TRegExpr Documentation

Version 0.952

Andrey Sorokin

nov. 22, 2019

Table des matières

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

TRegExpr est facile utiliser et un outil très puissant pour vérifier les entrées de chaîne de caractères dans les champs (dans les DBMS et les applications web), recherche/substitution de texte, utilitaire comme egrep & etc…

Vous pouvez aisément vérifier la syntaxe d'adresse e-mail, extraire un numéro de téléphone ou un code ZIP d'un texte non formaté ou n'importe quel autre information d'une page web et tout ce que vous pouvez imaginer ! Les modèles peuvent être changées sans recompilation du programme !

Cette librairie gratuite est une version étendue des routines de Henry Spencer V8-routins pour travailler avec un sous-ensemble de Perl ; Expressions Réguličres.

TRegExpr est écris en objet pascal avec les fichiers source disponible gratuitement.

Le fichier source original en C a été amélioré et encapsulé complètement dans la classe TRegExpr en un seul fichier : RegExpr.pas.

Aussi, vous n'aurez pas besoin de fichier DLL !

Prenez un simple regard une illustration et étudier la syntaxe des expressions régulières. (Vous pouvez utiliser le *projet demo* pour explorer et apprendre vos propres expressions régulières).

Vous pouvez utiliser les (WideString Delphi) - voir « Comment utiliser les unicode ».

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

CHAPITRE 1

Introduction

Les Expressions Régulières sont grandement utilisées pour spécifier des type de recherches pour le texte. Les métacaractères spéciaux vous permettent de spécifier, par exemple, qu'une chaîne particulière que vous chercher se trouve au début ou la fin d'une ligne, ou contient n récurrence d'un certain caractère.

Les Expressions Régulières ressemblent vraiment du charabia pour les débutants, mais elles sont réellement simples (bien, habituellement), et sont un outil pratique et puissant.

Je recommande fortement de vous amuser avec le demo Windows REStudio - il vous aidera comprendre le concept principal. De plus, il y a plusieurs exemple prédéfinis avec des commentaires inclus dans TestRExp.

Commençons notre voyage d'apprentissage !

CHAPITRE 2

Simple comparaison

Un simple se compare lui-même, sauf s'il est un métacaractère avec une spécification spéciale décris plus bas.

Une série de caractère se compare la même série de caractère dans la chaîne de destination, aussi le gabarit `bluh` se compare `bluh` dans la chaîne de destination. Relativement simple, n'est-ce pas ?

Vous pouvez obliger les métacaractères ou les Séquences d'échappements être interprétés littéralement avec un "échappement" en les précédents avec une barre oblique inverse `\`, par exemple : le métacaractère `^` normalement se compare au début de ligne, mais « `^` » se compare au caractère `^`, `\` se compare `\` et ainsi de suite.

2.1 Exemples :

```
foobar      compare la chane 'foobar'
\^FooBarPtr     compare la chane '^FooBarPtr'
```

2.1.1 Séquences d'échappements

Les caractères peuvent être spécifiés avec une Séquence d'échappement comme celles utilisé dans le langage C et Perl : `\n` se compare une nouvelle ligne, `\t` une tabulation, etc… Plus généralement, `\xnn`, où nn est un nombre hexadécimal, se compare aux caractères ASCII avec une valeur dans nn. Si vous avez besoin des caractères large (wide, ou Unicode), vous pouvez utiliser `\x{nnnn}`, d'où `nnnn` - un nombre de plus ou moins 4 caractères numérique hexadécimal.

```
\xnn     caractère hexa avec le code nn.
\x{nnnn} caractère hexa avec un code nnnn (un octet pour le texte ordinaire et 2␣
↪octets pour l'Unicode).
\t       Tabulation horizontale (HT/TAB), même chose que \x09.
\n       Nouvelle ligne (NL), même chose que \x0a.
\r       Retour de chariot (CR), même chose que \x0d.
\f       Avance page (FF), même chose que \x0c.
\a       Alarme (bell) (BEL), même chose que \x07.
\e       échappement (ESC), même chose que \x1b.
```

2.2 Exemples :

```
foo\x20bar      Se compare  'foo bar' (noter l'espace dans le milieu).
\tfoobar        Se compare  'foobar' prédéfinis avec une tabulation.
```

CHAPITRE 3

Classes de Caractères

Vous pouvez spécifier une Classe de caractères, en insérant une liste de caractères dans `[]`, lequel comparera tous les caractères inclus dans la liste.

Si le premier caractère après `[` est `^`, la classe se compare comme une négation en comparant tous les caractères qui ne sont pas dans la liste.

3.1 Exemples :

```
foob[aeiou]r        Trouve les chanes 'foobar', 'foober' etc... Mais pas 'foobbr',
↪'foobcr' etc...
foob[^aeiou]r       Trouve les chanes 'foobbr', 'foobcr' etc. Mais pas 'foobar',
↪'foober' etc...
```

Dans une liste, le caractère - est utilisé pour spécifier une distance (range), aussi a-z représente tous les caractères entre `a` et `z`, inclusivement.

Si vous voulez que - soit membre de la classe, veuillez le mettre au début ou la fin de la liste, ou encore placer un échappement (\) devant. Si vous voulez un `]` vous pouvez le placer au début de la liste ou le placer avec un échappement \.

3.2 Exemples :

```
[-az]       Compare 'a', 'z' et '-.'
[az-]       Compare 'a', 'z' et '-.'
[a\-z]      Compare 'a', 'z' and '-.'
[a-z]       Trouve tous les 26 petits caractres de 'a'  'z'.
[\n-\x0D]       Trouve tous les #10,#11,#12,#13.
[\d-t]      Trouve n'importe quel caractre numérique, '-' ou 't'.
[]-a]       Trouve n'importe quel caractre de ']'  'a'.
```

CHAPITRE 4

Métacaractères

Les Métacaractères sont des caractères spéciaux qui sont l'essence même des expressions régulières. Il y a différents types de métacaractères, décris plus bas.

```
^       Au début.
$        la fin.
\A      Début du texte.
\Z      Fin du texte.
.       N'importe quel caractre dans une ligne.
```

4.1 Exemples :

```
^foobar     Trouve la chane 'foobar' seulement s'il est au début.
foobar$     Trouve la chane 'foobar' seulement s'il est  la fin.
^foobar$        Trouve la chane 'foobar' seulement s'il y a ce mot.
foob.r      Trouve les chanes comme 'foobar', 'foobbr', 'foob1r' et ainsi de suite.
```

Le métacaractère `^` par défaut garantie de trouver le mot seulement s'il est au début de la chaîne ou du texte, le métacaractère `$` seulement la fin. Les séparateurs de ligne inclus dans le texte ne sont pas considérés comme valable par `^` ou `$`, donc la condition est fausse et la recherche n'est pas valide.

Vous pouvez, toutefois, désirer traiter une chaîne comme une chaîne de plusieurs ligne de texte, de cette façon `^` sera valable après le séparateur de ligne, et `$` sera valable avant un séparateur de ligne. Vous pouvez faire ceci avec le modifier `/m`.

Les métacaractères `\A` et `\Z` sont comme `^` et `$`, excepté qu'ils fonctionne seulement qu'une seule fois pour tout le texte quand le modifier `/m` est en usage, pendant que `^` et `$` recherchera chaque séparateur de ligne.

Le métacaractère `.` par défaut se compare n'importe quel caractère, mais si vous mettez off le modifier `/s`, les séparateurs de ligne ne seront plus inclus pour `.`.

TRegExpr travaille avec les séparateurs de lignes comme recommandé au site web www.unicode.org :

^ est au début de la chaîne d'entrée, et si le modifier /m est On, aussi suivant immédiatement n'importe quelle occurrence de \x0D\x0A ou \x0A ou \x0D (si vous utiliser la Version Unicode de TRegExpr, et ensuite \x2028 ou \x2029 ou \x0B ou \x0C ou \x85). Noter qu'il n'y a pas de ligne vide dans la séquence \x0D\x0A.

$ est la fin de la chaîne d'entrée, et si le modifier /m est On, aussi précédant immédiatement n'importe quelle occurrence de \x0D\x0A ou \x0A ou \x0D (si vous utiliser la Version Unicode de TRegExpr, et ensuite \x2028 ou \x2029 ou \x0B ou \x0C ou \x85). Noter qu'il n'y a pas de ligne vide dans la séquence \x0D\x0A.

. se compare n'importe quel caractère, mais si le modifier /s est a Off « . » ne correspondra plus \x0D\x0A et \x0A et \x0D (si vous utiliser la Version Unicode de TRegExpr, et ensuite \x2028 et \x2029 et \x0B et \x0C et \x85).

Noter que ^.*$ (un gabarit de ligne vide) ne correspond pas une chaîne vide, mais se compare une chaîne vide contenant la séquence \x0A\x0D.

Le traitement Multiligne peut facilement être ajusté selon vos besoins avec l'aide des propriétés LineSeparators et LinePairedSeparator de TregExpr. Vous pouvez utiliser le style Unix \n ou seulement le style DOS/Windows \r\n ou un mélange des deux (comme décris plus haut et utilisé par défaut) ou définir vos propres séparateurs !

4.1.1 Métacactères - classes prédéfinies

```
\w      Un caractre alphanumérique (incluant "_").
\W      Un caractre non alphanumérique.
\d      Un caractre numérique.
\D      Un caractre non numérique.
\s      N'importe quel espace (mme chose que [ \t\n\r\f]).
\S      Tout ce qui n'est pas un espace.
```

Vous pouvez utiliser \w, \d et \s l'intérieur de la classe de caractères.

4.2 Exemples :

```
foob\dr     Trouve les chane comme 'foob1r', ''foob6r', ... Mais pas 'foobar', 'foobbr
↪', ...
foob[\w\s]r     Trouve les chane comme 'foobar', 'foob r', 'foobbr', ... Mais pas
↪'foob1r', 'foob=r', ...
```

TRegExpr utilise les propriétés SpaceChars et WordChars pour définir les classes de caractères \w, \W, \s, \S, aussi vous pouvez aisément les redéfinir.

4.2.1 Métacaractères - itérateurs

N'importe quel item d'une expression régulière peut-être suivi par un autre type de métacaractère - les itérateurs. En utilisant ces métacaractères vous pouvez spécifier le nombre de fois que le caractère précédent sera représenté, métacaractères ou sous expression.

```
*       Zéro ou plus ("vorace"), similaire  {0,}.
+       Un ou plus ("vorace"), similaire  {1,}.
?       Zéro or un ("vorace"), similaire  {0,1}.
{n}     Exactement n fois ("vorace").
{n,}        Au moins n fois ("vorace").
{n,m}       Au moins n fois mais pas plus de m fois ("vorace").
*?      Zéro ou plus ("non-vorace"), similaire  {0,}?.
```

(suite sur la page suivante)

(suite de la page précédente)

```
+?      Un ou plus ("non-vorace"), similaire  {1,}?.
??      Zéro ou un ("non-vorace"), similaire  {0,1}?.
{n}?        Exactement n fois ("non-vorace").
{n,}?       Au moins n fois ("non-vorace").
{n,m}?      Au moins n fois mais pas plus de m fois ("non-vorace").
```

Donc, les nombres dans les accolades de la forme `{n,m}`, spécifie le nombre de fois minimum avec la lettre n et le nombre maximum avec la lettre `m`. La forme `{n}` est équivalente `{n,n}` et correspond exactement `n` fois. La forme `{n,}` correspond n ou plus. Il n'y a aucune limite quand la grosseur de `n` et `m`, mais les grands nombres prendront beaucoup plus de mémoire et vont ralentir l'exécution de l'e.r.

Si les accolades apparaissent dans un autre contexte, ils sont traitées comme un caractère régulier.

4.3 Exemples :

```
foob.*r     Se compare  'foobar',  'foobalkjdflkj9r' et 'foobr'.
foob.+r     Se compare  'foobar', 'foobalkjdflkj9r' mais pas 'foobr'.
foob.?r     Se compare  'foobar', 'foobbr' et 'foobr', mais pas 'foobalkj9r'.
fooba{2}r       Se compare  'foobaar'.
fooba{2,}r      Se compare  'foobaar', 'foobaaar', 'foobaaaar', etc...
fooba{2,3}r     Se compare  'foobaar', ou 'foobaaar', mais pas 'foobaaaar'.
```

Une petite explication propos de l'utilisation des termes « non-vorace » et « vorace ». « Vorace » prend autant que possible, « non-vorace » prend aussi peu que possible. Par exemple, `b+` et `b\*` appliqué la chaîne `abbbbc` retourne `bbbb`, `b+?` retourne `b`, `b\*?` retourne une chaîne vide, `b{2,3}?` retourne `bb`, `b{2,3}` retourne `bbb`.

Vous pouvez changer tous les itérateurs en mode « non-vorace » en utilisant le modifier `/g`.

4.3.1 Métacaractères - Alternatifs

Vous pouvez spécifier des alternatifs pour le modèle en utilisant `|` pour les séparer, donc `fee|fie|foe` correspond `fee`, `fie`, ou `foe` dans la chaîne de destination (comme `f(e|i|o)e` le ferait). La première alternative inclus tout du délimiteur précédent (`(`, `["`, ou le début du modèle) jusqu'au premier `|`, et la dernière alternative contient tout du dernier `|` jusqu'au dernier délimiteur. Pour cette raison, il est de pratique courante d'inclure les alternatives dans des parenthèses, pour minimiser le risque de confusion pour savoir quand c'est le départ et quand c'est la fin.

Les alternatifs sont évalués de gauche droite, donc la première alternative trouvé pour la correspondance est celle qui est choisi. Ceci signifie que les alternatives ne sont pas nécessairement vorace. Par exemple : quand vous faites correspondre `foo|foot barefoot`, seulement la partie `foo` correspond, comme c'est la première alternative essayée, elle correspond exactement la chaîne de destination. (Ceci ne semble pas important, mais ceci le deviens quand vous capturer du texte correspondant en utilisant les parenthèses.)

Aussi rappeler vous que `|` est interprété comme un littéral entre `[]`, donc si vous écrivez `[fee|fie|foe]` Vous réellement rechercher pour `[feio|]`.

4.4 Exemples :

```
foo(bar|foo) Trouve la chaîne 'foobar' ou 'foofoo'.
```

4.4.1 Métacaractères - sous expressions

Les parenthèses `( ... )` peuvent aussi être utilisées pour construire des sous expression régulière. (après l'analyse, vous pouvez trouver les positions des sous expressions, longueurs et valeurs actuelles dans MatchPos, MatchLen et les propriétés de Match dans TRegExpr, et les substituer dans les chaînes du gabarit de TRegExpr.Substitute).

Les Sous expressions sont numérotés de gauche droite selon les ouvertures des parenthèses. La première sous expression le numéro `1` (l'e.r. complète a le numéro `0` - vous pouvez le substituer dans TRegExpr.Substitute comme `$0` ou `$&`).

4.5 Exemples :

```
(foobar){8,10}      Se compare aux chanes qui contiennent 8, 9 ou 10 instances de
↪'foobar'.
foob([0-9]|a+)r     Trouve la chane 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar
↪', etc...
```

4.6 Métacaractères - Références Précédentes

Les Métacaractères `\1` jusqu' `\9` sont interprétés comme des références précédentes. `\<n>` compare la sous expression `#<n>` précédente trouvé.

4.7 Exemples :

```
(.)\\1+         Trouve 'aaaa' et 'cc'.
(.+)\\1+       Aussi se compare  'abab' et '123123'.
(\['"\]?)(\\d+)\\1 Trouve  '"13" (entre guillemets), ou '4' (en apostrophe) ou 77␣
↪(sans guillemet ou apostrophe), etc...
```

CHAPITRE 5

Modifier

Les Modifier existe dans le but de changer le comportement de TRegExpr.

Il y a plusieurs façon d'ajuster ces modifier. N'importe quel de ces modifier peuvent être incorporé dans l'expression régulière elle-même en utilisant la construction de `(?...)`.

Aussi, vous pouvez changer la propriété adéquate de TRegExpr (ModifierX par exemple pour changer `/x`, ou ModifierStr pour changer tous les modificateurs ensemble). Les valeurs par défaut de la nouvelle instance de l'objet TRegExpr sont définis dans les variables globales, par exemple la variable globale RegExprModifierX définie la valeur (ModifierX) d'une nouvelle instance de TRegExpr.

5.1 i

Faire des recherche sans égard la casse des caractères (utilisant les ajustements locaux définis dans votre système), voir aussi CasInversé.

5.2 m

Traite les chaînes comme des ligne multiples. Change la fonction de `^` et `$` pour chercher uniquement partir du début ou de la fin de la chaîne, ce sera maintenant partir du début d'une ligne ou la fin de la ligne, voir aussi Séparateurs de Ligne.

5.3 s

Traite les chaînes comme une simple ligne de texte. Change la fonction de `.` pour qu'il se compare n'importe quel caractère, même un séparateur de ligne (voir aussi Séparateur de Ligne), normalement il ignorerait les sauts de ligne.

5.4 g

Modifier non standard. En le mettant Off vous spécifier de mettre tous les opérateurs en mode non-vorace (par défaut, ce modifier est On). Aussi, si le modifier `/g` est Off, alors + fonctionne comme `+?`, `*` comme `*?` et ainsi de suite...

5.5 x

étend la lisibilité du modèle en vous permettant des espaces et des commentaires (voir l'explication plus bas).

5.6 r

Modificateur non standard. Si ajusté, les distances additionnelles de − inclus les lettres russe , − inclus additionnellement , et − inclus tous les symboles russe.

Désolé pour les utilisateurs de l'extérieur, mais ces valeurs sont ajusté par défaut. Si vous voulez les mettre off par défaut - changer la valeur de la variable globale RegExprModifierR.

Le modifier `/x` requiert des explications. Il dit TRegExpr d'ignorer les espaces qui ne sont pas avec un échappement ou qui ne sont pas dans une classe. Vous pouvez utiliser ceci pour casser l'expression régulière en morceaux plus petit et plus lisible. Le caractère # est aussi traité comme un métacaractère qui introduit les commentaires, par exemple :

```
(
(abc) \# commentaire 1
  |   \# Vous ne pouvez pas utiliser les espaces pour formater l'e.r. -
TRegExpr l'ignorera.
(efg) \# commentaire 2
)
```

Ceci signifie que si vous voulez avoir des espace ou des caractères # dans le modèle (l'extérieur de la classe, où ils ne sont pas affectés par `/x`), que vous aurez mettre des échappements ou les encoder avec des valeurs octal ou hexadécimale. Pris ensemble, cette option va plus loin pour l'écriture d'expressions régulière pour les rendent plus lisible.

CHAPITRE 6

Extensions Perl

Vous pouvez l'utiliser dans les e.r. pour les modifier sur le champ. Si la construction est encapsulé dans une sous expression, alors seulement la sous expression sera affecté.

6.1 Exemples :

```
(?i)Saint-Petersburg        Trouve 'Saint-petersburg' et 'Saint-Petersburg'.
(?i)Saint-(?-i)Petersburg       Trouve 'Saint-Petersburg' mais pas 'Saint-petersburg'.
(?i)(Saint-)?Petersburg     Trouve 'Saint-petersburg' et 'saint-petersburg'.
((?i)Saint-)?Petersburg     Trouve 'saint-Petersburg', mais pas 'saint-petersburg' .
```

.

6.1.1 (?#text)

Un commentaire, le texte est ignoré. Veuillez noter que TRegExpr ferme le commentaire aussitt qu'il voit une parenthèse) , Aussi il n'y a aucune façon de placer une parenthèse dans le commentaire sans fermer celui-ci.

N'oubliez pas de lire la FAQ (spécifiquement la section "non-vorace" ou cette Question).

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

CHAPITRE 7

Méthodes et propriétés publique de TRegExpr :

```
property Expression : string
```

7.1 Fonction

Contient l'expression Régulière. Pour L'optimisation, TRegExpr va automatiquement le compiler en "P-code" (vous pouvez le voir avec l'aide de la méthode Dump) et stocké dans sa structure interne. La vrai [re]compilation survient quand c'est réellement le cas - en appellant Exec[Next], Substitute, Dump, etc et seulement si l'expression ou un autre P-code a affecté les propriétés qui ont été changées après la dernière [re]compilation.

7.2 Erreur

Si une erreur survient durant la [re]compilation, une méthode d'erreur est appelée (par défaut une erreur d'exception est levée - voir plus bas).

```
property ModifierStr : string
```

7.3 Fonction

Ajuste/récupère les valeurs par défaut des modifications d'e.r.. Le format de chaîne est similaire `(?ismx-ismx)`. Par exemple si `ModifierStr := 'i-x'` va mettre On le modifier `/i`, Off `/x` et laisser les autres inchangés.

Valeurs Possibles

```
-i-s-m-x ou ismx ou,...
```

Valeurs par Défaut

```
-i-s-m-x
```

7.4 Erreur

Si vous essayez des modifications non supportées, une erreur sera appelée (par défaut les erreurs lève une exception dans ERegExpr).

```
property ModifierI : boolean
```

7.5 Fonction

Modifier /i - (« casse des caractères ignorée »), initialisé avec la valeur RegExprModifierI.

Valeur par Défaut False

```
property ModifierR : boolean
```

Modifier /r - (« extension de syntaxe Russe »), initialisé avec la valeur RegExprModifierR.

Valeur par Défaut True property ModifierS : boolean

7.6 Fonction

Modifier /s - "." veut dire n'importe quel caractère (normalement il ne comprend pas les LinesSeparators et LinePairedSeparator), initialisé avec la valeur RegExprModifierS.

Valeur par Défaut True

```
property ModifierG : boolean;
```

7.7 Fonction

Modifier /g. En le mettant Off tous les opérateurs fonctionne en mode non-vorace, donc si ModifierG = Faux, alors "*" est comme "* ?", tous les "+" comme "+ ?" et ainsi de suite, initialisé avec la valeur RegExprModifierG.

Valeur par Défaut True

```
property ModifierM : boolean;
```

7.8 Fonction

Modifier /m Traite les chaînes comme des lignes multiples . Ceci fait, changer '^" et '$" de correspondre au début ou la fin de la chaîne, partir d'une nouvelle ligne ou la fin d'une ligne, initialisé avec la valeur RegExprModifierM.

Valeur par Défaut False

```
property ModifierX : boolean;
```

7.9 Fonction

Modifier /x - (« syntaxe étendue »), initialisé avec la valeurRegExprModifierX.

Valeur par Défaut False

```
function Exec (const AInputString : string) : boolean;
```

7.10 Fonction

Compare une recherche la chaîne AInputString.

7.11 Note

La fonction Exec stocke AInputString dans la propriété InputString.

```
function ExecNext : boolean;
```

7.12 Fonction

Trouve l'occurrence suivante de Exec(AString) ;

7.13 Note

fonctionne comme

```
Exec (AString);

if MatchLen \[0\] = 0 then ExecPos(MatchPos \[0\] + 1)
    else ExecPos (MatchPos \[0\] + MatchLen \[0\]);
```

mais est plus simple !

```
function ExecPos (AOffset: integer = 1) : boolean;
```

7.14 Fonction

Trouve une occurrence de recherche pour de départ de InputString partir de la position Aoffset (AOffset=1 - premier caractère de InputString).

```
property InputString : string;
```

7.15 Fonction

Retourne le chaîne d'entrée courante (partir du dernier appel de Exec ou de la dernière désignation de cette propriété).

7.16 Note

Une modification cette propriété efface les propriétés Match* !

```
function Substitute (const ATemplate : string) : string;
```

7.17 Fonction

Retourne ATemplate avec "$&" ou "$0" remplacé par l'occurrence complète de l'e.r. et "$n" remplacé par l'occurrence de la sous expression #n.

7.18 Valeur de Retour

Contient la chaîne avec les modification apportées.

7.19 Note

Depuis la v.0.929 "$" utiliser plutt "" (pour les futures extensions et pour plus de compatibilité avec Perl) pour accepter plus d'un caractère numérique.

Si vous voulez placer le gabarit dans le modèle "$" ou "", utiliser le préfixe "".

7.20 Exemple :

```
'1\\$ is $2\\\\rub\\\\' -> '1$ est <Match\[2\]>\\rub\\'
```

Si vous voulez placer un caractère numérique après "$n" vous devez délimiter n avec des accolades "{}".

7.21 Exemple :

```
'a$12bc' -> 'a<Match\[12\]>bc', 'a${1}2bc' -> 'a<Match\[1\]>2bc'.

procedure Split (AInputStr : string; APieces : TStrings);
```

7.22 Fonction

Divise AInputStr en pièces dans APieces par les occurrences de l'e.r.

7.23 Note

Appelé au niveau interne Exec[Next].

```
function Replace (AInputStr : string; const AReplaceStr : string) : string;
```

7.24 Fonction

Retourne AInputStr avec les occurrences de l'e.r remplacé par AReplaceStr

7.25 Note

Appelé au niveau interne Exec[Next].

```
property SubExprMatchCount : integer; // LectureSeulement
```

7.26 Fonction

Le nombre de sous expressions qui a été trouvé dans la dernière exécution de Exec*.

7.27 Valeur de Retour

S'il n'y a aucune sous expression mais que l'expression complète été trouvé (Exec* retourné vrai), alors SubExprMatchCount=0, si aucune sous expression et aucune e.r. complète a été trouvé (Exec* retourne Faux) alors SubExprMatchCount=-1.

Noter que quelques sous expressions peuvent ne pas être trouvées et pour de telles sous expressions, MathPos=MatchLen=-1 and Match="".

7.28 Par exemple :

```
L'Expression := '(1)?2(3)?';
Exec ('123'): SubExprMatchCount=2, Match\[0\]='123', \[1\]='1', \[2\]='3'
Exec ('12'): SubExprMatchCount=1, Match\[0\]='12', \[1\]='1'
Exec ('23'): SubExprMatchCount=2, Match\[0\]='23', \[1\]='', \[2\]='3'
Exec ('2'): SubExprMatchCount=0, Match\[0\]='2'
Exec ('7') - return False: SubExprMatchCount=-1

property MatchPos \[Idx : integer\] : integer; // LectureSeulement
```

7.29 Fonction

La position d'entrée de la sous expression #Idx en test la dernière exécution de Exec*.

7.30 Paramètre

La première sous expression a une valeur de Idx=1, dernière - MatchCount, l'e.r. a une valeur de Idx=0.

7.31 Valeur de Retour

Retourne -1 si dans l'e.r. il n'y a pas de sous expression trouvée dans la chaîne.

```
property MatchLen \[Idx : integer\] : integer; // Lecture Seulement
```

7.32 Fonction

La longueur d'entrée de la sous expression #Idx e.r. en test la dernière exécution de Exec*. La première sous expression a la valeur Idx=1, dernière - MatchCount, l'e.r. entière a une valeur de Idx=0.

7.33 Valeur de Retour

Retourne -1 si dans l'e.r. il n'y a pas de sous expression ou que cette expression n'as pas été trouvé dans la chaîne.

```
property Match \[Idx : integer\] : string; // Lecture Seulement
```

7.34 Fonction

== copy (InputString, MatchPos [Idx], MatchLen [Idx])

7.35 Valeur de Retour

Retourne "" si dans l'e.r. il n'y a pas de sous expression ou que la sous expression n'as pas été trouvé dans la chaîne.

```
function LastError : integer;
```

7.36 Fonction

Retourne l'ID de la dernière erreur, 0 s'il y a aucune erreur (inutilisable si l'erreur a générée une erreur d'exception) et efface la valeur interne 0 (pas d'erreur).

```
function ErrorMsg (AErrorID : integer) : string; virtual;
```

7.37 Fonction

Retourne un message d'erreur pour l'erreur avec ID = AErrorID.

```
property CompilerErrorPos : integer; // Lecture Seulement
```

7.38 Fonction

Retourne la position dans l'e.r. ou le compilateur a stoppé. Très pratique pour diagnostiquer les erreurs.

```
property SpaceChars : RegExprString
```

7.39 Fonction

Contient les caractères traités comme \s (initialement remplit avec les valeurs de la variable globale RegExprSpaceChars).

```
property WordChars : RegExprString;
```

7.40 Fonction

Contient les caractères traités comme w (initialement remplit avec les valeurs de la variable globale RegExprWordChars).

```
property LineSeparators : RegExprString
```

7.41 Fonction

Les séparateurs de ligne (comme Unix \n), initialement remplit avec les valeurs de la variable globale RegExprLineSeparators). Voir aussi a propos des séparateurs de ligne.

```
property LinePairedSeparator : RegExprString
```

7.42 Fonction

Paire de séparateur de ligne (pour le Dos et Windows \rn). Doit contenir exactement deux caractères ou pas de caractères du tout, initialement remplit avec les valeurs de la variable globale RegExprLinePairedSeparato). Voir aussi a propos des séparateurs de ligne.

7.43 Note

Par exemple, si vous avez besoin du style Unix, assigner LineSeparators := #$a (caractère de nouvelle ligne) et LinePairedSeparator := “” (chaîne vide), si par contre vous voulez accepter les séparateurs « x0Dx0A » mais pas « x0D » ou « x0A » seul, alors assigner LineSeparators := “” (chaîne vide) et LinePairedSeparator := #$d#$a.

Par défaut le mode “mixe” est utilisé (définit par défaut dans les constantes globales RegExpr-Line[Paired]Separator[s]) : LineSeparators := #$d#$a ; LinePairedSeparator := #$d#$a. Le comportement de ce mode est décris dans la section syntaxe.

```
class function InvertCaseFunction  (const Ch : REChar) : REChar;
```

7.44 Fonction

Convertit Ch en majuscule si c’est minuscule et vice-versa (en utilisant les ajustement du système local).

```
property InvertCase : TRegExprInvertCaseFunction;
```

7.45 Fonction

Ajuster cette propriété si vous voulez éviter la fonctionnalité de l’ignorance des minuscules/majuscules.

7.46 Note

Crée une interdiction la fonction RegExprInvertCaseFunction (InvertCaseFunction par défaut).

```
procedure Compile;
```

7.47 Fonction

[Re]compile l’e.r. Très pratique pour les applications qui utilise les éditeurs graphique pour vérifier la validité des propriétés.

```
function Dump : string;
```

7.48 Fonction

Crée pour le visionnement une e.r. compilée en une forme plus compréhensive.

CHAPITRE 8

Constantes Globales

Valeurs par défaut des Modifiers :

```
RegExprModifierI : boolean = False;                 // TRegExpr.ModifierI
RegExprModifierR : boolean = True;                 // TRegExpr.ModifierR
RegExprModifierS : boolean = True;                 // TRegExpr.ModifierS
RegExprModifierG : boolean = True;                 // TRegExpr.ModifierG
RegExprModifierM : boolean = False;                 //TRegExpr.ModifierM
RegExprModifierX : boolean = False;                 //TRegExpr.ModifierX

RegExprSpaceChars : RegExprString = ' '\#$9\#$A\#$D\#$C;  // Valeur par défaut pour␣
↪la propriété SpaceChars

RegExprWordChars : RegExprString = '0123456789'
  + 'abcdefghijklmnopqrstuvwxyz'
  + 'ABCDEFGHIJKLMNOPQRSTUVWXYZ\_';
  // Valeur par défaut pour la propriété WordChars

 RegExprLineSeparators : RegExprString =
   \#$d\#$a{$IFDEF UniCode}\#$b\#$c\#$2028\#$2029\#$85{$ENDIF};
  // Valeur par défaut pour la propriété LineSeparators

 RegExprLinePairedSeparator : RegExprString =   \#$d\#$a;
  // Valeur par défaut pour la propriété LinePairedSeparator

 RegExprInvertCaseFunction : TRegExprInvertCaseFunction =
TRegExpr.InvertCaseFunction;
  // Valeur par défaut pour la propriété InvertCase
```

CHAPITRE 9

Fonctions globales pratiques

```
function ExecRegExpr (const ARegExpr, AInputStr : string) : boolean;
```

9.1 Fonction

Retourne vrai si la chaîne AInputString concorde l'expression ARegExpr.

9.2 Note

!Va lever une exception s'il y a une erreur de syntaxe dans ARegExpr.

```
procedure SplitRegExpr (const ARegExpr, AInputStr : string; APieces : TStrings);
```

9.3 Fonction

Sépare AInputStr en pièces dans APieces par les occurrences de l'e.r. ARegExpr.

```
function ReplaceRegExpr (const ARegExpr, AInputStr, AReplaceStr : string) : string;
```

9.4 Fonction

Retourne AInputStr avec l'occurrence de l'e.r. remplacé par AReplaceStr.

```
function QuoteRegExprMetaChars (const AStr : string) : string;
```

9.5 Fonction

Remplace tous les métacaractères avec une représentation simple, par exemple “abc$cd.(“ est converti en “abc$cd.(“.

9.6 Note

Cette fonction est très pratique pour l’autogénération d’e.r. partir d’entrée utilisateur.

```
function RegExprSubExpressions (const ARegExpr : string; ASubExprs :
    TStrings; AExtendedSyntax : boolean = False) : integer;
```

9.7 Fonction

Fabrique une liste de sous expression trouvé dans l’e.r. ARegExpr.

9.8 Note

Dans ASubExps chaque item représente une sous expression, partir de la première jusqu’ la dernière, dans le format :

```
Chane      -       texte de sous expression (sans les '()').
bas mot de l'objet      -       position de dpart dans ARegExpr, incluant  '(' s'il␣
↪existe ! (la premire position est 1).
haut mot de l'objet     -       La longueur, incluant le dpart '(' et la fin ')' s'il␣
↪existent!
AExtendedSyntax     -       Doit tre Vrai si le modifier /x est  On durant l
↪'utilisation de l'e.r.
```

Utile pour les éditeurs avec interface graphique (Vous pouvez trouver un exemple d’utilisation dans le projet TestRExp.dpr).

Code Résultat : Sens

0 : Succčs. Pas de parenthčse non balancées trouvé.

-1 : Il n’a pas assez de parenthčse de fermeture.

-(n+1) : Ŕ la position n était trouvé “[“ sans fermeture “]”.

n : Ŕ la position n était trouvé “)” sans ouverture “(“.

Si le résultat <> 0, alors ASubExprs peut contenir des items vide ou de items illégaux.

CHAPITRE 10

Type d'exception

Routine par Défaut des erreurs d'exception pour TRegExpr :

```
ERegExpr = class (Exception)
   public
    ErrorCode : integer; // code d'erreur. Les erreurs de compilation du code sont␣
↪avant 1000.
    CompilerErrorPos : integer; // Position dans l'e.r. où l'erreur est survenue.
  end;
```

CHAPITRE 11

Comment utiliser les Unicode

TRegExpr supporte maintenant les UniCode, mais il travaille très lentement.

Qui veut se risquer l'optimiser ?

L'utiliser seulement si vous avez vraiment besoin du support des Unicode !

Pour utiliser les WideString, enlever le "." dans {.$DEFINE UniCode} dans le fichier regexpr.pas.

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

CHAPITRE 12

Q. Comment utiliser TRegExpr avec Borland C++ Builder ?

J'ai un problème depuis qu'il n'y a plus aucun fichier d'en-tête (.h or .hpp) n'est pas disponible.

12.1 R.

— Ajouter RegExpr.pas au projet bcb.
— Compiler le projet. Ceci génre le fichier RegExpr.hpp.
— Maintenant vous pouvez utiliser l'unité RegExpr.
— Ne pas oublier d'inclure #include « RegExpr.hpp » au début de votre programme.

CHAPITRE 13

Q. Pourquoi TRegExpr retourne plus d'une ligne ?

Par exemple l'e.r. `<font .\*>` retourne le premier `<font`, et ensuite le reste du fichier incluant le dernier `</html>`.

13.1 R.

Pour la compatibilité précédente le modifier /s est `On` par défaut.

Le mettre off et ensuite le `.` concordera tout excepté les séparateurs de ligne - comme voulu.

Mais je suggère aussi l'e.r. suivante `<font (\[^\\n>\]\*)>`, dans Match[1] sera l'url.

CHAPITRE 14

Q. Pourquoi TRegExpr retourne plus que prévu ?

Par exemple l'e.r. `<p>(.+)</p>` appliqué la chaîne `<p>a</p><p>b</p>` retourne `a</p><p>b` mais pas `a` comme prévu.

14.1 R.

Par défaut tous les opérateurs fonctionne en mode « vorace », aussi il correspond compare le plus possible. Si vous voulez le mode « non-vorace » vous pouvez utiliser les opérateurs `+?` et ainsi de suite (nouveau dans la v. 0.940) ou changer tous les opérateurs en mode « non-vorace » avec l'aide du modifier « g » (utiliser les propriétés convenablement dans TRegExpr ou avec une inscription dans l'e.r. comme `?(-g)`).

CHAPITRE 15

Q. Comment analyser des sources comme du HTML avec l'aide deTRegExpr

15.1 R.

Désolé les gars, mais c'est pratiquement impossible !

Bien sur que vous pouvez utiliser TRegExpr pour extraire des informations comme démontré dans mes exemples, mais si vous voulez faire une analyse précise, vous devez utiliser un vrai analyseur, pas l'e.r. !

Vous pouvez lire les explications de Tom Christiansen et Nathan Torkington dans le document `Perl Cookbook`, par exemple. Pour faire une histoire courte, il y a plusieurs expressions qui peuvent être analyser facilement avec un vrai analyseur mais pas toutes par e.r., et les vrai analyseurs sont PLUS rapide pour faire l'analyse, parce que l'e.r. ne scanne pas l'entrée avant, il fait plutt une optimisation de recherche qui peut prendre beaucoup de temps.

CHAPITRE 16

Q. Est-ce qu'il y a une façon d'avoir une correspondance multiple d'un modèle sur TRegExpr ?

16.1 R.

Vous pouvez faire une loupe et procéder comparaison par comparaison avec la méthode ExecNext.

ça ne peut être plus fait plus facilement parce que delphi n'est pas un interpréteur comme Perl (et les interpréteurs fonctionnent généralement très lentement !).

Si vous voulez quelques exemples, svp visionner la méthode TRegExpr.Replace. ou aux exemples dans *HyperLinks-Decorator.pas*.

CHAPITRE 17

Q. Je vérifie l'entrée d'utilisateur. Pourquoi TRegExpr retourne "Vrai" pour une mauvaise chaîne d'entrée ?

17.1 R.

Dans plusieurs cas de TRegExpr les utilisateurs oublient qu'une expression régulière est pour chercher dans les chaînes d'entrées. Aussi, si vous voulez faire que l'utilisateur entre seulement 4 caractères numérique et que vous utiliser l'expression `\\d{4,4}`, vous pouvez ignorer les mauvaises entrées comme `12345` ou "n'importe quel caractère 1234 et n'importe quoi". Vous devez ajouter une vérification pour le début et la fin de la ligne et vous assurer qu'il n'y a rien d'autre comme dans l'expression suivante : `^\\d{4,4}$`.

CHAPITRE 18

Q. Pourquoi que le mode non-vorace quelquefois fonctionne comme le mode vorace ?

Par exemple, l'e.r. `a+?,b+?` appliqué `aaa,bbb` retourne `aaa,b`, mais normalement ne devrait-il pas retourner `a,b` cause de la nature non-vorace du premier itérateur ?

18.1 R.

C'est une limite d'utilisation par la mathématique de TRegExpr (et plusieurs autre comme Perl et Unix). E.r. effectue seulement une "simple" optimisation de recherche, et ne tente pas d'obtenir la meilleure optimisation. Dans plusieurs cas ce n'est pas bon, mais en général cette limite est plutt avantageuse, cause des performances et des prévisions de raison.

La règle générale est que premièrement e.r. essaie de trouver une correspondance partir de sa position actuelle et seulement si c'est complètement impossible de trouver une correspondance alors il avance d'un caractère et réessaie de nouveau partir de cet emplacement. Aussi si vous utiliser `a,b+?` il correspondra avec `a,b`, mais dans le cas de `a+?,b+?` ce "n'est pas recommandé" (cause du mode non-vorace) mais possible de correspondre plus d'un `a`, aussi TRegExpr le fait mais le résultat obtenu ne sera pas une correspondance optimum. TRegExpr comme Perl ou les e.r. de Unix ne tente pas de bouger en avant et vérifier qu'est-ce qu'il serait la meilleure correspondance. De plus, il ne peut comparer en terme `plus ou moins bon`.

SVP, lire le section Syntaxe du fichier d'aide pour plus d'explication.

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

CHAPITRE 19

Simple illustrations

Si vous n'êtes pas familier avec les expressions régulières, svp, aller au sujet syntaxe.

Utiliser les routines globales

C'est simple mais pas une façon flexible et pratique.

```
ExecRegExpr ('\\d{3}-(\\d{2}-\\d{2}|\\d{4})', 'Téléphone: 555-1234');
```

retourne Vrai

```
ExecRegExpr ('^\\d{3}-(\\d{2}-\\d{2}|\\d{4})', 'Téléphone: 555-1234');
```

retourne Faux, parce qu'il y a des symboles avant le numéro de téléphone on utilise le métasymbole "^" (qui signifie début de ligne).

ReplaceRegExpr ("au produit", "Tendez un regard au produit. TRegExpr est le meilleur !", " TRegExpr");

retourne "Tendez un regard TRegExpr. TRegExpr est le meilleur !".

CHAPITRE 20

Utiliser la classe TRegExpr

Vous avez tout le pouvoir de la librairie.

{% highlight pascal linenos %} // Cette simple fonction extrait tous les e-mail de la chaîne d'entrée. // et place la liste de tous les e-mail dans la chaîne sortante. function ExtractEmails (const AInputString : string) : string; const EmailRE = "[_a-zA-Zd-.]+@[_a-zA-Zd-]+(.[_a-zA-Zd-]+)+" var r : TRegExpr; begin Result := ""; r := TRegExpr.Create; // Crée L'objet try // s'assure de la relche de mémoire en cas d'erreurs d'exceptions. r.Expression := EmailRE; // Assigne le code source l'e.r. Il sera compilé quand ce sera nécessaire // (par exemple quand Exec sera appelé). S'il y a des erreurs dans l'e.r. // Des exceptions seront levées durant la compilation de l'e.r. if r.Exec (AInputString) then REPEAT Result := Result + r.Match [0] + ", "; UNTIL not r.ExecNext; finally r.Free; end; end; begin ExctractEmails ("My e-mails is anso@mail.ru and anso@usa.net"); // retourne "anso@mail.ru, anso@usa.net, " end. // Noter : La compilation de l'e.r. durant l'attribution de l'expression // prend quelque temps , si vous voulez utiliser cette fonction plusieurs fois // ce sera du travail inutile... // Pour l'optimiser de façon significative, créer TRegExpr // et précompiler l'expression durant la phase d'initialisation du programme. {% endhighlight %} {% highlight pascal linenos %} // Ce simple exemple extrait les numéros de téléphone et // l'analyse en partie (code régional, ville, numéro interne). // Ensuite il substitut les parties en gabarit. function ParsePhone (const AInputString, ATemplate : string) : string; const IntPhoneRE = "(+d *) ?((d+) *) ?d+(-d*)*"; var r : TRegExpr; begin r := TRegExpr.Create; // Crée l'objet try // s'assure de la relche de mémoire en cas d'erreursd'exceptions. r.Expression := IntPhoneRE; // Assigne le code source l'e.r. Il sera compilé quand nécessaire // (par exemple quand Exec sera appelé). S'il y a des erreurs dans l'e.r. // Des exceptions seront levées durant la compilation de l'e.r. if r.Exec (AInputString) then Result := r.Substitute (ATemplate) else Result := ""; finally r.Free; end; end; begin ParsePhone ("Phone of AlkorSoft (project PayCash) is +7(812) 329-44-69", "Zone code $1, city code $2. Whole phone number is $&."); // retourne "Code Régional +7, Code de Ville (812) . Téléphone complet +7(812) 329-44-69." end. {% endhighlight %}

CHAPITRE 21

Illustrations plus complexes

Vous pouvez trouver des illustrations plus complexes pour utiliser TRegExpr dans le projet TestRExp.dpr et *HyperLinkDecorator.pas*.

Voir aussi mon petit article *Delphi3000.com* (en Anglais) et *Delphi Kingdom* (en Russe).

Explication plus détaillée

Svp, voir la description d'interface de TregExpr.

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.
>
> *If you want to help to update the translation please* contact me. New translation is based on GetText and can be edited with weblate.org. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

Simple programme pour explorer et tester les e.r., distribué en fichiers sources (projet TestRExp.dpr) et avec l'exécutable TestRExp.exe.

Noter qu'il utilise les propriétés de plusieurs VCL qui existent seulement dans Delphi 4 ou plus récent. En compilant dans Delphi 3 ou Delphi 2 vous recevrez quelques message d'erreurs a propos de propriétés inconnues. Vous pouvez les ignorer, ces propriétés sont seulement nécessaire lorsque vous redimmensionner la fenêtre du programme.

Avec l'aide de ce programme, vous pourrez aisément déterminer le nombre de sous expression que vous êtes en train de modifier, aller n'importe quelle sous expression définie (dans le code compilé de l'e.r. autant que dans le résultat des chaînes d'entrée), jouer avec les substituts, remplacer, séparer et même plus.

Et de plus, le projet inclus une bonne quantité d'exemple - utiliser les pour apprendre la syntaxe des e.r. ou pour apprendre rapidement les avantages des fonctionnalités de TRegExpr.

CHAPITRE 22

Exemple : Décorateur HyperLinks

Fonctions pour décorer les liens hyperlinks en convertissant le texte standard en format HTML.

Par exemple, remplace `http://anso.da.ru` avec `<a href="http://anso.da.ru">anso.da.ru</a>` ou `anso@mail.ru` avec `<a href="mailto:anso@mail.ru">anso@mail.ru</a>`.

CHAPITRE 23

fonction Décorer les URLs

Trouve et remplace les liens comme `http://...` ou `ftp://..` aussi bien que les liens sans protocol, mais qui commence avec `www.` Si vous voulez décorer les e-mails, vous pouvez utiliser la fonction DecorateEMails.

```
function DecorateURLs (const AText : string; AFlags :
    TDecorateURLsFlagSet = \[durlAddr, durlPath\]) : string;
```

23.1 Description

retourne le texte d'entrée AText avec les liens hyperliens décorés.

AFlags décris quelle partie de l'hyperlien lien doit être inclus dans la partie VISIBLE du lien :

Par exemple, si [durlAddr] alors hyperlien `http://anso.da.ru/index.htm` sera décoré comme `<a href="http://anso.da.ru/index.htm">anso.da.ru</a>`.

```
type
 TDecorateURLsFlags = (durlProto, durlAddr, durlPort, durlPath, durlBMark, durlParam);
 TDecorateURLsFlagSet = jeux de TDecorateURLsFlags;
```

23.2 Description

Voici les valeurs possibles pour TDecorateURLsFlagSet :

durlProto : Protocole (comme `ftp://` ou `http://`).

durlAddr : Adresse TCP ou nom de domaine (comme `anso.da.ru`).

durlPort : Numéro de port si spécifié (comme `:8080`).

durlPath : Chemin au document (comme `index.htm`).

durlBMark : Bookmark (comme `#mark`).

durlParam : Paramètres URL (comme `?ID=2&User=13`).

CHAPITRE 24

Fonction Décorer les EMails

Remplace les e-mails avec `<a href="mailto:ADDR">ADDR</a>`. Par exemple, remplace `anso@mail.ru` avec `<a href="mailto:anso@mail.ru">anso@mail.ru</a>`.

```
function DecorateEMails (const AText : string) : string;
```

24.1 Description

Retourne le texte d'entrée avec les décoration e-mails dans Atext.