TRegExpr Documentation

Release 0.952

Andrey Sorokin

май 27, 2019

Съдържание

1 Documentation **3**

1.1 Въведение . . . 3
1.2 Прости съвпадения . . . 3
1.3 Escape последователности . . . 4
1.4 Клас от символи . . . 4
1.5 Метасимволи . . . 5
1.6 Модификатори . . . 8
1.7 Perl-разширения . . . 9
1.8 Публични методи и свойства на TRegExpr: . . . 10
1.9 Глобални константи . . . 14
1.10 Полезни глобални функции . . . 15
1.11 Как да използваме Unicode . . . 16
1.12 Q. Как мога да използвам TRegExpr в Borland C++ Builder? . . . 16
1.13 Q. Защо TRegExpr връща повече от един ред? . . . 17
1.14 Q. Защо TRegExpr връща повече, отколкото очаквам? . . . 17
1.15 Q. Как да правя parse на сорсове като HTML с помощта на TregExpr? . . . 17
1.16 Q. Има ли начин за получаване на многократни съвпадения на шаблон в TRegExpr? . . . 18
1.17 Q. Аз проверявам потребителския вход. Защо TRegExpr връща `True` при грешно въведен текст от потребителя? . . . 18
1.18 Q. Защо нежадните итератори понякога изглежда, че работят в жаден режим? . . . 18
1.19 Прости примери . . . 19
1.20 Използване на глобалните процедури . . . 19
1.21 Пример: Hyper Links Decorator . . . 20

This is very old and outdated translation.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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

TRegExpr е мощен и лесен за използване инструмент, който е предназначен за проверка, на базата на предварително дефинирани изрази (шаблони, regular expressions) на входни данни в текстови полета (в бази от данни, web-ориентирани приложения и др.), за търсене и замяна на текст, за създаване на инструменти, подобни на egrep от UNIX и т.н.

С негова помощ лесно можете да проверявате например дали един e-mail адрес е синтактично правилен, да извличате необходимите Ви данни (като телефонни номера, пощенски кодове и др.) от произволен неформатиран текст, да намирате определена информация в web-страници – възможностите са ограничени единствено от Вашето въображение! Обърнете внимание – правилата (изразите) за търсене и проверка може да бъдат изменяни без прекомпилиране на основната Ви програма!

Тази библиотека е freeware и представлява разширена Delphi-версия на библиотеката V8 на Henry Spencer за работа с подмножество на дефинираните в езика Perl нормални изрази (Regular Expressions).

TRegExpr е написан на чист Object Pascal, като всички сорс-кодове са достъпни безплатно.

Освен това към оригиналния C-код са добавени някои разширения и всичко е капсулирано в обектния клас TRegExpr, който е реализиран в един-единствен файл - RegExpr.pas.

За неговото използване не са необходими никакви DLL !

За начало може да погледнете някои прости примери на използване , а ако не сте запознати с изразите Regular Expressions, да разучите техният синтаксис. (За обучение и експерименти може да използвате и демонстрационната програма).

TregExpr може да работи и с Unicode-стрингове - вижте как се работи с Unicode.

Историята на разработката и измененията на библиотеката, както и новите неща, са описани в История.

Ако при работа с библиотеката откриете някакви грешки, ако имате забележки или нови идеи, изпращайте ги без притеснение.

- Guido Muehlwitz – намерил и изчистил грозен бъг в обработката на големи стрингове
- Stephan Klimek – тествал в CPPB и предложил/реализирал много възможности
- Steve Mudford – реализирал параметъра Offset
- Martin Baur (www.mindpower.com) - помощ на немски, полезни съвети
- Yury Finkel – реализирал поддръжката на UniCode, намерил и поправил грешки
- Ralf Junker – някои нови възможности, много съвети по оптимизацията
- Simeon Lilov – помощ на български
- Filip Jirsбk и Matthew Winter (wintermi@yahoo.com) - помощ при реализацията на нежадния режим
- Kit Eason – много примери, използвани в тази помощ
- Juergen Schroth – изчистване на грешки и полезни предложения
- Diego Calp (mail@diegocalp.com), Аржентина – помощ на испански

И на много други – за голямото чистене на грешки !

Глава 1

Documentation

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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

1.1 Въведение

Regular Expressions са широко използван метод за задаване на шаблони за търсене на текст. Специални метасимволи позволяват да се укаже например, дали текстът, който се търси, да е в началото или в края на реда, или пък съдържа n повторения на даден символ.

Regular expressions изглеждат доста страшно на пръв поглед, но те наистина са много прости (т.е. обикновено са прости;)), удобни и мощни средства.

Препоръчвам ви да си поиграете с regular expressions използвайки Windows REStudio – той ще ви помогне да разберете главните концепции. Освен това в него са включени много предварително дефинирани и коментирани изрази

Нека да започнем нашето обучение!

1.2 Прости съвпадения

Всеки прост символ съвпада сам със себе си, освен ако не е метасимвол със специално предназначение, описано по-долу.

Последователност от символи съвпада със същата последователност, например изразът `bluh` ще съвпадне с `bluh` в изследвания стринг. Засега е просто, а ?

Можете да задавате символи, които нормално принадлежат към групата на метасимволите или escape последователностите, да бъдат интерпретирани като нормален (литерален) символ, като просто поставите символа \ пред тях. Например ^ е метасимвол, означаващ начало на реда, а \^ определя самия символ ^. \\ определя \ и т.н..

1.2.1 Примери:

```
Foobar                открива 'foobar'
\^FooBarPtr         открива '^FooBarPtr'
```

1.3 Escape последователности

Всеки символ може да бъде задаван и със синтаксис за escape последователност – така, както е в C и Perl: \n определя символа ‚нов ред'; t определя tab; \r - return; \f - form feed, и т.н. В общия случай може да се използва \xnn, където nn е поредица от шестнадесетични цифри, указващи ASCII-кода на желания символ. Ако искате да зададете символ от таблицата UniCode, може да използвате \x{nnnn}, където nnnn са една или повече шестнадесетични цифри.

```
\xnn                    символ с hex код nn
\x{nnnn}                символ с hex код nnnn (един байт за чист текст и два - за Unicode)
\t                       tab (HT/TAB), също и  \\x09
\n                       newline (NL), също и  \\x0a
\r                      car.return (CR), също и  \\x0d
\f                       form feed (FF), също и \\x0c
\a                       alarm (bell) (BEL), също и \\x07
\e                       escape (ESC), също и  \\x1b
```

1.3.1 Примери:

```
foo\\x20bar               открива 'foo bar' (с интервал в средата)
\\tfoobar                открива 'foobar', предшестван от tab
```

1.4 Клас от символи

Може да зададете клас от символи, като ги затворите в квадратни скоби []. В този случай се проверява за съвпадение на който и да е символ от класа.

Ако първият символ в класа веднага след [е ^, се проверява за съвпадение на всички символи, които не са от този клас.

1.4.1 Примери:

```
foob[aeiou]r   открива 'foobar', 'foober' и т.н., но не 'foobbr', 'foobcr' и т.н.
foob[^aeiou]r открива 'foobbr', 'foobcr' и т.н., но не 'foobar', 'foober' и т.н.
```

Вътре в описанието на класа може да се използва символът - за задаване на обхват, например a-z представлява съкратен запис на всички букви от a до z включително.

Ако искате самият символ - да бъде член на класа, просто го поставете в началото или в края на класа, или пред него сложете \. Ако искате и] да бъде член на класа, поставете го в началото на списъка или пред него сложете \.

Примери:

```
[-az]     открива 'a', 'z' и '-'
[az-]     открива 'a', 'z' и '-'
[a\-z]     открива 'a', 'z' и '-'
[a-z]     открива всички 26 малки букви от 'a' до 'z'
[\n-\x0D] открива което и да е от #10,#11,#12,#13.
[\d-t]     открива всички цифри, '-' or 't'.
[]-a]     открива произволен символ в обхвата ']'..'a'.
```

1.5 Метасимволи

Метасимволите са специални символи, които са същността на Regular Expressions. Има различни типове метасимволиописани по-долу.

```
^               начало на ред
$               край на ред
\A               начало на текст
\Z               край на текст
.               произволен символ
```

1.5.1 Примери:

```
^foobar              открива 'foobar' само ако е в началото на реда
foobar$               открива 'foobar' само ако е в края на реда
^foobar$             открива 'foobar' само ако е единствения стринг на реда
foob.r                открива стрингове като
'foobar', 'foobbr', 'foob1r' и т.н.
```

По подразбиране символът ^ съвпада само с началото на обработвания фрагмент от текста, а символът $ - с неговия край (или пред \n в края), което позволява ускоряване на процеса по откриване на съвпадения. Всички символи \n, които се срещат вътре в текстовия блок, не се разпознават от ^ или $.

Ако обаче искате да третирате входния стринг като буфер с много редове текст, при което ^ да съвпада с началото на всеки ред в този буфер (след \n), а $ – с края на всеки ред (преди \n), това може да стане чрез активиране на модификатора /m.

Метасимволите \A и \Z съвпадат с ^ и $, с изключение на това, че при тях няма многократни съвпадения в случай че се използва модификаторът /m, докато в този случай ^ и $ ще откриват всички вътрешни за текста знаци за край на ред.

Метасимволът . по подразбиране съвпада с всеки символ в текста, но ако изключите модификатора /s, то . няма да открива вътрешните за текста знаци за край на ред.

TRegExpr работи със знаците за край на текст според препоръките на www.unicode.org:

^ съвпада с началото на входния стринг, и, ако модификаторът /m е включен, също така със символите, непосредствено следващи всяко срещане на комбинациите \x0D\x0A или \x0A или \x0D (Ако използвате Unicode-версията на TRegExpr, и след \x2028 или \x2029 или \x0B или \x0C или \x85). Забележете, че няма празен ред в комбинацията \x0D\x0A.

$ съвпада с края на входния стринг, и, ако модификаторът /m е включен, също така със символите, непосредствено предшестващи всяко срещане на комбинациите \x0D\x0A или \x0A или \x0D (Ако използвате Unicode-версията на TRegExpr, и преди \x2028 или \x2029 или \x0B или \x0C или \x85). Забележете, че няма празен ред в комбинацията \x0D\x0A.

. съвпада със всеки символ, но ако изключите модификатора /s тогава . не съвпада с \x0D\x0A и \x0A и \x0D (Ако използвате Unicode-версията на TRegExpr, и с \x2028 или \x2029 или \x0B или \x0C или \x85).

Забележете, че ^.*$ (шаблон за празен ред) няма да даде съвпадение за празен ред в комбинацията \x0D\x0A, но ще даде такова за комбинацията \x0A\x0D.

Обработката на много редове може да бъде настройвана лесно за Вашите нужди чрез свойствата на TRegExpr LineSeparators и LinePairedSeparator, Може да ползвате само разделителите /n в стил Unix , или \r\n на DOS/Windows, или да ги смесвате (както е описано по-горе и е по подразбиране), или дори да дефинирате собствени разделители!

Метасимволи – предварително дефинирани класове

```
\w     буквено-цифров символ (включително `_`)
\W   небуквено-цифров символ
\d     цифров символ
\D     нецифров символ
\s     интервален символ (същото като [\t\n\r\f])
\S     неинтервален символ
```

Може да използвате \w, \d и \s в произволни символни класове.

1.5.2 Примери:

```
foob\dr               съвпада със стрингове като 'foob1r', ''foob6r' и т.н., но не с  'foobar',
↪'foobbr' и т.н.
foob\[\w\s\]r      съвпада със стрингове като 'foobar', 'foob r', 'foobbr' и т.н., но не с 'foob1r
↪', 'foob=r' и т.н.
```

TRegExpr използва свойствата SpaceChars и WordChars за дефиниране на символните класове \w, \W, \s, \S, така, че Вие лесно можете да си ги предефинирате.

Метасимволи - итератори

Всяка част от един regular expression може да бъде последвана от дриг вид метасимволи - итератори. Чрез тях може да задавате броя на срещанията на предхождащите ги символ, , метасимвол или подизраз.

```
\*     нула или повече пъти ("жаден"), подобно на {0,}
+    един или повече пъти ("жаден"), подобно на {1,}
?    нула или един път ("жаден"), подобно на {0,1}
{n}    точно n пъти ("жаден")
{n,}   най-малко n пъти ("жаден")
{n,m} най-малко n но не повече от m пъти ("жаден")
\*?     нула или повече пъти ("нежаден"), подобно на {0,}
+?     един или повече пъти ("нежаден"), подобно на {1,}
??     нула или един път ("нежаден"), подобно на {0,1}
{n}?   точно n пъти ("нежаден")
```

(continues on next page)

(continued from previous page)

```
{n,}? най-малко n пъти ("нежаден")
{n,m}? най-малко n но не повече от m пъти ("нежаден")
```

И така, цифрите във фигурните скоби от типа {n,m} указват минималният брой съвпадения n и максималния m. Формата {n} е еквивалент на {n,n} и осигурява съвпадение точно n пъти. Формата {n,} осигурява съвпадения n или повече пъти. Няма огранияенич за големинатата на n или m, но по-големите числа водят до заемане на повече памет и до забавяне на изчислението на RE.

Ако фигурните скоби се срещнат на друго място, те се третират като обикновени символи.

1.5.3 Примери:

```
foob.\*r                                  съвпада със стрингове като 'foobar',  'foobalkjdflkj9r' и
↪'foobr'
foob.+r                 съвпада със стрингове като 'foobar', 'foobalkjdflkj9r' но не 'foobr'
foob.?r                                   съвпада със стрингове като 'foobar', 'foobbr' и 'foobr' но␣
↪не 'foobalkj9r'
fooba{2}r                 съвпада с 'foobaar'
fooba{2,}r                 съвпада със стрингове като 'foobaar', 'foobaaar', 'foobaaaar' и т.н..
fooba{2,3}r                съвпада със стрингове като 'foobaar' или 'foobaaar' но не 'foobaaaar'
```

Малко обяснение на „жаждата“. „Жаден“ дава колкото се може повече съвпадения, „нежаден“ дава колкото се може по-малко съвпадения. Например, `b+` и `b\*` , приложени към `abbbbc` връщат `bbbb`, `b+?` връща `b`, `b\*?` връща празен стринг, `b{2,3}?` връща `bb`, `b{2,3}` връща `bbb`.

Всички итератори може да бъдат превключени в „нежаден“ режим (виж модификатора `/g`).

Метасимволи - алтернативи

Може да задавате няколко алтернативи за даден израз, използвайки символа `|` като разделител между тях. Например `fee|fie|foe` ще съвпадне с която и да е от думите `fee`, `fie`, или `foe` във входния поток (същият резултат ще се получи и с `f(e|i|o)e`). Първата алтернатива включва всичко от последния разделител (`(`, `[`, или началото на RE) до първия `|`, а последната – всичко от последния `|` до следващия разделител. Затова е препоръчително да поставяте алтернативите в скоби, за да се избегнат недоразуменията за това къде започват или завършват.

Алтернативите се преглеждат от ляво на дясно, като при това се избира първият подходящ вариант. Това означава, че алтернатовите не е задължително да са „жадни“. Например: ако търсим `foo|foot` в `barefoot`, ще имаме съвпадение на алтернативата `foo`, понеже тя е най-лявата алтернатива, и удовлетворява съвпадението. (Това може и да не изглежда важно, но е – когато искате да вземете текста, отговарящ на съвпадението, използвайки скоби.)

Освен това не забравяйте, че `|` се интерпретира като литерален символ в квадратните скоби, така, че ако напишете `[fee|fie|foe]` всъщност това ще отговаря на `[feio|]`.

1.5.4 Примери:

```
foo(bar|foo) съвпада с 'foobar' или 'foofoo'.
```

Метасимволи - подизрази

Кръглите скоби `( ... )` може да бъдат използвани и за отделяне на подизрази (след изпълнение на RE имате информация за позицията, дължината и стойността на всеки подизраз, намираща се в променливите MatchPos, MatchLen и свойствата Match на TRegExpr, а също така можете и да ги замествате в шаблонни стрингове с помощта на TRegExpr.Substitute).

Подизразите са номерирани, като номерацията се базира на подреждането отляво надясно на отварящите скоби.

Първият подизраз има номер `1` (целия RE отговаря на номер `0` – по този начин може да го използвате за заместване в TRegExpr.Substitute като `$0` или `$&`).

1.5.5 Примери:

```
(foobar){8,10}          открива текст, съдържащ 8, 9 или 10 срещания на 'foobar'
foob([0-9]|a+)r      съвпада с 'foob0r','foob1r' , 'foobar', 'foobaar', 'foobaar' etc.
```

Метасимволи – обратни връзки (backreferences)

Метасимволите от `\1` до `\9` се интерпретират като обратни връзки. С метасимвола `\<n>` се задава съответствие с предварително намереният вече подизраз номер `<n>`.

1.5.6 Примери:

```
(.)\1+                          открива напр. 'aaaa' и 'cc'.
(.+)\1+                        открива напр. 'abab' и '123123'
(['"]?)(\d+)\1                открива '"13" (в двойни кавички), или '4' (в единични кавички) или␣
↪77 (без кавички) и т.н.
```

Метасимволи – граници на думи (word boundaries)

```
\b                Match a word boundary
\B                Match a non-(word boundary)
```

Граница на дума (`\b`) е мястото между два символа. То има `w` от едната си страна и `\W` от другата (независимо от реда им), и представлява въображаемите символи, които са между стринга, определен като `\W`.

1.6 Модификатори

Модификаторите служат за промяна на работата на TRegExpr.

Има много начини за задаване на описаните по-долу модификатори.

Всеки от тези модификатори може да бъде вграден в самия RE чрез конструкцията `(?...)`.

Също така може да присвоявате стойности на свойствата на TRegExpr (на ModifierX например за промяна на `/m`, или на ModifierStr за промяна на всички модификатори едновременно). Стойностите по подразбиране за всеки нов обект от тип TRegExpr са дефинирани в глобални променливи, например

RegExprModifierX дефинира първоначалната стойност на свойството ModifierX на новия обект от тип TRegExpr.

Проверката става без да се влияе от регистъра на символите (малки/главни букви) (сравнението се базира на инсталирания език на Вашата система). Виж също InvertCase.

Възприема входния стринг като съставен от много редове, т.е. `^` и `$` съвпадат с разделителите вътре в стринга (виж метасимволи-разделители)

Възприема входния стринг като един текстов ред, като метасимволът `.` съвпада с всеки символ от стринга, включително и с разделителите (виж метасимволи-разделители), с които той нормално не съвпада.

Нестандартен модификатор. Изключете го, ако искате всички следващи операции да са в нежаден режим (по подражбиране е включен). Ако /g е изключен, то `+` работи като `+?`, `*` като `*?` и т.н.

Разширен синтаксис – допуска празни символи и коментарии за по-ясно оформяне на RE (виж обяснението по-долу).

Нестандартен модификатор. Ако той е включен, в диапазона `[а-я]` ще се включи и символа ё, `[А-Я]` ще включва Ё, и `[а-Я]` ще включва всички символи от руската азбука (с българската азбука този проблем не съществува).

По подразбиране този модификатор е включен, а ако това не ви харесва, изключете го посредством глобалната променлива RegExprModifierR.

Модификаторът `/x` се нуждае от малко повече обяснение. Той казва на TRegExpr да игнорира интервалите, които не са нито предшествани от ‚‘, нито са в символен клас. Може да го използвате, за да разделите RE на (малко) по-разбираеми части. Символът `#` се третира като метасимвол, след който следва коментар, например:

```
(
(abc) # Коментар 1
  |     # Можете да използвате интервали за разделители в RE - TRegExpr ги игнорира
(efg) # Коментар 2
)
```

Това също означава, че ако желаете да имате истински интервали или символи `#` във Вашия шаблон (извън символен клас, където те не се влияят от `/x`), ще трябва или да ги въвеждате с escape-последователности, или да ги кодирате с осмични или шестнадесетични последователности. Като цяло обаче тези елементи правят RE доста по-удобни за четене.

1.7 Perl-разширения

Може да го използвате вътре в RE за временно включване или изключване на модификаторите. Ако тази конструкция се срещне в подизраз, ря влияе само на него.

1.7.1 Примери:

```
(?i)Saint-Petersburg                  открива 'Saint-petersburg' и 'Saint-Petersburg'
(?i)Saint-(?-i)Petersburg            открива 'Saint-Petersburg' но не 'Saint-petersburg'
(?i)(Saint-)?Petersburg              открива 'Saint-petersburg' и 'saint-petersburg'
((?i)Saint-)?Petersburg              открива 'saint-Petersburg', но не 'saint-petersburg'
```

(?#text)

Коментар, текстът ‚text‘ се игнорира. Обърнете внимание, че TRegExpr счита за край на коментара първият срещнат символ), затова няма възможност да използвате този символ като част от коментара.

Засега не забравяйте да прочетете FAQ (по-специално въпросът за нежадната оптимизация).

> **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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

1.8 Публични методи и свойства на TRegExpr:

```
property Expression : string
```

Представлява самият RE.

С цел оптимизация, присвоявате RE на това свойство, TRegExpr автоматично го компилира в ‚P-код‘ (Може да го видите с помощта на метода Dump) и го записва във вътрешните си структури. Истинско [pe]компилиране се прави само когато възникне нужда от това – при извикване на Exec[Next], Substitute, Dump и т.н. и само ако самият израз или друго свойство, влияещи върху P-кода бъдат променени след последната [pe]компилация.

В случай на грешка при [pe]компилиране, се извиква методът Error (по подразбиране методът Error предизвиква изключение – виж по-долу)

```
property ModifierStr : string
```

За проверка/установяване на стойностите на модификаторите на RE. Форматът на стринга е подобен на този при модификаторите (?ismx-ismx). Например, ModifierStr := ‚i-x‘ ще включи модификатора /i, ще изключи /x и няма да промени останалите.

Ако се опитате да зададете непознат модификатор, ще се извика методът Error (по подразбиране методът Error предизвиква изключение ERegExpr).

```
property ModifierI : boolean
```

Модификатор /i – проверка без отчитане главни/малки букви. Приема начална стойност от RegExprModifierI.

```
property ModifierR : boolean
```

Модификатор /r – използване на диапазони за руски език. Приема начална стойност от RegExprModifierR.

```
property ModifierS : boolean
```

Модификатор /s - „.‘ съвпада с всеки символ (иначе не съвпада с LineSeparators и LinePairedSeparator). Приема начална стойност от RegExprModifierS.

```
property ModifierG : boolean;
```

Модификатор /g - изключването на /g превключва всички операции в „нежаден“ стил така, че ако ModifierG = False, то всички ‚*‘ работят като ‚*?‘, всички ‚+‘ като ‚+?‘ и т.н. Приема начална стойност от RegExprModifierG.

```
property ModifierM : boolean;
```

Модификатор /m - третира стринга като съставен от много редове. Т.е. задава за ‘^‘ и ‘$‘ вместо да съвпадат със самото начало или край на стринга, да съвпадат с началото и края на всяки ред вътре в самия стринг. Приема начална стойност от RegExprModifierM.

```
property ModifierX : boolean;
```

Модификатор /x – eXtended синтаксис. Приема начална стойност от RegExprModifierX.

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

Проверка за съвпадение на RE с входния стринг AInputString

!!! Exec запазва AInputString в свойството InputString

```
function ExecNext : boolean;
```

търси за следващо съвпадение:

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

работи по същия начин, както

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

но е доста по-просто за разбиране !

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

Търси за съвпадение на RE с входния стринг, започвайки от позиция AOffset

(AOffset=1 – първият символ на InputString)

```
property InputString : string;
```

текущият входен стринг (присвоен явно или от последния Exec).

сяко присвояване на стринг на това свойство изтрива стойностите на свойствата Match* !

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

Връща стринга Atemplate, в който ‚$&‘ или ‚$0‘ е заменено с целия RE, а ‚$n‘ се заменя с подизраза с номер n.

След версия v.0.929 се използва ‚$‘ вместо ‚‘ (за бъдещи разширения и за по-голяма Perl-съвместимост) и приема повече от една цифра.

Ако искате да запишете просто символите ‚$‘ или ‚‘, запишете пред тях ‚‘

Пример: ‚1$ is $2rub‘ -> ‚1$ is <Match[2]>rub‘

Ако искате да запишете обикновена цифра след ‚$n‘, ще трябва да заградите n с фигурни скоби ‚{}‘.

Пример: ‚a$12bc‘ -> ‚a<Match[12]>bc‘, a${1}2bc‘ -> ‚a<Match[1]>2bc‘.

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

Разделя AInputStr на парчета в APieces според срещането на RE. Използва Exec [Next].

```
function Replace (AInputStr : RegExprString;  const AReplaceStr :
```

RegExprString; AUseSubstitution : boolean = False) : RegExprString;

Връща AinputStr, в който срещанията на RE са заменени с AreplaceStr. Ако AUseSubstitution е true, тогава AReplaceStr ще се използва като шаблон за методите Substitution.

1.8.1 Пример:

```
Expression := '({-i}block|var)\\s\*\\(\\s\*(\[^ \]\*)\\s\*\\)\\s\*';
Replace ('BLOCK( test1)', 'def "$1" value "$2"', True);
```

ще върне: `def 'BLOCK' value 'test1'`

```
Replace ('BLOCK( test1)', 'def "$1" value "$2"', False)
```

ще върне: `def "$1" value "$2"`

Използва Exec[Next]

```
function ReplaceRegExpr (const ARegExpr, AInputStr, AReplaceStr :
```

string; AUseSubstitution : boolean = False) : string;

Връща AinputStr, в който срещанията на RE са заменени с AreplaceStr. Ако AUseSubstitution е true, тогава AReplaceStr ще се използва като шаблон за методите Substitution.

1.8.2 Пример:

```
ReplaceRegExpr ('({-i}block|var)\\s\*\\(\\s\*(\[^ \]\*)\\s\*\\)\\s\*', 'BLOCK( test1)', 'def "$1"␣
↪value "$2"', True)
```

ще върне: `def 'BLOCK' value 'test1'`

```
ReplaceRegExpr ('({-i}block|var)\\s\*\\(\\s\*(\[^ \]\*)\\s\*\\)\\s\*', 'BLOCK( test1)', 'def "$1"␣
↪value "$2"')
```

ще върне: `def "$1" value "$2"`

```
property SubExprMatchCount : integer; // ReadOnly
```

Връща броя на подизразите, намерени при последното извикване на Exec*.

Ако няма подизрази, а е намерен само целият израз (Exec* е върнала True), то SubExprMatchCount=0, а ако не са намерени нито подизрази, нито целия RE (Exec* е върнала false), то `SubExprMatchCount=-1`.

Забележете, че някои подизрази може и да не бъдат намерени и за тях `MathPos=MatchLen= -1` и `Match=''`.

1.8.3 Пример:

```
Израз := '(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') - връща False: SubExprMatchCount=-1
```

. property MatchPos [Idx : integer] : integer; // ReadOnly

позиция на намерения подизраз. #Idx е номера на подизраза в последния стринг на Exec*. Първият подизраз има Idx=1, последния - MatchCount, целия израз има Idx=0.

Връща -1 ако в израза няма такъв подизраз или ако той не е намерен.

```
property MatchLen \[Idx : integer\] : integer; // ReadOnly
```

дължина на намерения подизраз. #Idx е номера на подизраза в последния стринг на Exec*. Първият подизраз има Idx=1, последния - MatchCount, целия израз има Idx=0.

Връща -1 ако в израза няма такъв подизраз или ако той не е намерен.

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

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

Връща ‚‘ ако в израза няма такъв подизраз или ако той не е намерен.

```
function LastError : integer;
```

Връща номера на последната грешка или 0 ако няма грешки (неизползваем ако методът Error генерира изключение) и установява вътрешния статус в 0 (няма грешки).

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

Връща съобщение за грешка с номер = AErrorID.

```
property CompilerErrorPos : integer; // ReadOnly
```

Връща позицията в израза, където компилаторът е спрял.

Полезно при диагностика на грешки

```
property SpaceChars : RegExprString
```

Съдържа символите, третирани като \s (инициализира се с глобалната константа RegExprSpaceChars.

```
property WordChars : RegExprString;
```

Съдържа символите, третирани като \w (инициализира се с глобалната константа RegExprWordChars.

```
property LineSeparators : RegExprString
```

разделителите на редове (като \n в Unix), (инициализира се с глобалната константа RegExprLineSeparators.

Виж също за разделителите

```
property LinePairedSeparator : RegExprString
```

сдвоени разделители на редове (като \rn в DOS и Windows).

Трябва да съдържа точно два символа или да не съдържа нищо (инициализира се с глобалната константа RegExprLinePairedSeparator).

Виж също за разделителите

Например, ако имате нужда от Unix-стил, задайте `LineSeparators := \#$a` (символ за нов ред) и `LinePairedSeparator := ''` (празен стринг), а ако искате да обработвате разделителя `\x0D\x0A`, но не `\x0D` или `\x0A`, задайте `LineSeparators := ''` (празен стринг) и `LinePairedSeparator := \#$d\#$a`.

По подразбиране се използва ‚смесен' режим (дефиниран в глобалните константи `RegExprLine\[Paired\]Separator\[s\]`): `LineSeparators := \#$d\#$a; LinePairedSeparator := $\#d\#$a`. Действието на този режим е подробно обяснено в раздела, касаещ синтаксиса.

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

Конвертира Ch в горен регистър (ако е бил в долен) или в долен регистър (ако е бил в горен), като използва системните настройки.

```
property InvertCase : TRegExprInvertCaseFunction;
```

Ако искате да работите със зависимост от регистъра на символите, установете това свойство да сочи RegExprInvertCaseFunction (по подразбиране сочи InvertCaseFunction)

```
procedure Compile;
```

[Pe]компилира RE. Полезна за редактори, използващи RE в GUI (за проверка на валидността на всички свойства).

```
function Dump : string;
```

показва компилираният израз в по-подробен вид.

```
class function VersionMajor: integer;
class function VersionMinor: integer;
```

Връщат главната и второстепенната версия, например за v. 0.944 VersionMajor = 0 и VersionMinor = 944

1.9 Глобални константи

Стойности по подразбиране на модификаторите:

```
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; // стойност по подразбиране за свойството␣
↪SpaceChars
```

```
RegExprWordChars : RegExprString =  '0123456789'
 + 'abcdefghijklmnopqrstuvwxyz'
 + 'ABCDEFGHIJKLMNOPQRSTUVWXYZ\_';
// стойност по подразбиране за свойството WordChars

RegExprLineSeparators : RegExprString =
   \#$d\#$a{$IFDEF UniCode}\#$b\#$c\#$2028\#$2029\#$85{$ENDIF};
// стойност по подразбиране за свойството LineSeparators

RegExprLinePairedSeparator : RegExprString = \#$d\#$a;
// стойност по подразбиране за свойството LinePairedSeparator

RegExprInvertCaseFunction : TRegExprInvertCaseFunction = TRegExpr.InvertCaseFunction;
// стойност по подразбиране за свойството InvertCase
```

1.10 Полезни глобални функции

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

връща истина ако в стринга AInputString се открие съвпадение на израза ARegExpr

! ако в AregExpr има синтактични грешки, генерира изключение

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

разделя AInputStr на парчета в APieces според срещанията на ARegExpr

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

Връща AinputStr, като срещанията на израза ARegExpr са заменени с AReplaceStr

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

Заменя всички метасимволи с тяхната безопасна форма, например ‚abc$cd.(‚ се конвертира в ‚abc$cd.(‚

Тази функция е полезна при автоматично генериране на изрази от потребителски вход

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

Прави списък на подизразите, намерени в RE ARegExpr

В ASubExps всяка стойност представлява подизраз, от първия до последния, във формат:

```
String - текст на подизраза (без '()')
low word на Object - стартова позиция в  ARegExpr, вкл. '(' ако съществува! (първата позиция е 1)
high word на Object - дължината, вкл. '(' и ')' ако съществуват!
AExtendedSyntax - трябва да е True ако модификаторът /x е включен при използването на RE.
```

Полезна за редактори на RE в GUI и т.н. (Пример за използване има в проекта TestRExp.dpr)

Резултат	Значение
0	Успех. Няма намерени небалансирани скоби.
-1	няма достатъчно затварящи скоби ')'

(continues on next page)

(continued from previous page)

```
-(n+1)     на позиция n е намерена отваряща '\[' без съответстваща й затваряща '\]'
n           на позиция n е намерена затваряща ')' без съответстваща й отваряща '('
```

ако резултатът е <> 0, то ASubExpr съдържа празни или некоректни стрингове.

Той обработва грешките на TRegExpr:

```
ERegExpr = class (Exception)
   public
      ErrorCode : integer; // код за грешка. Кодовете за грешка при компилиране са < 1000.
      CompilerErrorPos : integer; // Позиция в израза, където е възникнала грешката
end;
```

1.11 Как да използваме Unicode

TRegExpr поддържа UniCode, но работи с него много бавно :(

Някой иска ли да го оптимизира ? ;)

Използвайте го само ако наистина ви трябва поддръжка на Unicode !

Премахнете `'.'` от `{.$DEFINE UniCode}` в `regexpr.pas`. След това всички стрингове ще бъдат третирани като `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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

Q : въпрос

A : отговор

1.12 Q. Как мога да използвам TRegExpr в Borland C++ Builder?

Имам проблем, понеже няма хедър-файлове (.h или .hpp).

1.12.1 A.

- Добавете `RegExpr.pas` в проект bcb
- Компилирайте проекта. Това ще генерира файла RegExpr.hpp
- Вече може да се пише код, който използва unit-а RegExpr
- Не забравяйте да добавите `\#include "RegExpr.hpp"` където е необходимо

1.13 Q. Защо TRegExpr връща повече от един ред?

Например RE `<font .\*>` връща първия `<font`, и след това останалата част от файла, вкл. последния `</html>`...

1.13.1 A.

За съвместимост с по-стари версии модификаторът /s е включен по подразбиране.

Изключете го (например чрез `ModifierS := false` или за всички НОВИ обекти `RegExprModifierS := false`) и `.` ще съвпада с всичко освен \n – така, както искате.

BTW препоръчвам `<font (\[^\\n>\]\*)>`, тогава в `Match\[1\]` ще бъде URL.

1.14 Q. Защо TRegExpr връща повече, отколкото очаквам?

Например, шаблонът `<p>(.+)</p>` , приложен към стринг `<p>a</p><p>b</p>` връща `a</p><p>b` , а не `a`, както очаквам.

1.14.1 A.

По подразбиране всички оператори работят в **жаден** режим, т.е. дават колкото е възможно по-голямо съвпадение.

Ако искате да работят в **нежаден** режим, можете или да използвате **нежадни** оператори като `+?` и др. (ново във версия v. 0.940), или да превключите всички оператори в **нежаден** режим с помощта на модофокатора `g` (използвайте съответните свойства на TRegExpr или конструкции от типа `?(-g)` направо в RE).

1.15 Q. Как да правя parse на сорсове като HTML с помощта на TregExpr?

1.15.1 A. Съжалявам, това е почти невъзможно!

Разбира се, може да използвате TRegExpr за лесно извличане на информация от HTML, както съм показал в примера си, но ако искате истински parsing ще трябва да използвате истинсли parser, не RE!

Може да прочетете подробното обяснение в книгата на Tom Christiansen и Nathan Torkington `Perl Cookbook` например. Накратко – има много конструкции, на които може лесно да се прави parse от истински parser, но не и от RE, и истинският parser МНОГ ПО-БЪРЗО прави parsing-а, защото RE не сканират просто входния поток, той прави и оптимизиращи търсения, което отнема доста повече време.

1.16 Q. Има ли начин за получаване на многократни съвпадения на шаблон в TRegExpr?

1.16.1 A.

Може да направите цикъл и да правите постъпкови съвпадения с метода ExecNext.

Това не може да стане лесно, понеже Delphi не е интерпретатор, какъвто е Perl (и в това е предимството му – интерпретаторите работят много бавно!).

Ако имате нужда от пример, разгледайте реализаията на метода TRegExpr.Replace или погледнете примерите в *HyperLinksDecorator.pas*

1.17 Q. Аз проверявам потребителския вход. Защо TRegExpr връща `True` при грешно въведен текст от потребителя?

1.17.1 A.

В много от случаите потребителите на TRegExpr забравят, че RE е за ТЪРСЕНЕ във входния стринг. Така, че ако искате да накарате потребителя да въвежда само 4 цифри и за това използвате шаблона `\\d{4,4}`, може да пропуснете грешни въвеждания от типа `12345` или `букви 1234 и други букви`. Трябва да добавите проверки за начало и край на реда, за да сте сигурни, че няма нищо наоколо: `^\\d{4,4}$`.

1.18 Q. Защо нежадните итератори понякога изглежда, че работят в жаден режим?

Например, RE `a+?,b+?` , приложена към стринга `aaa,bbb` , връща `aaa,b`, а не би ли трябвало да връща `a,b` заради нежадния първи итератор?

1.18.1 A.

Това е ограничение на използваната в TRegExpr (и в Perl и в много RE под Unix) математика – RE прави само `проста` оптимизация при търсене, и не се опитва да прави най-добрата оптимизация. В някои случаи това е лошо, но като цяло е по-скоро предимство, отколкото ограничение – заради производителността и предвидимостта на резултатите.

Основното правило е – RE най-напред се опитва да направи съвпадение от текущата позиция и само ако е напълно невъзможно, се премества напред с един символ и отново опитва от новата позиция. Така, че ако използвате `a,b+?` , се открива `a,b`, но в случай на `a+?,b+?` , не е задължително (заради нежадността), но е възможно да се даде съвпадение за повече от едно `a`, така, че TRegExpr го прави и накрая връща коректния (но не оптимален) резултат. TregExpr, както и RE на Perl или Unix не се опитват да се придвижват напред и да проверяват дали ще има `по-добро` съвпадение. Нещо повече, те изобщо не могат да определят кое съвпадение е по-добро и кое – по-лошо.

Мпля, прочетете `[Syntax](regular_expressions.html#engine)`.

> **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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

1.19 Прости примери

Ако не сте запознати с Regular Expressions, можете да прочетете моето описанието на техния синтаксис.

1.20 Използване на глобалните процедури

Това е най-простият, но не най-гъвкавият и ефективен начин:

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

връща True

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

връща False, защото има символи преди самия телефонен номер, а ние използваме метасимвола ‚^‘ (BeginningOfLine)

ReplaceRegExpr (‚product‘, ‚Take a look at product. product is the best !‘, ‚TRegExpr‘);

връща ‚Take a look at TRegExpr. TRegExpr is the best !‘ ;)

1.20.1 Използване на обекти от клас TRegExpr

Това дава в ръцете ви мощността на цялата библиотека.

- 1

Тази проста функция извлича всички e-mail адреси от входния стринг и връща техният списък в резултантния стринг:

{% highlight pascal linenos %} 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; // Създаваме обекта try // гарантира освобождаването на паметта r.Expression := EmailRE; // при това присвояване RE се компилира автоматично // ако има грешка, възниква Exception 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‘); // връща ‚anso@mail.ru, anso@usa.net, , end. {% endhighlight %} Забележка: компилирането на RE при присвояването й отнема време.

Затова ако използвате тази функция многократно, ще се получи значително забавяне.

Значително оптимизиране се постига ако по-рано създадете TRegExpr и още при инициализирането компилирате RE.

- 2

Този прост пример извлича телефонен номер от входящия стринг и го разделя на съставни части (код на страната, на града и вътрешен номер). След това тези части се поставят в шаблон.

{% highlight pascal linenos %} function ParsePhone (const AInputString, ATemplate : string) : string; const IntPhoneRE = ‚(+d *)?((d+) *)?d+(-d*)*‘; var r : TRegExpr; begin r := TRegExpr.Create; // Създаваме обекта try // гарантира освобождаване на паметта r.Expression := IntPhoneRE; // Присвоява сорс-кода на RE. Той ще бъдекомпилиран при необходимост // (например ако се извика Exec). Ако в RE имагрешки, // ще възникне run-time exception прикомпилацията му 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 $&.‘); // връща ‚Zone code +7, city code (812) . Whole phone number is +7(812) 329-44-69.‘ end. {% endhighlight %}

1.20.2 По-сложни примери

Може да намерите по-сложни примери на използване на TRegExpr в проекта TestRExp.dpr и *HyperLinkDecorator.pas*.

Вижте и статията ми в на английски език и на руски език.

Детайлно описание

Вижте пълното описание на интерфейса на 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 transifex.com. It is already machine-translated and need only proof-reading and may be some copy-pasting from here.

Това е проста програма, която можете да използвате за изследване и тестване на RE. Разпространява се като сорс (проекта TestRExpr.dpr) и като TestRExpr.exe.

Отбележете, че тя използва някои свойства на VCL които съществуват само в Delphi 4 или по-висока версия. Ако компилирате в Delphi 3 или Delphi 2, ще получите съобщения за грешки, отнасящи се за непознати свойства. Можете да ги игнорирате – тези свойства за необходими за промяна на размерите и наместване на компонентите, ако формата промени размерите си.

С помощта на тази програма може лесно да определяте броя на подизразите, които да редактирате, да отидете на някой дефиниран подизраз (както в изходния код на RE, така и в резултатите, върнати от него), да си поиграете с функциите Substitude, Replace и Split и др.

Нещо повече – в демонстрационния проект са включени много примери – използвайте ги за разучаване на синтаксиса на RE или за бързо разучаване на възможностите на TRegExpr.

1.21 Пример: Hyper Links Decorator

DecorateURLs DecorateEMails

Функции за създаване на хипервръзки при конвертиране на обикновен текст в HTML.

Например, заменя ‚http://anso.da.ru‘ с ‚anso.da.ru‘ или ‚anso@mail.ru‘ с ‚anso@mail.ru‘.

функция DecorateURLs

Тя е за търсене и замяна на хипервръзки като ‚http://. . . ‘ или ‚ftp://..‘ или такива без протокол, но започващи с ‚www.‘ Ако искате да конвертирате и e-mail адрес, използвайте функцията DecorateEMails.

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

Описание

Връща текста AText форматиран като хипервръзка.

AFlags описва кои части от хипервръзката да се включат в нейната ВИДИМА част:

Например, ако е [durlAddr], то хипервръзката ‚http://anso.da.ru/index.htm‘ ще се форматира като ‚anso.da.ru‘

type

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

TDecorateURLsFlagSet = set of TDecorateURLsFlags;

Описание

Това са възможните стойности:

Стойност	Значение
durlProto	Protocol (like ‚ftp://‘ or ‚http://‘)
durlAddr	TCP address or domain name (like ‚anso.da.ru‘)
durlPort	Port number if specified (like ‚:8080‘)
durlPath	Path to document (like ‚index.htm‘)
durlBMark	Book mark (like ‚#mark‘)
durlParam	URL params (like ‚?ID=2&User=13‘)

функция DecorateEMails

Заменя всички синтактически правилни e-mail адреси с ‚ADDR‘. Например, заменя ‚anso@mail.ru‘ с ‚anso@mail.ru‘.

function DecorateEMails (const AText : string) : string;

Описание

Връща AText с фарматирани e-mail адреси