TRegExpr Documentation

Выпуск 0.952

Andrey Sorokin

дек. 14, 2019

Оглавление

	English	Русский	Deutsch	Български	Français	Español

Библиотека TRegExpr реализует регулярные выражения.

Регулярные выражения - простой в использовании и мощный инструмент для сложного поиска и замены, а также для проверки текста на основе шаблонов.

Это особенно полезно для проверки пользовательского ввода в формах ввода - для проверки адресов электронной почты и так далее.

Также вы можете извлекать номера телефонов, почтовые индексы и т.д. из веб-страниц или документов, искать сложные шаблоны в файлах журналов и все, что вы можете себе представить. Правила (шаблоны) могут быть изменены без перекомпиляции вашей программы.

TRegExpr реализован на чистом Паскале. Является частью Lazarus (Free Pascal) проекта. Но также существует как отдельная библиотека, которая может быть скомпилирована Delphi 2-7, Borland C ++ Builder 3-6.

Глава 1

Отзывы

Как библиотека была встречена.

Глава 2

Быстрый старт

Чтобы использовать библиотеку, просто добавьте исходники в ваш проект и далее используйте класс TRegExpr.

Благодаря FAQ вы можете учиться на чужих ошибках.

Готовое к запуску приложение Windows REStudio поможет вам выучить и отладить регулярные выражения.

Глава 3

Обратная связь

Если вы видите какие-либо проблемы, пожалуйста, создайте баг.

Глава 4

Исходный код

Чистый Object Pascal.

- Оригинальная версия
- FreePascal fork (GitHub зеркало SubVersion)

Глава 5

Документация

	English	Русский	Deutsch	Български	Français	Español

5.1 Регулярные выражения (RegEx)

5.1.1 Вступление

Регулярные выражения - удобный способ описывать шаблоны текстов.

С помощью регулярных выражений вы можете проверять пользовательский ввод, искать некоторые шаблоны, такие как электронные письма телефонных номеров на веб-страницах или в некоторых документах и так далее.

Ниже приведена исчерпывающая шпаргалка по регулярных выражениям всего на одной странице.

5.1.2 Символы

Простые совпадения

Любой отдельный символ соответствует самому себе.

Серия символов соответствует этой серии символов во входной строке.

RegEx	Находит
`foobar`	`foobar`

Непечатные символы (escape-коды)

Для представления непечатаемого символа в регулярном выражении вы используете `\x..`:

RegEx	Находит
\xnn	символ с шестнадцатеричным кодом “ nn“
\x{nnnn}	символ с шестнадцатеричным кодом nnnn (один байт для простого текста и два байта для Unicode)
foo\x20bar	foo bar (обратите внимание на пробел в середине)

Существует ряд предопределенных escape-кодов для непечатных символов, как в языке C:

RegEx	Находит
\t	tab (HT/TAB), тоже что \x09
\n	символ новой строки (NL), то же что \x0a
\r	car.return (CR), тоже что \x0d
\f	form feed (FF), то же что \x0c
\a	звонок (BEL), тоже что \x07
\e	escape (ESC), то же что \x1b
\cx	Ctrl-комбинация (Ctrl-x) Например \ci соответствует \x09, поскольку ctrl-i имеет код 0x09

Эскейпинг

Если вы хотите использовать символ \ сам по себе, а не как часть escape-кода, просто добавьте к нему префикс \, например: \\.

На самом деле вы можете поставить перед префиксом (или escape)“‘‘ любой символ, имеющий особое значение в регулярных выражениях.

RegEx	Находит
\^FooBarPtr	^FooBarPtr здесь ^ не означает *начало строки*
\[a\]	[a] это не *класс символов*

5.1.3 Классы символов

Пользовательские классы

Символьный класс - это список символов внутри []. Класс соответствует любому **одному** символу, указанному в этом классе.

RegEx	Находит
foob[aeiou]r	foobar, foober и т. д., но не foobbr, foobcr и т. д.

Вы можете инвертировать класс - если первый символ после [является ^, то класс соответствует любому символу, **кроме** символов, перечисленных в классе.

RegEx	Находит
“ Foob [^ AEIOU] r“	foobbr, foobcr и т. д., но не foobar, foober и т. д.

Внутри списка символ - используется для указания диапазона, так что `a-z` представляет все символы между `a` и `z` включительно.

Если вы хотите, чтобы - сам был членом класса, поместите его в начало или конец списка или *escape* с обратной косой чертой.

Если вы хотите буквально использовать символы `]` или `[`, поместите их в начало списка или *escape* обратной косой чертой.

RegEx	Находит
`[-az]`	`a`, `z` и `-`
`[az-]`	`a`, `z` и `-`
`[A\-z]`	`a`, `z` и `-`
`[a-z]`	символы от `a` до `z`
`[\n-\x0D]`	символы от #10 до #13

Предопределенные классы символов

Существует ряд предопределенных классов символов, которые делают регулярные выражения более компактными.

RegEx	Находит
`\w`	буквенно-цифровой символ (включая `_`)
`\W`	не буквенно-цифровой
`\d`	числовой символ (такой же как `[0123456789]`)
`\D`	нечисловой
`\s`	любой пробел (такой же как `[\t\n\r\f]`)
`\S`	не пробел
`\h`	горизонтальный разделитель. Табуляция, пробел и все символы которые определены в Unicode как «space separator».
`\H`	не горизонтальный разделитель
`\v`	вертикальные разделители. новая строка и все символы которые входят в набор «разделители строк» Unicode.
`\V`	не вертикальный разделитель

Вы можете использовать `\w`, `\d` и `\s` *в классах пользовательских символов*.

RegEx	Находит
`foob\dr`	`foob1r`, `foob6r` и т. д., но не `foobar`, `foobbr` и т. д.
`foob[\w\s]r`	`foobar`, `foob r`, `foobbr` и т. д., но не `foob1r`, `foob=r` и т. д.

Примечание: TRegExpr

Свойства SpaceChars и WordChars определяют, какие символы входят в классы \w, \W,“s“, \S.

Таким образом, вы можете переопределить эти классы.

5.1.4 Разделители

Разделители строк

RegEx	Находит
^	начало строки
$	конец строки
\A	начало текста
\Z	конец текста
.	любой символ в строке
^Foobar	foobar только если он находится в начале строки
foobar$	foobar, только если он в конце строки
^foobar$	foobar только если это единственная строка в строке
foob.r	foobar, foobbr, foob1r и так далее

Метасимвол ^ по умолчанию соответствует началу входной строки. $ - конец.

Однако вы можете захотеть рассматривать строку как многострочный текст, так что ^ будет соответствовать месту перед разделителем строк во входном тексте, а $ - месте после любого разделителя строк. Для этого переключите *modifier /m*.

Обратите внимание, что в последовательности \x0D\x0A нет пустой строки.

Примечание: TRegExpr

Если вы используете Unicode версию, то ^/$ также соответствует \x2028, \x2029, \x0B, \x0C или \x85.

\A и \Z похожи на ^ и $, за исключением того, что они не будут совпадать несколько раз, когда *modifier /m* используется.

Метасимвол . по умолчанию соответствует любому символу, но если вы переключите Off на *modifier /s*, то . не будет совпадать с разделителями строк внутри строки.

Обратите внимание, что выражение ^.*$ не соответствует точке между \x0D\x0A, потому что это неразрывный разделитель строк. Но оно соответствует пустой строке в последовательности \x0A\x0D, потому из-за неправильного порядка кодов он не воспринимается как разделитель строк и считается просто двумя символами.

Примечание: TRegExpr

Многострочная обработка может быть настроена с помощью свойств LineSeparators и LinePairedSeparator.

Таким образом, вы можете использовать разделители стиля Unix \n или стиль DOS / Windows \r\n или смешивать их вместе (как описано выше по умолчанию).

Если вы предпочитаете математически правильное описание, вы можете найти его на сайте www.unicode.org.

Разделители слов

RegEx	Находит
\b	разделитель слов
\B	разделительс с **не**-словом

Граница слова \b - это точка между двумя символами, у которой \w с одной стороны от нее и \W с другой стороны (в любом порядке).

5.1.5 Повторы

Повтор

За любым элементом регулярного выражения может следовать допустимое число повторений элемента.

RegEx	Находит
{n}	ровно n раз
{n,}	по крайней мере n раз
{n,m}	по крайней мере n, но не более чем m раз
*	ноль или более, аналогично {0,}
+	один или несколько, похожие на {1,}
?	ноль или единица, похожая на {0,1}

То есть цифры в фигурных скобках {n,m} определяются минимальное n и максимальное m количество повторов (совпадений во входном тексте).

{n} эквивалентно {n,n} и означает точно n раз.

{n,} соответствует n или более раз.

Нет ограничений на величину n и m.

Если фигурная скобка встречается в любом другом контексте, она рассматривается как обычный символ.

RegEx	Находит
foob.*r	foobar, foobalkjdflkj9r и foobr
foob.+r	foobar, foobalkjdflkj9r, но не foobr
foob.?r	foobar, foobbr и foobr, но не foobalkj9r
fooba{2}r	foobaar
fooba{2}r	foobaar, foobaaar, foobaaaar и т. д.
fooba{2,3}r	foobaar, или foobaaar, но не foobaaaar
(foobar){8,10}	8, 9 или 10 экземпляров foobar (() это *Subexpression*)

Жадность

Повторы в жадном режиме захватывают как можно больше из входного текста, в не жадном режиме - как можно меньше.

По умолчанию все повторы являются жадными. Используйте ? Чтобы сделать любой повтор не жадным.

Для строки abbbbc:

RegEx	Находит
b+	bbbb
b+?	b
b*?	пустой строки
b{2,3}?	bb
b{2,3}	bbb

Вы можете переключить все квантификаторы в режим не жадный (*modifier /g*, ниже мы используем *in-line модификатор change*).

RegEx	Находит
(?-g)b+	b

5.1.6 Альтернативы

Выражения в списке альтернатив разделяются |.

Таким образом, fee|fie|foe будет соответствовать любому из fee, fie или foe (также как и f(e|i|o)e).

Первое выражение включает в себя все от последнего разделителя шаблона ((, [или начало шаблона) до первого |, а последнее выражение содержит все от последнего | к следующему разделителю шаблона.

Звучит сложно, поэтому обычной практикой является заключение списка альтернатив в скобки, чтобы минимизировать путаницу относительно того, где он начинается и заканчивается.

Выражения в списке альтернатив пробуются слева направо, принимается первое же совпадение.

Например, регулярное выражение foo|foot в строке barefoot будет соответствовать foo - первое же совпадение.

Также помните, что | в квадратных скобках воспринимается просто как символ, поэтому, если вы напишите [fee|fie|foe], это тоже самое что [feio|].

RegEx	Находит
foo(bar|foo)	foobar или foofoo

5.1.7 Подвыражения

Скобки (...) также могут использоваться для определения подвыражений регулярного выражения.

Примечание: TRegExpr

Позиция, длина и фактические значения подвыражений будут в MatchPos, MatchLen и Match.

Вы можете заменить их на Заменить.

Подвыражения нумеруются слева направо по открывающим их скобкам (включая вложенные подвыражения).

Первое подвыражение имеет номер 1. Целое регулярное выражение имеет номер 0.

Например, для входной строки `foobar` регулярное выражение `(foo(bar))` найдет:

подвыражение	значение
0	`foobar`
1	`foobar`
2	`bar`

5.1.8 Backreferences

Метасимволы от `\1` до `\9` интерпретируются как обратные ссылки. `\n` соответствует ранее найденному подвыражению `n`.

RegEx	Находит
`(.)\1+`	`aaaa` и `cc`
`(.+)\1+`	также `abab` и `123123`

`(['"]?)(\d+)\1` соответствует `"13"` (в двойных кавычках) или `'4'` (в одинарных кавычках) или `77` (без кавычек) и т. д.

5.1.9 Модификаторы

Модификаторы предназначены для изменения поведения регулярных выражений.

Вы можете установить модификаторы глобально в вашей системе или изменить их внутри регулярного выражения, используя *(?imsxr-imsxr)*.

Примечание: TRegExpr

Для изменения модификаторов используйте ModifierStr или соответствующие `TRegExpr` свойства Модификатор *.

Значения по умолчанию определены в глобальных переменных. Скажем, глобальная переменная `RegExprModifierX` определяет значение по умолчанию для свойства `ModifierX`.

---

i, без учета регистра

Регистро-независимые сравнения. Использует установленные в вашей системе языковые настройки, см. также InvertCase.

m, многострочные строки

Обрабатывать строку как несколько строк. Таким образом, `^` и `$` соответствуют началу или концу любой строки в любом месте строки.

Смотрите также *Разделители строк*.

s, одиночные строки

Обрабатывать строку как одну строку. Так что . соответствует любому символу, даже разделителям строк.

Смотрите также *Разделители строк*, которые обычно не совпадают.

r, жадность

Примечание: Специфичный для TRegExpr модификатор.

Отключив его `Off`, вы переключите *повторитель* в *не-жадный* режим.

Итак, если модификатор `/g` имеет значение `Off`, то `+` работает как `+?`, `*` как `*?` и так далее.

По умолчанию этот модификатор имеет значение `Выкл`.

x, расширенный синтаксис

Позволяет комментировать регулярные выражения и разбивать их на несколько строк.

Если модификаторор `Вкл`, мы игнорируем все пробелы, которые не заэскейплены обратной косой чертой, и не включены в класс символов.

Также символ `#` отделяет комментарии.

Обратите внимание, что вы можете использовать пустые строки для форматирования регулярного выражения для лучшей читаемости:

```
(
(abc) # комментарий 1
#
(efg) # комментарий 2
)
```

Это также означает, что если вам нужно вставить пробел или символ `#` в шаблон (вне класса символов, где они не затрагиваются `/x`), вам придется либо эскейпить их, либо кодировать, используя шестнадцатеричный код.

r, русские диапазоны

Примечание: Специфичный для TRegExpr модификатор.

В русской таблице ASCII символы ё / Ё размещаются отдельно от других.

Большие и маленькие русские символы находятся в отдельных диапазонах, это не отличается от ситуации с английскими символами, но, тем не менее, я хотел иметь краткую форму.

С этим модификатором вместо `[а-яА-ЯёЁ]` вы можете написать `[а-Я]`, если вам нужны все русские символы.

Когда модификатор `Вкл`:

RegEx	Находит
`а-я`	символы от `а` до `я` и `ё`
`А-Я`	символы от `А` до `Я` и `Ё`
`а-Я`	все русские символы

Модификатор по умолчанию установлен на `Вкл`.

5.1.10 Расширения

(?=<lookahead>)

`Заглядывание вперед`. Проверяет совпадение для регулярного выражения в `<look-ahead>`, но не включает это совпадение в результат.

Примечание: TRegExpr

`Заглядывание вперед` не реализовано в TRegExpr.

Во многих случаях вы можете заменить `Заглядывание вперед` на *Sub-expression* и просто игнорировать то, что будет записано в этом подвыражении.

Например, `(blah)(?=aoobar)(blah)` совпадает с `(blah)(foobar)(blah)`. Но в варианте с подвыражениями вы должны исключить среднее подвыражение вручную - используйте `Match [1] + Match [3]` и игнорируйте `Match[2]`.

Это просто не так удобно, как с `Заглядыванием вперед`, где вы можете использовать весь `Match[0]`, поскольку захваченное `<look-ahead>` совпадение не будет включено в найденное регулярное выражение.

(?:<non-capturing group>)

`?:` можно использовать, если вы хотите сгруппировать какую-то часть регулярного выражения, но вам не нужно чтобы соответствующей ей текст захватывался.

Таким образом это способ организации регулярного выражения без захламления результата:

RegEx	Находит
`(https?|ftp)://([^/ \r\n]+)`	в `https://sorokin.engineer` захватит подвыражения `https` и `sorokin.engineer`
`(?:https?|ftp)://([^/ \r\n]+)`	в `https://sorokin.engineer` захватит только `sorokin.engineer`

(?imsgxr-imsgxr)

Вы можете использовать его внутри регулярного выражения для изменения модификаторов на лету.

Это может быть особенно удобно, поскольку оно имеет локальную область видимости в регулярном выражении. Оно влияет только на ту часть регулярного выражения, которая следует за оператором `(?imsgxr-imsgxr)`.

И если оно находится внутри подвыражения, оно будет влиять только на это подвыражение, а именно на ту часть подвыражения, которая следует за оператором. Таким образом, в `((?i)Saint)-Petersburg` это влияет только на подвыражение `((?i)Saint)`, поэтому оно будет соответствовать `saint-Petersburg`, но не `saint-petersburg`.

RegEx	Находит
`(?i)Saint-Petersburg`	`Saint-petersburg` и `Saint-Petersburg`
`(?i)Saint-(?-i)Petersburg`	`Saint-Petersburg`, но не `Saint-petersburg`
`(?i)(Saint-)?Petersburg`	`Saint-petersburg` и `saint-petersburg`
`((?i)Saint-)?Petersburg`	`saint-Petersburg`, но не `saint-petersburg`

(?#текст)

Комментарий, текст игнорируется.

Обратите внимание, что комментарий закрывается ближайшим `)`, поэтому нет способа вставить литерал `)` в комментарий.

5.1.11 Послесловие

В этой древней статье из прошлого века есть примеры использования регулярных выражений.

| | English | Русский | Deutsch | Български | Français | Español |

5.2 TRegExpr

Реализует регулярные выражения в чистом паскале. Совместим с Free Pascal, Delphi 2-7, Borland C ++ Builder 3-6.

Чтобы использовать это просто скопируйте исходный код в ваш проект.

Библиотека уже включена в Lazarus (Free Pascal) проект, поэтому вам не нужно ничего копировать, если вы используете Lazarus.

5.2.1 Класс TRegExpr

VersionMajor, VersionMinor

Вернуть мажорную и минорную версию, например, для `version 0.944`

```
VersionMajor = 0 VersionMinor = 944
```

Expression

Регулярное выражение.

Для оптимизации регулярное выражение автоматически компилируется в `P-код`. Читаемая человеком форма `P-кода` возвращает *Dump*.

В случае каких-либо ошибок при компиляции вызывается метод `Error` (по умолчанию `Error` вызывает исключение *ERegExpr*)

ModifierStr

Установить или получить значения модификаторов регулярных выражений.

Формат строки такой же, как в (? Ismx-ismx). Например, `ModifierStr: = &#39;i-x&#39;` включит модификатор /i, выключить /x и оставляйте без изменений других.

Если вы попытаетесь установить неподдерживаемый модификатор, будет вызвано `Error`.

ModifierI

Модификатор /i, «регистро-нечуствительный», инициализируется значением *RegExprModifierI*.

ModifierR

Модификатор /r, «расширение русского диапазона», инициализируется значением *RegExprModifierR*.

модификаторы

Модификатор /s, «однострочный текст», инициализируется значением *RegExprModifierS*.

ModifierG

Модификатор /g, «жадность», инициализируется значением *RegExprModifierG*.

ModifierM

Модификатор /m, «многострочный текст», инициализируется значением *RegExprModifierM*.

ModifierX

Модификатор /x, «расширенный синтаксис», инициализируется значением *RegExprModifierX*.

Exec

Ищет регулярное выражение в `AInputString`.

Доступна перегруженная версия `Exec` без `AInputString` - она использует `AInputString` из предыдущего вызова.

Смотрите также глобальную функцию *ExecRegExpr*, которую вы можете использовать без явного создания объекта `TRegExpr`.

ExecNext

Найти следующее совпадение.

Без параметра работает так же, как

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

Вызывает исключение, если используется без предшествующего успешного вызова *Exec*, *ExecPos* или *ExecNext*.

Таким образом, вы всегда должны использовать что-то вроде

```
if Exec (InputString)
  then
    repeat
      { Обработка }
    until not ExecNext;
```

ExecPos

Находит совпадение для `InputString`, начиная с позиции `AOffset`

```
AOffset = 1 // первый символ InputString
```

Строка ввода

Возвращает текущую входную строку (из последнего вызова *Exec* или последнего присвоения этому свойству).

Любое присвоение этому свойству очищает *Match*, *MatchPos* и *MatchLen*.

Substitute

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

Возвращает `ATemplate` с `$&` или `$0`, замененным целым регулярным выражением, а `$n` заменяется вхождением числа подвыражения `n`.

Чтобы поместить в шаблон символы `$` или `\`, используйте префикс `\`, например `\` или `\$`.

условное обозначение	описание
$&	полное совпадение регулярного выражения
$0	полное совпадение регулярного выражения
$N	регулярное подвыражение n совпадение
\n	для Windows \r\n
\l	строчный следующий символ
\L	делает строчными все символы после этого
\u	делает заглавным один следующий символ
\U	делает заглавными все символы после этого

```
'1\$ is $2\\rub\\' -> '1$ это <Match[2]>\rub\'
'\U$1\\r' преобразуется '<Match[1] в верхнем регистре>\r'
```

Если вы хотите поместить необработанную цифру после `n`, вы должны разделить `n` фигурными скобками `{}`.

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

Split

Разделяет AInputStr на APieces по найденным регулярным выражениям

Внутренне вызывает *Exec* / *ExecNext*

Смотрите также глобальную функцию *SplitRegExpr*, которую вы можете использовать без явного создания объекта `TRegExpr`.

Replace, ReplaceEx

```
function Replace (Const AInputStr : RegExprString;
  const AReplaceStr : RegExprString;
  AUseSubstitution : boolean= False)
 : RegExprString; overload;

function Replace (Const AInputStr : RegExprString;
  AReplaceFunc : TRegExprReplaceFunction)
 : RegExprString; overload;

function ReplaceEx (Const AInputStr : RegExprString;
  AReplaceFunc : TRegExprReplaceFunction):
  RegExprString;
```

Возвращает строку с повторениями, замененными строкой замены.

Если последний аргумент (`AUseSubstitution`) равен true, то `AReplaceStr` будет использоваться в качестве шаблона для методов подстановки.

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

Возвращает `def "BLOCK" value "test1"`

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

Возвращает `def "$1" value "$2"`

Внутренне вызывает *Exec* / *ExecNext*

Перегруженная версия и `ReplaceEx` работают с функцией обратного вызова, поэтому вы можете реализовать действительно сложную функциональность.

Смотрите также глобальную функцию *ReplaceRegExpr*, которую вы можете использовать без явного создания объекта `TRegExpr`.

SubExprMatchCount

Количество подвыражений было найдено в последнем вызове *Exec* / *ExecNext*.

Если нет подвыражений но был найдено все выражение (Exec* вернул True), то `SubExprMatchCount=0`, если ни подвыражений, ни всего выражения не найдено (*Exec* / *ExecNext* вернул false), тогда `SubExprMatchCount=-1`.

Обратите внимание, что некоторые субэкспр. может быть не найден и для такого подэкспр. `MathPos = MatchLen = -1` и `Match = &#39;&#39;`.

```
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
```

MatchPos

позиция найденного подвыражения `#Idx` в строке после последнего `Exec*`. Первое подвыражение будет `Idx=1`, последнее - `MatchCount`, все выражение `Idx=0`.

Возвращает `-1`, если нет такого подвыражения или это подвыражение не найдено во входной строке.

MatchLen

длина найденного подвыражения `#Idx` в последнем `Exec*`. Первое подвыражение `Idx=1`, последнее - MatchCount, все выражение `Idx=0`.

Возвращает -1, если в такого подвыражения нет в выражении или оно не найдено во входной строке.

Match

Возвращает `&#39;&#39;`, если такого подвыражении нет или оно не найдено во входящей строке.

LastError

Возвращает `ID` последней ошибки, `0`, если ошибок нет (невозможно использовать, если метод `Error` вызывает исключение) и очищает внутренний статус в `0` (без ошибок).

ErrorMsg

Возвращает сообщение об ошибке `Error` с `ID = AErrorID`.

CompilerErrorPos

Возвращает pos в re там компилятор остановился.

Полезно для диагностики ошибок

SpaceChars

Содержит символы, которые рассматриваются как `\s` (изначально заполнены глобальной константой *RegExprSpaceChars*)

WordChars

Содержит символы, которые рассматриваются как `\w` (изначально заполнены глобальной константой *RegExprWordChars*)

LineSeparators

разделители строк (например, `\n` в Unix), изначально заполненные глобальной константой *RegExprLineSeparators*)

смотрите также Разделители строк

LinePairedSeparator

разделитель парных строк (например, `\r\n` в DOS и Windows).

должен содержать ровно два символа или вообще не содержать символов, изначально заполненных глобальной константой RegExprLinePairedSeparator)

смотрите также Разделители строк

Например, если вам нужно поведение в стиле Unix, присвойте `LineSeparators: = #$a` и `LinePairedSeparator: = ''` (пустая строка).

Если вы хотите принять в качестве разделителей строк только `\x0D \ x0A`, но не `\x0D` или `\x0A`, тогда присвойте `LineSeparators: = &#39;&#39;` (пустая строка) и `LinePairedSeparator: = #$d#$a`.

По умолчанию используется смешанный режим (определенный в глобальных константах разделителя RegExprLine [Paired]):

```
LineSeparators: = #$d#$a; LinePairedSeparator: = #$d#$a
```

Поведение этого режима подробно описано в Разделителях строк.

InvertCase

Инвертирует регистр символов для регистро-независимого поиска. Переопределите, если хотите другое поведение.

Compile

Компилирует регулярное выражение.

Полезно, например, для редакторов регулярных выражений в графическом интерфейсе - для проверки регулярных выражений без их использования.

Dump

Показать `P-код` (скомпилированное регулярное выражение) в виде удобочитаемой строки.

5.2.2 Глобальные константы

EscChar

Escape-char, по умолчанию \.

RegExprModifierI

Модификатор i значение по умолчанию

RegExprModifierR

Модификатор r значение по умолчанию

RegExprModifierS

Модификатор s значение по умолчанию

RegExprModifierG

Модификатор g значение по умолчанию

RegExprModifierM

Модификатор m значение по умолчанию

RegExprModifierX

Модификатор x значение по умолчанию

RegExprSpaceChars

По умолчанию для свойства *SpaceChars*

RegExprWordChars

Значение по умолчанию для свойства *WordChars*

RegExprLineSeparators

Значение по умолчанию для свойства *LineSeparators*

RegExprLinePairedSeparator

Значение по умолчанию для свойства *LinePairedSeparator*

RegExprInvertCaseFunction

По умолчанию для свойства *InvertCase*

5.2.3 Глобальные функции

Вся эта функциональность доступна как методы `TRegExpr`, но с глобальными функциями вам не нужно создавать экземпляр `TReExpr`, поэтому ваш код будет более простым, если вам просто понадобится одна функция.

ExecRegExpr

Значение true, если строка соответствует регулярному выражению. Так же, как *Exec* в `TRegExpr`.

SplitRegExpr

Разбивает строку по регулярным выражениям. Смотрите также *Split*, если вы предпочитаете явно создавать экземпляр `TRegExpr`.

ReplaceRegExpr

```
function ReplaceRegExpr (
    const ARegExpr, AInputStr, AReplaceStr : RegExprString;
    AUseSubstitution : boolean= False
) : RegExprString; overload;

Type
  TRegexReplaceOption = (rroModifierI,
                         rroModifierR,
                         rroModifierS,
                         rroModifierG,
                         rroModifierM,
                         rroModifierX,
                         rroUseSubstitution,
                         rroUseOsLineEnd);
  TRegexReplaceOptions = Set of TRegexReplaceOption;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
function ReplaceRegExpr (
    const ARegExpr, AInputStr, AReplaceStr : RegExprString;
    Options :TRegexReplaceOptions
) : RegExprString; overload;
```

Возвращает строку с регулярными выражениями, замененными на `AReplaceStr`. Смотрите также *Replace*, если вы предпочитаете создавать экземпляр TRegExpr явно.

Если последний аргумент (`AUseSubstitution`) равен true, то `AReplaceStr` будет использоваться в качестве шаблона для `методов подстановки`:

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

Возвращает `def 'BLOCK' value 'test1'`

Но этот (обратите внимание, что нет последнего аргумента):

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

Возвращает `def "$1" value "$2"`

Версия с опциями

С `Options` вы управляете поведением `\n` (если `rroUseOsLineEnd`, то `\n` заменяется на `\n\r` в Windows и `\n` в Linux). И так далее.

```
Type
  TRegexReplaceOption = (rroModifierI,
                         rroModifierR,
                         rroModifierS,
                         rroModifierG,
                         rroModifierM,
                         rroModifierX,
                         rroUseSubstitution,
                         rroUseOsLineEnd);
```

QuoteRegExprMetaChars

Замените все мета-символы своим безопасным представлением, например, `abc'cd.(` преобразуется в `abc\'cd\.\(`

Эта функция полезна для повторного автогенерации из пользовательского ввода

RegExprSubExpressions

Составляет список подвыражений, найденных в `ARegExpr`

В `ASubExps` каждый элемент представляет подвыражение, от первого до последнего, в формате:

String - текст подвыражения (без „()“)

младшее слово Object - начальная позиция в ARegExpr, включая ‘(’ если существует! (первая позиция 1)

старшее слово Object - длина, включая начало ((и окончание)), если существует!

`AExtendedSyntax` - должно быть `True`, если модификатор `/x` будет `On` при использовании r.e.

Полезно для графических редакторов и т. Д. (Пример использования можно найти в REStudioMain.pas)

Код результата	Имея в виду
0	Успех. Не найдено несбалансированных скобок
-1	недостаточно закрывающих скобок)
-(n+1)	в позиции n было найдено открытие [без соответствующего закрытия]
N	в позиции n была найдена закрывающая скобка) без соответствующего открывания (

Если `Result <> 0`, то `ASubExprs` может содержать пустые или недопустимые элементы.

5.2.4 ERegExpr

```
ERegExpr = class (Exception)
  public
   ErrorCode : integer; // error code. Ошибка компиляции менее 1000
   CompilerErrorPos : integer; // Позиция в которой найдена ошибка
 end;
```

5.2.5 Unicode

UniCode замедляет производительность, поэтому используйте его, только если вам действительно нужна поддержка Unicode.

Чтобы использовать Unicode раскомментируйте `{$DEFINE UniCode}` в regexpr.pas (удалить `off`).

После этого все строки будут рассматриваться как WideString.

| | English | Русский | Deutsch | Български | Français | Español |

5.3 Часто задаваемые вопросы

5.3.1 Я обнаружил ужасную ошибку: TRegExpr вызывает исключение Access Violation!

Ответ

Вы должны создать объект перед использованием. Итак, после того, как вы объявили что-то вроде:

```
r: TRegExpr
```

не забудьте создать экземпляр объекта:

```
r: = TRegExpr.Create.
```

5.3.2 Регулярные выражения с (? = . . .) не работают

Look-ahead не реализованы в TRegExpr. Но во многих случаях вы можете легко заменить их простыми подвыражениями.

5.3.3 Поддерживает ли он Юникод?

Ответ

Как использовать Юникод

5.3.4 Почему TRegExpr возвращает более одной строки?

Например, регулярное выражение `<font .\*>` возвращает первый же `<font`, далее весь последующий текст до финального“</html>“.

Ответ

Для обратной совместимости модификатор /s по умолчанию `Вкл`.

Выключите его, и `.` будет соответствовать любому символу, кроме Разделителей строк - именно так, как вы хотите.

Я лично предлагаю `<font ([^\n>] *)>`, тогда в `Match [1]` будет URL.

5.3.5 Почему TRegExpr возвращает больше, чем я ожидаю?

Например `<p> (. +) </p>` для строки `<p>a </p><p> b </p>` возвращает `a </p><p> b` но не `a`, как ожидается.

Ответ

По умолчанию все операторы работают в `жадном` режиме, поэтому они совпадают как можно больше.

Если вам нужен режим `не жадный режим`, вы можете использовать `не жадные` варианты операторов, такие как `+?` и т. д., или переключить все операторы в `не жадный` режим с помощью модификатора `g` (используйте соответствующие свойства TRegExpr или оператор `?(-g)` внутри выражения).

5.3.6 Как анализировать HTML, с помощью TRegExpr?

Ответ

Извините, ребята, но это почти невозможно!

Конечно, вы можете легко использовать TRegExpr для извлечения некоторой информации из HTML, как показано в моих примерах, но если вам нужен точный синтаксический анализ, вы должны использовать полноценный парсер, а не регулярные выражения

Вы можете прочитать полное объяснение в Том Кристиансен и Натан Торкингтон `Perl Cookbook`, например.

Вкратце - есть много структур, которые могут быть легко проанализированы реальным парсером, но не могут быть проанализированы регулярными выражениями. Полноценный парсер намного быстрее выполнит синтаксический анализ.

5.3.7 Есть ли способ получить несколько совпадений шаблона на TRegExpr?

Ответ

Вы искать последующие совпадения с помощью метода ExecNext.

Если вам нужен какой-то пример, посмотрите на реализацию метода `TRegExpr.Replace` или на примеры для HyperLinksDecorator

5.3.8 Я проверяю пользовательский ввод. Почему TRegExpr возвращает `True` для неправильных входных строк?

Ответ

Во многих случаях пользователи TRegExpr забывают, что регулярное выражение предназначено для **поиска** во входной строке.

Так, например, выражение `\d{4,4}` совпадет и с `12345` и с `любые буквы 1234`.

Вы должны проверить от начала строки до конца строки, чтобы убедиться, что вокруг ничего больше нет: `^\d{4,4}$`.

5.3.9 Почему не жадные итераторы иногда работают в жадном режиме?

Например, `a+?,b+?`, для строки `aaa,bbb`, найдет `aaa,b`, но не `a,b` хотя первый итератор не жаден?

Ответ

Регулярные выражения только ищут первое же совпадение и не пытаются найти «наилучшее» совпадение.

В некоторых случаях это плохо, но в целом это скорее преимущество, чем ограничение, по причинам производительности и предсказуемости.

Основное правило - сначала пытаемся найти соотвествие, начиная с текущей позиции в строке и, только если это невозможно, продвигаемся на один символ вперед и попробуем снова со следующей позиции в тексте.

Если вы используете `a,b+?` то это будет соответствовать `a,b`. В случае `a+?,b+?`, не смотря на не жадный модификатор, все же возможно захватить более одного `a`, поэтому TRegExpr сделает это.

Регулярные выражения, не пытаются двигаться дальше по тексту и проверять - удастся ли найти «лучшее» совпадение. Хотя бы потому, что нельзя сказать, что такое «лучше».

5.3.10 Как использовать TRegExpr с Borland C ++ Builder?

У меня проблема, нет файла заголовка (`.h` или `.hpp`).

Ответ

- Добавьте `RegExpr.pas` к проекту `bcb`.
- Скомпилировать проект. Это создает заголовочный файл `RegExpr.hpp`.
- Теперь вы можете писать код, использующий модуль `RegExpr`.
- Не забудьте добавить `#include "RegExpr.hpp"` там, где это необходимо.
- Не забудьте заменить все \ в регулярных выражениях на \ или переопределить EscChar const.

5.3.11 Почему многие примеры (включая примеры из документации) работают неправильно в Borland C ++ Builder?

Ответ

Подсказка есть в предыдущем вопросе;) Символ \ имеет особое значение в `C ++`, поэтому вы должны `эскейпить` его (как описано в предыдущем ответе). Но если вам не нравится, как выглядит `\w+\\w+\.\w+`, вы можете переопределить константу `EscChar` (в `RegExpr.pas`). Скажем, `EscChar = "/"`. Затем вы можете написать `/w+/w+/. /W+` - выглядит необычно, но более читабельно.

| | English | Русский | Deutsch | Български | Français | Español |

5.4 Demos

Демо-код для TRegExpr

5.4.1 Вступление

Если вы не знакомы с регулярными выражениями, посмотрите на синтаксис регулярных выражений.

Интерфейс TRegExpr описан в TRegExpr.

5.4.2 Text2HTML

Text2HTML исходники

Преобразует текст в HTML

Использует блок HyperLinksDecorator, основанный на TRegExpr.

Этот блок содержит функции для оформления гиперссылок.

Например, замените " www.masterAndrey.com" на " www.masterAndrey.com " или `filbert@yandex.ru` на `<a href="mailto:filbert@yandex.ru">filbert@yandex.ru</a>`.

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

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

(continues on next page)

(продолжение с предыдущей страницы)

```
TDecorateURLsFlagSet = set of TDecorateURLsFlags;

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

Значение	Имея в виду
durlProto	Протокол (например, `ftp://` или `http://`)
durlAddr	TCP-адрес или доменное имя (например, `masterAndrey.com`)
durlPort	Номер порта, если указан (например, `: 8080`)
durlPath	Путь к документу (например, `index.html`)
durlBMark	Закладка (например, “ # mark“)
durlParam	Параметры URL (например, `? ID = 2 &amp; User = 13`)

Возвращает введенный текст `AText` с оформленными гиперссылками.

`AFlags` описывает, какие части гиперссылки должны быть включены в видимую часть ссылки.

Например, если *AFlags* равно `[durlAddr]`, то гиперссылка `www.masterAndrey.com / contacts.htm` будет оформлена `<a href="www.masterAndrey.com/contacts.htm">www.masterAndrey.com</a>`.

5.4.3 TRegExprRoutines

Очень простые примеры, см. Комментарии внутри блока

5.4.4 TRegExprClass

Чуть более сложные примеры, см. Комментарии внутри блока

Глава 6

Переводы

Документация доступна на английском и русском языках.

Есть также старые переводы на немецкий, болгарский, французский и испанский языки. Если вы хотите помочь обновить эти старые переводы, пожалуйста, свяжитесь со мной.

Новые переводы основаны на GetText и могут быть отредактированы с помощью transifex.com.

Они уже переведены автоматически и нуждаются только в корректуре, и, возможно, копировании каких-то частей из старых переводов.

Глава 7

Благодарности

Сообществом предложено и реализовано множество функций TRegExpr.

Я не могу перечислить здесь всех, но я ценю все сообщения об ошибках, предложения функций и вопросы, которые я получаю от вас.

- Guido Muehlwitz - обнаружена и исправлена ошибка в обработке больших строк
- Stephan Klimek - тестирование в CPPB и предложение / реализация многих функций
- Steve Mudford - реализован параметр Offset
- Martin Baur (www.mindpower.com) - немецкий перевод, полезные предложения
- Yury Finkel - реализовал поддержку UniCode, нашел и исправил некоторые ошибки
- Ralf Junker - Реализованы некоторые функции, много предложений по оптимизации
- Симеон Лилов - болгарский перевод
- Филип Джирсбк и Мэтью Винтер - помогли в реализации не жадного режима
- Kit Eason много примеров для документации
- Juergen Schroth - поиск ошибок и полезные советы
- Martin Ledoux - французский перевод
- Diego Calp, Аргентина - испанский перевод