TRegExpr Documentation

Release 0.952

Andrey Sorokin

14.12.2019

Inhaltsverzeichnis

1 Rezensionen **3**

2 Schnellstart **5**

3 Feedback **7**

4 Quellcode **9**

5 Dokumentation **11**
5.1 Reguläre Ausdrücke (RegEx) 11
5.2 TRegExpr 20
5.3 FAQ 28
5.4 Demos 31

6 Übersetzungen **33**

7 Dankbarkeit **35**

	English		Deutsch		Français	Español

Die TRegExpr-Bibliothek implementiert reguläre Ausdrücke.

Reguläre Ausdrücke sind ein benutzerfreundliches und leistungsfähiges Werkzeug für anspruchsvollere Such- und Ersetzungsaufgaben sowie für vorlagenbasiertes Überprüfen von Text.

Besonders nützlich sind sie zum Prüfen von Benutzereingaben in Eingabemasken, zum Validieren von E-Mail-Adressen usw.

Auch können Sie damit Telefonnummern, Postleitzahlen usw. aus Webseiten oder Dokumenten extrahieren, nach komplexen Mustern in Protokolldateien suchen und was Sie sich sonst noch in der Art vorstellen können. Die Regeln (Vorlagen) lassen sich ändern, ohne die gesamte Anwendung neu kompilieren zu müssen.

TRegExpr ist 100% in Pascal implementiert. Es ist Bestandteil von Lazarus (Free Pascal) , aber auch als separate Bibliothek nutzbar und kann auch mit Delphi 2-7 oder dem Borland C ++ Builder 3-6 kompiliert werden.

KAPITEL 1

Rezensionen

Machen Sie sich ein Bild von der‘Resonanz <https://sorokin.engineer/posts/en/regexpstudio_site_is_lunched.html>‘_ bei den Anwendern.

KAPITEL 2

Schnellstart

Um die Bibliothek zu verwenden, fügen Sie einfach den Quelltext Ihrem Projekt hinzu und verwenden Sie die Klasse TRegExpr.

In den FAQ können Sie aus den Problemen anderer Nutzer lernen.

Die benutzerfreundliche Windows-Anwendung REStudio hilft Ihnen dabei, reguläre Ausdrücke zu erlernen und zu debuggen.

KAPITEL 3

Feedback

Wenn Sie auf ein Problem stoßen, erstellen Sie bitte einen Fehlerbericht.

KAPITEL 4

Quellcode

Ausschließlich Object Pascal.

- Originalversion
- FreePascal-Fork (GitHub-Spiegel der Subversion)

KAPITEL 5

Dokumentation

	English		Deutsch		Français	Español

5.1 Reguläre Ausdrücke (RegEx)

5.1.1 Einführung

Reguläre Ausdrücke sind eine praktische Möglichkeit, Textmuster anzugeben.

Mit regulären Ausdrücken können Sie Benutzereingaben validieren, nach Mustern wie E-Mail-Adressen oder Telefonnummern auf Webseiten oder in Dokumenten suchen und so weiter.

Nachfolgend auf einer einzigen Seite ein Spickzettel zum Thema reguläre Ausdrücke.

5.1.2 Zeichen

Einfache Übereinstimmungen

Jedes einzelne Zeichen passt zu sich selbst.

Eine Reihe von Zeichen stimmt mit dieser Reihe von Zeichen in der Eingabezeichenfolge überein.

RegEx	Streichhölzer
`foobar`	`foobar`

Nicht druckbare Zeichen (Escape-Codes)

Um nicht druckbare Zeichen in regulären Ausdrücken darzustellen, verwenden Sie `\x..`:

RegEx	Streichhölzer
‘‘ xnn‘‘	Zeichen mit Hex-Code ‘‘ nn‘‘
‘‘ x {nnnn} ‘‘	Zeichen mit Hex-Code `nnnn` (ein Byte für Klartext und zwei Byte für Unicode)
‘‘ foo x20bar‘‘	`foo bar` (Notizfeld in der Mitte)

Es gibt eine Reihe von vordefinierten "Escape-Codes" für nicht druckbare Zeichen, genau wie in der Sprache "C":

RegEx	Streichhölzer
‘‘ t‘‘	Registerkarte (HT / TAB), identisch mit `\x09`
`\n`	Newline (NL), wie ‘‘ x0a‘‘
‘‘ r‘‘	car.return (CR), das gleiche wie ‘‘ x0d‘‘
‘‘ f‘‘	Formularvorschub (FF), wie ‘‘ x0c‘‘
‘‘ a‘‘	Alarm (BEL), wie ‘‘ x07‘‘
‘‘ e‘‘	Escape (ESC), genauso wie ‘‘ x1b‘‘
`\cx`	Control Escape Sequence (`Ctrl-x`) For example, `\ci` matches the target sequence `\x09`, because `ctrl-i` has the value `0x09`

Flucht

Wenn Sie das Zeichen \ für sich und nicht als Teil von `escape code` verwenden möchten, fügen Sie einfach \ wie folgt ein: `\\`.

Tatsächlich können Sie jedem Zeichen, das in regulären Ausdrücken eine besondere Bedeutung hat, ein \ vorangestellt (oder `escape`).

RegEx	Streichhölzer
`\^FooBarPtr`	‘‘ ^ FooBarPtr‘‘ Dies ist ^ und nicht *Zeilenanfang*
‘‘ [a] ‘‘	‘‘ [a] ‘‘ Dies ist keine *Zeichenklasse*

5.1.3 Charakterklassen

Benutzerzeichenklassen

Die Zeichenklasse ist eine Liste von Zeichen in `[]`. Die Klasse entspricht einem beliebigen ** Zeichen in dieser Klasse.

RegEx	Streichhölzer
‘‘ foob [aeiou] r‘‘	`foobar`, `foober` usw., aber nicht `foobbr`, `foobcr` etc

Sie können die Klasse „umkehren“ - wenn das erste Zeichen nach dem `[` ‘‘ ‘‘ ‘‘ ist, dann entspricht die Klasse einem beliebigen Zeichen **, aber ** Zeichen, die in der Klasse aufgeführt sind.

RegEx	Streichhölzer
‘‘ foob [^ aeiou] r‘‘	`foobbr`, `foobcr` usw., aber nicht `foobar`, `foober` etc

Innerhalb einer Liste wird das Zeichen `-` verwendet, um einen Bereich anzugeben, so dass `a-z` alle Zeichen zwischen `a` und `z` einschließlich darstellt.

Wenn Sie möchten, dass `-` selbst Mitglied einer Klasse ist, setzen Sie es an den Anfang oder das Ende der Liste oder *escape* mit einem Backslash.

Wenn Sie `]` oder `[` möchten, können Sie es an den Anfang der Liste setzen oder es mit einem Backslash entkommen.

RegEx	Streichhölzer
`[-az]`	`a`, `z` und `-`
`[az-]`	`a`, `z` und `-`
`‘‘ [a -z] ‘‘`	`a`, `z` und `-`
`‘‘ [az] ‘‘`	Zeichen von `a` bis `z`
`‘‘ [n- x0D] ‘‘`	Zeichen von `#10` bis `#13`

Vordefinierte Zeichenklassen

Es gibt eine Reihe vordefinierter Zeichenklassen, die reguläre Ausdrücke kompakter halten.

RegEx	Streichhölzer
`\w`	ein alphanumerisches Zeichen (einschließlich _)
`\W`	eine nichtalphanumerische
`\d`	ein numerisches Zeichen (dasselbe wie *[0123456789] ‘*)
`‘‘ D‘‘`	eine nicht numerische
`s`	ein beliebiges Leerzeichen (dasselbe wie *[t n r f] ‘*)
`‘‘ S‘‘`	ein Nichtraum
`\h`	horizontal whitespace. the tab and all characters in the „space separator“ Unicode category.
`\H`	not a horizontal whitespace
`\v`	vertical whitespace. all characters treated as line breaks in the Unicode standard.
`\V`	not a vertical whitespace

Sie können `\w`, `\d` und `\s` innerhalb von]Benutzerzeichenklassen verwenden <User Character *Classes*>‘_.

RegEx	Streichhölzer
`‘‘ foob dr‘‘`	`foob1r`, `foob6r` usw., aber nicht `foobar`, `foobbr` usw.
`‘‘ foob [w s] r‘‘`	*foobar*‘,‘ foob r‘‘, *foobbr*‘ und so weiter, aber nicht‘ foob1r‘‘, *foob = r*‘ usw.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Eigenschaften SpaceChars und‘ WordChars <tregexpr.html#wordchars>‘_ Zeichenklassen definieren `\w`, `\W`,‘ *s*‘,‘ *S’*.

So können Sie diese Klassen neu definieren.

5.1.4 Grenzen

Zeilengrenzen

RegEx	Streichhölzer
^	Zeilenanfang
$	Ende der Linie
‘‘ A‘‘	Beginn des Textes
‘‘ Z‘‘	Ende des Textes
.	ein beliebiges Zeichen in einer Reihe
‘‘ ^ foobar‘‘	`foobar` nur, wenn es am Anfang der Zeile steht
‘‘ foobar $ ‘‘	`foobar` nur wenn es am Zeilenende ist
‘‘ ^ foobar $ ‘‘	`foobar` nur, wenn es die einzige Saite in der Reihe ist
‘‘ foob.r‘‘	`foobar`, `foobbr`, `foob1r` usw.

^ Metazeichen entsprechen standardmäßig dem Anfang der Eingabezeichenfolge. $ - das Ende.

Möglicherweise möchten Sie jedoch eine Zeichenfolge als mehrzeiligen Text behandeln, so dass ^ nach einem beliebigen Trennzeichen innerhalb der Zeichenfolge und ’‘‘ ‘‘ vor jedem Zeilentrennzeichen übereinstimmt. Sie können dies tun, indem Sie *modifier / m* wechseln.

Beachten Sie, dass sich in der Sequenz `\x0D\x0A` keine leere Zeile befindet.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Wenn Sie die Unicode-Version verwenden <tregexpr.html#unicode>‘__, dann‘ ^ */‘‘$‘* stimmt auch mit‘ *x2028‘*,‘ x2029‘‘, *x0B‘*,‘ *x0C‘* oder‘ *x85* überein ‘.

‘‘ A’‘ und ‘‘ Z‘‘ sind genau wie ^ und ‘‘ ‘‘ *, außer dass sie nicht mehrmals übereinstimmen, wenn der* Modifier / m <#m>‘_ wird eingesetzt.

Das ‘‘ .‘‘-Metazeichen entspricht standardmäßig jedem Zeichen, aber wenn Sie `Off` mit dem *Modifier / s* ausschalten, stimmt ‘‘ .‘‘ nicht mit den Trennzeichen in der Zeichenfolge überein.

Beachten Sie, dass ^ . ∗$ Nicht mit einer Zeichenfolge zwischen ‘‘ x0D x0A‘‘ übereinstimmt, da dies ein unzerbrechlicher Trenner ist. Er stimmt jedoch mit der leeren Zeichenfolge in der Sequenz `\x0A\x0D` überein, da dies einfach eine falsche Reihenfolge ist, um als Zeilentrennzeichen behandelt zu werden.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Die mehrzeilige Verarbeitung kann mit den Eigenschaften LineSeparators abgestimmt werden <tregexpr.html#lineseparators>‘__ und‘ LinePairedSeparator <tregexpr.html#linepairedseparator>‘_.

Sie können also die Unix-Stil-Trennzeichen `\n` oder den DOS / Windows-Stil `\r\n` verwenden oder sie zusammenmischen (wie oben beschrieben, Standardverhalten).

Wenn Sie eine mathematisch korrekte Beschreibung bevorzugen, können Sie sie unter www.unicode.org finden <http://www.unicode.org/unicode/reports/tr18/>__.

Wortgrenzen

RegEx	Streichhölzer
`\b`	eine Wortgrenze
`\B`	eine Nicht-Wortgrenze

Eine Wortgrenze "b" ist eine Stelle zwischen zwei Zeichen, die auf einer Seite ein "w" und auf der anderen Seite ein "W" (in beliebiger Reihenfolge) hat.

5.1.5 Quantifizierung

Quantor

Auf jedes Element eines regulären Ausdrucks kann ein Quantifizierer folgen. Quantifier gibt die Anzahl der Wiederholungen des Elements an.

RegEx	Streichhölzer
‘‘ {n} ‘‘	genau "n" mal
‘‘ {n,} ‘‘	mindestens "n" mal
‘‘ {n, m} ‘‘	zumindest "n" aber nicht mehr als "m" mal
`*`	Null oder mehr, ähnlich wie `{0,}`
`+`	eine oder mehrere, ähnlich wie `{1,}`
`?`	Null oder Eins, ähnlich wie `{0,1}`

Geben Sie also in geschweiften Klammern ‘‘ {n, m} ‘‘ die Mindestanzahl an Übereinstimmungen mit `n` und die maximale Anzahl von *m‘* an.

Das ‘‘ {n} ‘‘ entspricht ‘‘ {n, n} ‘‘ und stimmt genau mit `n`-Zeiten überein.

Das ‘‘ {n,} ‘‘ entspricht mehr als `n`.

Es gibt keine Begrenzung für die Größe von `n` oder `m`.

Wenn eine geschweifte Klammer in einem anderen Kontext auftritt, wird sie als normales Zeichen behandelt.

RegEx	Streichhölzer
‘‘ foob. * r‘‘	`foobar`, `foobalkjdflkj9r` und `foobr`
‘‘ foob. + r‘‘	`foobar`, `foobalkjdflkj9r`` aber nicht `foobr`
"foob." r ""	`foobar`, `foobbr` und `foobr` aber nicht `foobalkj9r`
‘‘ fooba {2} r‘‘	`foobaar`
‘‘ fooba {2,} r‘‘	`foobaar`, `foobaaar`, `foobaaaar` usw.
‘‘ fooba {2,3} r‘‘	`foobaar` oder `foobaaar` aber nicht `foobaaaar`
‘‘ (foobar) {8,10} ‘‘	`8`, `9` oder `10`-Instanzen von `foobar` (`()` ist *Subexpression*)

Gier

Quantifiers in `greedy` Modus so viele wie möglich stattfindet, in `nicht-greedy` Modus - so wenig wie möglich.

Standardmäßig sind alle Quantifizierer ‘‘ gierig‘‘. Verwenden Sie `?`, Um einen Quantifizierer ‘‘ nicht-gierig‘‘ zu machen.

Für den String `abbbbc`:

RegEx	Streichhölzer
‘‘ b + ‘‘	`bbbb`
`b+?`	"b"
`b*?`	leerer String
‘‘ b {2,3}? ‘‘	"bb"
‘‘ b {2,3} ‘‘	‘‘ bbb‘‘

Sie können alle Quantifizierer in den ‘‘ non-greedy‘‘-Modus umschalten (*modifier / g*, unten verwenden wir *inline modifier change*).

RegEx	Streichhölzer
‘‘ (? -g) b + ‘‘	"b"

5.1.6 Die Wahl

Ausdrücke in der Auswahl werden durch | getrennt.

‘‘ Fee | fie | foe‘‘ passt also zu einem beliebigen von `fee`, `fie` oder `foe` in der Zielzeichenfolge (ebenso wie ‘‘ f (e | i | o) e ‘‘).

Der erste Ausdruck enthält alles vom letzten Musterbegrenzer (‘‘ (‘‘ ‘‘ ‘‘) `oder vom Anfang des Musters) bis zum ersten `` | `, und der letzte Ausdruck enthält alles vom letzten`` | zum nächsten Musterbegrenzer.

Klingt etwas kompliziert, daher ist es üblich, die Auswahl in Klammern zu setzen, um die Verwirrung über den Anfang und das Ende zu minimieren.

Ausdrücke in der Auswahl werden von links nach rechts ausprobiert. Der erste passende Ausdruck ist derjenige, der ausgewählt wird.

Beispielsweise wird der reguläre Ausdruck `foo|foot` in der Zeichenfolge `barefoot` mit `foo` übereinstimmen. Nur ein erster Ausdruck, der passt.

Denken Sie auch daran, dass | in eckigen Klammern als Literal interpretiert wird. Wenn Sie also ‘‘ [fee | fie | foe] ‘‘ schreiben, passen Sie wirklich nur ‘‘ [feio |] ‘‘ zusammen.

RegEx	Streichhölzer
‘‘ foo (bar | foo) ‘‘	`foobar` oder `foofoo`

5.1.7 Unterausdrücke

Die Klammern ‘‘ (. . .) ‘‘ können auch verwendet werden, um Unterausdrücke regulärer Ausdrücke zu definieren.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Positionen, Längen und tatsächliche Werte des Unterausdrucks werden in MatchPos angegeben,‘ MatchLen <tregexpr.html#matchlen>‘_ und‘ Match <tregexpr.html#match>‘_.

Sie können sie durch ‘Substitute ’ersetzen <tregexpr.html#substitute> ‘_.

Unterausdrücke werden von links nach rechts durch ihre öffnenden Klammern nummeriert (einschließlich verschachtelter Unterausdrücke).

Der erste Unterausdruck hat die Nummer "1". Der gesamte reguläre Ausdruck hat die Nummer "0".

Zum Beispiel für die Eingabezeichenfolge `foobar` wird der reguläre Ausdruck ‘‘(foo(bar)) ‘‘ gefunden:

Unterausdruck	Wert
"0"	`foobar`
"1"	`foobar`
"2"	`bar`

5.1.8 Rückreferenzen

Die Metazeichen "‘ 1 "bis" 9 "werden als Rückreferenzen interpretiert. `\n` stimmt mit dem vorher übereinstimmenden Unterausdruck `n` überein.

RegEx	Streichhölzer
(.) 1 + ‘	`aaaa` und `cc`
(. +) 1 + ‘	auch `abab` und `123123`

‘‘ (['"??) (d +) 1‘‘ entspricht‘ *13' (in doppelten Anführungszeichen) oder '4'* (in einfachen Anführungszeichen) oder‘ *77*‘ (ohne Anführungszeichen) etc

5.1.9 Modifikatoren

Modifikatoren dienen zum Ändern des Verhaltens regulärer Ausdrücke.

Sie können Modifikatoren global in Ihrem System festlegen oder den regulären Ausdruck mithilfe von *(? Imsxr-imsxr)* ändern.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Um Modifikatoren zu ändern, verwenden Sie ModifierStr oder entsprechende‘ *TRegExpr*‘-Eigenschaften‘ Modifier * <tregexpr.html#modifieri> __.

Die Standardwerte sind in globalen Variablen definiert <tregexpr.html#global-constants>‘_. Beispielsweise definiert die globale Variable ‘‘ RegExprModifierX‘‘ den Standardwert für die ‘‘ ModifierX‘‘-Eigenschaft.

---

ich, Groß- und Kleinschreibung wird nicht berücksichtigt

Groß- und Kleinschreibung wird nicht berücksichtigt Verwenden Sie die in Ihrem Systemgebietsschema installierten Einstellungen, siehe auch InvertCase <tregexpr.html#invertcase> __.

m, mehrzeilige Zeichenketten

String als mehrere Zeilen behandeln. ^ Und ‘‘ ‘‘ ‘entspricht also dem Anfang oder Ende einer beliebigen Zeile innerhalb des Strings.

Siehe auch *Zeilenumgrenzungen*.

s, einzeilige Zeichenfolgen

Zeichenfolge als einzelne Zeile behandeln. `.` passt also zu jedem beliebigen Charakter, sogar zu Trennzeichen.

Siehe auch *Line Boundaries*, die normalerweise nicht übereinstimmen würden.

g, gierigkeit

Bemerkung: TRegExpr nur Modifikator.

Wenn Sie `Off` umschalten, schalten Sie *Quantifier* in den *nicht-gierigen*-Modus.

Wenn also der Modifikator `/g Off` ist, funktioniert + *'als +?* , * *Als *?* ‘Und so weiter.

Standardmäßig ist dieser Modifikator `On`.

x, erweiterte Syntax

Ermöglicht den regulären Ausdruck zu kommentieren und in mehrere Zeilen aufzuteilen.

Wenn der Modifizierer ‘‘ On‘‘ ist, ignorieren wir alle Leerzeichen, die weder zurückgesetzt noch innerhalb einer Zeichenklasse sind.

Das Zeichen ‘‘ # ‘trennt die Kommentare.

Beachten Sie, dass Sie leere Zeilen verwenden können, um reguläre Ausdrücke für eine bessere Lesbarkeit zu formatieren:

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

Dies bedeutet auch, dass, wenn Sie echte Whitespace- oder # Zeichen in dem Muster (außerhalb einer Zeichenklasse, wo sie nicht von `/x` betroffen sind) möchten, diese entweder entkommen oder sie mit codieren müssen Oktal- oder Hex-Fluchten.

r, russische Reichweiten

Bemerkung: TRegExpr nur Modifikator.

In der russischen ASCII-Tabelle werden die Zeichen / getrennt von anderen platziert.

Große und kleine russische Schriftzeichen liegen in getrennten Bereichen, dies ist die gleiche wie bei englischen Schriftzeichen, aber ich wollte etwas Kurzform.

Mit diesem Modifikator anstelle von `[--]` können Sie `[-]` schreiben, wenn Sie alle russischen Zeichen benötigen.

Wenn der Modifikator "Ein" ist:

RegEx	Streichhölzer
‘‘ a-‘‘	Zeichen von bis und
‘‘ -‘‘	Zeichen von ‘‘ À‘‘ bis und
‘‘ a-‘‘	alle russischen symbole

Der Modifikator ist standardmäßig auf "Ein" gesetzt.

5.1.10 Erweiterungen

(?=<lookahead>)

Behauptung "Look Ahead". Es prüft die Eingabe für den regulären Ausdruck ‘‘ <look-ahead>‘‘, aber erfassen Sie es nicht.

Bemerkung: ‘TRegExpr <tregexpr.html> __

Look-Ahead ist in TRegExpr nicht implementiert.

In vielen Fällen können Sie `look ahead` durch *Sub-expression* ersetzen und einfach ignorieren, was in diesem Subausdruck erfasst wird.

Zum Beispiel ist *(blah) (? = Foobar) (blah) dasselbe wie (blah) (foobar) (blah) . In der letzteren Version müssen Sie jedoch den mittleren Unterausdruck manuell ausschließen - verwenden Sie* ‘ Match [1] + Match [3] ‘‘ und ignorieren Sie `Match[2]`.

Dies ist einfach nicht so praktisch wie in der vorherigen Version, in der Sie den gesamten `Match[0]` *verwenden können, da der von Look Ahead‘* erfasste Teil nicht in den regulären Ausdruck einbezogen wird.

(?:<non-capturing group>)

`?:` is used when you want to group an expression, but you do not want to save it as a matched/captured portion of the string.

So this is just a way to organize your regex into subexpressions without overhead of capturing result:

RegEx	Streichhölzer
`(https?|ftp)://([^/ \r\n]+)`	in `https://sorokin.engineer` matches `https` and `sorokin.engineer`
`(?:https?|ftp)://([^/ \r\n]+)`	in `https://sorokin.engineer` matches only `sorokin.engineer`

(? imsgxr-imsgxr)

Sie können es in regulären Ausdrücken verwenden, um Modifikatoren im Handumdrehen zu ändern.

Dies kann besonders praktisch sein, da es in einem regulären Ausdruck lokalen Gültigkeitsbereich hat. Es betrifft nur den Teil des regulären Ausdrucks, der auf den Operator ‘‘ (?imsgxr-imsgxr) ‘folgt.

Wenn es sich innerhalb des Teilausdrucks befindet, wirkt sich dies nur auf diesen Teilausdruck aus, insbesondere auf den Teil des Teilausdrucks, der auf den Operator folgt. In `((?i)Saint)-Petersburg` betrifft es nur den Teilausdruck ‘‘ ((??) Saint) , `so dass `` Sankt-Petersburg`, aber nicht ‘‘ Sankt-Petersburg‘‘ passt .

RegEx	Streichhölzer
"((i)) Sankt-Petersburg"	‘‘ Saint-petersburg‘‘ und ‘‘ Saint-Petersburg‘‘
"(? i) Saint - (? - i) Petersburg"	‘‘ Sankt Petersburg‘‘ aber nicht ‘‘ Sankt Petersburg‘‘
"((i)) (Sankt Petersburg)"	‘‘ Saint-petersburg‘‘ und ‘‘ s-petersburg‘‘
"((?)) Saint -)? Petersburg"	"Sankt-Petersburg", aber nicht "Sankt-Petersburg"

(?#Text)

Ein Kommentar, der Text wird ignoriert.

Beachten Sie, dass der Kommentar durch das nächste) geschlossen wird. Es gibt also keine Möglichkeit, ein Literal) in den Kommentar einzufügen.

5.1.11 Nachwort

In diesem alten Blogeintrag aus dem vorigen Jahrhundert Ich erläutere einige Verwendungen von regulären Ausdrücken.

	English		Deutsch		Français	Español

5.2 TRegExpr

Implementiert ‘reguläre Ausdrücke <regular_expressions.html> in reinem Pascal, kompatibel zu Free Pascal, Delphi 2 bis 7 und Borland C++ Builder 3 bis 6.

Um es zu benutzen, kopiere einfach den Quellcode in Dein Projekt.

In der IDE Lazarus (Free Pascal) ist diese Bibliothek bereits enthalten . Falls Sie also Lazarus verwenden, müssen Sie gar nichts kopieren.

5.2.1 Klasse TRegExpr

Haupt- und Nebenversion

Gibt die Haupt- und Nebenversionsnummer zurück, zum Beispiel für ‘‘ Version 0.944‘‘

```
Hauptversion = 0 Nebenversion = 944
```

Ausdruck

Regulären Ausdruck.

Zur Optimierung wird der reguläre Ausdruck automatisch in `P-Code` kompiliert. Von Menschen lesbare Form des `P-Codes` wird von *Dump* zurückgegeben.

Bei Fehlern beim Kompilieren wird die `Error`-Methode aufgerufen (standardmäßig `Error` löst die Ausnahme *ERegExpr* aus)

ModifierStr

Werte für reguläre Ausdrücke einstellen oder abrufen <regular_expressions.html#modifiers> __.

Das Format der Zeichenfolge ist ähnlich wie in *(? Ismx-ismx). <regular_expressions.html#inlinemodifiers> __. Zum Beispiel ‘‘ModifierStr: = ’i-x‘‘* schaltet den Modifikator / i ein, schalten Sie‘/x aus <regular_expressions.html#x> und lassen andere unverändert.

Wenn Sie versuchen, einen nicht unterstützten Modifikator einzustellen, wird `Error` aufgerufen.

ModifierI

Modifier / i," Groß- und Kleinschreibung wird nicht berücksichtigt ", initialisiert mit *RegExprModifierI* value.

ModifierR

Modifier / r," Russische Reichweitenerweiterung ", initialisiert mit *RegExprModifierR*-Wert.

ModifierS

Modifikator / en," einzeilige Zeichenketten ", initialisiert mit *RegExprModifierS*-Wert.

ModifierG

Modifikator / g," Gier ", initialisiert mit *RegExprModifierG*-Wert.

ModifierM

Modifier / m," mehrzeilige Zeichenketten ", initialisiert mit *RegExprModifierM*-Wert.

ModifierX

'Modifier / x," erweiterte Syntax " <regular_expressions.html#x> '_, initialisiert mit *RegExprModifierX*-Wert.

Exec

Vergleichen Sie den regulären Ausdruck mit `AInputString`.

Verfügbare überladene `Exec`-Version ohne `AInputString` - es verwendet `AInputString` aus dem vorherigen Aufruf.

Siehe auch die globale Funktion *ExecRegExpr*, die Sie ohne explizite `TRegExpr`-Objekterstellung verwenden können.

ExecNext

Nächste Übereinstimmung finden.

Ohne Parameter funktioniert das Gleiche wie

```
wenn MatchLen [0] = 0, dann ExecPos (MatchPos [0] + 1) oder ExecPos (MatchPos [0] +␣
↪MatchLen [0]);
```

Löst eine Ausnahme aus, wenn sie ohne vorherigen Aufruf von *Exec*, *ExecPos* oder *ExecNext* verwendet wird.

Du musst also sowas immer verwenden

```
wenn Exec (InputString), dann wiederholen Sie {continue results}, bis nicht ExecNext;
```

ExecPos

Sucht nach `InputString` ab `AOffset`

```
AOffset = 1 // erstes Zeichen von InputString
```

InputString

Gibt die aktuelle Eingabezeichenfolge zurück (vom letzten *Exec*-Aufruf oder der letzten Zuweisung an diese Eigenschaft).

Jede Zuweisung zu dieser Eigenschaft löscht *Match*, *MatchPos* und *MatchLen*.

Ersatz

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

Gibt `ATemplate` zurück, wobei `$&` oder `$0` durch den ganzen regulären Ausdruck ersetzt werden und `$n` durch das Auftreten der Unterausdrucknummer `n` ersetzt wird.

Verwenden Sie das Präfix `\` wie `\\` oder ‘‘ $ ‘‘.

Symbol	Beschreibung
`$&`	ganze Übereinstimmung des regulären Ausdrucks
`$0`	ganze Übereinstimmung des regulären Ausdrucks
`$n`	regulärer Unterausdruck `n`
`\n`	in Windows durch `\r\n` ersetzt
`\l`	Kleinbuchstabe ein nächstes Zeichen
`\L`	Kleinbuchstaben alle Zeichen danach
`\u`	ein weiteres Zeichen
`\U`	Großbuchstaben alle Zeichen danach

```
'1\$ is $2\\rub\\' -> '1$ is <Match[2]>\rub\'
'\U$1\\r' wird zu '<Match[1] in uppercase>\r'
```

Wenn Sie eine rohe Ziffer nach ’$ n’ setzen möchten, müssen Sie `n` mit geschweiften Klammern `{}` abgrenzen.

```
&#39;a $ 12bc&#39; -&gt; &#39;a <Match[12]> bc &#39;&#39; a $ {1} 2bc &#39;-&gt;&#39;␣
↪a <Match[1]> 2bc &#39;.
```

Teilt

Trennen Sie AInputStr nach Wiederholungen in APieces

Ruft intern *Exec* / *ExecNext* auf

Siehe auch die globale Funktion *SplitRegExpr*, die Sie ohne explizite `TRegExpr`-Objekterstellung verwenden können.

Ersetzen, ErsetzenEx

```
function Replace (Const AInputStr: RegExprString; const AReplaceStr: RegExprString;␣
↪AUseSubstitution: boolean = False): RegExprString; Überlast; function Replace␣
↪(Const AInputStr: RegExprString; AReplaceFunc: TRegExprReplaceFunction):␣
↪RegExprString; Überlast; function ReplaceEx (Const AInputStr: RegExprString;␣
↪AReplaceFunc: TRegExprReplaceFunction): RegExprString;
```

Gibt die Zeichenfolge mit erneuten Vorkommen durch die Ersetzungszeichenfolge zurück.

Wenn das letzte Argument ("AUseSubstitution") wahr ist, wird "AReplaceStr" als Vorlage für Substitutionsmethoden verwendet.

```
Ausdruck: = &#39;((? I) block | var) \ s * (\ s * \ ([^] * \) \ s *) \ s *&#39;;␣
↪Ersetzen Sie (&#39;BLOCK (test1)&#39;, &#39;def &quot;$ 1&quot; -Wert &quot;$ 2&
↪quot;&#39;, True);
```

Liefert ‘‘def „BLOCK“ Wert "test1" ‘‘

```
Ersetzen (&#39;BLOCK (test1)&#39;, &#39;def &quot;$ 1&quot; Wert &quot;$ 2&quot;&#39;,
↪ False)
```

Liefert ‘‘ def "$ 1" Wert "$ 2" ‘‘

Ruft intern *Exec* / *ExecNext* auf

Die überladene Version und `ReplaceEx` arbeiten mit der Rückruffunktion, sodass Sie sehr komplexe Funktionen implementieren können.

Siehe auch die globale Funktion *ReplaceRegExpr*, die Sie ohne explizite `TRegExpr`-Objekterstellung verwenden können.

SubExprMatchCount

Die Anzahl der Unterausdrücke wurde im letzten *Exec* / *ExecNext*-Aufruf gefunden.

Wenn es keine Subexpressions gibt, jedoch ein vollständiger Ausdruck gefunden wurde (Exec * hat True zurückgegeben), dann `SubExprMatchCount = 0`, falls weder Unterausdrücke noch vollständige gefunden wurden (*Exec* / *ExecNext* lieferte False), dann `SubExprMatchCount = -1`.

Beachten Sie, dass einige Unterausdrücke. kann nicht gefunden werden und für einen solchen Unterausdruck. ‘‘ MathPos = MatchLen = -1‘‘ und ‘‘ Match = ” ‘‘.

```
Ausdruck: = &#39;(1) 2 (3)&#39; &#39;; Exec (&#39;123&#39;): SubExprMatchCount = 2,␣
↪Match [0] = &#39;123&#39;, [1] = &#39;1&#39;, [2] = &#39;3&#39; Exec (&#39;12&#39;
↪): SubExprMatchCount = 1, Match [0] = &#39;12 &#39;, [1] =&#39; 1 &#39;Exec (&#39;␣
↪23 &#39;): SubExprMatchCount = 2, Match [0] =&#39; 23 &#39;, [1] =&#39; &#39;, [2]␣
↪=&#39; 3 &#39;Exec (&#39; 2 &#39;): SubExprMatchCount = 0, Match [0] =&#39; 2 &#39;
↪Exec (&#39; 7 &#39;) - Rückgabe False: SubExprMatchCount = -1
```

MatchPos

pos von Eingang subexpr. `#Idx` wurde in der letzten ‘‘ Exec * ‘‘ Zeichenfolge getestet. Erster Unterausdruck habe `Idx=1`, zuletzt - ‘‘ MatchCount‘‘, ganze hat `Idx=0`.

Gibt `-1` zurück, wenn kein solcher Subexpr vorliegt. oder dieser subexpr. nicht in Eingabezeichenfolge gefunden.

MatchLen

len von entry subexpr. `#Idx` wird in der letzten ‘‘ Exec * ‘‘ Zeichenfolge getestet. Erster Unterausdruck habe `Idx=1`, letzte - MatchCount, ganze hat `Idx=0`.

Gibt -1 zurück, wenn kein solcher Unterausdruck vorhanden ist. oder dieser subexpr. nicht in Eingabezeichenfolge gefunden.

Spiel

Gibt ‘‘ ” ‘‘ zurück, wenn in der Eingabezeichenfolge kein solcher Unterausdruck oder dieser Unterausdruck gefunden wurde.

LastError

Gibt `ID` des letzten Fehlers zurück, `0`, wenn keine Fehler vorliegen (unbrauchbar, wenn die ’Fehler’-Methode eine Ausnahmebedingung auslöst) und den internen Status in `0` löschen (keine Fehler).

ErrorMsg

Gibt die `Error`-Nachricht für einen Fehler mit ‘‘ ID = AErrorID‘‘ zurück.

CompilerErrorPos

Gibt pos in re zurück, wo der Compiler gestoppt wurde.

Nützlich für die Fehlerdiagnose

SpaceChars

Enthält Zeichen, die als `\s` behandelt werden (anfänglich mit der globalen Konstante *RegExprSpaceChars* gefüllt)

WordChars

Enthält Zeichen, die als `\w` behandelt werden (anfänglich mit der globalen Konstante *RegExprWordChars* gefüllt)

LineSeparators

Zeilentrennzeichen (wie `\n` in Unix), anfänglich gefüllt mit *RegExprLineSeparators* (globale Konstante)

Siehe auch ‘Zeilengrenzen <regular_expressions.html#lineseparators> __

LinePairedSeparator

gepaartes Trennzeichen (wie `\r\n` in DOS und Windows).

muss genau zwei oder gar keine Zeichen enthalten und ist anfangs mit dem Inhalt der globalen Konstante RegExprLinePairedSeparator vorbefüllt

Siehe auch ‘Zeilengrenzen <regular_expressions.html#lineseparators> __

Wenn Sie beispielsweise ein Verhalten im Unix-Stil benötigen, weisen Sie ‘‘ LineSeparators: = # $ a‘‘ und ‘‘ LinePairedSeparator: = ” ‘‘ (leere Zeichenfolge) zu.

Wenn Sie als Linientrennzeichen nur *x0D x0A* ‘ aber nicht‘ *x0D* ‘ oder‘ *x0A* ‘ allein akzeptieren möchten, weisen Sie‘ *LineSeparators: = ” (leere Zeichenfolge) und* ‘ LinePairedSeparator: = # $ d # $ a‘‘.

Standardmäßig wird der gemischte Modus verwendet (definiert in RegExprLine [Paired] Separator [s] globale Konstanten):

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

Das Verhalten dieses Modus wird ausführlich in den Zeilengrenzen beschrieben <regular_expressions.html#lineseparators> __.

InvertCase

Invertierung des Zeichenkastens. Definieren Sie es neu, wenn Sie ein anderes Verhalten wünschen.

Kompilieren

Kompiliert regulären Ausdruck.

Nützlich zum Beispiel für GUI-Editoren für reguläre Ausdrücke, um reguläre Ausdrücke zu überprüfen, ohne ihn zu verwenden.

Dump

Zeigen Sie `P-Code` (kompilierter regulärer Ausdruck) als vom Menschen lesbare Zeichenfolge.

5.2.2 Globale Konstanten

EscChar

Escape-char, standardmäßig `\`.

RegExprModifierI

Modifier i <regular_expressions.html#i> _ Standardwert

RegExprModifierR

Modifier r <regular_expressions.html#r> _ Standardwert

RegExprModifierS

‘Modifier s <regular_expressions.html#s> _ Standardwert

RegExprModifierG

‘Modifikator g <regular_expressions.html#g> _ Standardwert

RegExprModifierM

Modifier m <regular_expressions.html#m> _ Standardwert

RegExprModifierX

‘Modifikator x <regular_expressions.html#x> _ Standardwert

RegExprSpaceChars

Standardwert für die *SpaceChars*-Eigenschaft

RegExprWordChars

Standardwert für die Eigenschaft *WordChars*

RegExprLineSeparators

Standardwert für die *LineSeparators*-Eigenschaft

RegExprLinePairedSeparator

Standardwert für die *LinePairedSeparator*-Eigenschaft

RegExprInvertCaseFunction

Standard für die Eigenschaft *InvertCase*

5.2.3 Globale Funktionen

Alle diese Funktionen stehen als Methoden von `TRegExpr` zur Verfügung, aber mit globalen Funktionen müssen Sie keine `TReExpr`-Instanz erstellen, sodass Ihr Code einfacher wäre, wenn Sie nur eine Funktion benötigen.

ExecRegExpr

true, wenn die Zeichenfolge mit dem regulären Ausdruck übereinstimmt. So wie *Exec* in `TRegExpr`.

SplitRegExpr

Teilt den String mit regulären Ausdrücken. Siehe auch *Split*, wenn Sie die `TRegExpr`-Instanz explizit erstellen möchten.

ReplaceRegExpr

```
Funktion ReplaceRegExpr (const ARegExpr, AInputStr, AReplaceStr: RegExprString;␣
↪AUseSubstitution: boolean = False): RegExprString; Überlast; Typ␣
↪TRegexReplaceOption = (rroModifierI, rroModifierR, rroModifiersS, rroModifierG,␣
↪rroModifierM, rroModifierX, rroUseSubstitution, rroUseOsLineEnd);␣
↪TRegexReplaceOptions = Set von TRegexReplaceOption; Funktion ReplaceRegExpr (const␣
↪ARegExpr, AInputStr, AReplaceStr: RegExprString; Optionen: TRegexReplaceOptions):␣
↪RegExprString; Überlast;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

Gibt den String mit regulären Ausdrücken zurück, die durch `AReplaceStr` ersetzt werden. Siehe auch *Replace*, wenn Sie es vorziehen, die TRegExpr-Instanz explizit zu erstellen.

Wenn das letzte Argument (`AUseSubstitution`) wahr ist, wird ‘‘ AReplaceStr‘‘ als Vorlage für ‘‘ Substitutionsmethoden‘‘ verwendet:

```
ReplaceRegExpr (&#39;((?)) Block | var) \ s * (\ s * \ ([^] * \) \ s *) \ s *&#39;, &
↪#39;BLOCK (test1)&#39;, &#39;def &quot;$ 1&quot; value &quot; $ 2 &quot;, wahr)
```

Gibt den ‘‘ def ’BLOCK’-Wert’ test1 ’‘zurück

Aber dieses hier (Anmerkung: Es gibt kein letztes Argument):

```
ReplaceRegExpr (&#39;((?)) Block | var) \ s * (\ s * \ ([^] * \) \ s *) \ s *&#39;, &
↪#39;BLOCK (test1)&#39;, &#39;def &quot;$ 1&quot; value &quot; $ 2 &quot;&#39;)
```

Liefert ‘‘ def "$ 1" Wert "$ 2" ‘‘

Version mit Optionen

Mit `Options` steuert man das Verhalten von *n‘* (mit `rroUseOsLineEnd` wird `\ n` unter Windows durch `\ n \ r` ersetzt , unter Linux bleibt es ein `\ n`). Und so weiter.

```
Typ TRegexReplaceOption = (rroModifierI, rroModifierR, rroModifierS, rroModifierG,␣
↪rroModifierM, rroModifierX, rroUseSubstitution, rroUseOsLineEnd);
```

QuoteRegExprMetaChars

Ersetzen Sie alle Metachars durch ihre sichere Darstellung, zum Beispiel ‘‘ abc’cd. (‘‘ Konvertiert in `abc\'cd\.\(.)`

Diese Funktion ist nützlich für die erneute Generierung von Benutzereingaben

RegExprSubExpressions

Erstellt eine Liste der Unterausdrücke, die in `ARegExpr` gefunden werden

In `ASubExps` repräsentiert jedes Element einen ersten Ausdruck vom ersten bis zum letzten Format im Format:

String - Unterausdruck (ohne ’()’)

Low word of Object - Startposition in ARegExpr, einschließlich ’(’ falls vorhanden! (erste Position ist 1)

hohes Wort der Objektlänge, einschließlich "(" und "Endung") "falls vorhanden!

`AExtendedSyntax` - muss `True` sein, wenn der Schalter `/x` mit dem regulären Ausdruck verwendet werden soll.

Nützlich für GUI-Editoren von re etc (ein Beispiel für die Verwendung finden Sie in ‘REStudioMain.pas) <https://github.com/andgineer/TRegExpr/blob/74ab342b639fc51941a4eea9c7aa53dcdf783592/restudio/REStudioMain.pas#L474> _)

Ergebniscode	Bedeutung
0	Erfolg. Es wurden keine unausgeglichenen Klammern gefunden
-1	Es gibt nicht genug schließende Klammern)
-(n+1)	An Position n wurde das Öffnen von [ohne entsprechende Schließung ‘‘] ‘‘ gefunden
n	In Position n wurde das Schließen der Klammer) ohne entsprechende Öffnung () gefunden

Wenn `Result <> 0 `ASubExprs` kann leere oder illegale Elemente enthalten

5.2.4 ERegExpr

```
ERegExpr = Klasse (Ausnahme) public ErrorCode: integer; // Fehlercode.␣
↪Kompilierungsfehlercodes liegen vor 1000 CompilerErrorPos: integer; // Position in␣
↪re, an der der Kompilierungsfehler aufgetreten ist end;
```

5.2.5 Unicode

UniCode verlangsamt die Leistung. Verwenden Sie ihn daher nur, wenn Sie wirklich Unicode-Unterstützung benötigen.

Um Unicode zu verwenden, kommentieren Sie `{$DEFINE UniCode}` *in* regexpr.pas <https://github.com/andgineer/TRegExpr/blob/29ec3367f8309ba2ecde7d68d5f14a514de94511/src/RegExpr.pas#L86>‘__ (entferne `off`).

Danach werden alle Zeichenfolgen als WideString behandelt.

	English		Deutsch		Français	Español

5.3 FAQ

5.3.1 Ich habe einen schrecklichen Fehler gefunden: TRegExpr löst Zugriffsverletzung aus!

Antworten

Sie müssen das Objekt vor der Verwendung erstellen. Nachdem Sie also etwas erklärt haben:

```
r: TRegExpr
```

Vergessen Sie nicht, die Objektinstanz zu erstellen:

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

5.3.2 Reguläre Ausdrücke mit (? = . . .) funktionieren nicht

Look Ahead ist in TRegExpr nicht implementiert. In vielen Fällen können Sie sie jedoch durch einfache Unteraus-drücke leicht ersetzen.

5.3.3 Unterstützt es Unicode?

Antworten

‘Wie verwende ich Unicode? <tregexpr.html#unicode> __

5.3.4 Warum gibt TRegExpr mehr als eine Zeile zurück?

Zum Beispiel gibt re ‘‘ ‘‘ das erste ‘‘ zurück <font‘‘, then the rest of the file including last `</html>` `<font .\*>`.

Antworten

Für Rückwärtskompatibilität Modifier / s ist standardmäßig‘ *Ein*‘.

Schalten Sie es aus und . passt zu allen anderen als ’Trennzeichen’ <regular_expressions.html#syntax_line_separators> __ - genau wie Sie es wünschen.

Übrigens empfehle ich ‘‘] *)> ‘‘, in ‘‘ Match [1] ‘‘ wird die URL.

5.3.5 Warum gibt TRegExpr mehr als ich erwarte?

Zum Beispiel: ‘‘ (. +) ‘‘ auf String ‘‘ angewendet ein b ‘‘ gibt ‘‘ a zurück b‘‘ aber nicht `a` wie ich erwartet hatte.

Antworten

Standardmäßig arbeiten alle Operatoren im `gierigen` Modus, so dass sie so weit wie möglich zusammenpassen.

Wenn Sie den `nicht-gierigen` Modus wollen, können Sie ‘‘ nicht-gierige‘‘-Operatoren wie `+?` Usw. verwenden, oder alle Operatoren mit Hilfe des Modifikators ‘‘ `g (verwenden Sie die entsprechenden TRegExpr-Eigenschaften oder den Operator‘ *? (- g)* ‘in re).

5.3.6 Wie kann man mit TRegExpr Quellen wie HTML analysieren?

Antworten

Sorry Leute, aber es ist fast unmöglich!

Natürlich können Sie TRegExpr problemlos zum Extrahieren einiger Informationen aus HTML verwenden, wie in meinen Beispielen gezeigt. Wenn Sie jedoch eine genaue Analyse wünschen, müssen Sie einen echten Parser verwenden, nicht re

Die vollständige Erklärung finden Sie beispielsweise in Tom Christiansen und Nathan Torkington, ‘‘ Perl Cookbook‘‘.

Kurz gesagt, es gibt viele Strukturen, die mit echtem Parser einfach geparst werden können, von re aber gar nicht, und realer Parser ist beim Parsing viel schneller, da re nicht nur den Eingabestrom scannt, sondern eine Optimierungssuche ausführt, die dauern kann viel Zeit.

5.3.7 Gibt es eine Möglichkeit, mehrere Übereinstimmungen eines Musters auf TRegExpr abzurufen?

Antworten

Sie können Übereinstimmungen mit der ExecNext-Methode wiederholen.

Wenn Sie ein Beispiel wünschen, werfen Sie einen Blick auf die Implementierung der *TRegExpr.Replace'*-Methode oder auf die Beispiele für' HyperLinksDecorator <demos.html> _

5.3.8 Ich überprüfe die Benutzereingaben. Warum gibt TRegExpr für falsche Eingabezeichenfolgen "True" zurück?

Antworten

In vielen Fällen vergessen TRegExpr-Benutzer, dass reguläre Ausdrücke für ** Suche ** in der Eingabezeichenfolge sind.

Wenn Sie beispielsweise '' d {4,4} '' Ausdruck verwenden, erhalten Sie Erfolg bei falschen Benutzereingaben wie `12345` oder '' any letters 1234''.

Sie müssen vom Anfang der Zeile bis zum Ende der Zeile überprüfen, um sicherzustellen, dass nichts anderes vorhanden ist: '' ^ d {4,4} $ ''.

5.3.9 Warum funktionieren nichtgierige Iteratoren manchmal wie im gierigen Modus?

Zum Beispiel entspricht das re `a+?,b+?`, Das auf den String `aaa,bbb` angewendet wird, `aaa,b`, sollte aber nicht mit `a,b` wegen Nicht-Gier passen des ersten Iterators?

Antworten

Dies liegt an TRegExprs Arbeitsweise. Tatsächlich arbeiten viele andere re-Engines genau gleich: Sie führen nur eine "einfache" Suchoptimierung durch und versuchen nicht, die beste Optimierung zu erreichen.

In manchen Fällen ist es schlecht, aber im Allgemeinen ist es aus Gründen der Leistung und Vorhersagbarkeit eher ein Vorteil als eine Einschränkung.

Die Hauptregel - versuchen Sie zuerst, vom aktuellen Ort aus zu passen. Nur wenn dies völlig unmöglich ist, bewegen Sie sich um ein Zeichen vorwärts und versuchen Sie es erneut von der nächsten Position im Text.

Wenn Sie also `a,b+?` Wird es `a,b` passen. Im Falle von `a+?,b+?` Wird es jetzt nicht empfohlen (wir fügen einen nicht-gierigen Modifikator hinzu), aber es ist immer noch möglich, mehr als einen `a` zu finden, also macht TRegExpr dies.

TRegExpr wie Perl oder Unix versucht nicht, sich vorwärts zu bewegen und zu prüfen - würde es "besser" passen. Fisrt von allem, nur weil es keine Möglichkeit gibt zu sagen, dass es mehr oder weniger gut passt.

5.3.10 Wie kann ich TRegExpr mit Borland C ++ Builder verwenden?

Ich habe ein Problem, da keine Header-Datei (`.h` oder `.hpp`) verfügbar ist.

Antworten

- Fügen Sie `RegExpr.pas` zu `bcb` hinzu.
- Projekt kompilieren Dadurch wird die Header-Datei `RegExpr.hpp` generiert.
- Jetzt können Sie Code schreiben, der die `RegExpr`-Einheit verwendet.
- Vergessen Sie nicht, '' #include "RegExpr.hpp" '' bei Bedarf hinzuzufügen.
- Vergessen Sie nicht, alle \ in regulären Ausdrücken durch \ \ oder neu definierte 'EscChar zu ersetzen <tregexpr.html#escchar> __ const.

5.3.11 Warum arbeiten viele (einschließlich TRegExpr-Hilfe und -Demo) in Borland C ++ Builder falsch?

Antworten

Der Hinweis ist in der vorherigen Frage;) Das Symbol \ hat eine besondere Bedeutung in ‘‘ C ++ , also müssen Sie es ``entkommen (wie in der vorherigen Antwort beschrieben). Wenn Sie nicht gerne \\w + \\\\ w + \\. \\ w + `` mögen, können Sie die Konstante ``EscChar (in RegExpr.pas) neu definieren. Zum Beispiel ‘‘ EscChar = "/" ‘‘. Dann können Sie ‘‘ / w + / w + /. / W + ‘‘ schreiben, sieht ungewöhnlich aus, ist aber lesbarer.

	English		Deutsch		Français	Español

5.4 Demos

Demo-Code für ‘TRegExpr <index.html> __

5.4.1 Einführung

Wenn Sie sich mit dem regulären Ausdruck nicht auskennen, werfen Sie einen Blick auf die Resyntax <regular_expressions.html> __.

TRegExpr-Schnittstelle, beschrieben in TRegExpr-Schnittstelle.

5.4.2 Text2HTML

‘Text2HTML-Quellen <https://github.com/andgineer/TRegExpr/tree/master/examples/Text2HTML> _

Veröffentlichen Sie Nur-Text als HTML

Verwendet die Einheit HyperLinksDecorator, das auf TRegExpr basiert.

Diese Einheit enthält Funktionen zum Verzieren von Hyperlinks.

Ersetzt beispielsweise ‘‘ www.masterAndrey.com‘‘ durch ‘‘ www.masterAndrey.com ‘‘ oder filbert@yandex.ru durch filbert@yandex.ru.

```
Funktion DecorateURLs (const AText: string; AFlags: TDecorateURLsFlagSet = [durlAddr,␣
↪durlPath]): string; type TDecorateURLsFlags = (durlProto, durlAddr, durlPort,␣
↪durlPath, durlBMark, durlParam); TDecorateURLsFlagSet = Satz von TDecorateURLsFlags;
↪ Funktion DecorateEMails (const AText: string): string;
```

Wert	Bedeutung
durlProto	Protokoll (wie ‘‘ ftp: // *oder http: //* ‘)
durlAddr	TCP-Adresse oder Domänenname (wie masterAndrey.com)
durlPort	Portnummer, falls angegeben (wie : 8080)
DurlPath	Pfad zum Dokument (wie index.html)
durlBMark	Buchmarke (wie #mark)
DurlParam	URL-Parameter (wie ? ID = 2 & User = 13)

Gibt den eingegebenen Text "AText" mit verzierten Hyperlinks zurück.

"AFlags" beschreibt, welche Teile des Hyperlinks in den sichtbaren Teil des Links eingefügt werden müssen.

Wenn zum Beispiel AFlags ‘‘ [durlAddr] ‘‘ ist, wird der Hyperlink `www.masterAndrey.com/contact.htm` als ``<a href="www.masterAndrey.com/contacts.htm">www.masterAndrey.com</a>`` verziert.

5.4.3 ‘TRegExprRoutines <https://github.com/andgineer/TRegExpr/tree/master/examples/TRegE

—

Sehr einfache Beispiele, siehe Kommentare im Gerät

5.4.4 ‘TRegExprClass <https://github.com/andgineer/TRegExpr/tree/master/examples/TRegExp

—

Etwas komplexere Beispiele, siehe Kommentare innerhalb der Einheit

KAPITEL 6

Übersetzungen

Die Dokumentation ist in Englisch verfügbar.

Es gibt auch alte Übersetzungen in Deutsch, Bulgarisch, Französisch und Spanisch. Wenn Sie bei der Aktualisierung dieser alten Übersetzungen mithelfen möchten, kontaktieren Sie mich bitte <https://github.com/andgineer>‘_.

Neue Übersetzungen basieren auf GetText <https://en.wikipedia.org/wiki/Gettext>‘_ und kann mit‘ transifex.com bearbeitet werden <https://www.transifex.com/masterAndrey/tregexpr/dashboard/>‘_.

Sie sind bereits maschinell übersetzt und müssen nur korrigiert werden. Möglicherweise werden auch alte Übersetzungen kopiert.

KAPITEL 7

Dankbarkeit

Viele Funktionen wurden vorgeschlagen und viele Fehler wurden von TRegExpr-Mitarbeitern begründet (und sogar behoben).

Ich kann hier nicht alle aufführen, aber ich freue mich über alle Fehlerberichte, Vorschläge und Fragen, die ich von Ihnen bekomme.

- Guido Muehlwitz - hässlicher Fehler in der Verarbeitung großer Seiten gefunden und behoben
- Stephan Klimek - Testen in CPPB und Vorschlagen / Implementieren vieler Funktionen
- Steve Mudford - Offset-Parameter implementiert
- Martin Baur (www.mindpower.com)) - deutsche Übersetzung, nützliche Vorschläge
- Yury Finkel - Unterstützung für UniCode implementiert, einige Fehler gefunden und behoben
- Ralf Junker - Einige Funktionen implementiert, viele Optimierungsvorschläge
- Simeon Lilov - Bulgarische Übersetzung
- Filip Jirsk und Matthew Winter - Hilfe bei der Implementierung des nichtgierigen Modus
- Kit Eason viele Beispiele für die Einführung
- Jürgen Schroth - Käferjagd und nützliche Vorschläge
- Martin Ledoux - französische Übersetzung
- Diego Calp, Argentinien - spanische Übersetzung