Назначение

Универсальный инструмент для проверки пользовательского ввода на соответствие шаблонам, для самого изощренного поиска, а также замены подстрок.

Вы можете проверить синтаксическую корректность e-mail адреса, извлечь номера телефонов из неформализованного текста, найти необходимую информацию в web-странице - возможности ограничены только Вашим воображением. Правила (шаблоны) можно изменять не перекомпилируя Вашу программу !

В качестве языка правил используется подмножество регулярных выражений Перла (Perl regular expressions, regexp).

Распространяется в исходных текстах, полностью на Object Pascal (совместим с Delphi, Kylix, FreePascal), не нужны никакие DLL.

Документация на русском, английском, немецком, болгарском, французском и испанском доступна
на домашней странице TRegExpr.

Установка элементарна, работа осуществляется через удобную объектную оболочку TRegExpr.

Для начала Вы можете ознакомится с рядом небольших примеров использования, если Вы не знакомы с регулярными выражениями, рекомендую просмотреть краткое описание синтаксиса (для изучения и экспериментов удобно использовать R.E.Studio).

TRegExpr может работать с unicode-строками (см. как это делать).

История изменений описана в файле History.txt.

Лицензия

Установка

- Распакуйте архив в любой (вновь созданный) каталог.
- При необходимости, в этот же каталог распакуйте архив с локализацией документации и примеров.
- Добавьте RegExpr.pas (размещен в подкаталоге Source) в список файлов Вашего проекта (Гл.меню Delphi -> Project -> Add to project..).
- Теперь просто используйте класс TRegExpr или глобальные процедуры из regexpr.pas в Вашем проекте (см. примеры использования). Не забудьте добавить 'uses RegExpr' (или выполнить Гл.меню Delphi -> File -> Use Unit..) в соответствующие модули Вашего проекта.

Точно также устанавливается библиотека HyperLinksDecorator.pas (для ее работы нужно установить TRegExpr).

Для более удобной работы, а также для быстрого освоения регулярных выражений рекомендуется использовать R.E.Studio.

Синтаксис регулярных выражений

Введение

Регулярные выражения - это широкоиспользуемый способ описания шаблонов для поиска текста и проверки соответствия текста шаблону. Специальные **метасимволы** позволяют определять, например, что Вы ищете подстроку в начале входной строки или определенное число повторений подстроки.

На первый взгляд регулярные выражения выглядят страшновато (ну хорошо, на второй - еще страшнее ;)).
Однако Вы очень быстро оцените всю их мощь. Они съэкономят Вам многие часы ненужного кодирования, а в некоторых случаях будут и быстрее работать, чем вручную закодированные проверки.

Я настоятельно рекомендую Вам "поиграть" с поставляемой в дистрибутиве демо-программой TestRExp.dpr - это позволит Вам лучше понять принцип работы регулярных выражений и отладить Ваши собственные выражения. Кроме того, в TestRExp входит множество примеров выражений.

Давайте начнем наше знакомство с регулярными выражениями!

Простое сравнение

Любой символ совпадает с самим собой, если он не относится к специальным метасимволам описанным чуть ниже.

Последовательность символов совпадает с такой же последовательностью во входной строке, так что шаблон "bluh" совпадет с подстрокой "bluh" во входной строке. Пока все просто, не так ли ?

Если необходимо, чтобы метасимволы или **escape-последовательности** воспринимались как обычные символы, их нужно предварять символом "\", например, метасимвол "^" обычно совпадает с началом строк, однако, если записать его как "\^", то он будет совпадать с символом "^", "\\" совпадает с "\" и т.д.

Примеры:
foobar *находит 'foobar'*
\^FooBarPtr *находит '^FooBarPtr'*

Escape-последовательности

Любой символ может быть определен с помощью escape последовательности, так же как это делается в языках C или Perl: "\n" означает начало строки, "\t" - табуляцию и т.д.. Вообще, \xnn, где nn это последовательность шестнадцатеричных цифр, означает символ с ASCII-кодом nn. Если необходимо определить двухбайтный (Unicode) символ, используйте формат '\x{nnnn}', где 'nnnn' - одна или более шестнадцатеричных цифр.

\xnn *символ с шестнадцатеричным кодом nn*
\x{nnnn} *символ с шестнадцатеричным кодом nnnn (более одного байта можно задавать только в режиме Unicode)*
\t *табуляция (HT/TAB), можно также \x09*
\n *новая строка (NL), можно также \x0a*
\r *возврат каретки (CR), можно также \x0d*
\f *перевод формата (FF), можно также \x0c*
\a *звонок (BEL), можно также \x07*
\e *escape (ESC), можно также \x1b*

Примеры:

foo\x20bar *находит 'foo bar' (обратите внимание на пробел посередине)*
\tfoobar *находит 'foobar' предшествуемый табуляцией*

Перечни символов

Вы можете определить перечень, заключив символы в []. Перечень будет совпадать с любым **одним** символом перечисленным в нем.

Если первый символ перечня (сразу после "[") - "^", то такой перечень совпадает с любым символом **не** перечисленным в перечне.

Примеры:
foob[aeiou]r *находит 'foobar', 'foober' и т.д. но не 'foobbr', 'foobcr' и т.д.*
foob[^aeiou]r *находит 'foobbr', 'foobcr' и т.д.. но не 'foobar', 'foober' и т.д.*

Внутри перечня символ "-" может быть использован для определения **диапазонов** символов, например a-z представляет все символы между "a" и "z", включительно.

Если Вам необходимо включить в перечень сам символ "-", поместите его в начало или конец перечня или предварите '\'. Если Вам необходимо поместить в перечень сам символ ']', поместите его в самое начало или предварите '\'.

Примеры:
[-az] *'a', 'z' и '-'*
[az-] *'a', 'z' и '-'*
[a\-z] *'a', 'z' и '-'*
[a-z] *все 26 малых латинских букв от 'a' до 'z'*
[\n-\x0D] *#10, #11, #12, #13.*
[\d-t] *цифра, '-' или 't'.*
[]-a] *символ из диапазона ']'..'a'.*

Метасимволы

Метасимволы - это специальные символы, являющиеся важнейшим понятием в регулярных выражениях. Существует несколько групп метасимволов.

Метасимволы - разделители строк

^ *начало строки*
$ *конец строки*
\A *начало текста*
\Z *конец текста*
. *любой символ в строке*

Примеры:
^foobar *находит 'foobar' только если он в начале строки*
foobar$ *находит 'foobar' только если он в конце строки*
^foobar$ *находит 'foobar' только если это единственное слово в строке*
foob.r *находит 'foobar', 'foobbr', 'foob1r' и т.д.*

Метасимвол "^" по умолчанию совпадает только в начале входного текста, а метасимвол "$" - только в конце текста. Внутренние разделители строк, имеющиеся в тексте, не будут совпадать с "^" и "$".
Однако, если Вам необходимо работать с текстом как с многострочным, чтобы "^" совпадал после каждого разделителя строки внутри текста, а "$" - перед каждым разделителем, то Вы можете включить модификатор /m.

Метасимволы \A и \Z аналогичны "^" и "$", но на них не действует модификатор /m, т.е. они всегда совпадают только с началом и концом всего входного текста.

Метасимвол "." по умолчанию совпадает с любым символом, однако, если Вы выключите модификатор /s, то '.' не будет совпадать с разделителями строк.

TRegExpr интерпретирует разделители строк так, как это рекомендовано на www.unicode.org (http://www.unicode.org/unicode/reports/tr18/):

"^" совпадает с началом входного текста, а также, если включен модификатор /m, с точкой непосредственно следующей после \x0D\x0A, \x0A или \x0D (если Вы используете Unicode-версию TRegExpr, то также \x2028 или \x2029 или \x0B или \x0C или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

"$" совпадает с концом входного текста, а также, если включен модификатор /m, с точкой непосредственно предшествующей \x0D\x0A, \x0A или \x0D (если Вы используете Unicode-версию TRegExpr, то также \x2028 или \x2029 или \x0B или \x0C или \x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности \x0D\x0A.

"." совпадает с любым символом, но если выключен модификаторr /s, то "." не совпадает с \x0D\x0A и \x0A и \x0D (если Вы используете Unicode-версию TRegExpr, то не совпадает также с \x2028 и \x2029 и \x0B и \x0C и \x85).

Обратите внимание, что "^.*$" (шаблон для пустой строки) не совпадает с пустой строкой вида \x0D\x0A, но совпадает с \x0A\x0D.

Вы можете перенастроить вышеописанное поведение при обработке многострочных текстов - см. описания свойств LineSeparators и LinePairedSeparator, скажем, Вы можете перенастроиться на использование только Unix-разделителей строк \n или только DOS/Windows-разделителей \r\n или же смешанных разделителей (так и настроено по умолчанию) или вообще определить свои собственные разделители строк!

Метасимволы - стандартные перечни символов

\w *буквенно-цифровой символ или "_"*
\W *не \w*
\d *цифровой символ*
\D *не \d*
\s *любой "пробельный" символ (по умолчанию - [\t\n\r\f])*
\S *не \s*

Стандартные перечни \w, \d и \s можно использовать и внутри **перечней символов**.

Примеры:
foob\dr *находит 'foob1r', "foob6r' и т.д. но не 'foobar', 'foobbr' и т.д.*
foob[\w\s]r *находит 'foobar', 'foob r', 'foobbr' и т.д. но не 'foob1r', 'foob=r' и т.д.*

TRegExpr использует свойства SpaceChars и WordChars для того, чтобы определять стандартные перечни \w, \W, \s, \S, т.е. Вы легко можете переопределить состав этих перечней.

Метасимволы - границы слов

\b *Совпадает на границе слова*
\B *Совпадает не на границе слова*

Граница слова (\b) это точка между двумя символами, один из которых удовлетворяет \w, а другой - \W (в любом порядке), при этом перед началом и после конца строки подразумевается \W.

Метасимволы - повторения

После любого элемента регулярного выражения может следовать очень важный тип метасимвола - **повторитель**. Используя их Вы можете определить число допустимых повторений предшествующего символа, метасимвола или подвыражения.

* *ноль или более раз ("жадный"), то же что {0,}*
\+ *один или более раз ("жадный"), то же что {1,}*
? *ноль или один раз ("жадный"), то же что {0,1}*
{n} *точно n раз ("жадный")*
{n,} *не менее n раз ("жадный")*
{n,m} *не менее n но не более m раз ("жадный")*
*? *ноль или более раз ("не жадный"), то же что {0,}?*
+? *один или более раз ("не жадный"), то же что {1,}?*
?? *ноль или один раз ("не жадный"), то же что {0,1}?*
{n}? *точно n раз ("не жадный")*
{n,}? *не менее n раз ("не жадный")*
{n,m}? *не менее n но не более m раз ("не жадный")*

Т.о. {n,m} задает **минимум** n повторов и **максимум** - m. Повторитель {n} эквивалентен {n,n} и задает точно n повторов. Повторитель {n,} задает минимум n повторов. Теоретически величина параметров n и m не ограничена, но рекомендуется не задавать большие значения, поскольку в некоторых ситуациях это может потребовать существенных затрат времени и ОЗУ при обработке такого повторителя в связи с рекурсивным характером работы.

Если фигурные скобки встречаются в "неправильном" месте, где они не могут быть восприняты как повторитель, то они вопринимаются просто как символы.

Примеры:
foob.*r *находит 'foobar', 'foobalkjdflkj9r' и 'foobr'*
foob.+r *находит 'foobar', 'foobalkjdflkj9r' но не 'foobr'*
foob.?r *находит 'foobar', 'foobbr' и 'foobr' но не 'foobalkj9r'*
fooba{2}r *находит 'foobaar'*
fooba{2,}r *находит 'foobaar', 'foobaaar', 'foobaaaar' и т.д.*
fooba{2,3}r *находит 'foobaar', или 'foobaaar' но не 'foobaaaar'*

Небольшое пояснение по поводу "жадности". "Жадные" варианты повторителей пытаются захватить как можно большую часть входного текста, в то время как "не жадные" - как можно меньшую. Например, 'b+' как и 'b*' примененные к входной строке 'abbbbc' найдут 'bbbb', в то время как 'b+?' найдет только 'b', а 'b*?' - вообще - пустую строку; 'b{2,3}?' найдет 'bb', в то время как 'b{2,3}' найдет 'bbb'.

Вы можете переключить все повторители в выражении в "не жадный" режим, воспользовавшись модификатором /g.

Метасимволы - варианты

Вы можете определить перечень **вариантов**, используя метасимвол "|" для их разделения, например "fee|fie|foe" найдет "fee" или "fie" или "foe", (так же как "f(e|i|o)e"). В качестве первого варианта воспринимается все от предыдущего метасимвола "(" или "[" или от начала выражения до первого метасимвола "|", в качестве последнего - все от последнего "|" до конца выражения или до ближайшего метасимвола ")". Обычно, чтобы не запутаться, набор вариантов всегда заключают в скобки, даже если без этого можно было бы обойтись. Варианты пробуются начиная с первого и попытки завершаются сразу же как удастся подобрать такой при котором совпадет вся последующая часть выражения (подробнее см. Механизм работы). Это означает, что варианты не обязательно обеспечат "жадное" поведение. Например, если применить выражение "foo|foot" ко входной строке "barefoot", то

будет найдено "foo" так это первый вариант который позволил совпасть всему выражению. Обратите внимание, что метасимвол "|" воспринимается как обычный символ внутри перечней символов, например, [fee|fie|foe] означает ровно то же самое что и [feio|].

Примеры:
foo(bar|foo) *находит 'foobar' или 'foofoo'.*

Метасимволы - подвыражения

Метасимволы (...) могут также использоваться для задания подвыражений - по завершении поиска выражения Вы можете обратиться к любому подвыражению используя свойства MatchPos, MatchLen и Match, а также подставлять подвыражения в некий шаблон, используя метод Substitute).

Подвыражения нумеруются слева направо, в порядке появления открывающих скобок. Первое подвыражение имеет номер '1' (выражение в целом - '0', к нему можно обращаться в Substitute как '$0' так и '$&').

Примеры:
(foobar){8,10} *находит строку содержащую 8, 9 или 10 копий 'foobar'*
foob([0-9]|a+)r *находит 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' и т.д.*

Метасимволы - обратные ссылки

Метасимволы от \1 до \9 воспринимаются как обратные ссылки. \<n> совпадает с ранее найденным подвыражением #<n>.

Примеры:
(.)\1+ *находит 'aaaa' и 'cc'.*
(.+)\1+ *также находит 'abab' и '123123'*
(["']?)(\d+)\1 *находит "13" (в дв.кавычках), или '4' (в один.кавычках) или 77 (без кавычек) и т.д.*

Модификаторы

Модификаторы служат для изменения режимов работы TRegExpr.

Вы можете изменять модификаторы несколькими способами.
Любой модификатор может меняться с помощью специальной конструкции (?...) внутри регулярного выражения.
Также, Вы можете присвоить значение соответствующему свойству экземпляра объекта TRegExpr (например, ModifierX для изменения модификатора /x, или ModifierStr для изменения сразу нескольких модификаторов). Значения по умолчанию для новых экземпляров объектов TRegExpr определены в глобальных константах, например RegExprModifierX определяет значение по умолчанию для ModifierX.

i
: Регистро-независимый режим (по умолчанию использует выбранный в ОС язык по умолчанию), (см. также InvertCase)

m
: Воспринимать входной текст как многострочный, при этом метасимволы "^" и "$" будут совпадать не только в начале и конце текста в целом, но и в начале и в конце всех имеющихся в тексте строк (см. также Разделители строк)

s
: Воспринимать входной текст как одну строку. При этом метасимвол "." совпадает с любым символом, если же этот модификатор выключен, то он не совпадает с разделителями строк (см.также Разделители строк).

g

Не стандартный модификатор. Выключая его Вы переключаете все повторители в "не жадный" режим (по умолчанию этот модификатор включен). Т.е. если его отключить, то все '+' работают как '+?', '*' как '*?' и т.д.

x

Позволяет форматировать шаблон чтобы обеспечить более легкую читаемость (см. описание ниже).

r

Не стандартный модификатор. Если включен, то диапазоны вида а-я включают также букву 'ё', А-Я включают 'Ё', а а-Я включает вообще все русские буквы.

Модификатор /x заставляет TRegExpr игнорировать пробелы, табуляции и разделители строк, что позволяет форматировать текст выражения. Кроме того, если встречается символ #, то все последующие символы до конца строки воспринимаются как комментарий, например:

```
(
(abc) # Комментарий 1
  |   # Пробелы внутри выражения также игнорируются
(efg) # Комментарий 2
)
```

Естественно, это означает что, если Вам нужно вставить в выражение пробел, табуляцию или разделитель строки или #, то в расширенном (/x) режиме это можно сделать только предваряя их '/' или используя /xnn (внутри перечней символов все эти символы воспринимаются как обычно)

Расширения Perl

(?imsxr-imsxr)

Позволяет изменять значения модификаторов

Примеры:

(?i)Saint-Petersburg	*находит 'Saint-petersburg' и 'Saint-Petersburg'*
(?i)Saint-(?-i)Petersburg	*находит 'Saint-Petersburg' но не 'Saint-petersburg'*
(?i)(Saint-)?Petersburg	*находит 'Saint-petersburg' и 'saint-petersburg'*
((?i)Saint-)?Petersburg	*находит 'saint-Petersburg', но не 'saint-petersburg'*

(?#text)

Комментарий, просто игнорируется. Обратите внимание, что в комментарии такого вида невозможно поместить символ ")", поскольку он воспринимается как конец комментария.

Внутренний механизм работы

Вам любопытно, как устроен TRegExpr изнутри?
К сожалению, этот раздел пока что в стадии написания, подождите немного!
А пока не забудьте заглянуть в ЧАВО (особенно на вопрос о нежадном режиме).

Интерфейс
Методы и свойства TRegExpr:

class function VersionMajor : integer;
class function VersionMinor : integer;
Возвращают соответственно старшую и младшую часть версии, например, для v. 0.950 VersionMajor = 0 и VersionMinor = 950

property Expression : string;
Собственно регулярное выражение.
Для ускорения работы TRegExpr автоматически выполняет компиляцию выражения во внутреннее представления (его можно посмотреть через Dump).
Однако, делается это только тогда, когда это реально необходимо, т.е. при обращении к методам Exec[Next], Substitute, Dump и т.п. и только в том случае, если после последней [пере]компиляции было изменено свойство Expression или какие-либо другие влияющие на откомпилированное выражение свойства.
При ошибках компиляции вызывается метод Error (по умолчанию он генерирует исключение ERegExpr - см. ниже)

property ModifierStr : string;
Проверка и установка модификаторов с помощью строки в том же формате, что и в конструкции (?ismx-ismx). Т.е., например ModifierStr := 'i-x' включит регистро-независимый режим и выключит режим расширенного синтаксиса, прочие модификаторы останутся без изменений.
Если указать несуществующий модификатор, вызывается Error

property ModifierI : boolean;
Модификатор /i - ("регистро-независимый режим"), инициализируется из RegExprModifierI

property ModifierR : boolean;
Модификатор /r ("русские диапазоны"), инициализируется из RegExprModifierR

property ModifierS : boolean
Модификатор /s - если установлен, то '.' совпадает с любым символом, (если сброшен, то '.' не совпадает с LineSeparators и LinePairedSeparator, , инициализируется из RegExprModifierS

property ModifierG : boolean;
Модификатор /g, отключение приводит к тому, что все операторы работают в "не жадном" (non-greedy) режиме, т.е. когда ModifierG = False, то все '*' работают как '*?', все '+' как '+?' и т.д.., инициализируется из RegExprModifierG

property ModifierM : boolean;
Модификатор /m -воспринимать входной текст как многострочный. Если выключен, то метасимволы `^' и `$' "срабатывают" только в начале и конце входного текста.
Если включен, то эти символы срабатывают также и в начале и в конце каждой строки входного текста., инициализируется из RegExprModifierM

property ModifierX : boolean;
Модификатор /x - ("расширенный синтаксис"), инициализируется из RegExprModifierX

function Exec (const AInputString : string) : boolean;
Выполнить выражение применительно к входной строке AInputString
!!! также, запоминает AInputString в ствойстве InputString
For Delphi 5 and higher available overloaded versions:
function Exec : boolean;
without parameter (uses already assigned to InputString property value)
function Exec (AOffset: integer) : boolean;
is same as ExecPos

function ExecNext : boolean;

поиск следующего совпадения. Фактически:
```
   Exec (AString);
```
означает то же что и
```
  if MatchLen [0] = 0 then ExecPos (MatchPos [0] + 1)
    else ExecPos (MatchPos [0] + MatchLen [0]);
```
но воспринимается гораздо нагляднее!
Выдает исключение если этому вызову не предшествовал **успешный** вызов метода Exec* (Exec, ExecPos, ExecNext). Т.е. необходимо использовать что-то вида
```
 if Exec (InputString) then repeat { обработка} until not ExecNext;
```

function ExecPos (AOffset: integer = 1) : boolean;
выполняет выражение для строки в InputString начиная с позиции AOffset (AOffset=1 - первый символ InputString)

property InputString : string;
текущая входная строка (присвоенная явно или в последнем Exec).
Присвоение этому свойству значений делает неопределенными свойства Match* !

function Substitute (const ATemplate : string) : string;
Возвращает ATemplate в котором все '$&' и '$0' заменены на найденное регулярное выражение, а '$n' заменены на подвыражения #n.
Начиная с версии v.0.929 используется '$' вместо '\' (для расширений типа \n\r и т.п. а также для большей схожести с Perl) и допускаются n > 9.
Если Вам необходим просто символ '$' или '\', предваряйте их '\'.
Например: '1\$ is $2\\rub\\' -> '1$ is <Match[2]>\rub\'
Если Вам необходимо сразу после '$n' поместить цифру, заключайте n в фигурные скобки '{}'.
Например: 'a$12bc' -> 'a<Match[12]>bc', 'a${1}2bc' -> 'a<Match[1]>2bc'.

procedure Split (AInputStr : string; APieces : TStrings);
Режет входную строку AInputStr на помещаемые в APieces куски разделяемые вхождениями выражения. Внимание! Этот метод вызывает методы Exec[Next]

function Replace (AInputStr : RegExprString;
```
  const AReplaceStr : RegExprString;
  AUseSubstitution : boolean = False) : RegExprString;
```
function Replace (AInputStr : RegExprString;
```
  AReplaceFunc : TRegExprReplaceFunction) : RegExprString;
```
function ReplaceEx (AInputStr : RegExprString;
```
  AReplaceFunc : TRegExprReplaceFunction)  : RegExprString;
```
Заменяет в AInputStr все вхождения выражения на AReplaceStr
Если AUseSubstitution = true, то AReplaceStr будет восприниматься как шаблон для метода Substitution.
Например:
```
  Expression := '({-i}block|var)\s*\(\s*([^ ]*)\s*\)\s*';
  Replace ('BLOCK( test1)', 'def "$1" value "$2"', True);
   вернет:  def 'BLOCK' value 'test1'
  Replace ('BLOCK( test1)', 'def "$1" value "$2"', False)
   вернет:  def "$1" value "$2"
```
Внимание! Этот метод вызывает методы Exec[Next]
Вариант с параметром-функцией и ReplaceEx отличаются тем, что передается не строка, а ссылка на функцию, которая в динамике формирует строку для замены, что позволяет реализовать сложные варианты замен.

property SubExprMatchCount : integer; // ReadOnly
Число подвыражений, найденных в последнем Exec*. если найдено только само выражение в целом, то SubExprMatchCount=0, если и само выражение не найдено (Exec* вернул false) то SubExprMatchCount=-1. Обратите внимание, что часть подвыражений может быть не найдено и для них MathPos=MatchLen=-1 и Match=''.

Например, для 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') - возвращает False: SubExprMatchCount=-1

property MatchPos [Idx : integer] : integer; // ReadOnly
Позиция начала подвыражения #Idx во входной строке
Первое подвыражение имеет Idx=1, последнее - MatchCount, выражение в целом Idx=0.
Возвращает -1 если нет такого подвыражения или если оно не найдено во входной строке.

property MatchLen [Idx : integer] : integer; // ReadOnly
Длина подвыражения #Idx во входной строке
Первое подвыражение имеет Idx=1, последнее - MatchCount, выражение в целом Idx=0.
Возвращает -1 если нет такого подвыражения или если оно не найдено во входной строке.

property Match [Idx : integer] : string; // ReadOnly
== copy (InputString, MatchPos [Idx], MatchLen [Idx])
Возвращает '' если нет такого подвыражения или если оно не найдено во входной строке.

function LastError : integer;
Код последней ошибки, 0 если нет ошибки (бессмысленно использовать эту функцию, если Вы не изменяете реализацию Error, поскольку текущая реализация генерирует исключительную ситуацию).
Вызов этой функции очищает внутреннюю переменную и повторный вызов всегда вернет 0.

function ErrorMsg (AErrorID : integer) : string; virtual;
Возвращает текст сообщения об ошибке с кодом AErrorID.

property CompilerErrorPos : integer; // ReadOnly
Возвращает позицию, в которой случилась последняя ошибка компиляции (упрощает отладку выражений)

property SpaceChars : RegExprString
Содержит символы, трактуемые как \s (инициализируется из глобальной константы RegExprSpaceChars)

property WordChars : RegExprString;
Содержит символы, трактуемые как \w (инициализируется из глобальной константы RegExprWordChars)

property LineSeparators : RegExprString
Разделители строк (например, \n для Unix) (инициализируется из глобальной константы RegExprLineSeparators)
см.подробнее о разделителях строк

property LinePairedSeparator : RegExprString
Сдвоенный разделитель строк (как, \r\n для DOS и Windows) (инициализируется из глобальной константы RegExprLinePairedSeparator)
см.подробнее о разделителях строк

Например, если Вам необходимо отслеживать только Unix-разделители строк, присвойте LineSeparators := #$a (символ новой строки) и LinePairedSeparator := '' (пустую строку), если необходимо воспринимать как разделители строк только \x0D\x0A но не отдельные \x0D или \x0A, присвойте LineSeparators := '' (пустую строку) и LinePairedSeparator := #$d#$a.

По умолчанию используется "смешанный вариант" (им инициализированы константы RegExprLine[Paired]Separator[s]): LineSeparators := #$d#$a; LinePairedSeparator := #$d#$a подробно описанный в описании синтаксиса.

class function InvertCaseFunction (const Ch : REChar) : REChar;
Преобразует символ Ch в верхний регистр если это символ нижнего регистра и в нижний - если верхнего (используются текущие установки операционной системы)

property InvertCase : TRegExprInvertCaseFunction;
Позволяет определить свою собственную реализацию регистро-независимого режима работы TRegExpr. Инициализируется из глобальной константы RegExprInvertCaseFunction (по умолчанию она указывает на InvertCaseFunction)

procedure Compile;
Вызывает принудительную [пере]компиляцию регулярного выражения.
Может быть полезной, например, для проверки корректности всех свойств при создании визуальных редакторов рег.выражений и т.п.

function Dump : string;
возвращает внутренний формат в который было откомпилировано выражение. Предназначено для особо любознательных ;)

Глобальные константы

EscChar = '\'; // 'Escape'-char ('\' in common r.e.) used for escaping metachars (\w, \d etc).
// it's may be usefull to redefine it if You are using C++ Builder - to avoide ugly constructions
// like '\\w+\\\\\\w+\\.\\w+' - just define EscChar='/' and use '/w+\/w+/./w+'

Значения по умолчанию для модификаторов:
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 // Значение по умолчанию для SpaceChars
= ' '#$9#$A#$D#$C;

RegExprWordChars : RegExprString // Значение по умолчанию для WordChars
= '0123456789'
+ 'abcdefghijklmnopqrstuvwxyz'
+ 'ABCDEFGHIJKLMNOPQRSTUVWXYZ_';

RegExprLineSeparators : RegExprString // Значение по умолчанию для LineSeparators
= #$d#$a{$IFDEF UniCode}#$b#$c#$2028#$2029#$85{$ENDIF};
RegExprLinePairedSeparator : RegExprString // Значение по умолчанию для LinePairedSeparator
= #$d#$a;

RegExprInvertCaseFunction : TRegExprInvertCaseFunction // Значение по умолчанию для InvertCase
= TRegExpr.InvertCaseFunction;

Глобальные функции

function ExecRegExpr (const ARegExpr, AInputStr : string) : boolean;
true если строка AInputString совпадает с выражением ARegExpr
! При ошибках в ARegExpr будет генерировать exception

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

Режет AInputStr на помещаемые в APieces куски по вхождениям выражения ARegExpr (например, разбиение строки на отдельные поля, разделенные некой последовательностью символов)

function ReplaceRegExpr (const ARegExpr, AInputStr, AReplaceStr : string;
AUseSubstitution : boolean = False) : string;
Возвращает AInputStr в которой все вхождения выражения ARegExpr заменены на AReplaceStr. Если AUseSubstitution = true, то AReplaceStr будет восприниматься как шаблон для Substitution:

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

function QuoteRegExprMetaChars (const AStr : string) : string;
Заменяет все метасимволы во входной строке так, чтобы ее можно было безопасно использовать внутри регулярного выражения.
Например 'abc$cd.(' преобразуется в 'abc\$cd\.\('
Эта функция полезна для автоматического синтеза регулярного выражения на основании пользовательских данных

function RegExprSubExpressions (const ARegExpr : string;
ASubExprs : TStrings; AExtendedSyntax : boolean = False) : integer;
Создает список найденные в рег.выражении ARegExpr подвыражений
Каждому подвыражению соответствует элемент в ASubExps, где
String - исходный текст подвыражения (без обрамляющих '()')
мл.слово Object - начальная позиция подвыражения в ARegExpr, включая обрамляющий '(' если он есть!
ст.слово Object - длина подвыражения, включая '(' и ')' если они есть!
AExtendedSyntax - должен быть True если модификатор /x будет включен при использовании данного регулярного выражения.
Эта функция полезна для написания визуальных редакторов рег.выражений и т.п. Пример использования есть в TestRExp.dpr

Результат	**Комментарий**
0	Успех. Все скобки сбалансированы;
-1	Не хватает как минимум одной закрывающей скобки ')';
-(n+1)	В позиции n обнаружен незакрытый '[';
n	В позиции n обнаружена закрывающая ')' для которой нет открывающей '('.

Возвращает 0 если все скобки сбалансированы, или -1 если недостаточно закрывающих скобок ')', или n если в позиции n встречена закрывающая скобка ')' которой не соответствует ни одна открывающая '('.
Естественно, если Result <> 0, то ASubExprs может быть некорректен.

Тип генерируемого при ошибках Exception

Обработчик ошибок TRegExpr по умолчанию (Вы вполне можете его перекрыть и изменить его поведение) генерирует exception:

```
ERegExpr = class (Exception)
  public
   ErrorCode : integer; // Код ошибки. Ошибки компиляции выражения меньше 1000, что позволяет их отличить от ошибок выполнения выражения.
   CompilerErrorPos : integer; // Позиция в выражении где произошла последняя ошибка компиляции выражения
 end;
```

Как использовать Unicode

TRegExpr теперь поддерживает работу с UniCode.

Обратите внимание, что этот режим практически неоптимизирован и работает **чрезвычайно медленно** (по сравнению со стандартным режимом).
Используйте его только если Вам действительно не обойтись без UniCode (а лучше примите участие в разработке TRegExpr и оптимизируйте UniCode-режим).

Чтобы переключить TRegExpr на работу с unicode уберите '.' из {.$DEFINE UniCode} в файле regexpr.pas.
Все строки после этого будут восприниматься как WideString.

R.E.Studio

Приложение для изучения регулярных выражений, а также для создания своих выражений, их оптимизации и отладки. Если в вашем дистрибутивной архиве TRegExpr нет этого приложения в подкаталоге REStudio, то Вы можете его скачать с домашней страницы TRegExpr.

Программа позволяет работать с подвыражение (самого регулярного выражения, так и текущего результата выполнения выражения на входных данных), искать синтаксические ошибки, оценивать время выполнения разных вариантов выражения, использовать функции Substitude, Replace и Split и т.п.

R.E.Studio поддерживает репозиторий регулярных выражений, который изначально заполнен множеством полезных на практике выражений, а также примеров для изучения регулярных выражений. Вы можете сохранять в этом репозитории как свои собственные выражения, так и тестовые примеры и комментарии для них.

Используя R.E.Studio Вы сможете изучить работу TRegExpr "изнутри", просматривая сгенерированный при компиляции регулярного выражения псевдо-код, что дает возможность лучше понять нюансы работы и лучше оптимизировать выражения.

ЧАсто задаваемые ВОпросы

В.

Ничего не работает! Какой-то Access violation выдает!

О.

Пожалуйста. Проверьте. Что. Вы. Создали. Эземпляр. Объекта.
После того как вы описали что-то вроде var r : TRegExpr; нужно еще и сам объект создать r := TRegExpr.Create. Рекомендую вообще почитать что-нибудь по программированию на Delphi language, не относящиеся к TRegExpr вопросы эффективнее задавать в ФИДО, на delphi.mastak.ru или в Королевстве Delphi.

В.

Как использовать TRegExpr в Borland C++ Builder?

Я не могу это сделать, потому что нет заголовочных файлов (.h или .hpp).

О.

- Добавьте RegExpr.pas в Ваш bcb-проект
- Откомпилируйте проект. В результате автоматически будет создан hpp-файл (сообщения об ошибках можно проигнорировать)
- Теперь Вы можете использовать класс TRegExpr в своем проекте. Не забывайте добавлять #include "RegExpr.hpp" в соответствующие cpp-файлы
- Не забудьте заменить в регулярных выражениях все символы '\' на '\\'.

В.

А почему почти все выражения (даже приведенные здесь как примеры) неверно работают в Borland C++ Builder?

О.

А Вы еще раз перечитайте ответ на пред.вопрос ;) Символ '\' воспринимается в C++ как "эскейп"-символ, поэтому его самого нужно всегда "эскейпить". Когда Вам надоест работать с кошмаром вида '\\w+\\\\\\w+\\.\\w+', Вы можете переопределить константу EscChar (RegExpr.pas) , например, если EscChar='/' - то приведенное выше выражение нужно записать как '/w+\/w+/./w+' - немножко непривычно\нестандартно, но намного обозримее\читаемее..

В.

Почему TRegExpr возвращает более одной строки?

Например, выражение <font .*> возвращает <font, а затем - весь оставшийся файл включая завершающий </html>...

О.

Для обеспечения обратной совместимости, модификатор /s по умолчанию включен.
Если Вы его выключите, то мета-символ '.' перестанет совпадать с разделителями строк - и Вы получите ожидаемый Вами результат.
Кстати, данное конкретное выражение было бы эффективнее переписать как '<font ([^\n>]*)>', (в Match[1] Вы получите URL).

В.

Почему TRegExpr возвращает больше чем я ожидал?

Например, выражение '<p>(.+)</p>' примененное к строке '<p>a</p><p>b</p>' возвращает 'a</p><p>b' а не 'a' как я ожидал.

О.

По умолчанию, все операторы работают в "жадном" ('greedy') режиме и пытаются захватить как можно большую часть входной строки.
Если Вам необходим "не жадный" ('non-greedy') режим, то Вы можете использовать либо специальные "не жадные" варианты операторов ('+?' и т.п.) либо вообще переключить модификатор "жадности" ModifierG. Прим.: эта возможность появилась начиная с версии 0.940).

В.

Как анализировать HTML с помощью TRegExpr

О.

Мне очень часто задают этот вопрос! Должен вас огорчить, корректный ответ - "никак".
Конечно, вполне возможно (и очень удобно) использовать TRegExpr для извлечения каких-то специфичных фрагментов HTML, я сам часто этим пользуюсь, но если Вам нужен полноценный синтаксический разбор, то возьмите подходящий инструмент - синтаксический

анализатор, а не регулярные выражения!
Мне не хочется занимать здесь место объяснениями почему это так - просто поверьте или почитайте например книжку Tom Christiansen и Nathan Torkington 'Perl Cookbook' (в русском переводе вышла в серии "Библиотека программиста" и называется просто "Perl"). Если в двух словах - мало того что отдельные случаи даже теоретически невозможно обработать с помощью регулярных выражений, так еще и не стоит забывать, что регулярные выражения это достаточно ресурсоемкий механизм выполняющий оптимизационный поиск, в то время как синтаксический анализ обычно линеен и работает гораздо быстрее. Используйте подходящие инструменты для каждого вида работ!

В.
Как мне получить все вхождения регулярного выражения а не только первое?
О.
Очень просто - напишите цикл, который будет вызывать метод ExecNext и тем самым переберет все вхождения.
Пример Вы можете посмотреть в реализации метода TRegExpr.Replace или в исходных текстах модуля HyperLinksDecorator.pas

В.
Я хочу проверить вводимые пользователем строки. Но почему-то TRegExpr иногда возвращает True для явно ошибочных строк.
О.
Возможно, Вы просто забыли о том, что регулярные выражения ИЩУТ заданный Вами шаблон во входной строке. Поэтому, если например задать шаблон типа '\d{4,4}' то он "сработает" не только для строк, состоящих из четырех цифр, но и для таких строк как '12345' или даже 'что угодно 1234 и опять что угодно'. Если Вам необходимо убедиться что во входной строке ТОЛЬКО искомый шаблон, не забудьте обрамлять выражение метасимволами начала и конца строки: '^\d{4,4}$'.

В.
Почему "не жадные" повторители иногда ведут себя "жадно"?
Например, выражение 'a+?,b+?', примененное к строке 'aaa,bbb' находит 'aaa,b', в то время как я ожидал что будет найдено 'a,b', ведь первый повторитель написан в "нежадной" форме!
О.
Это особенность используемой TRegExpr (а также Perl-ом и многими Unix-скими регулярными выражениями) математики - выполняется только "простая" оптимизация при поиске и не делается попыток найти наиболее **оптимальный вариант**. В некоторых ситуациях это неудобно, однако, в основном, это можно воспринимать как преимущество а не недостаток, поскольку это обеспечивает большую скорость работы и предсказуемость результатов. Основное правило - вначале TRegExpr пытается найти соответствие выражению начиная с текущей позиции во входном тексте. И только если это абсолютно невозможно, передвигается на символ вперед и повторяет все сначала. Поэтому, для выражения 'a,b+?' будет найдено 'a,b', а вот в случае 'a+?,b+?' выражение **не рекомендует** (с помощью "нежадной" формы повторителя) но и **не запрещает** вовсе совпадение более одного символа 'a', поэтому TRegExpr, пытаясь найти соответствие для текущей позиции, захватывает все больше этих символов, пока не находит соответствие. На этом он завершает поиск, не пытаясь выяснить, нельзя ли, сдвинувшись вперед, найти лучший вариант. Впрочем, строго говоря, нет способа понять - что значит "лучший"
Вы можете также найти более детальные пояснения в разделе 'Синтаксис'.

Автор

Andrey V. Sorokin,
Saint Petersburg, Russia
anso@mail.ru, anso@paycash.ru
http://RegExpStudio.com (http://anso.da.ru)

Пожалуйста, прежде чем присылать мне сообщения об ошибках или какие-либо вопросы, касающиеся использования библиотеки, вначале скачайте последнюю версию с моей домашней странички и прочитайте ЧАВО!

Эта библиотека основана на библиотеке Henry Spencer.
Она была переведена с Си на Object Pascal, существенно дополнена синтаксическими конструкциями из других реализаций регулярных выражений и снабжена объектным интерфейсом.
Многие доработки были предложены (а иногда и реализованы) пользователями TRegExpr (см. благодарности ниже).

Благодарности

- Guido Muehlwitz - нашел и уничтожил очень серьезную ошибку обработки строк
- Stephan Klimek - тестирование в CPPB, предложил многие доработки
- Steve Mudford - реализовал параметр Offset
- Martin Baur (www.mindpower.com) - перевод этого справочного файла на немецкий, полезные предложения
- Yury Finkel - реализация поддержки UniCode, нашел и исправил неприятный баг
- Ralf Junker - весьма педантично просмотрел код и реализовал ряд блестящих оптимизаций.
- Simeon Lilov - перевод этого справочного файла на болгарский язык
- Filip Jirsбk and Matthew Winter (wintermi@yahoo.com) - помощь в реализации non-greedy режима
- Kit Eason - предоставил массу примеров выражений для описания синтаксиса
- Juergen Schroth - поиск ошибок и полезные предложения
- Martin Ledoux - перевод этого справочного файла на французский язык
- Diego Calp (mail@diegocalp.com), Аргентина - перевод этого справочного файла на испанский язык

И многим другим за неоценимую помощь в охоте на баги !

Автор с удовольствием примет помощь в переводе этого руководства на другие языки.

Демо-примеры

Здесь перечислены демо-проекты, иллюстрирующие основные приемы использования TRegExpr.

Обратите внимание, что существуют локализованные варианты (с комментариями на разных языках). Если у Вас в каталоге Demos примеры с комментариями только на английском, то русскоязычные
Вы можете найти в составе полного русского дистрибутива TRegExpr или в архиве с русской документацией (при распаковке архива в каталог TRegExpr, русифицированные примеры записываются поверх английских, замещая их).

Demos\TRegExprRoutines
Самый простой способ использовать TRegExpr, пояснения см.в исходных текстах.

Demos\TRegExprClass
Более эффективный способ использовать TRegExpr, пояснения см.в исходных текстах.

Demos\Text2HTML
см. описание

Если Вы не знакомы с регулярными выражениями, изучите раздел Синтаксис. Кроме того, для понимания примеров нужно просмотреть описание интерфейса TRegExpr.

Не забудьте также прочитать мои статьи на Delphi3000.com (только на английском) и Королевстве Delphi, и проголосовать там за эти статьи ;).

Примечание
Обратите внимание, что если Вы используете Delphi версии 3 и ниже, то при открытии этого проекта Вы получите серию предупреждений о **несуществующих свойствах**. Это не нарушит работу программы (речь идет о расширениях, появившихся в Delphi 4 и позволяющих более интеллектуально изменять размеры и положение компонентов при изменении размеров содержащей их формы).

Text2Http

Простейшая утилита для конвертации текста в HTML-код.
Использует модуль HyperLinksDecorator

Написана исключительно как пример использования TRegExpr.

Модуль оформления гипер-ссылок

DecorateURLs DecorateEMails

Содержит функции для поиска URL в обычном тексте и оформления их как HTML-ссылки (используется в программе преобразования текста, в HTML-код, Text2Html).

Например, подстрока 'www.RegExpStudio.com' будет заменена на 'www.RegExpStudio.com', а подстрока 'anso@mail.ru' заменится на 'anso@mail.ru'.

function DecorateURLs

Оформляет ссылки найденные как по сигнатуре 'http://...' или 'ftp://..' так и ссылки в которых протокол не указан, но они начинаются с 'www.' Прим. если нужно также оформить как ссылки e-mail адреса, воспользуйтесь функцией DecorateEMails.

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

Описание

Возвращает текст AText с оформленными гипер-ссылками.

AFlags определяет, какая часть гипер-ссылки будет помещена в видимую часть. Например, если указать [durlAddr] то гипер-ссылка 'www.RegExpStudio.com/contacts.htm' будет оформлена как 'www.RegExpStudio.com'.

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

Описание

Возможные значения:

Значение	Описание
durlProto	Протокол ('ftp://' или 'http://')
durlAddr	TCP адрес или доменное имя сервера (например, 'anso.da.ru')
durlPort	Номер порта, если указан (например, ':8080')
durlPath	Путь к файлу (например, 'index.htm')
durlBMark	Закладка (например, '#mark')
durlParam	URL-параметры (например, '?ID=2&User=13')

function DecorateEMails

Заменяет все обнаруженные адреса e-mails на гипер-ссылки вида 'ADDR'. Например, адрес 'anso@mail.ru' будет заменен на 'anso@mail.ru'.

function DecorateEMails (const AText : string) : string;

Описание

Возвращает текст AText с оформленными как гипер-ссылки адресами e-mails