TRegExpr Documentation

Release 1.147

Andrey Sorokin

13.05.2023

Inhaltsverzeichnis

	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 bei den Anwendern.

KAPITEL 2

Schnellstart

Um die Bibliothek zu verwenden, fügen Sie einfach den Quelltext Ihrem Projekt hinzu und nutzen 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

	Englisch		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.

Below is the complete regular expressions cheat sheet.

5.1.2 Zeichen

Einfache Übereinstimmungen

Any single character (except special regex characters) matches itself. A series of (not special) characters matches that series of characters in the input string.

RegEx	passt auf
`foobar`	`foobar`

Nicht druckbare Zeichen (Escape-Codes)

To specify character by its Unicode code, use the prefix `\x` followed by the hex code. For 3-4 digits code (after U+00FF), enclose the code into braces.

RegEx	passt auf
`\xAB`	character with 2-digit hex code `AB`
`\x{AB20}`	character with 1..4-digit hex code `AB20`
`foo\x20bar`	`foo bar` (Beachten Sie das Leerzeichen in der Mitte)

There are a number of predefined escape-codes for non-printable characters, like in C language:

RegEx	passt auf
`\t`	(Horizontal-)Tabulator (HT / TAB), identisch mit `\x09`
`\n`	line feed (LF), same as `\x0a`
`\r`	carriage return (CR), same as `\x0d`
`\f`	Formularvorschub (FF), wie `\x0c`
`\a`	Alarm (BEL), gleichbedeutend mit `\x07`
`\e`	Escape (ESC), genauso wie `\x1b`
`\cA ... \cZ`	chr(0) to chr(25). For example, `\cI` matches the tab-char. Lower-case letters „a“...“z“ are also supported.

Maskierung (Escaping)

To represent special regex character (one of `.+*?|\()[]{}^$`), prefix it with a backslash `\`. The literal backslash must be escaped too.

RegEx	passt auf
`\^FooBarPtr`	`^FooBarPtr`, this is `^` and not *start of line*
`\[a\]`	`[a]`, this is not *character class*

5.1.3 Zeichenklassen

Benutzerdefinierte Zeichenklassen

Character class is a list of characters inside square brackets `[]`. The class matches any **single** character listed in this class.

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

You can „invert“ the class - if the first character after the `[` is `^`, then the class matches any character **except** the characters listed in the class.

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

Within a list, the dash `-` character is used to specify a range, so that `a-z` represents all characters between `a` and `z`, inclusive.

If you want the dash − itself to be a member of a class, put it at the start or end of the list, or *escape* it with a backslash.

If you want `]` as part of the class you may place it at the start of list or *escape* it with a backslash.

RegEx	passt auf
`[-az]`	`a`, `z` und −
`[az-]`	`a`, `z` und −
`[a\-z]`	`a`, `z` und −
`[az]`	Zeichen von `a` bis `z`
`[\n-\x0D]`	characters from chr(10) to chr(13)

Meta-Classes

There are a number of predefined character classes that keeps regular expressions more compact, „meta-classes“:

RegEx	passt auf
`\w`	an alphanumeric character, including _
`\W`	a non-alphanumeric
`\d`	a numeric character (same as `[0-9]`)
`\D`	ein nichtnumerisches Zeichen
`s`	ein beliebiger Leerraum (dasselbe wie `[ \t\n\r\f]`)
`\S`	a non-space
`\h`	horizontal whitespace: the tab and all characters in the „space separator“ Unicode category
`\H`	jedes Zeichen, das kein horizontaler Leerraum ist
`\v`	vertical whitespace: all characters treated as line-breaks in the Unicode standard
`\V`	jedes Zeichen, das *nicht* einen vertikalen Abstand liefert

You may use all meta-classes, mentioned in the table above, within *user character classes*.

RegEx	passt auf
`foob\dr`	`foob1r`, `foob6r` and so on, but not `foobar`, `foobbr` and so on
`foob[\w\s]r`	`foobar`, `foob r`, `foobbr` and so on, but not `foob1r`, `foob=r` and so on

Bemerkung: TRegExpr

Die Zeichenklassen `\w`, `\W`, `\s` und `\S` sind in den Properties SpaceChars und WordChars definiert.

So you can redefine these classes.

5.1.4 Grenzen

Zeilengrenzen

Meta-char	passt auf
`.`	any character, can include line-breaks
`^`	zero-length match at start of line
`$`	zero-length match at end of line
`\A`	zero-length match at the very beginning
`\z`	zero-length match at the very end
`\Z`	like `\z` but also matches before the final line-break

Examples:

RegEx	passt auf
`^foobar`	`foobar` nur, wenn es am Anfang der Zeile steht
`foobar$`	`foobar` nur wenn es am Zeilenende steht
`^foobar$`	`foobar` nur wenn davor und danach kein sonstiges Zeichen innerhalb der Zeile steht
`foob.r`	`foobar`, `foobbr`, `foob1r` usw.

Meta-char `^` matches zero-length position at the beginning of the input string. `$` - at the ending. If *modifier /m* is **on**, they also match at the beginning/ending of individual lines in the multi-line text.

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

Bemerkung: TRegExpr

Wenn Sie die Unicode-Version verwenden, dann stimmen `^`/`$` auch mit `\x2028`, `\x2029`, `\x0B`, `\x0C` or `\x85` überein.

Meta-char `\A` matches zero-length position at the very beginning of the input string, `\z` - at the very ending. They ignore *modifier /m*. `\Z` is like `\z` but also matches before the final line-break (LF and CR LF). Behaviour of `\A`, `\z`, `\Z` is made like in most of major regex engines (Perl, PCRE, etc).

Meta-char `.` (dot) by default matches any character, but if you turn **off** the *modifier /s*, then it won't match line-breaks inside the string.

Note that `^.*$` does not match a string between `\x0D\x0A`, because this is unbreakable line separator. But it matches the empty string within the sequence `\x0A\x0D` because this is 2 line-breaks in the wrong order.

Bemerkung: TRegExpr

Multi-line processing can be tuned by properties LineSeparators and UseLinePairedBreak.

Sie können also Zeilenschaltungen im Unix-Stil `\n` oder im DOS/Windows-Stil `\r\n` verwenden oder beides (Standardverhalten wie oben beschrieben).

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

Wortgrenzen (Ankerpunkte)

RegEx	passt auf
`\b`	eine Wortgrenze
`\B`	keine Wortgrenze (Ankerpunkt, der entweder beidseits von w oder W umgeben ist)

Eine Wortgrenze `\b` ist eine Stelle zwischen zwei Zeichen, von denen eines `\w` und das andere ein `\W` ist (Reihenfolge egal).

5.1.5 Quantifizierung

Quantifiers

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

RegEx	passt auf
`{n}`	genau `n` mal
`{n,}`	mindestens `n` mal
`{n,m}`	zumindest "n" aber nicht mehr als "m" mal
`*`	nicht bis beliebig oft vorkommend, alternative Notation: `{0,}`
`+`	mindestens einmal vorkommend, alternative Notation: `{1,}`
`?`	höchstens einmal vorkommend, alternative Notation: `{0,1}`

Innerhalb der geschweiften Klammern `{n, m}` geben Sie also die Mindest-(`n`) und Höchstanzahl (`m`) für das Vorkommen des voranstehenden Elements an.

The `{n}` is equivalent to `{n,n}` and matches exactly `n` times. The `{n,}` matches `n` or more times.

There is no practical limit to the values n and m (limit is maximal signed 32-bit value).

RegEx	passt auf
`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...10 instances of `foobar` (`()` is *group*)

Gier (greediness)

Quantifiers in „greedy“ mode takes as many as possible, in „lazy“ mode - as few as possible.

By default all quantifiers are „greedy“. Append the character `?` to make any quantifier „lazy“.

Für den String `abbbbc`:

RegEx	passt auf
`b+`	`bbbb`
`b+?`	`b`
`b*?`	leerer String
`b{2,3}?`	`bb`
`b{2,3}`	`bbb`

You can switch all quantifiers into „lazy“ mode (*modifier /g*, below we use *in-line modifier change*).

RegEx	passt auf
`(?-g)b+`	`b`

Possessive Quantifier

The syntax is: `a++`, `a*+`, `a?+`, `a{2,4}+`. Currently it's supported only for simple braces, but not for braces after group like `(foo|bar){3,5}+`.

This regex feature is described here. In short, possessive quantifier speeds up matching in complex cases.

5.1.6 Choice

Expressions in the choice are separated by vertical bar `|`.

`fee|fie|foe` passt also zu einem `fee`, `fie` oder `foe` in der Zielzeichenfolge (ebenso wie `fee|fie|foe`).

Zum ersten Ausdruck zählt alles vom letzten Musterbegrenzer (`(`, `[` oder dem Anfang des Musters) bis vor das erste `|``, und der letzte Ausdruck umfasst alles nach dem letzten ``|` zum nächsten Musterbegrenzer.

Klingt etwas kompliziert, daher setzt man üblicherweise die Alternativausdrücke in Klammern, um damit ihren Anfang und das Ende deutlicher erkennbar zu machen.

Die alternativen Ausdrücke werden von links nach rechts durchgetestet, und der erste passende wird verwendet.

Beispielsweise wird der reguläre Ausdruck `foo|foot` in der Zeichenfolge `barefoot` nur `foo` ergeben. Dies war nämlich die erste passende Alternative.

Denken Sie auch daran, dass `|` innerhalb eckiger Klammern als Literal interpretiert wird. Wenn Sie also `[fee|fie|foe]` schreiben, bedeutet das wirklich nur die Zeichenklasse `[efio |]`.

RegEx	passt auf
`foo(bar|foo)`	`foobar` oder `foofoo`

5.1.7 Groups

The brackets `()` are used to define groups (ie subexpressions).

Bemerkung: TRegExpr

Group positions, lengths and actual values will be in MatchPos, MatchLen and Match.

Sie können sie durch Substitute ersetzen.

Groups are numbered from left to right by their opening parenthesis (including nested groups). First group has index 1. The entire regex has index 0.

For string `foobar`, the regex `(foo(bar))` will find:

Group	Value
0	`foobar`
1	`foobar`
2	`bar`

5.1.8 Rückreferenzen

Meta-chars `\1` through `\9` are interpreted as backreferences to groups. They match the previously found group with the specified index.

RegEx	passt auf
`(.)\1+`	`aaaa` und `cc`
`(.+)\1+`	auch `abab` und `123123`

RegEx `(['"]?)(\d+)\1` matches `"13"` (in double quotes), or `'4'` (in single quotes) or `77` (without quotes) etc.

5.1.9 Named Groups and Backreferences

To make some group named, use this syntax: `(?P<name>expr)`. Also Perl syntax is supported: `(?'name'expr)`.

Name of group must be valid identifier: first char is letter or „_“, other chars are alphanumeric or „_“. All named groups are also usual groups and share the same numbers 1 to 9.

Backreferences to named groups are `(?P=name)`, the numbers `\1` to `\9` can also be used.

RegEx	passt auf
`(?P<qq>['"])\w+(?P=qq)`	`"word"` and `'word'`

5.1.10 Modifikatoren

Modifikatoren gestatten eine Verhaltensänderung regulärer Ausdrücke.

You can set modifiers globally in your system or change inside the regular expression using the *(?imsxr-imsxr)*.

Bemerkung: TRegExpr

Um den Wert eines Modifikators festzulegen, verwenden Sie entweder ModifierStr oder die entsprechende `TRegExpr`-Eigenschaft namens ‘Modifier* <tregexpr.html#modifieri> __.

Die Standardwerte sind in den globalen Variablen definiert. Beispielsweise legt die globale Variable ‘‘ RegExprModifierX‘‘ den Standardwert für die `ModifierX`-Eigenschaft fest.

---

i: Groß- und Kleinschreibung ignorieren

Groß- und Kleinschreibung wird nicht berücksichtigt. Verwendet ansonsten die in Ihrem System eingestellten Spracheinstellungen, siehe auch InvertCase.

m: mehrzeilige Zeichenketten

String zeilenweise als mehrzeiligen Text behandeln. `^` und `$` finden damit Anfang und Ende in jeder beliebigen Zeile innerhalb des Strings.

Siehe auch *Zeilengrenzen*.

s: einzeilige Zeichenfolgen

Gesamte Zeichenfolge als einzelne Zeile behandeln. `.` passt dann auf *jedes* beliebige Zeichen, insbesondere auch auf Zeilenwechsel.

Siehe auch *Zeilengrenzen*, die normalerweise nicht gefunden würden.

g: Gierigkeit

Bemerkung: In TRegExpr nur als Modifikator verfügbar.

Wenn Sie auf `Off` umschalten, bringen Sie alle *Quantifier* vom gierigen (greedy) in den ‘‘trägen (non-greedy)<#greedy>‘__-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 dieser Modifikator `On` ist, ignorieren wir alle Leerräume, sofern sie weder maskiert noch innerhalb einer Zeichenklasse stehen.

Und das Zeichen ‘‘ #‘‘ trennt Kommentare ab.

Eine mehrzeilige Darstellung macht übrigens reguläre Ausdrücke oft besser lesbar:

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

Dies bedeutet auch: wenn Sie literale Whitespace- oder `#` Zeichen in dem Muster (außerhalb einer Zeichenklasse, wo sie nicht von `/x` betroffen sind) angeben möchten, müssen Sie diese entweder maskieren oder als Oktal- oder Hex-Codes schreiben.

r: russische Zeichenbereiche

Bemerkung: In TRegExpr nur als Modifikator verfügbar.

In der russischen ASCII-Tabelle sind die Zeichen `/` separat untergebracht.

Die übrigen großen und kleinen russische Schriftzeichen liegen jeweils in getrennten Bereichen, analog wie bei den englischen. Ich wollte eine praktische Kurzform.

Mit diesem Modifikator können Sie also statt `[--]` einfach `[-]` schreiben, wenn Sie sämtliche russischen Zeichen benötigen.

Wenn dieser Modifikator `On` ist:

RegEx	passt auf
–	Kleinbuchstaben, von bis sowie
–	Großbuchstaben, von `A` bis sowie
–	Sämtliche russischen Schriftzeichen

Dieser Modifikator ist standardmäßig `Ein`.

5.1.11 Assertions

Positive lookahead assertion: `foo(?=bar)` matches „foo“ only before „bar“, and „bar“ is excluded from the match.

Negative lookahead assertion: `foo(?!bar)` matches „foo“ only if it's not followed by „bar“.

Positive lookbehind assertion: `(?<=foo)bar` matches „bar“ only after „foo“, and „foo“ is excluded from the match.

Negative lookbehind assertion: `(?<!foo)bar` matches „bar“ only if it's not prefixed with „foo“.

Limitations:

- Brackets for lookahead must be at the very ending of expression, and brackets for lookbehind must be at the very beginning. So assertions between choices `|`, or inside groups, are not supported.
- For lookbehind `(?<!foo)bar`, regex „foo“ must be of fixed length, ie contains only operations of fixed length matches. Quantifiers are not allowed, except braces with the repeated numbers `{n}` or `{n,n}`. Char-classes are allowed here, dot is allowed, `\b` and `\B` are allowed. Groups and choices are not allowed.
- For other 3 assertion kinds, expression in brackets can be of any complexity.

5.1.12 Non-capturing Groups

Syntax is like this: `(?:expr)`.

Such groups do not have the „index“ and are invisible for backreferences. Non-capturing groups are used when you want to group a subexpression, 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	passt auf
`(https?|ftp)://([^/\r\n]+)`	`https` und `sorokin.engineer` in `https://sorokin.engineer`
`(?:https?|ftp)://([^/\r\n]+)`	nur `sorokin.engineer` in `https://sorokin.engineer`

5.1.13 Atomic Groups

Syntax is like this: `(?>expr|expr|...)`.

Atomic groups are special case of non-capturing groups. Description of them.

5.1.14 Inline Modifiers

Syntax for one modifier: `(?i)` to turn on, and `(?-i)` to turn off. Many modifiers are allowed like this: `(?msgxr-imsgxr)`.

You may use it inside regular expression for modifying modifiers on-the-fly. This can be especially handy because it has local scope in a regular expression. It affects only that part of regular expression that follows `(?imsgxr-imsgxr)` operator.

And if it's inside group, it will affect only this group - specifically the part of the group that follows the modifiers. So in `((?i)Saint)-Petersburg` it affects only group `((?i)Saint)` so it will match `saint-Petersburg` but not `saint-petersburg`.

RegEx	passt auf
`(?i)Sankt-Petersburg`	'' Sankt-petersburg'' und '' Sankt-Petersburg''
`(?i)Sankt-(?-i)Petersburg`	`Sankt Petersburg` aber nicht `Sankt petersburg`
`(?i)(Sankt-)?Petersburg`	`Sankt-petersburg` und `sankt-petersburg`
`((?i)Sankt-)?Petersburg`	`saint-Petersburg`, aber nicht `saint-petersburg`

5.1.15 Comments

Syntax is like this: `(?#text)`. Text inside brackets is ignored.

Beachten Sie, dass der Kommentar durch die nächstfolgende `)` geschlossen wird. Es gibt also keine Möglichkeit, eine schließende runde Klammer `)` in den Kommentar einzufügen.

5.1.16 Recursion

Syntax is `(?R)`, the alias is `(?0)`.

The regex `a(?R)?z` matches one or more letters „a" followed by exactly the same number of letters „z".

The main purpose of recursion is to match balanced constructs or nested constructs. The generic regex is `b(?:m|(?R))*e` where „b" is what begins the construct, „m" is what can occur in the middle of the construct, and „e" is what occurs at the end of the construct.

If what may appear in the middle of the balanced construct may also appear on its own without the beginning and ending parts then the generic regex is `b(?R)*e|m`.

5.1.17 Subroutine calls

Syntax for call to numbered groups: `(?1) ... (?90)` (maximal index is limited by code).

Syntax for call to named groups: `(?P>name)`. Also Perl syntax is supported: `(?&name)`.

This is like recursion but calls only code of capturing group with specified index.

5.1.18 Unicode Categories

Unicode standard has names for character categories. These are 2-letter strings. For example „Lu" is uppercase letters, „Ll" is lowercase letters. And 1-letter bigger category „L" is all letters.

- Cc - Control
- Cf - Format

- Co - Private Use
- Cs - Surrrogate
- Ll - Lowercase Letter
- Lm - Modifier Letter
- Lo - Other Letter
- Lt - Titlecase Letter
- Lu - Uppercase Letter
- Mc - Spacing Mark
- Me - Enclosing Mark
- Mn - Nonspacing Mark
- Nd - Decimal Number
- Nl - Letter Number
- No - Other Number
- Pc - Connector Punctuation
- Pd - Dash Punctuation
- Pe - Close Punctuation
- Pf - Final Punctuation
- Pi - Initial Punctuation
- Po - Other Punctuation
- Ps - Open Punctuation
- Sc - Currency Symbol
- Sk - Modifier Symbol
- Sm - Math Symbol
- So - Other Symbol
- Zl - Line Separator
- Zp - Paragraph Separator
- Zs - Space Separator

Meta-character `\p` denotes one Unicode char of specified category. Syntax: `\pL` and `\p{L}` for 1-letter name, `\p{Lu}` for 2-letter names.

Meta-character `\P` is inverted, it denotes one Unicode char **not** in the specified category.

These meta-characters are supported within character classes too.

5.1.19 Nachwort

In diesem alten Blogeintrag aus dem vorigen Jahrhundert erläutere ich einige Anwendungsfälle von regulären Ausdrücken.

	Englisch		Deutsch		Français	Spanisch

5.2 TRegExpr

Implements regular expressions in pure Pascal. Compatible with Free Pascal, Delphi 2-7, C++Builder 3-6.

To use it, copy files „regexpr.pas“, „regexpr_unicodedata.pas“, „regexpr_compilers.inc“, to your project folder.

The library is already included into Lazarus (Free Pascal) project so you do not need to copy anything if you use Lazarus.

5.2.1 Klasse TRegExpr

Haupt- und Nebenversion

Return major and minor version of the component.

```
VersionMajor = 1
VersionMinor = 101
```

Ausdruck

Ein regulärer Ausdruck.

For optimization, regular expression is automatically compiled into P-code. Human-readable form of the P-code is returned by *Dump*.

In case of any errors in compilation, `Error` method is called (by default `Error` raises exception *ERegExpr*).

ModifierStr

Werte für reguläre Ausdrücke setzen oder auslesen.

Format of the string is similar to (?ismx-ismx). For example `ModifierStr := 'i-x'` will switch on the modifier /i, switch off /x and leave unchanged others.

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

ModifierI

Modifier /i, „case-insensitive“, initialized with *RegExprModifierI* value.

ModifierR

Modifier /r, “ russische Zeichenbereiche“, initialisiert mit dem Wert von *RegExprModifierR*.

ModifierS

Modifier /s, „einzeilige Zeichenketten „, initialisiert mit dem Wert von *RegExprModifierS* .

ModifierG

Modifier /g, „Gier (greediness)“, initialisiert mit dem Wert von *RegExprModifierG* .

ModifierM

Modifier /m, „mehrzeilige Zeichenkette“, initialisiert mit dem Wert von *RegExprModifierM* .

ModifierX

Modifier /x, „erweiterte Syntax“, initialisiert mit dem Wert von *RegExprModifierX* .

Exec

Finds regular expression against `AInputString`, starting from the beginning.

The overloaded `Exec` version without `AInputString` exists, it uses `AInputString` from previous call.

Siehe auch die globale Funktion *ExecRegExpr*, die sich ohne explizite Erstellung eines `TRegExpr`-Objekts nutzen lässt.

ExecNext

Finds next match. If parameter `ABackward` is True, it goes downto position 1, ie runs backward search.

Without parameter it works the same as:

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

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

So you always must use something like:

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

ExecPos

Finds match for `AInputString` starting from `AOffset` position (1-based).

Parameter `ABackward` means going from `AOffset` downto 1, ie backward search.

Parameter `ATryOnce` means that testing for regex will be only at the initial position, without going to next/previous positions.

InputString

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

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

Substitute

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

Returns `ATemplate`, where `$&` or `$0` are replaced with the found match, and `$1` to `$9` are replaced with found groups 1 to 9.

To use in template the characters `$` or `\`, escape them with a backslash `\`, like `\\` or `\$`.

Symbol	Description
`$&`	ganze Übereinstimmung des regulären Ausdrucks
`$0`	ganze Übereinstimmung des regulären Ausdrucks
`$1 .. $9`	contents of numbered group 1 .. 9
`\n`	in Windows durch `\r\n` ersetzt
`\l`	lowercase one next char
`\L`	Kleinbuchstaben alle Zeichen danach
`\u`	uppercase one next char
`\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 Ziffer unmittelbar hinter `$n` schreiben möchten, müssen Sie `n` mit geschweiften Klammern `{}` abgrenzen.

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

To use found named groups, use syntax `${name}`, where „name“ is valid identifier of previously found named group (starting with non-digit).

Split

Splits `AInputStr` into `APieces` by regex occurrences.

Ruft intern *Exec* / *ExecNext* auf

Siehe alternativ die globale Funktion *SplitRegExpr*, die sich verwenden läßt, ohne explizit ein‘‘TRegExpr‘‘-Objekt zu erstellen.

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;
```

Returns the string with regex occurencies replaced by the replace string.

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

Overloaded version and `ReplaceEx` operate with callback function, so you can implement really complex functionality.

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

SubExprMatchCount

Count of groups (subexpressions) found in last *Exec* / *ExecNext* call.

If there are no groups found, but some string was found (Exec* returned True), it returns 0. If no groups nor some string were found (*Exec* / *ExecNext* returned false), it returns -1.

Note, that some group may be not found, and for such group `MathPos=MatchLen=-1` and `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

Position (1-based) of group with specified index. Result is valid only after some match was found. First group has index 1, the entire match has index 0.

Returns -1 if no group with specified index was found.

MatchLen

Length of group with specified index. Result is valid only after some match was found. First group has index 1, the entire match has index 0.

Returns -1 if no group with specified index was found.

Spiel

String of group with specified index. First group has index 1, the entire match has index 0. Returns empty string, if no such group was found.

MatchIndexFromName

Returns group index (1-based) from group name, which is needed for „named groups“. Returns -1 if no such named group was found.

LastError

Returns Id of last error, or 0 if no errors occured (unusable if `Error` method raises exception). It also clears internal status to 0 (no errors).

ErrorMsg

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

CompilerErrorPos

Returns position in regex, where P-code compilation was stopped.

Useful for error diagnostics.

SpaceChars

Contains chars, treated as `\s` (initially filled with *RegExprSpaceChars* global constant).

WordChars

Contains chars, treated as `\w` (initially filled with *RegExprWordChars* global constant).

LineSeparators

Line separators (like `\n` in Unix), initially filled with *RegExprLineSeparators* global constant).

See also Line Boundaries

UseLinePairedBreak

Boolean property, enables to detect paired line separator CR LF.

See also Line Boundaries

For example, if you need only Unix-style separator LF, assign `LineSeparators := #$a` and `UseLinePairedBreak := False`.

If you want to accept as line separators only CR LF but not CR or LF alone, then assign `LineSeparators := ''` (empty string) and `UseLinePairedBreak := True`.

By default, „mixed“ mode is used (defined in RegExprLineSeparators global constant):

```
LineSeparators := #$d#$a;
UseLinePairedBreak := True;
```

Behaviour of this mode is described in the Line Boundaries.

Kompilieren

Compiles regular expression to internal P-code.

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

Dump

Shows P-code (compiled regular expression) as human-readable string.

5.2.2 Globale Konstanten

EscChar

Escape character, by default backslash `'\'`.

SubstituteGroupChar

Char used to prefix groups (numbered and named) in Substitute method, by default `'$'`.

RegExprModifierI

Modifier i default value.

RegExprModifierR

Modifier r default value.

RegExprModifierS

Modifier s default value.

RegExprModifierG

Modifier g default value.

RegExprModifierM

Modifier m default value.

RegExprModifierX

Modifier x default value.

RegExprSpaceChars

Default for *SpaceChars* property.

RegExprWordChars

Default value for *WordChars* property.

RegExprLineSeparators

Default value for *LineSeparators* property.

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

Returns True if the string matches the regular expression. Just like *Exec* in `TRegExpr`.

SplitRegExpr

Splits the string by regular expression occurences. See also *Split* if you prefer to create `TRegExpr` instance explicitly.

ReplaceRegExpr

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

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.

If last argument (`AUseSubstitution`) is True, then `AReplaceStr` will be used as template for `Substitution methods`:

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

Returns `def 'BLOCK' value 'test1'`

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 der Option `rroUseOsLineEnd` wird jedes `\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

Replace all metachars with its safe representation, for example `abc'cd.(` is converted to `abc\'cd\.\(`

This function is useful for regex auto-generation from user input.

RegExprSubExpressions

Makes list of subexpressions found in `ARegExpr`.

In `ASubExps` every item represents subexpression, from first to last, in format:

String - Unterausdruck (ohne ’()’)

Low word of Object - starting position in ARegExpr, including ‘(’ if exists! (first position is 1)

High word of Object - length, including starting ‘(’ and ending ‘)’ if exist!

`AExtendedSyntax` - must be True if modifier `/x` os on, while using the regex.

Usefull for GUI editors of regex (you can find example of usage in REStudioMain.pas)

Ergebniscode	Bedeutung
0	Success. No unbalanced brackets were found.
-1	Not enough closing brackets `)`.
-(n+1)	At position n it was found opening `[` without corresponding closing `]`.
n	At position n it was found closing bracket `)` without corresponding opening `(`.

If `Result <> 0`, then `ASubExprs` can contain empty items or illegal ones.

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

In Unicode mode, all strings (InputString, Expression, internal strings) are of type UnicodeString/WideString, instead of simple „string“. Unicode slows down performance, so use it only if you really need Unicode support.

To use Unicode, uncomment `{$DEFINE UniCode}` in regexpr.pas (remove `off`).

	Englisch		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’ <regu-lar_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 `<a href="mailto:filbert@yandex.ru">filbert@yandex.ru</a>`.

```
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 &amp; 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.

- Alexey Torgashin - added many features in 2019-2020, e.g. named groups, non-capturing groups, assertions, backward search
- Guido Muehlwitz - hässlicher Fehler in der Verarbeitung großer Seiten gefunden und behoben
- Stephan Klimek - testing in C++Builder and suggesting/implementing many features
- Steve Mudford - Offset-Parameter implementiert
- Martin Baur (www.mindpower.com)) - deutsche Übersetzung, nützliche Vorschläge
- Yury Finkel - implemented Unicode support, found and fixed some bugs
- Ralf Junker - implemented some features, many optimization suggestions
- Simeon Lilov - Bulgarische Übersetzung
- Filip Jirsk and Matthew Winter - help in implementation non-greedy mode
- Kit Eason - many examples for introduction help section
- Jürgen Schroth - Käferjagd und nützliche Vorschläge
- Martin Ledoux - französische Übersetzung
- Diego Calp, Argentinien - spanische Übersetzung