TRegExpr Documentation

Versión 0.952

Andrey Sorokin

22 de noviembre de 2019

Índice general

> **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.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 es una herramienta poderosa y fácil de usar para controlar entrada de datos de cadenas de caracteres en base a plantillas (en DBMS y aplicaciones para web), búsqueda y sustitución de texto, utilidades tipo egrep, etc.

Se puede verificar fácilmente una dirección de e-mail, extraer un número de teléfono o código postal de texto sin formato, cualquier tipo de información de una página web, y todo lo que pueda imaginar!. Las reglas (plantillas) pueden ser modificadas sin recompilar el programa!

Esta librería freeware es una versión extendida para Delphi de las rutinas V8 de Henry Spencer,. Trabajan con un subconjunto de las Expresiones Regulares de Perl.

TRegExpr está programado en Pascal puro, con el código fuente completo gratis.

El original en C ha sido mejorado y encapsulado completamente en la clase TRegExpr implementada en un sólo archivo: RegExpr.pas.

Por lo tanto, no se necesita ninguna DLL!

Ver los ejemplos simples y estudiar la sintaxis de las expresiones regulares (Se puede usar el proyecto demo para probar y depurar las expresiones regulares propias).

Se puede usar Unicode (WideString de Delphi) - ver «Cómo usar 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 **This is very old and outdated translation.** *If you can read English or Russian please use up-to-date* English version *or* Russian version.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.

CAPÍTULO 1

Introducción

Las Expresiones Regulares son un método ampliamente empleado para especificar «plantillas» de texto a buscar. Los metacaracteres especiales permiten especificar, por ejemplo, que una cadena en particular que se está buscado aparezca al inicio o al fin de una línea, o que contenga n repeticiones de cierto caracter.

Las expresiones regulares lucen incomprensibles para los novatos, pero en realidad son muy simples (bueno, usualmente simples ;)), y son una herramienta práctica y poderosa.

Les recomiendo especialmente hacer pruebas con e.r. usando Windows REStudio - ayuda a comprender muchos conceptos. Además hay muchos ejemplos predefinidos con comentarios, incluidos en este proyecto.

Iniciemos el recorrido de aprendizaje!

CAPÍTULO 2

Búsquedas simples

Cualquier caracter se encuentra a sí mismo, a menos que se trate de un metacaracter con significado especial, descriptos abajo.

Una serie de caracteres encuentra esa misma serie en la cadena objetivo, por lo tanto la plantilla «bluh» encontrará «bluh"" en la cadena objetico. Simple, no?

Se puede conseguir que los caracteres que normalmente funcionan como metacaracteres o secuencias de escape sean interpretadas literalmente precediéndolas con el símbolo \ (backslash). Por ejemplo, el metacaracter ^ significa inicio de la cadena, pero \^ encuentra el símbolo ^, \\ encuentra \, y así para todos los casos especiales.

2.1 Ejemplos:

```
foobar          encuentra la cadena 'foobar'
\^FooBarPtr      encuentra '^FooBarPtr'
```

CAPÍTULO 3

Secuencias de escape

Algunos casos especiales pueden ser especificados usando sintaxis de secuencias de escape como las empleadas en C y Perl: `\n` significa nueva línea, `\t` equivale a tab, etc. Más generalmente, `\xnn`, donde nn es un número hexadecimal, encuentra el caracter cuyo valor ASCII es nn. Para usar códigos dobles de Unicode, se puede especificar `\x{nnnn}`, donde `nnnn` - es uno o más valores hexadecimales.

```
\xnn     caracter con código hexadecimal nn
\x{nn} caracter con código hexadecimal nnnn (un byte para texto común y dos para␣
↪[Unicode](tregexpr.html))
\t       tab (HT/TAB), lo mismo que \x09
\n       línea nueva (NL), lo mismo que \x0a
\r       retorno de carro (CR), lo mismo que \x0d
\f       salto de hoja (FF), lo mismo que \x0c
\a       alarma (bell) (BEL), lo mismo que \x07
\e       escape (ESC), lo mismo que \x1b
```

3.1 Ejemplos:

```
foo\x20bar   encuentra 'foo bar' (notar el espacio en el medio)
\tfoobar     encuentra 'foobarar' precedido por tab
```

CAPÍTULO 4

Clases de caracteres

Se pueden especificar clases de caracteres encerrando una lista de caracteres entre corchetes `[]`, la que que encontrará uno cualquiera de los caracteres de la lista.

Si el primer símbolo después deI `[` es `^`, la clase encuentra cualquier caracter que no está en la lista.

4.1 Ejemplos:

```
foob\[aeiou\]r   encuentra las cadenas 'foobar', 'foober' etc. pero no 'foobbr',
↪'foobcr' etc.
foob\[^aeiou\]r encuentra las cadenas 'foobbr', 'foobcr' etc. pero no 'foobar',
↪'foober' etc.
```

Dentro de una lista, el síbolo `-` es utilizado para especificar un rango, entonces `a-z` representa todos los caracteres entre `a` y `z` inclusive.

Para que `-` forme parte de la clase hay que ubicarlo al inicio o final de la lista, o usar la secuencia de escape `\-`. Para usar `]` en la lista hay que ubicarlo al inicio de la lista o usar la secuencia `\]`.

4.2 Ejemplos:

```
[-az]     encuentra 'a', 'z' y '-'
[az-]     encuentra 'a', 'z' y '-'
[a\-z]     encuentra 'a', 'z' y '-'
[a-z]     encuentra todas las minúsculas de 'a' hasta 'z'
[\n-\x0D] encuentra cualquiera de #10,#11,#12,#13.
[\d-t]     encuentra cualquier dígito, '-' or 't'.
[]-a]     encuentra cualquier caracter de ']' hasta 'a'.
```

CAPÍTULO 5

Metacaracteres

Los metacaracteres son caracteres especiales que son la esencia de las Expresiones Regulares. Hay diferentes tipos:

```
^     inicio de línea
$     fin de línea
\A     inicio de texto
\Z     fin de texto
.     cualquier caracter en la línea
```

5.1 Ejemplos:

```
^foobar     encuentra 'foobar' sólo si está al principio de una línea
foobar$     encuentra 'foobar' sólo si está al final de una línea
^foobar$   encuentra 'foobar' sólo si es la única cadena en la línea
foob.r     encuentra cadenas tipo 'foobar', 'foobbr', 'foob1r'
```

El metacaracter `^` por defecto sólo garantiza encontrar coincidencias al principio de la cadena/texto analizados, y `$` sólo al final. Los separadores de línea intermedios no son encontrados por `^` o `$`.

Sin embargo, se puede tratar una cadena como multilínea, de esta forma `^` encontrará coincidencias después de cualquier separador de línea dentro de esta cadena, y `$` dará resultados positivos antes de cualquier separador. Esto se logra activando el modificador `/m`.

Las secuencias `\A` y `\Z` son como `^` y `$`, excepto que no dan resultados múltiples aunque esté activado el modificador `/m`, mientras que `^` y `$` encontrarán coincidencias en todos los separadores de línea internos.

El metacaracter `.` por defecto encuentra cualquier caracter, pero si se desactiva el modificador `/s` entonces `.` no encuentra separadores de línea internos.

TRegExpr trabaja con los separadores de línea según las recomendaciones de []www.unicode.org](http://www.unicode.org/unicode/reports/tr18/):

`^` al inicio de la cadena ingresada, y si el modificador `/m` está activado, también inmediatamente después de toda ocurrencia de `\x0D\x0A`, `\x0A` o `\x0D` (si se usa la versión Unicode de TRegExpr, también `\x2028`, `\x2029`,

\x0B, \x0C o \x85). Notar que no hay una línea vacía dentro de la secuencia \x0D\x0A.

$ al final de la cadena ingresada, y si el modificador /m está activado, también inmediatamente antes de toda ocurrencia de \x0D\x0A, \x0A, o \x0D (si se usa la versión Unicode de TRegExpr, también \x2028, \x2029, \x0B, \x0C o \x85). Notar que no hay una línea vacía dentro de la secuencia \x0D\x0A.

. encuentra cualquier caracter, pero si se desactiva el modificador /s entonces . no encuentra \x0D\x0A, \x0A y \x0D (si se usa la versión Unicode de TRegExpr, también \x2028, \x2029, \x0B, \x0C y \x85).

Notar que ^.*$ (plantilla de línea vacía) no encuentra la cadena vacía dentro de la secuencia \x0D\x0A, pero sí dentro de la secuencia \x0A\x0D.

El procesamiento multilínea puede ser fácilmenet afinado para sus propósitos con la ayuda de las propiedades LineSeparators y LinePairedSeparator de TRegExpr. Se pueden usar sólo separadores tipo Unix (\n) o sólo tipo DOS/Windows (\r\n) o todos juntos (como se describe arriba y usado por defecto), o incluso definir sus propios deparadores de línea !

5.1.1 Metacaracteres - clases predefinidas

```
\w     un caracter alfanumérico (incluye "_")
\W     un caracter no alfanumérico
\d     un caracter numérico
\D     un caracter no numérico
\s     cualquier espacio (lo mismo que [ \t\n\r\f])
\S     un no espacio
```

Se pueden usar las clases \w, \d y \s dentro de las clases de caracteres personales.

5.2 Ejemplos:

```
foob\dr     encuentra cadenas como 'foob1r', ''foob6r' pero no 'foobar', 'foobbr'
foob\[\w\s]r encuentra cadenas como 'foobar', 'foob r', 'foobbr' pero no 'foob1r',
↪'foob=r'
```

TRegExpr usa las propiedades SpaceChars y WordChars para definir las clases de caracteres \w, \W, \s, \S, por lo tanto se pueden redefinir fácilmente.

NOTA PARA USUARIOS DE IDIOMA ESPAOL:

La propiedad WordChars por defecto está definida con el siguiente conjunto de caracteres:

```
WordChars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ\_'
```

Para nuestro idioma lo correcto sería:

```
WordChars = '0123456789aábcdeéfghiíjklmnoópqrstuúvwxyzAáBCDEéFGHIJKLMNOPQRSTUúVWXYZ\_'
```

5.2.1 Metacaracteres - límites de palabras

```
\b                              encuentra límite de palabra
\B                              encuentra distinto a límite de palabra
```

Un límite de palabra (\b) es un punto entre dos caracteres que está limitado por un \w de un lado y un \W en el otro (en cualquier orden), contando los caracteres imaginarios del inicio y final de la cadena como coincidencias con \W.

5.2.2 Metacaracteres - iteradores

Cualquier item de una expresión regular puede ser seguido por otro tipo de metacaracteres, los iteradores. Usando estos metacaracteres se puede especificar el número de ocurrencias del caracter previo, de un metacaracter o de una subexpresion.

```
*     cero o más ("voraz"), similar a {0,}
+   una o más ("voraz"), similar a {1,}
?   cero o una ("voraz"), similar a {0,1}
{n}   exactamente n veces ("voraz")
{n,}   por lo menos n veces ("voraz")
{n,m} por lo menos n pero no más de m veces ("voraz")
*?     cero o más ("no voraz"), similar a {0,}?
+?     una o más ("no voraz"), similar a {1,}?
??     cero o una ("no voraz"), similar a {0,1}?
{n}?   exactamente n veces ("no voraz")
{n,}? por lo menos n veces ("no voraz")
{n,m}? por lo menos n pero no más de m veces ("no voraz")
```

Entonces, los dígitos entre llaves de la forma `{n,m}`, especifican el mínimo número de ocurrencias en `n` y el máximo en `m`. La forma `{n}` es equivalente a `{n,n}` y coincide exactamente `n` veces. La forma `{n,}` encuentra ocurrencias de `n` o más veces. No hay límites para los número `n` o `m`, pero si son muy grandes se consume más memoria y la ejecución de la e.r. se hace más lenta.

Si una llave aparece en otro contexto se la trata como un caracter regular.

5.3 Ejemplos:

```
foob.*r     encuentra cadenas como 'foobar', 'foobalkjdflkj9r' y 'foobr'
foob.+r     encuentra cadenas como 'foobar', 'foobalkjdflkj9r' pero no 'foobr'
foob.?r     encuentra cadenas como 'foobar', 'foobbr' y 'foobr' pero no 'foobalkj9r'
fooba{2}r   encuentra la cadena 'foobaar'
fooba{2,}r encuentra cadenas como 'foobaar', 'foobaaar', 'foobaaaar' etc.
fooba{2,3}r encuentra cadenas como 'foobaar', o 'foobaaar' pero no 'foobaaaar'
```

Una mínima explicacion acerca de la `voracidad`. `Voraz` toma la mayor cantidad posible, `no voraz` toma la mínima cantidad posible. Por ejemplo, `b+` y `b*` aplicados a la cadena `abbbbc` devuelven `bbbb`, `b+?` devuelve `b`, `b*?` devuelve unacadena vacía, `b{2,3}?` devuelve `bb`, `b{2,3}` devuelve `bbb`.

Se pueden activar todos los iteradores para que funcione en modo «no voraz» (ver el modificador `/g`).

5.3.1 Metacaracteres - alternativas

Se puede especificar una serie de alternativas para una plantilla usando `|` para separarlas, entonces `fee|fie|foe` encontrará cualquier `fee`, `fie`, o `foe` en la cadena objetivo (lo mismo sería `f(e|i|o)e`). La primera alternativa incluye todo desde el ultimo delimitador («(“”, `[`, o el inicio de la plantilla) hasta el primer `|`, y la última alternativa contiene todo desde el último `|` hasta el siguiente delimitador de plantilla. Por esta razón es una práctica común incluir las alternativas entre paréntesis, para minimizar la confusión de dónde se inician y dónde terminan.

Las alternativas son evaluadas de izquierda a derecha, por lo tanto la primera alternativa que coincide plenamente con la expresión analizada es la que se selecciona. Esto significa que las alternativas no son necesariamente voraces. Por ejemplo: si se buscam foolfoot en `barefoot`, sólo la parte `foo` da resultado positivo, porque es la primera alternativa probada, y porque tiene éxito en la búsqueda de la cadena analizada. (Esto puede no parecer importante, pero lo es cuando se está capturando el texto buscado usando paréntesis.)

También recordar que | es interpretado literalmente dentro de los corchetes, entonces si se escribe [fee|fie|foe] lo único que se encuentra es [feio|].

5.4 Ejemplo:

```
foo(bar|foo) encuentra las cadenas 'foobar' o 'foofoo'.
```

5.4.1 Metacaracteres - subexpresiones

La construcción (...) también puede ser empleada para definir subexpresiones de e.r. (después del análisis se obtienen las posiciones de las subexpresiones, su longitud y el valor actual en las propiedades MatchPos, MatchLen y Match de TRegExpr; y se pueden substituir en cadenas de plantillas con TRegExpr.Substitute).

Las subexpresiones son numeradas de izquierda a derecha en base al orden de sus paréntesis de apertura. La primera subexpresión es la "1" (la e.r. completa tiene el número "0" - Se puede substituir en TRegExpr.Substitute como "$0" o "$&").

5.5 Ejemplos:

```
(foobar){10} encuentra cadenas que contienen 8, 9 o 10 instancias de 'foobar'
foob([0-9]|a+)r encuentra 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.
```

5.5.1 Metacaracteres - memorias (backreferences)

Los metacaracteres \1 a \9 son interpretados como memorias. \<n> encuentra la subexpresión previamente encontrada #<n>.

5.6 Ejemplos:

```
(.)\1+          encuentra 'aaaa' y 'cc'.
(.+)\1+        también encuentra 'abab' y '123123'
(\['"]?)(\d+)\1 encuentra '"13" (entre comillas dobles), o '4' (entre comillas␣
↪simples) o 77 (sin comillas) etc
```

CAPÍTULO 6

Modificadores

Los modificadores son para cambiar el comportamiento de TRegExpr.

Hay varias formas de configurar los modificadores.

Cualquiera de estos modificadores pueden incluirse dentro de las expresiones regulares usando la estructura `(?...)`.

También se pueden asignar las propiedades adecuadas de TRegExpr (ModifierX por ejemplo, para cambiar `/x`, o ModifierStr para cambiar todos los modificadores simultáneamente). Los valores por defecto para nuevas instancias de TRegExpr están definidos en variables globales, por ejemplo la variable global RegExprModifierX define el valor de la propiedad ModifierX en las nuevas instacias del objeto TRegExpr.

Búsquedas insensibles a mayúsculas, ver también InvertCase.

Tratamiento de cadenas como múltiples líneas. Esto es, cambia a `^` y `$` de encontrar solo en el inicio y fin de la cadena al inicio y fin de cada línea dentro de la cadena, ver también Separadores de líneas.

Tratamiento de cadenas cómo línea simple. Esto es, cambia `.` para encontrar cualquier caracter en cualquier lado, incluso separadores de línea (ver Separadores de línea), que normalmente no son encontradod.

Modificador no standard. Al desactivarlo se cambian todos los operadores siguientes en modo no voraz (por defecto este modificador esta activado). Entonces, si el modificador `/g` está Off entonces `+` funciona como `+?`, `*` como `*?`, etc.

Aumenta la legibilidad de las plantillas al permitir espacios en blanco y comentarios (ver la explicación más abajo).

Modificador no standard para incluir letras rusas en el rango de caracteres.

Perdón a los usuarios extranjeros, pero está activado por defecto. Para desactivarlo por defecto cambiar a False la variable global RegExprModifierR.

El modificador `/x` necesita un poco más de explicación. Le dice a TRegExpr que ignore los espacios blancos que no están precedidos por `\` o no incluídos en una clase de caracteres. Se puede usar para separar las expresiones regulares en partes más legibles. El caracter `#` es tratado como metacaracter para comentarios, por ejemplo:

```
(
(abc) #comentario 1
  |   #Se pueden usar espacios para formatear e.r.
```

(continué en la próxima página)

(proviene de la página anterior)

```
(efg) #comentario 2
)
```

Esto significa que si se desea usar realmente un espacio o # en una plantilla (fuera de clases de caracteres, donde no son afectadas por /x), hay que usar \ o codificarlos en su valor ASCII en octal o hexadecimal.

CAPÍTULO 7

Extensiones de Perl

Se pueden usar dentro de las e.r. para cambiar modificaciones instantáneamente. Si esta construcción está incluída dentro de una subexpresión, entonces sólo afecta a la subexpresión.

7.1 Ejemplos:

```
(?i)Saint-Petersburg       encuentra 'Saint-petersburg' y 'Saint-Petersburg'
(?i)Saint-(?-i)Petersburg encuentra 'Saint-Petersburg' pero no 'Saint-petersburg'
(?i)(Saint-)?Petersburg   encuentra 'Saint-petersburg' y 'saint-petersburg'
((?i)Saint-)?Petersburg   encuentra 'saint-Petersburg', pero no 'saint-petersburg'
```

7.1.1 (?#text)

Comentario, el texto es ignorado. Notar que TRegExpr cierra el comentario apenas encuentra un «)», por lo tanto no hay forma de poner un «)» literal en el comentario.

> **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.

CAPÍTULO 8

Métodos públicos y propiedades de TRegExpr:

```
función VersionMajor  : integer;
función VersionMinor : integer;
```

Devuelve versiones mayor y menor, por ejemplo, para v. 0.944 VersionMajor = 0 y VersionMinor = 944

```
propiedad Expression : string
```

Expresión regular.

Para optimización, TRegExpr la compilará automáticamente en "P-code" (se puede ver con el método Dump) y la almacena en estructuras internas. La [re]compilación real ocurre sólo cuando es necesario - al llamar a Exec[Next], Substitute, Dump, etc y sólo si Expression u otra propiedad P-code fue modificada después de la última [re]compilación.

Si se produce cualquier error durante la [re]compilación de llama al método Error (por defecto Error genera una excepción - ver abajo)

```
propiedad ModifierStr : string
```

valores por defecto para los modificadores de e.r. El formato de la cadena es similar a (?ismx-ismx). Por ejemplo ModifierStr := "i-x" activa el modificador /i, desactiva /x y deja sin cambios el resto.

Si se intenta cambiar un modificador inexistente, se llamará al procedimiento Error (por defecto Error genera una excepción ERegExpr).

```
propiedad ModifierI : boolean
```

Modificador /i - («insensible a mayúsculas»), inicializado por el valor de RegExprModifierI.

```
propiedad ModifierR : boolean
```

Modificador /r - («Extensiones para Ruso»), inicializado con el valor de RegExprModifierR.

propiedad ModifierS : boolean

Modificador /s - "." funciona como cualquier char (no encuentra Separadores de línea y LinePairedSeparator), inicializado con el valor de RegExprModifierS.

propiedad ModifierG : boolean;

Modificador /g Desactiva el modificador /g cambiando todos los operadores a estilo no voraz, entonces si ModifierG = False todos los "*" trabajan como "*?", todos los "+" como "+?", etc, inicializado con el valor de RegExprModifierG.

propiedad ModifierM : boolean;

Modificador /m Tratamiento de cadenas como líneas múltiples. Esto es, cambia '^" y '$" de encontrar sólo al inicio o final de la cadena al inicio o final de cualquier salto de línea DENTRO de la cadena, inicializado con el valor de RegExprModifierM.

propiedad ModifierX : boolean;

Modificador /x - («sintaxis eXtendida»), inicializado con RegExprModifierX.

función Exec (const AInputString : string) : boolean;

ejecuta el programa sobre la cadena AInputString. Exec guarda AInputString en la propiedad InputString.

función ExecNext : boolean;

busca la siguiente coincidencia:

```
  Exec (AString); ExecNext;
```

funciona igual que

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

pero es más simple !

función ExecPos (AOffset: integer = 1) : boolean;

busca coincidencias en InputString comenzando el la posición AOffset

(AOffset=1 - primer caracter de InputString)

propiedad InputString : string;

devuelve la cadena corriente (desde la última llamada a Exec o la última asignación a esta propiedad).

Cualquier asignación de esta propiedad limpia las propiedades Match* !

función Substitute (const ATemplate : string) : string;

Devuelve ATemplate con "$&" o "$0" reemplazados por la ocurrencia completa de la e.r. y "$n" reemplazado por la ocurrencia de la subexpresión #n.

Desde la v.0.929 "$" se usa en vez de "" (para ampliaciones futuras y por mayor compatibilidad con Perl) y acepta más de un dígito.

Si es necesario incluir en la plantilla los símbolos "$" o "", usar el prefijo ""

Ejemplo: "1$ es $2\rub' -> "1$ es <Match[2]>rub"

Si hay que incluir un dígito después de "$n" se debe delimitar n con llaves "{}".

Ejemplo: "a$12bc" -> "a<Match[12]>bc", "a${1}2bc" -> "a<Match[1]>2bc".

procedimiento Split (AInputStr : string; APieces : TStrings);

Parte AInputStr en APieces por las ocurrencias de la e.r.

Internamente llama a Exec[Next]

function Replace (AInputStr : RegExprString;

const AReplaceStr : RegExprString;

AUseSubstitution : boolean = False) : RegExprString;

Devuelve AInputStr con las ocurrencias de la e.r. reemplazadas por AReplaceStr

Si AUseSubstitution es verdadero se usa AReplaceStr como plantilla para métodos de sustitución.

Por ejemplo:

Expression := "({-i}block|var)s*(s*([^]*)s*)s*";

Replace ("BLOCK(test1)", "def «$1» value «$2»", True);

devolverá: def "BLOCK" value "test1"

Replace ("BLOCK(test1)", "def «$1» value «$2»", False)

devolverá: def «$1» value «$2»

Internamente llama a Exec[Next]

propiedad SubExprMatchCount : integer; // ReadOnly

Número de subexpresiones que han sido encontradas en la última llamada a Exec*.

Si no hay subexpresiones pero se encontró la expresión entera (Exec* devolvió True), entonces SubExprMatchCount=0, si no hay subexpresiones ni expresión completa de la e.r. encontradas (Exec* devolvió False) entonces SubExprMatchCount=-1.

Por ejemplo: 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") - devuelve False: SubExprMatchCount=-1

propiedad MatchPos [Idx : integer] : integer; // ReadOnly

Ubicación de inicio de la subexpresión número #Idx en la ejecución de la última llamada a Exec*. La primera subexpresión tiene Idx=1, la última es igual a MatchCount, la e.r. completa tiene Idx=0.

Devuelve -1 si en la e.r. no hay subexpresiones o no se encontraron en la cadena ingresada.

propiedad MatchLen [Idx : integer] : integer; // ReadOnly

Longitud de la cadena de la subexpresion número Idx en la ejecución de la última llamada a Exec*. La primera subexpresión tiene Idx=1, la última es igual a MatchCount, la e.r. completa tiene Idx=0.

Devuelve -1 si en la e.r. no hay subexpresiones o no se encontraron en la cadena ingresada.

propiedad Match [Idx : integer] : string; // ReadOnly

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

Devuelve -1 si en la e.r. no hay subexpresiones o no se encontraron en la cadena ingresada.

función LastError : integer;

Devuelve el código de identificación del último error, 0 si no hay errores (No se puede usar si el método Error genera una excepción) y limpia el status interno a 0 (sin errores).

función ErrorMsg (AErrorID : integer) : string; virtual;

Devuelve el mensaje de error de código AErrorID.

propiedad CompilerErrorPos : integer; // ReadOnly

Devuelve la posición en la e.r. donde se detuvo el compilador.

Util para diagnosticar errores.

propiedad SpaceChars : RegExprString

Contiene los caracteres que son tratados como \s (inicializada con la constante global RegExprSpaceChars)

propiedad WordChars : RegExprString;

Contiene los caracteres que son tratados como \w (inicializada con la constante global RegExprWordChars)

```
propiedad LineSeparators : RegExprString
```

Caracteres que son separadores de línea (como \n en Unix), inicializada con la constante global RegExprLineSeparators)

Ver separadores de línea

propiedad LinePairedSeparator : RegExprString

Pares de separadores de línea (como \rn en DOS y Windows).

Debe contener exactamente dos caracteres o ninguno, inicializada con la constante global RegExprLinePairedSeparator)

Ver separadores de línea

Por ejemplo, si se necesita comportamiento tipo Unix asignar a LineSeparators := #$a (caracter de línea nueva) y a LinePairedSeparator := "" (cadena vacía), si se quiere aceptar como separadores de línea \x0Dx0A pero no \x0D o \x0A solos, entonces asignar LineSeparators := "" (cadena vacía) y a LinePairedSeparator := #$d#$a.

Por defecto se usa el modo "mixto" (definido en las constantes globales RegExprLine[Paired]Separator[s]): LineSeparators := #$d#$a; LinePairedSeparator := #$d#$a. El comportamiento de este modo es descripto con más detalle en la sección sintaxis.

función de clase InvertCaseFunction (const Ch : REChar) : REChar;

Convierte Ch en mayúsculas si está en minúsculas o en minúsculas si está en mayúsculas (usa la configuración local corriente)

propiedad InvertCase : TRegExprInvertCaseFunction;

Activar esta propiedad si se desea anular la funcionalidad de insensibilidad a mayúsculas.

Create la inicializa a RegExprInvertCaseFunction (InvertCaseFunction por defecto)

procedimiento Compile;

[Re]compila la e.r. Util para editores GUI de e.r. (para controlar la validez de todas las propiedades).

función Dump : string;

Descarga una e.r. compilada en una forma vagamente comprensible.

CAPÍTULO 9

Constantes globales

Valores por defecto de los modificadores:

```
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;

// valor por defecto de la propiedad SpaceChars

RegExprWordChars : RegExprString =

"0123456789"

+ "abcdefghijklmnopqrstuvwxyz"

+ "**ABCDEFGHIJKLMNOPQRSTUVWXYZ_**";

// valor por defecto de la propiedad WordChars

//NOTA AYUDA EN ESPAOL

// agregar áéíóú

RegExprLineSeparators : RegExprString =

#$d#$a{$IFDEF UniCode}#$b#$c#$2028#$2029#$85{DIF};

// valor por defecto de la propiedad LineSeparators

RegExprLinePairedSeparator : RegExprString =

#$d#$a;

// valor por defecto de la propiedad LinePairedSeparator

RegExprInvertCaseFunction: TRegExprInvertCaseFunction = TRegExpr.InvertCaseFunction;

// valor por defecto de la propiedad

Funciones globales prácticas

función ExecRegExpr (const ARegExpr, AInputStr : string) : boolean;

True si la cadena AInputString es encontrada en la e.r. ARegExpr

Se genera una excepción si hay errores de sintaxis en ARegExpr

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

Parte la cadena AInputStr en subcadenas APieces por las ocurrencia de la e.r. ARegExpr

función ReplaceRegExpr (const ARegExpr, AInputStr, AReplaceStr : string;

AUseSubstitution : boolean = False) : string;

Devuelve AInputStr con las ocurrencias de la e.r. reemplazadas por AReplaceStr

Si AUseSubstitution es verdadero AReplaceStr será usado como plantilla para los métodos de sustitución.

Por ejemplo:

ReplaceRegExpr (“({-i}block|var)s*(s*([^]*)s*)s*”,

“BLOCK(test1)”, “def «$1» value «$2»”, True)

devolverá: def “BLOCK” value “test1”

ReplaceRegExpr (“({-i}block|var)s*(s*([^]*)s*)s*”,

“BLOCK(test1)”, “def «$1» value «$2»”)

devolverá: def «$1» value «$2»

función QuoteRegExprMetaChars (const AStr : string) : string;

Reemplaza todos los metacaracteres por su representación segura , por ejemplo “abc$cd.(“ es convertida en “abc$cd.(“

Esta función es práctica para autogeneración de e.r. a partir de datos del usuario.

función RegExprSubExpressions (const ARegExpr : string;

ASubExprs : TStrings; AExtendedSyntax : boolean = False) : integer;

Genera una lista de subexpresiones encontradas en la e.r. ARegExpr

En ASubExps cada item representa una subexpresión, en el formato:

String - texto de la subexpresión (sin “()”)

low word of Object - posición inicial en ARegExpr, incluyendo “(“ si existe! (la primera posición es 1)

high word of Object - longitud, incluyendo el “(“ inicial y el “)” final si existen!

AExtendedSyntax - debe ser True si el modificador /x estará activado mientras se usa la e.r.

Práctico para editores GUI de e.r., etc (se puede ver un ejemplo de uso en el proyecto *TestRExp.dpr*)

Resultado	Significado
0	Exito. No se encontraron paréntesis desbalanceados;
-1	No hay suficientes paréntesis de cierre “)”;
-(n+1)	En la posición n se encontró un “[“ abriendo sin el correspondiente “]” de cierre;
n	En la posición n se encontró un “)” cerrando sin el correspondiente “(“ de apertura.

Si el resultado es <> 0, entonces ASubExprs puede contener items vacíos o ilegales

Exception type

El administrador de errores por defecto de TRegExpr genera una excepción:

```
ERegExpr = class (Exception)
  public
   ErrorCode : integer; // código de error. Los errores de compilación son menores a 1000.
   CompilerErrorPos : integer; // Posición en la e.r. donde se ocurrió el error de compilación
 end;
```

CAPÍTULO 10

Cómo usar Unicode

TRegExpr ahora soporta UniCode, pero funciona muy lentamente :(

Quién quiere optimizarlo ? ;)

Usarlo exclusivamente si realmente se necesita soporte de Unicode !

Sacar el "." en {.$DEFINE UniCode} en regexpr.pas. Después de ésto todas las cadenas serán tratadas como WideString.

> **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.

CAPÍTULO 11

P. Cómo puedo usar TRegExpr con Borland C++ Builder?

Tengo un problema porque no hay un archivo de cabecera (.h or .hpp) disponible.

11.1 R.

- Agregar RegExpr.pas al proyecto bcb.
- Compilar el proyecto. Esto genera el archivo RegExpr.hpp.
- Ahora se puede escribir código que use la unidad RegExpr.
- No olvidar agregar #include «RegExpr.hpp» donde haga falta.

CAPÍTULO 12

P. Porqué TRegExpr devuelve más de una línea?

Por ejemplo, e.r. `<font .\*>` devuelve la primera línea `<font`, y entonces el resto del archivo incluso el último `</html>`.

12.1 R.

Por compatibilidad con versiones anteriores el modificador /s está activado por defecto.

Desactivarlo y `.` encotrará todo menos separadores de línea.

A propósito, le sugiero `<font (\[^\\n>\]\*)>`, será la URL en Match[1].

CAPÍTULO 13

P. Porqué TRegExpr devuelve más de lo esperado?

Por ejemplo, la e.r. `<p>(.+)</p>` aplicada a la cadena `<p>a</p><p>b</p>` devuelve `a</p><p>b` pero no `a` como esperaba.

13.1 R.

Por defecto, todos los operadores funcionan en modo `voraz`, entonces devuelven lo máximo posible.

Para operación `no voraz` se pueden usar operadores no voraces como `+?` (nuevo en v. 0.940) o cambiar todos los operadores a modo `no voraz` con la ayuda del modificador `g` (usando las propiedades de TRegExpr o construcciones como `?(-g)` en la e.r.).

CAPÍTULO 14

P. Cómo se pueden descomponer textos como HTML con la ayuda de TRegExpr

14.1 R.

Lo siento amigos, pero es prácticamente imposible!

Por supuesto, se puede usar fácilmente TRegExpr para extraer alguna información del HTML, como se muestra en mis ejemplos, pero para desomponer en forma precisa hay que usar un código real de descomposición, no e.r.!

Pueden obtener la explicación completa en el libro `Perl Cookbook` de Tom Christiansen y Nathan Torkington. Brevemente, hay muchas construcciones que són fácilmente descompuestas por el programa apropiado, pero en absoluto por una e.r., y un descomponedor real es MUCHO más rápido porque la e.r. no hace simplemente una búsqueda, incluye una optimización que puede llevar una gran cantidad de tiempo.

CAPÍTULO 15

P. Hay forma de obtener múltiples coincidencias de una plantilla en TRegExpr?

15.1 R.

Se puede hacer un bucle e iterar una por una con el método ExecNext.

No se puede hacer más fácil porque Delphi no es un intérprete como Perl (y es un beneficio, los intérpretes son muy lentos).

Para ver algún ejemplo ver la implementación del método TRegExpr.Replace, o los ejemplos en *HyperLinksDecorator.pas*

CAPÍTULO 16

P. Estoy controlando entradas de usuarios. Porqué TRegExpr devuelve `True` para cadenas incorrectas?

16.1 R.

En muchos casos los usuarios de TRegExpr olvidan que las expresiones regulares son para BUSCAR en una cadena. Entonces, si se pretende que un usuario ingrese sólo 4 dígitos y se usa para ello la expresión `\\d{4,4}`, se puede errar la detección de parámetros incorrectos como `12345` o `cualquier letra 1234 y cualquier otra cosa`. Hay que agregar control para inicio y fin de línea para asegurarse de que no hay nada alrededor: `^\\d{4,4}$`.

CAPÍTULO 17

P. Porqué los iteradores no voraces a veces funcionan como en modo voraz?

Por ejemplo, la e.r. `a+?,b+?` aplicada a la cadena `aaa,bbb` encuentra `aaa,b`, pero debería No encontrar `a,b` a causa de la no voracidad del primer iterador?

17.1 R.

Esta es la limitación de las matemáticas usadas por TRegExpr (y de las e.r. de Perl y muchos Unix) - e.r. realiza sólo una optimización de búsqueda `simple`, y no trata de hacer la mejor optimización. En algunos casos esto es malo, pero en los comunes es mayor la ventaja que la limitación - por motivos de rapidez y predecibilidad.

La regla principal - la e.r. antes que nada intenta encontrar coincidencia desde la posición actual y sólo si es completamente imposible avanzar un caracter e intentar nuevamente desde ese lugar. Entonces, si se usa `a,b+?` se encuentra `a,b`, pero en el caso de `a+?,b+?` es `no recomendado` (a causa de la no voracidad) pero posible encontrar más de una `a`, entonces TRegExpr lo hace y finalmente obtiene una correcta (pero no óptima) coincidencia. TRegExpr como las e.r. de Perl o Unix no intenta avanzar y volver a chequear - lo que sería una `mejor` coincidencia. Más aún, esto no puede ser comparado en absoluto en términos de `mejor o pero coincidencia`.

Por favor, leer `[Sintaxis](regular_expressions.html)`.

> **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.

CAPÍTULO 18

Ejemplos simples

Si no tiene experiencia con las expresiones regulares, por favor vea la sección sintaxis.

CAPÍTULO 19

Usando las rutinas globales

Es simple pero poco flexible y efectivo

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

devuelve True

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

devuelve False, porque hay algunos símbolos antes del número de teléfono y estamos usando el metasímbolo "^" (BeginningOfLine = Inicio de línea)

ReplaceRegExpr ("producto", "Pruebe producto. producto es el mejor !", "TRegExpr");

devuelve "Pruebe TRegExpr. TRegExpr es el mejor !"; ;)

CAPÍTULO 20

Usando la clase TRegExpr

Se obtiene todo el poder de la librería

{ % highlight pascal linenos %} // Esta simple función extrae todas las direcciones de email de la cadena ingresada // y devuelve una lista de estos email en el resultado function ExtraeEmails (const AInputString : string) : string; const EmailRE =”[_a-zA-Zd-.]+@[_a-zA-Zd-]+(.[_a-zA-Zd-]+)+” var r : TRegExpr; begin Result := “”; r := TRegExpr.Create; // Crea el objeto try // asegura la liberación de memoria r.Expression := EmailRE; // La e.r. se compila automáticamente en estructuras internas // cuando se asigna la propiedad Expression if r.Exec (AInputString) then REPEAT Result := Result + r.Match [0] + “, “; UNTIL not r.ExecNext; finally r.Free; end; end; begin ExtraeEmails (“Mis e-mails son anso@mail.ru y anso@usa.net”); // devuelve “anso@mail.ru, anso@usa.net, “ end. // Nota: la compilación de la r.e. realizada al asignar ;a propiedad Expression // toma cierto tiempo, si se usa esta función muchas veces // se sobrecarga inútilmente. // Esto se puede optimizar significativamente creando el objeto TRegExpr // y precompilando la expresión durante la inicialización del programa. { % endhighlight %} { % highlight pascal linenos %} // Este ejemplo extrae números de teléfono // y los descompone en partes (códigos de Ciudad y país, númerotelefónico). // Después substituye estas partes en la máscara ingresada. function ParseTel (const AInputString, ATemplate : string) : string; const IntPhoneRE = “(+d *)?((d+) *)?d+(-d*)*”; var r : TRegExpr; begin r := TRegExpr.Create; // Crea el objeto try // asegura la liberación de memoria r.Expression := IntPhoneRE; // La e.r. se compila automáticamente en estructuras internas // cuando se asigna la propiedad Expression if r.Exec (AInputString) then Result := r.Substitute (ATemplate) else Result := “”; finally r.Free; end; end; begin ParseTel (“El teléfono de AlkorSoft (proyecto PayCash) es +7(812) 329-44-69”, “Código de País $1, código de ciudad $2. El número telefónico completo es $&.”); // devuelve “Código de País +7, código de ciudad (812) . El número telefónico completo es +7(812) 329-44-69.” end. { % endhighlight %}

CAPÍTULO 21

Ejemplos más complejos

Se pueden encontrar ejemplos más complejos del uso de TRegExpr en el proyecto *TestRExp.dpr* y *HyperLinkDecorator.pas*.

Ver también mis artículos en *Delphi3000.com* (Inglés) y *Delphi Kingdom* (Ruso).

CAPÍTULO 22

Explicación detallada

Por favor, ver la descripcion de la 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.

Programa simple para explorar y probar e.r., distribuída como código fuente (proyecto TestRExp.dpr) y TestRExp.exe.

Nota: usa algunas propiedades de VCL que sólo existen en Delphi 4 o superior. Mientras se compila en Delphi 3 o Delphi 2 se recibirán mensajes de error acerca de propiedades desconocidas. Se pueden ignorar, estas propiedades son sólo para ajustar tamao y justificación de componentes cuando el formulario cambia su tamao.

Con la ayuda de este programa se puede determinar fácilmente el número de subexpresiones, saltar a cualquiera de ellas (en el código de la e.r. o en los resultados de la cadena exprorada), probar las funciones Substitute, Replace y Split.

Además se incluyen muchos ejemplos que se pueden usar mientras se aprende la sintaxis de e.r. o en la exploración rápida de las capacidades de TRegExpr.

Ejemplo: Hyper Links Decorator

Funciones para decorar hipervínculos mientras se convierte texto puro en HTML.

Por ejemplo, reemplaza “http://anso.da.ru” con “anso.da.ru” o “anso@mail.ru” con “anso@mail.ru”.

Función DecorateURLs

Busca y reemplaza hipervínculos como “http://. . . ” or “ftp://..” así como vínculos sin protocolo pero que comienzan con “www.” Si quiere modificar direcciones de correo electrónico tiene que usar la función DecorateEmails (ver más abajo).

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

Descripción

Devuelve el texto AText con los hipervínculos decorados.

AFlags indica qué parte del hipervínculo debe ser incluída en la parte VISIBLE del link:

Por ejemplo, si el flag es [durlAddr] entonces el link "http://anso.da.ru/index.htm" será decorado como "anso.da.ru"

type

TDecorateURLsFlags = (durlProto, durlAddr, durlPort, durlPath, durlBMark, durlParam);

TDecorateURLsFlagSet = set of TDecorateURLsFlags;

Descripción

Estos son los valores posibles:

Valor	Significado
durlProto	Protocolo (como "ftp://" or "http://")
durlAddr	Dirección TCP o nombre de dominio (como "anso.da.ru")
durlPort	Número de puerto, si está especificado (como ":8080")
durlPath	Ruta al documento (como "index.htm")
durlBMark	Book mark (como "#mark")
durlParam	Parámetros de la URL (como "?ID=2&User=13")

Función DecorateEMails

Reemplaza todos los e-mails de sintaxis correcta con "ADDR". Por ejemplo, reemplaza "anso@mail.ru" con "anso@mail.ru".

function DecorateEMails (const AText : string) : string;

Descripción

Devuelve el texto AText con los e-mails decorados.