg.IgnoreCase = True
        strBuff = reg.Replace(strBuff, "$1<span class=commentaire>$2</span>")

        '  les apostrophes (')
        reg.Pattern = "(\n)(([^\x22\n]*\x22[^\x22\n]*\x22)*)([^\x22\n']*)('.*)"
        reg.Multiline = False
        reg.Global = True
        reg.IgnoreCase = True
        strBuff = reg.Replace(strBuff, "$1$2$4<span class=commentaire>$5</span>")

        ' 3- les types
        TypesList = "Boolean©Byte©Date©Double©Integer©Long©Object©Short©Single©String©Unicode©Variant"
        Types = Split(TypesList, "©")
        For i = 0 To UBound(Types)
            reg.Pattern = "(\W|^)(" & Types(i) & ")(\W|$)"
            reg.Multiline = False
            reg.Global = True
            reg.IgnoreCase = True
            strBuff = reg.Replace(strBuff, "$1<span class=type>$2</span>$3")
        Next i

        ' 4- les chaines
        reg.Pattern = "(\x22[^\x22\n]*\x22)"
        reg.Multiline = False
        reg.Global = True
        reg.IgnoreCase = True
        strBuff = reg.Replace(strBuff, "<span class=chaine>$1</span>")

        ' Highlight dans un Highlight
        reg.Pattern = "(<span class=\w{6,11}>)(.*)(<span class=\w{3,11}>)(.*)(</span>)(.*</span>)"
        reg.Multiline = False
        reg.Global = True
        reg.IgnoreCase = True
        Do While reg.Test(strBuff)
            strBuff = reg.Replace(strBuff, "$1$2$4$6")
        Loop

        ' les espaces
        strBuff = Replace(strBuff, "  ", "  ")

        ' écriture de la chaîne dans le fichier
        Print #Fic, strBuff

        ' fermeture du module
        DoCmd.Close acModule, modName
```

La fonction d'export

```
        Print #Fic, "</BODY>"
        Print #Fic, "</HTML>"

        ' libération des objets mémoire
        Reset
        Set reg = Nothing

        'Ouverture du fichier HTML
        ' si un Hwnd de formulaire est passé en argument ...
        If MeHwnd <> 0 Then
            ShellExecute MeHwnd, "open", "d:\temp\export " & modName & " (" & Format(Now, "yy-mm-dd") &
    ").html", "", CurrentProject.Path, 1
        End If

        t1 = Timer

        Debug.Print "Job done @ " & Format(t1 - t0, "0.000") & " s"

    End Function
```

V - Conclusion

Je tiens à préciser, si besoin était, que je suis loin d'être un expert en Expressions Rationnelles et que celles que j'utilise ici sont sans aucun doute améliorables.

Ce tutoriel ne prétend pas à l'expertise ou l'optimisation de RegExp mais simplement à l'initiation.

J'espère que ce tutoriel vous donnera envie de mieux connaître les expressions rationnelles, ou mieux vous aura permis de débloquer des situations complexes.

Merci aux membres de la rédaction pour leurs relectures attentives.

Et merci à tous de m'avoir lu ;-)