<%SUMMARY%>

This Word layout template for Help & Manual defines the entire layout - front page, table of contents, chapter headings, keyword index.

To **customize this template**, use the original file ("Word-Template-A4.docx") from the H&M installation folder, make a copy of it, then tailor it with MS Word to your requirements. You **can use variables** in every place of the document. To insert a variable, insert a COMMENT field in Word, then type the variable name as the field content. Like the  field above this text.

Move on to the next page in this document to customize the table of contents (a placeholder for the real TOC later) and modify text and paragraph format of the chapter headings in the middle of this template. You might also want to change the numbering style.

**Important:** before you use the template for publishing in H&M, make sure that general page format and margins are set correctly. H&M will export images and tables to make them fit in the space available.

and EC Software GmbH

**[Type the company address]**

**[Type the phone number]**

**[Type the fax number]**

**2017**

**TRegExpr - Delphi Regular Expressions**

Contents

<%CHAPTER\_HEADING1%> 3

<%CHAPTER\_HEADING2%> 3

<%CHAPTER\_HEADING3%> 3

<%CHAPTER\_HEADING4%> 3

<%CHAPTER\_HEADING5%> 3

<%CHAPTER\_HEADING6%> 3

# About

TRegExpr is easy to use and powerfull tool for sophisticated search and substitutioning and for template-based text checking (especially usefull for user input validation in DBMS and web projects).

You can validate e-mail adresses, extract phone numbers or ZIP-codes from web-pages or documents, search for complex patterns in log files and all You can imagine! Rules (templates) can be changed without Your program recompilation!

As a language for rules used subset of Perl's [regular expressions](#regexp_syntax) (regexp).

Full source code included, pure Object Pascal. Thus, You need no DLL! The library source code is compatible with Delphi 2-7, Borland C++ Builder 3-6, Kylix, FreePascal (if You see any incompatibility problems, please. drop the bug-report to [author](#author)).

Documentation in English, Russian, German, Bulgarian, French and Spanish available at TRegExpr [home page](http://RegExpStudio.com/TRegExpr/TRegExpr.html) (author will be beholden for any remarks about English grammatics of this document and for translation into other languages).

[Installation](#installation) is very simple, the implementation encapsulated completely into class [TRegExpr](#tregexpr_interface).

[Demos projects](#demos) and [usage articles](#Articles) illustrate simplicity and power of text processing with the library.

Do not miss Visual [RegExp Studio](#RegExpStudio) - the best tool for regular expressions exploring, designing and debugging.

If You need Unicode (so called 'WideString' in Delphi) - see "[How to use unicode](#tregexpr_interface_unicode_support)".

Refer to [What's new](#whats_new) section for latest changes.

# Legal issues

Copyright (c) 1999-2004 by Andrey V. Sorokin <[anso@mail.ru]('mailto:anso@mail.ru')>

You may use TRegExpr library in any kind of development, including comercial, redistribute, and modify it freely, under the following restrictions :

1. TRegExpr library is provided as it is, without any kind of warranty given. Use it at Your own risk.The author is not responsible for any consequences of use of this software.

2. The origin of this software may not be mispresented, You must not claim that You wrote the original software. If You use this software in any kind of product, it would be appreciated that there in a information box, or in the documentation would be an acknowledgement like "Partial Copyright (c) 1999-2004 by Andrey V. Sorokin, anso@mail.ru, http://RegExpStudio.com"

3. You may not have any income from distributing this source (or altered version of it) to other developers. When You use this product in a comercial package, the source may not be charged seperatly.

4. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software.

5. RegExp Studio application and all the visual components as well as documentation is not part of the TRegExpr library and is not free for usage.

Some ideas of this library had been derived from Henry Spencer's sources (regexp V8 implementaion):

\* Copyright (c) 1986 by University of Toronto.

\* Written by Henry Spencer. Not derived from licensed software.

\*

\* Permission is granted to anyone to use this software for any

\* purpose on any computer system, and to redistribute it freely,

\* subject to the following restrictions:

\* 1. The author is not responsible for the consequences of use of

\* this software, no matter how awful, even if they arise

\* from defects in it.

\* 2. The origin of this software must not be misrepresented, either

\* by explicit claim or by omission.

\* 3. Altered versions must be plainly marked as such, and must not

\* be misrepresented as being the original software.

# What's New

Legend:

(+) - new feature

(-) - bugfix

(^) - improvement

v. 0.952 2004.01.11

* (+) FreePascal support, thanks to Yaroslav Romanchenko (SAGE)

v. 0.951 2002.09.29

* (+) Defined constant EscChar ('\' by default). Can be usefull for C++ Builder users who are tired with r.e. like '\\w+\\\\\\w+\\.\\w+'. Now they can define for example EscChar='/' and write '/w+\/w+/./w+' - sligtly unusual but more readable.. Suggested by unonimous CPPB user.
* (+) Overloaded versions of Exec (only for Delphi 5 and higher).
* (-) ExecNext in some cases didn't raise exception if called after unsuccessful exec\*, that is wrong (ExecNext must be called ONLY after successful Exec\* ). Bug reported by Craig A. Peterson
* (+) ReplaceEx and overloaded version of Replace - for complex, context-sensitive replacements. Implemented by Thor Asmund.
* (+) Use StrLCopy instead of StpPCopy - in SetExpression() as 0.927 did in SetInputString(). Fixed by Maley, Scott D.
* (-) Bug in HyperLinkDecorator.DecorateURLs.
* (-) Delphi 2 compatibility restored.
* (^) Delphi 7 compatibility tested, Ok.
* (+) New demos.
* (^) Documentation and distribution package were reorganized.

v. 0.948 2002.01.03

* (-) back-offset calculation in UniCode version due to Delphi's PWideChar subtraction 'feature' (produced
* Integer overflow (if Overflow checking is on) and, sometimes, wrong operation), bug reported by Craig A. Peterson
* (-) Bug that was found by Lars Karlslund was fixed by Martin Fuller (in addition Martin made some excellent source code optimizations). The bug was: "If I do '(something|^$)' on '' I get false (which is wrong ...).". Fixed ExecPrim (empty strings were not allowed to pass thru in UseFirstCharSet mode) and FillFirstCharSet (many bugs).
* (-) Visualization bug in Dump method. Fixed by Martin Fuller.

v. 0.947 2001.10.03

* (+) [Word boundary](#regexp_syntax_syntax_word_boundaries) (\b & \B) metachar
* (-) Bug in processing predefined char.classes in non-UseSetOfChar mode
* (+) Spanish help - translated by Diego Calp (mail@diegocalp.com), Argentina
* (+) [VersionMajor/Minor](#tregexpr_interface_tregexpr_version) class method of TRegExpr ;)
* (-) Bug in CompileRegExpr, Thanks to Oleg Orlov <orlov@diasoft.ru>
* (^) Method RegExprSubExpressions wasn't compatible with D2-D3.

Thanks to Eugene Tarasov for bug report.

* (+) [Method Replace](#tregexpr_interface_tregexpr_replace) can now do substitution as well

Thanks to Warren Bare, Ken Friesen and many others who suggested it.

* (+) Updated ReplaceRegExpr to use new Replace method functionality
* (^) Restored UniCode compatibility lost in some previous version

Thanks to Stephan Klimek for bug report

* (^) Updated TestRE project, new examples for Replace with substitution included.

v. 0.942+ 2001.03.01

* (+) Published French help for TRegExpr, translated by Martin Ledoux, Quebec, Canada.

v. 0.942 2001.02.12

* (-) Range-check error in DEMO-project (due to bug in RegExprSubExpressions), Thanks to Juergen Schroth
* (^) RegExprSubExpressions - added error codes for "unclosed "[" error
* (^) Help file bug fixing

v. 0.941 2001.02.01

* (^) Attension! Behaviour of '\w', '\W' was changed! Now it really match alphanum characters and '\_' as described in documentation, not only alpha as it was before. Thanks to Vadim Alexandrov.

If You want to restore previous behaviour, reassign RegExprWordChars (exclude '0123456789' from it).

* (+) Full compatible with recommended at unicode.org implementation of [modifier /m](#regexp_syntax_modifier_m), including DOS-styled line separators (\r\n) mixed with Unix styled (\n) - see properties [LineSeparators](#regexp_syntax_syntax_line_separators), LinePairedSeparator
* (^) Attension! Behaviour of '.' was changed! Now if [modifier /s](#regexp_syntax_modifier_s) is off it doesn't match all chars from [LineSeparators](#regexp_syntax_syntax_line_separators) and LinePairedSeparator (by default \r and \n)
* (^) Attension! To prevent unneeded recompilation of r.e., now assignment to Expression or changing modifiers doesn't cause immidiate [re]compilation. So, now You don't get exception while assigning wrong expression, but can get exception while calling Exec[Next], Substitute, Dump, etc if there are errors in Expression or other properties.
* (+) Non-greedy style [iterators](#regexp_syntax_metacharacters_iterators) (like '\*?'), [modifier /g](#regexp_syntax_modifier_g). Implemented with help from Matthew Winter and Filip Jirsбk
* (+) [modifier /x](#regexp_syntax_modifier_x) (eXtended syntax - allow formating r.e.)
* (+) Procedure Compile to [re]compile r.e. Usefull for GUI r.e. editors and so on (to check all properties validity).
* (+) [FAQ](#faq) in documentation. I am too tired to answer to the same questions again and again :(
* (^) [DEMO project](#RegExpStudio) have been significantly improved. Now this is the real r.e. debugger! Thanks to Jon Smith for his ideas.
* (+) function RegExprSubExpressions, usefull for GUI editors of r.e. (see example of using in TestRExp.dpr project)
* (+) [HyperLinkDecorator](#HyperLinksDecorator) unit - practical example of TRegExpr using
* (-) Range checking error in some cases if ComplexBraces defined Thanks to Juergen Schroth
* (^) 'ComplexBraces' now is defined by default
* (+) Kit Eason sent to me many examples for "[syntax](#regexp_syntax)" help section and I decided to complitely rewrite this section. I hope, You'll enjoy the results ;)
* (+) The \A and \Z metacharacters are just like "^'' and "$'', except that they won't match multiple times when the [modifier /m](#regexp_syntax_modifier_m) is used

v. 0.938 2000.07.23

* (^) Exeptions now jump to appropriate source line, not to Error procedure (I am not quite sure this is safe for all compiler versions (PLEASE, LET ME KNOW about any problems with this). You can turn it off - remove reRealExceptionAddr definition from regexpr.pas).
* (^) Forgotten BSUBEXP[CI] in FillFirstCharSet caused exeption 'memory corruption' in case if back reference can be first op, like this: (a)\*\1 (first subexpression can be skipped and we'll start matching with back reference..).

v. 0.937 2000.06.12

* (-) Bug in optimization engine (since v.0.934). In some cases TRegExpr didn't catch right strings.

Thanks to Matthias Fichtner

v. 0.936 2000.04.22

* (+) Back references, like '<font size=(['"]?)(\d+)\1>' see [syntax description](#regexp_syntax)
* (+) Wide hex char support, like '\x{263a}'

v. 0.935 2000.04.19 (by Yury Finkel)

* (-) fInvertCase now isn't readonly ;)
* (-) UniCode mode compiling errors

v. 0.934 2000.04.17

* (^) New ranges implementation (range matching now is very fast - uses one(!) CPU instruction)
* (^) Internal p-code structure converted into 32-bits - works faster and now there is no 64K limit for compiled r.e.
* (^)'{m,n}' now use 32-bits arguments (up to 2147483646) - specially for Dmitry Veprintsev ;)
* (^) Ranges now support metachars: [\n-\x0D] -> #10,#11,#12,#13; Changed '-' processing, now it's like in Perl: [\d-t] -> '0'..'9','-','t'; []-a] -> ']'..'a'
* (-) Bug with \t and etc macro (they worked only in ranges) Thanks to Yury Finkel
* (^) Added new preprocessing optimization (see FirstCharSet). Incredible fast (!). But be carefull it isn's properly tested. You can switch it Off - remove UseFirstCharSet definition.
* (^) Many other speed optimizations
* (-) Case-insensitive mode now support system-defined national charset (due to bug in v.0.90 .. 0.926 supported only english one)
* (^) Case-insensitive mode implemented with InvertCase (param & result of REChar type) - works 10 .. 100 times faster.
* (^) Match and ExecNext interfaces optimized, added IsProgrammOk by Ralf Junker
* (^) Increased max.subexpression number (NSUBEXP, now 15) and fixed code for this, now you can simply increase NSUBEXP constant by yourself. Suggested by Alexander V. Akimov.
* (^+) Substitute adapted for NSUBEXP > 10 and significant (!) optimized, improved error checking. ATTENTION! Read new Substitute description - syntax was changed !
* (+) SpaceChars & WordChars property - now you may change chars treated as \s & \w. By defauled assigned RegExprSpaceChars/WordChars
* (+) Now \s and \w supported in ranges
* (-) Infinite loop if end of range=#$FF Thanks to Andrey Kolegov
* (+) Function QuoteRegExprMetaChars (see description)
* (+) UniCode support - sorry, works VERY slow (remove '.' from {.$DEFINE UniCode} in regexpr.pas for unicode version). Implemented by Yury Finkel

v. 0.926 2000.02.26

* (-) Old bug derived from H.Spencer sources - SPSTART was set for '?' and '\*' instead of '\*', '{m,n}' and '+'.
* (-^) Now {m,n} works like Perl's one - error occures only if m > n or n > BracesMax (BracesMax = 255 in this version). In other cases (no m or nondigit symbols in m or n values, or no '}') symbol '{' will be compiled as literal. Note: so, you must include m value (use {0,n} instead of {,n}). Note: {m,} will be compiled as {m,BracesMax}.
* (-^) CaseInsensitive mode now support ranges '(?i)[a]' == '[aA]'
* (^) Roman-number template in TestRExp ;)
* (+^) Beta version of complex-braces - like ((abc){1,2}|d){3} By default its turned off. If you want take part in beta-testing, please, remove '.' from {.$DEFINE ComplexBraces} in regexpr.pas.
* (-^) Removed \b metachar (in Perl it isn't BS as in my implementation, but word bound)
* (+) Add /s modifier. But I am not sure that it's ok for Windows. I implemented it as [^\n] for '.' metachar in non-/s mode. But lines separated by \n\r in windows. I need you suggestions !
* (^) Sorry, but I had to rename Modifiers to ModifierStr (ModifierS uses for /s now)

v. 0.91 2000.02.02

* (^) some changes in documentation and demo-project.

v. 0.90 2000.02.01

(+) implemented braces repetitions {min,max}. Sorry - only simple cases now - like '\d{2,3}' or '[a-z1-9]{,7}', but not (abc){2,3} .. I still too short in time. Wait for future versions of TRegExpr or

implement it by youself and share with me ;)

* (+) implemented case-insensitive modifier and way to work with other modifiers - see properties Modifiers, ModifierR, ModifierI and [(?ismxr-ismxr)](#regexp_syntax_inline_modifiers) Perl extension. You may use global variables [RegExpr\*](#tregexpr_interface_modifier_defs) for assigning default modifier values.
* (+) property ExtSyntaxEnabled changed to 'r'-modifier (russian extensions - [see documentation](#regexp_syntax_modifier_r))
* (+) implemented [(?#comment)](#regexp_syntax_inline_comment) Perl extension - very hard and usefull work ;)
* (^) property MatchCount renamed to [SubExprMatchCount](#tregexpr_interface_subexprmatchcount). Sorry for any inconvenients, but it's because new version works slightly different and if you used MatchCount in your programms you have to rethink it ! (see comments to this property)
* (+) add InputString property - stores input string from last Exec call. You may directly assign values to this property for using in ExecPos method.
* (+) add ExecPos method - for working with assigned to InputString property. You may use it like this *InputString := AString;**ExecPos;* or this *InputString := AString; ExecPos (AOffset);* Note: ExecPos without parameter works only in Delphi 4 or higher.
* (+) add ExecNext method - simple and fast (!) way to finding multiple occurences of r.e. in big input string.
* (^) Offset parameter removed from Exec method, if you used it in your programs, please replace all *Exec (AString, AOffset)* with combination *InputString := AString; ExecPos (AOffset)*Sorry for any inconvenients, but old design (see v.0.81) was too ugly :( In addition, multiple Exec calls with same input string produce fool overhead because each Exec reallocate input string buffer.
* (^) optimized implementation of Substitution, Replace and Split methods
* (-) fixed minor bug - if r.e. compilation raise error during second pass (!!! I think it's impossible in really practice), TRegExpr stayed in 'compiled' state.
* (-) fixed bug - Dump method didn't check program existance and raised 'access violation' if previouse Exec was finished with error.
* (+) changed error handling (see functions Error, ErrorMsg, LastError, property CompilerErrorPos, type ERegExpr).
* (-^) TRegExpr.Replace, Split and ExecNext made a infinite loop in case of r.e. match empty-string. Now ExecNext moves by MatchLen if MatchLen <> 0 and by +1 if MatchLen = 0 Thanks to Jon Smith and George Tasker for bugreports.
* (-) While playing with null-matchs I discovered, that null-match at tail of input string is never found. Well, I fixed this, but I am not sure this is safe (MatchPos[0]=length(AInputString)+1, MatchLen = 0). Any suggetions are very appreciated.
* (^) Demo project and documentation was upgraded
* (^) Documentation and this version was published on my home page

v. 0.81 1999.12.25 // Merry Christmas ! :)

* (+) \s (AnySpace) and \S (NotSpace) meta-symbols - implemented by Stephan Klimek with minor fixes by AVS
* (+) \f, \a and \b chars (translates into FF, BEL, BS)
* (-) removed meta-symbols 'Ў' & 'г' - sorry for any inconvenients
* (+) Match property (== copy (InputStr, MatchPos [Idx], MatchLen [Idx]))
* (+) extra parameter Offset to Exec method (thanks to Steve Mudford)

v. 0.7 1999.08.22

* (-) in some cases the r.e. [^...] incorrectly processed (as any symbol) (thanks to Jan Korycan)
* (^) Some changes and improvements in TestRExp.dpr

v. 0.6 1999.08.13 // Friday 13 !

* (^) changed header of TRegExpr.Substitute
* (+) Split, Replace & appropriate global wrappers (thanks to Stephan Klimek for suggetions)

v. 0.5 1999.08.12

* (+) TRegExpr.Substitute routine
* (^) Some changes and improvements in TestRExp.dpr
* (-) Errors in english version of documentation (Thanks to Jon Buckheit)

v. 0.4 1999.07.20

* (-) bug with parsing of strings longer then 255 bytes (thanks to Guido Muehlwitz)
* (-) bug in RegMatch - mathes only first occurence of r.e. (thanks to Stephan Klimek)

v. 0.3 1999.06.13

* (+) ExecRegExpr function

v. 0.2 1999.06.10

* (^) packed into object-pascal class
* (^) code slightly rewriten for pascal
* (^) now macro correct proceeded in ranges
* (+) r.e.ranges syntax extention for russian letters ranges:
* р-  - replaced with all small russian letters (Win1251)
* L-- - replaced with all capital russian letters (Win1251)
* р-- - replaced with all russian letters (Win1251)
* (+) macro '\d' (opcode ANYDIGIT) - match any digit
* (+) macro '\D' (opcode NOTDIGIT) - match not digit
* (+) macro '\w' (opcode ANYLETTER) - match any english letter or '\_'
* (+) macro '\W' (opcode NOTLETTER) - match not english letter or '\_' (all r.e.syntax extensions may be turned off by flag ExtSyntax)

v. 0.1 1999.06.09

* (+) Just first version

# Installation

* Download [distribution package](http://RegExpStudio.com/TRegExpr/TRegExpr.html) if You don't have one
* Unpack it into any (newly created) directory.
* If neccesery, unpack into the same directory package with localized documentation and demos
* Add RegExpr.pas (situated in subdirectory Source) into Your project (Main Delphi menu -> Project -> Add to project..)
* Now You can use it (see [articles with usage illustrations](#Articles) and [Demos projects](#demos)), do not forget to add 'Uses RegExpr' (or performe Main Delphi menu -> File -> Use Unit..) in appropriate units.

Installation of bonus library [HyperLinksDecorator.pas](#HyperLinksDecorator) is the same - just add the file into Your project (You need TRegExpr also installed).

The [RegExp Studio](#RegExpStudio) will help You a much in r.e. learning, designing, debugging and profiling.

# Syntax of Regular Expressions

**Important note**

Below is the description of regular expressions implemented in freeware library [**TRegExpr**](http://RegExpStudio.com/). Please note, that the library widely used in many free and commertial software products. The author of TRegExpr library cannot answer direct questions from this products' users. Please, send Your questions to the product's support first.

**Introduction**

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special **metacharacters** allow You to specify, for instance, that a particular string You are looking for occurs at the beginning or end of a line, or contains **n** recurrences of a certain character.

Regular expressions look ugly for novices, but really they are very simple (well, usually simple ;) ), handly and powerfull tool.

I recommend You to play with regular expressions using [RegExp Studio](#RegExpStudio) - it'll help You to uderstand main conceptions. Moreover, there are many predefined examples with comments included into repository of R.e. visual debugger.

Let's start our learning trip!

**Simple matches**

Any single character matches itself, unless it is a **metacharacter** with a special meaning described below.

A series of characters matches that series of characters in the target string, so the pattern "bluh" would match "bluh'' in the target string. Quite simple, eh ?

You can cause characters that normally function as **metacharacters** or **escape sequences** to be interpreted literally by 'escaping' them by preceding them with a backslash "\", for instance: metacharacter "^" match beginning of string, but "\^" match character "^", "\\" match "\" and so on.

**Examples:**

foobar *matchs string 'foobar'*

\^FooBarPtr *matchs '^FooBarPtr'*

***Note for C++ Builder users***

*Please, read in FAQ answer on question* [*Why many r.e. work wrong in Borland C++ Builder?*](#faq_CPPBEscChar)

**Escape sequences**

Characters may be specified using a **escape sequences** syntax much like that used in C and Perl: "\n'' matches a newline, "\t'' a tab, etc. More generally, \xnn, where nn is a string of hexadecimal digits, matches the character whose ASCII value is nn. If You need wide (Unicode) character code, You can use '\x{nnnn}', where 'nnnn' - one or more hexadecimal digits.

\xnn *char with hex code nn*

\x{nnnn} *char with hex code nnnn (one byte for plain text and two bytes for* [*Unicode*](#tregexpr_interface_unicode_support)*)*

\t *tab (HT/TAB), same as \x09*

\n *newline (NL), same as \x0a*

\r *car.return (CR), same as \x0d*

\f *form feed (FF), same as \x0c*

\a *alarm (bell) (BEL), same as \x07*

\e *escape (ESC), same as \x1b*

**Examples:**

foo\x20bar *matchs 'foo bar' (note space in the middle)*

\tfoobar *matchs 'foobar' predefined by tab*

**Character classes**

You can specify a **character class**, by enclosing a list of characters in [], which will match any **one** character from the list.

If the first character after the "['' is "^'', the class matches any character **not** in the list.

**Examples:**

foob[aeiou]r *finds strings 'foobar', 'foober' etc. but not 'foobbr', 'foobcr' etc.*

foob[^aeiou]r *find strings 'foobbr', 'foobcr' etc. but not 'foobar', 'foober' etc.*

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

If You want "-'' 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 ']' you may place it at the start of list or escape it with a backslash.

**Examples:**

[-az] *matchs 'a', 'z' and '-'*

[az-] *matchs 'a', 'z' and '-'*

[a\-z] *matchs 'a', 'z' and '-'*

[a-z] *matchs all twenty six small characters from 'a' to 'z'*

[\n-\x0D] *matchs any of #10,#11,#12,#13.*

[\d-t] *matchs any digit, '-' or 't'.*

[]-a] *matchs any char from ']'..'a'.*

**Metacharacters**

Metacharacters are special characters which are the essence of Regular Expressions. There are different types of metacharacters, described below.

**Metacharacters - line separators**

^ *start of line*

$ *end of line*

\A *start of text*

\Z *end of text*

. *any character in line*

**Examples:**

^foobar *matchs string 'foobar' only if it's at the beginning of line*

foobar$ *matchs string 'foobar' only if it's at the end of line*

^foobar$ *matchs string 'foobar' only if it's the only string in line*

foob.r *matchs strings like 'foobar', 'foobbr', 'foob1r' and so on*

The "^" metacharacter by default is only guaranteed to match at the beginning of the input string/text, the "$" metacharacter only at the end. Embedded line separators will not be matched by "^'' or "$''.

You may, however, wish to treat a string as a multi-line buffer, such that the "^'' will match after any line separator within the string, and "$'' will match before any line separator. You can do this by switching On the [modifier /m](#regexp_syntax_modifier_m).

The \A and \Z are just like "^'' and "$'', except that they won't match multiple times when the [modifier /m](#regexp_syntax_modifier_m) is used, while "^'' and "$'' will match at every internal line separator.

The ".'' metacharacter by default matches any character, but if You switch Off the [modifier /s](#regexp_syntax_modifier_s), then '.' won't match embedded line separators.

TRegExpr works with line separators as recommended at www.unicode.org ( http://www.unicode.org/unicode/reports/tr18/ ):

"^" is at the beginning of a input string, and, if [modifier /m](#regexp_syntax_modifier_m) is On, also immediately following any occurrence of \x0D\x0A or \x0A or \x0D (if You are using [Unicode version](#tregexpr_interface_unicode_support) of TRegExpr, then also \x2028 or \x2029 or \x0B or \x0C or \x85). Note that there is no empty line within the sequence \x0D\x0A.

"$" is at the end of a input string, and, if [modifier /m](#regexp_syntax_modifier_m) is On, also immediately preceding any occurrence of \x0D\x0A or \x0A or \x0D (if You are using [Unicode version](#tregexpr_interface_unicode_support) of TRegExpr, then also \x2028 or \x2029 or \x0B or \x0C or \x85). Note that there is no empty line within the sequence \x0D\x0A.

"." matchs any character, but if You switch Off [modifier /s](#regexp_syntax_modifier_s) then "." doesn't match \x0D\x0A and \x0A and \x0D (if You are using [Unicode version](#tregexpr_interface_unicode_support) of TRegExpr, then also \x2028 and \x2029 and \x0B and \x0C and \x85).

Note that "^.\*$" (an empty line pattern) doesnot match the empty string within the sequence \x0D\x0A, but matchs the empty string within the sequence \x0A\x0D.

Multiline processing can be easely tuned for Your own purpose with help of TRegExpr properties [LineSeparators](#tregexpr_interface_lineseparators) and [LinePairedSeparator](#tregexpr_interface_linepairedseparator), You can use only Unix style separators \n or only DOS/Windows style \r\n or mix them together (as described above and used by default) or define Your own line separators!

**Metacharacters - predefined classes**

\w *an alphanumeric character (including "\_")*

\W *a nonalphanumeric*

\d *a numeric character*

\D *a non-numeric*

\s *any space (same as [ \t\n\r\f])*

\S *a non space*

You may use \w, \d and \s within custom **character classes**.

**Examples:**

foob\dr *matchs strings like 'foob1r', ''foob6r' and so on but not 'foobar', 'foobbr' and so on*

foob[\w\s]r *matchs strings like 'foobar', 'foob r', 'foobbr' and so on but not 'foob1r', 'foob=r' and so on*

TRegExpr uses properties [SpaceChars](#tregexpr_interface_tregexpr_spacechars) and [WordChars](#tregexpr_interface_tregexpr_wordchars) to define character classes \w, \W, \s, \S, so You can easely redefine it.

**Metacharacters - word boundaries**

\b *Match a word boundary*

\B *Match a non-(word boundary)*

A word boundary (\b) is a spot between two characters that has a \w on one side of it and a \W on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a \W.

**Metacharacters - iterators**

Any item of a regular expression may be followed by another type of metacharacters - **iterators**. Using this metacharacters You can specify number of occurences of previous character, **metacharacter** or **subexpression**.

\* *zero or more ("greedy"), similar to {0,}*

+ *one or more ("greedy"), similar to {1,}*

? *zero or one ("greedy"), similar to {0,1}*

{n} *exactly n times ("greedy")*

{n,} *at least n times ("greedy")*

{n,m} *at least n but not more than m times ("greedy")*

\*? *zero or more ("non-greedy"), similar to {0,}?*

+? *one or more ("non-greedy"), similar to {1,}?*

?? *zero or one ("non-greedy"), similar to {0,1}?*

{n}? *exactly n times ("non-greedy")*

{n,}? *at least n times ("non-greedy")*

{n,m}? *at least n but not more than m times ("non-greedy")*

So, digits in curly brackets of the form {n,m}, specify the minimum number of times to match the item n and the maximum m. The form {n} is equivalent to {n,n} and matches exactly n times. The form {n,} matches n or more times. There is no limit to the size of n or m, but large numbers will chew up more memory and slow down r.e. execution.

If a curly bracket occurs in any other context, it is treated as a regular character.

**Examples:**

foob.\*r *matchs strings like 'foobar', 'foobalkjdflkj9r' and 'foobr'*

foob.+r *matchs strings like 'foobar', 'foobalkjdflkj9r' but not 'foobr'*

foob.?r *matchs strings like 'foobar', 'foobbr' and 'foobr' but not 'foobalkj9r'*

fooba{2}r *matchs the string 'foobaar'*

fooba{2,}r m*atchs strings like 'foobaar', 'foobaaar', 'foobaaaar' etc.*

fooba{2,3}r *matchs strings like 'foobaar', or 'foobaaar' but not 'foobaaaar'*

A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible. For example, 'b+' and 'b\*' applied to string 'abbbbc' return 'bbbb', 'b+?' returns 'b', 'b\*?' returns empty string, 'b{2,3}?' returns 'bb', 'b{2,3}' returns 'bbb'.

You can switch all iterators into "non-greedy" mode (see the [modifier /g](#regexp_syntax_modifier_g)).

**Metacharacters - alternatives**

You can specify a series of **alternatives** for a pattern using "|'' to separate them, so that fee|fie|foe will match any of "fee'', "fie'', or "foe'' in the target string (as would f(e|i|o)e). The first alternative includes everything from the last pattern delimiter ("('', "['', or the beginning of the pattern) up to the first "|'', and the last alternative contains everything from the last "|'' to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end.

Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching foo|foot against "barefoot'', only the "foo'' part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.)

Also remember that "|'' is interpreted as a literal within square brackets, so if You write [fee|fie|foe] You're really only matching [feio|].

**Examples:**

foo(bar|foo) *matchs strings 'foobar' or 'foofoo'.*

**Metacharacters - subexpressions**

The bracketing construct ( ... ) may also be used for define r.e. subexpressions (after parsing You can find subexpression positions, lengths and actual values in MatchPos, MatchLen and [Match](#tregexpr_interface_tregexpr_match) properties of TRegExpr, and substitute it in template strings by [TRegExpr.Substitute](#tregexpr_interface_tregexpr_substitute)).

Subexpressions are numbered based on the left to right order of their opening parenthesis.

First subexpression has number '1' (whole r.e. match has number '0' - You can substitute it in [TRegExpr.Substitute](#tregexpr_interface_tregexpr_substitute) as '$0' or '$&').

**Examples:**

(foobar){8,10} *matchs strings which contain 8, 9 or 10 instances of the 'foobar'*

foob([0-9]|a+)r *matchs 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' etc.*

**Metacharacters - backreferences**

**Metacharacters** \1 through \9 are interpreted as backreferences. \<n> matches previously matched **subexpression** #<n>.

**Examples:**

(.)\1+ *matchs 'aaaa' and 'cc'.*

(.+)\1+ *also match 'abab' and '123123'*

*(['"]?)(\d+)\1 matchs '"13" (in double quotes), or '4' (in single quotes) or 77 (without quotes) etc*

**Modifiers**

Modifiers are for changing behaviour of TRegExpr.

There are many ways to set up modifiers.

Any of these modifiers may be embedded within the regular expression itself using the [(?...)](#regexp_syntax_inline_modifiers) construct.

Also, You can assign to appropriate TRegExpr properties ([ModifierX](#tregexpr_interface_tregexpr_modifier_x) for example to change /x, or ModifierStr to change all modifiers together). The default values for new instances of TRegExpr object defined in [global variables](#tregexpr_interface_modifier_defs), for example global variable RegExprModifierX defines value of new TRegExpr instance ModifierX property.

**i**

Do case-insensitive pattern matching (using installed in you system locale settings), see also [InvertCase](#tregexpr_interface_invertcase).

**m**

Treat string as multiple lines. That is, change "^'' and "$'' from matching at only the very start or end of the string to the start or end of any line anywhere within the string, see also [Line separators](#regexp_syntax_syntax_line_separators).

**s**

Treat string as single line. That is, change ".'' to match any character whatsoever, even a line separators (see also [Line separators](#regexp_syntax_syntax_line_separators)), which it normally would not match.

**g**

Non standard modifier. Switching it Off You'll switch all following operators into non-greedy mode (by default this modifier is On). So, if modifier /g is Off then '+' works as '+?', '\*' as '\*?' and so on

**x**

Extend your pattern's legibility by permitting whitespace and comments (see explanation below).

**r**

Non-standard modifier. If is set then range а-я additional include russian letter 'ё', А-Я additional include 'Ё', and а-Я include all russian symbols.

Sorry for foreign users, but it's set by default. If you want switch if off by default - set false to global variable [RegExprModifierR](#tregexpr_interface_modifier_defs).

The [modifier /x](#regexp_syntax_modifier_x) itself needs a little more explanation. It tells the TRegExpr to ignore whitespace that is neither backslashed nor within a character class. You can use this to break up your regular expression into (slightly) more readable parts. The # character is also treated as a metacharacter introducing a comment, for example:

*(*

*(abc) # comment 1*

*| # You can use spaces to format r.e. - TRegExpr ignores it*

*(efg) # comment 2*

*)*

This also means that if you want real whitespace or # characters in the pattern (outside a character class, where they are unaffected by /x), that you'll either have to escape them or encode them using octal or hex escapes. Taken together, these features go a long way towards making regular expressions text more readable.

**Perl extensions**

**(?imsxr-imsxr)**

You may use it into r.e. for modifying modifiers by the fly. If this construction inlined into subexpression, then it effects only into this subexpression

**Examples:**

(?i)Saint-Petersburg *matchs 'Saint-petersburg' and 'Saint-Petersburg'*

(?i)Saint-(?-i)Petersburg *matchs 'Saint-Petersburg' but not 'Saint-petersburg'*

(?i)(Saint-)?Petersburg *matchs 'Saint-petersburg' and 'saint-petersburg'*

((?i)Saint-)?Petersburg *matchs 'saint-Petersburg', but not 'saint-petersburg'*

**(?#text)**

A comment, the text is ignored. Note that TRegExpr closes the comment as soon as it sees a ")", so there is no way to put a literal ")" in the comment.

**Internal mechanism explanation**

It seems You need some internal secrets of TRegExpr?

Well, it's under constraction, please wait some time..

Just now don't forget to read the [FAQ](#faq) (expecially 'non-greediness' optimization [question](#faq_nongreedyoptimization)).

# TRegExpr interface

**Public methods and properties of TRegExpr class**:

**class function VersionMajor** : integer;

**class function VersionMinor** : integer;

Return major and minor version, for example, for v. 0.944 VersionMajor = 0 and VersionMinor = 944

**property Expression** : string

Regular expression.

For optimization, TRegExpr will automatically compiles it into 'P-code' (You can see it with help of Dump method) and stores in internal structures. Real [re]compilation occures only when it really needed - while calling Exec[Next], Substitute, Dump, etc and only if Expression or other P-code affected properties was changed after last [re]compilation.

If any errors while [re]compilation occures, Error method is called (by default Error raises exception - see below)

**property ModifierStr** : string

Set/get default values of [r.e.modifiers](#RegExp_Syntax_about_modifiers). Format of the string is similar as in [(?ismx-ismx)](#regexp_syntax_inline_modifiers). For example ModifierStr := 'i-x' will switch on modifier /i, switch off /x and leave unchanged others.

If you try to set unsupported modifier, Error will be called (by defaul Error raises exception ERegExpr).

**property ModifierI** : boolean

[Modifier /i](#RegExp_Syntax_modifier_i) - ("caseinsensitive"), initialized with [RegExprModifierI](#tregexpr_interface_modifier_defs) value.

**property ModifierR** : boolean

[Modifier /r](#regexp_syntax_modifier_r) - ("Russian.syntax extensions), initialized with [RegExprModifierR](#tregexpr_interface_modifier_defs) value.

**property ModifierS** : boolean

[Modifier /s](#regexp_syntax_modifier_s) - '.' works as any char (else doesn't match [LineSeparators](#tregexpr_interface_lineseparators) and [LinePairedSeparator](#tregexpr_interface_linepairedseparator)), initialized with [RegExprModifierS](#tregexpr_interface_modifier_defs) value.

**property ModifierG** : boolean;

[Modifier /g](#regexp_syntax_modifier_g) Switching off modifier /g switchs all operators in non-greedy style, so if ModifierG = False, then all '\*' works as '\*?', all '+' as '+?' and so on, initialized with [RegExprModifierG](#tregexpr_interface_modifier_defs) value.

**property ModifierM** : boolean;

[Modifier /m](#regexp_syntax_modifier_m) Treat string as multiple lines. That is, change `^' and `$' from matching at only the very start or end of the string to the start or end of any line anywhere within the string, initialized with [RegExprModifierM](#tregexpr_interface_modifier_defs) value.

**property ModifierX** : boolean;

[Modifier /x](#regexp_syntax_modifier_x) - ("eXtended syntax"), initialized with [RegExprModifierX](#tregexpr_interface_modifier_defs) value.

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

match a programm against a string AInputString

!!! Exec store AInputString into InputString property

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;

Find next match:

ExecNext;

Works same as

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

else ExecPos (MatchPos [0] + MatchLen [0]);

but it's more simpler !

Raises exception if used without preceeding **successful** call to

Exec\* (Exec, ExecPos, ExecNext). So You always must use something like

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

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

find match for InputString starting from AOffset position

(AOffset=1 - first char of InputString)

**property InputString** : string;

returns current input string (from last Exec call or last assign to this property).

Any assignment to this property clear Match\* properties !

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

Returns ATemplate with '$&' or '$0' replaced by whole r.e. occurence and '$n' replaced by occurence of subexpression #n.

Since v.0.929 '$' used instead of '\' (for future extensions and for more Perl-compatibility) and accept more then one digit.

If you want place into template raw '$' or '\', use prefix '\'

Example: '1\$ is $2\\rub\\' -> '1$ is <Match[2]>\rub\'

If you want to place raw digit after '$n' you must delimit n with curly braces '{}'.

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

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

Split AInputStr into APieces by r.e. occurencies

Internally calls 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;

Returns AInputStr with r.e. occurencies replaced by AReplaceStr

If AUseSubstitution is true, then AReplaceStr will be used

as template for Substitution methods.

For example:

Expression := '({-i}block|var)\s\*\(\s\*([^ ]\*)\s\*\)\s\*';

Replace ('BLOCK( test1)', 'def "$1" value "$2"', True);

will return: def 'BLOCK' value 'test1'

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

will return: def "$1" value "$2"

Internally calls Exec[Next]

Overloaded version and ReplaceEx operate with call-back function,

so You can implement really complex functionality.

**property SubExprMatchCount** : integer; // ReadOnly

Number of subexpressions has been found in last Exec\* call.

If there are no subexpr. but whole expr was found (Exec\* returned True), then SubExprMatchCount=0, if no subexpressions nor whole r.e. found (Exec\* returned false) then SubExprMatchCount=-1.

Note, that some subexpr. may be not found and for such subexpr. MathPos=MatchLen=-1 and Match=''.

For example: Expression := '(1)?2(3)?';

Exec ('123'): SubExprMatchCount=2, Match[0]='123', [1]='1', [2]='3'

Exec ('12'): SubExprMatchCount=1, Match[0]='12', [1]='1'

Exec ('23'): SubExprMatchCount=2, Match[0]='23', [1]='', [2]='3'

Exec ('2'): SubExprMatchCount=0, Match[0]='2'

Exec ('7') - return False: SubExprMatchCount=-1

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

pos of entrance subexpr. #Idx into tested in last Exec\* string. First subexpr. have Idx=1, last - MatchCount, whole r.e. have Idx=0.

Returns -1 if in r.e. no such subexpr. or this subexpr. not found in input string.

**property MatchLen** [Idx : integer] : integer; // ReadOnly

len of entrance subexpr. #Idx r.e. into tested in last Exec\* string. First subexpr. have Idx=1, last - MatchCount, whole r.e. have Idx=0.

Returns -1 if in r.e. no such subexpr. or this subexpr. not found in input string.

**property Match** [Idx : integer] : string; // ReadOnly

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

Returns '' if in r.e. no such subexpr. or this subexpr. not found in input string.

**function LastError** : integer;

Returns ID of last error, 0 if no errors (unusable if Error method raises exception) and clear internal status into 0 (no errors).

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

Returns Error message for error with ID = AErrorID.

**property CompilerErrorPos** : integer; // ReadOnly

Returns pos in r.e. there compiler stopped.

Usefull for error diagnostics

**property** **SpaceChars** : RegExprString

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

**property** **WordChars** : RegExprString;

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

**property** **LineSeparators** : RegExprString

line separators (like \n in Unix), initially filled with RegExprLineSeparators global constant)

see also [about line separators](#regexp_syntax_syntax_line_separators)

**property** **LinePairedSeparator** : RegExprString

paired line separator (like \r\n in DOS and Windows).

must contain exactly two chars or no chars at all, initially filled with RegExprLinePairedSeparator global constant)

see also [about line separators](#regexp_syntax_syntax_line_separators)

For example, if You need Unix-style behaviour, assign LineSeparators := #$a (newline character) and LinePairedSeparator := '' (empty string), if You want to accept as line separators only \x0D\x0A but not \x0D or \x0A alone, then assign LineSeparators := '' (empty string) and LinePairedSeparator := #$d#$a.

By default 'mixed' mode is used (defined in RegExprLine[Paired]Separator[s] global constants): LineSeparators := #$d#$a; LinePairedSeparator := #$d#$a. Behaviour of this mode is detailed described in the [syntax section](#regexp_syntax_syntax_line_separators).

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

Converts Ch into upper case if it in lower case or in lower if it in upper (uses current system local setings)

**property InvertCase** : TRegExprInvertCaseFunction;

Set this property if you want to override case-insensitive functionality.

Create set it to RegExprInvertCaseFunction (InvertCaseFunction by default)

**procedure Compile**;

[Re]compile r.e. Usefull for example for GUI r.e. editors (to check all properties validity).

**function Dump** : string;

dump a compiled regexp in vaguely comprehensible form

**Global constants**

**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+'

Modifiers default values:

**RegExprModifierI** : boolean = False; // [TRegExpr.ModifierI](#TRegExpr_Interface_tregexpr_modifier_i)

**RegExprModifierR** : boolean = True; // [TRegExpr.ModifierR](#TRegExpr_Interface_tregexpr_modifier_r)

**RegExprModifierS** : boolean = True; // [TRegExpr.ModifierS](#TRegExpr_Interface_tregexpr_modifier_s)

**RegExprModifierG** : boolean = True; // [TRegExpr.ModifierG](#TRegExpr_Interface_tregexpr_modifier_g)

**RegExprModifierM** : boolean = False; // [TRegExpr.ModifierM](#TRegExpr_Interface_tregexpr_modifier_m)

**RegExprModifierX** : boolean = False; // [TRegExpr.ModifierX](#tregexpr_interface_tregexpr_modifier_x)

**RegExprSpaceChars** : RegExprString = ' '#$9#$A#$D#$C;

// default for SpaceChars property

**RegExprWordChars** : RegExprString =

'0123456789'

+ 'abcdefghijklmnopqrstuvwxyz'

+ 'ABCDEFGHIJKLMNOPQRSTUVWXYZ\_';

// default value for WordChars property

**RegExprLineSeparators** : RegExprString =

#$d#$a{$IFDEF UniCode}#$b#$c#$2028#$2029#$85{$ENDIF};

// default value for LineSeparators property

**RegExprLinePairedSeparator** : RegExprString =

#$d#$a;

// default value for LinePairedSeparator property

**RegExprInvertCaseFunction** : TRegExprInvertCaseFunction = TRegExpr.InvertCaseFunction;

// default for InvertCase property

**Usefull global functions**

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

true if string AInputString match regular expression ARegExpr

! will raise exeption if syntax errors in ARegExpr

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

Split AInputStr into APieces by r.e. ARegExpr occurencies

**function ReplaceRegExpr** (const ARegExpr, AInputStr, AReplaceStr : string;

AUseSubstitution : boolean = False) : string;

Returns AInputStr with r.e. occurencies replaced by AReplaceStr.

If AUseSubstitution is true, then AReplaceStr will be used as template for Substitution methods.

For example:

ReplaceRegExpr ('({-i}block|var)\s\*\(\s\*([^ ]\*)\s\*\)\s\*',

'BLOCK( test1)', 'def "$1" value "$2"', True)

will return: def 'BLOCK' value 'test1'

ReplaceRegExpr ('({-i}block|var)\s\*\(\s\*([^ ]\*)\s\*\)\s\*',

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

will return: def "$1" value "$2"

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

Replace all metachars with its safe representation, for example 'abc$cd.(' converts into 'abc\$cd\.\('

This function usefull for r.e. autogeneration from user input

**function RegExprSubExpressions** (const ARegExpr : string;

ASubExprs : TStrings; AExtendedSyntax : boolean = False) : integer;

Makes list of subexpressions found in ARegExpr r.e.

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

String - subexpression text (without '()')

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 will be On while using the r.e.

Usefull for GUI editors of r.e. etc (You can find example of using in [TestRExp.dpr](#RegExpStudio) project)

**Result code** **Meaning**

**0** Success. No unbalanced brackets was found;

**-1** there are not enough closing brackets ')';

**-(n+1)** at position **n** was found opening '[' without corresponding closing ']';

**n** at position **n** was found closing bracket ')' without corresponding opening '('.

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

**Exception type**

Default error handler of TRegExpr raise exception:

**ERegExpr** = class (Exception)

public

ErrorCode : integer; // error code. Compilation error codes are before 1000.

CompilerErrorPos : integer; // Position in r.e. where compilation error occured

end;

**How to use Unicode**

TRegExpr now supports UniCode, but it works **very slow** :(

Who want to optimize it ? ;)

Use it only if you really need Unicode support !

Remove '.' in {.$DEFINE UniCode} in regexpr.pas. After that all strings will be treated as WideString.

# RegExp Studio

Application for visual regular expressions development (designing, exploring, debuging and profiling). If You don't see the application in 'RegExpStudio' folder of Your TRegExpr distribution package, You can download RegExp Studio from TRegExpr [home page](#author).

With help of RegExp Studio, You can easely jump to any r.e. subexpression (in r.e. source code as well as in current search results), check syntax errors, profile r.e. execution using precise time measurement, play with Substitude and Replace templates and so on.

RegExp Studio has customizable r.e. repository with common regular expressions and r.e. learning examples. You can use this repository to store Your own regular expressions as well as 'test-cases' for them.

Also RegExp Studio provides access to 'internal secrets' of TRegExpr - You can explore p-code of pre-compiled regular expressions and performe fine tunning and optimization.

# FAQ

Q.

**I found a terrible bug: TRegExpr raises Access Violation exception!**

A.

You must create the object before usage. So, after You **declared** something like r : TRegExpr, do not forget to **create the object instance**: r := TRegExpr.Create. Please!!! Read something about Delphi language. I recieve such a questions at least every week. I don't have time to learn how to work with objects and so on.

Q.

**How can I use TRegExpr with Borland C++ Builder?**

I have a problem since no header file (.h or .hpp) is available.

A.

* Add RegExpr.pas to bcb project.
* Compile project. This generates the header file RegExpr.hpp.
* Now one can write code which uses the RegExpr unit.
* Don't forget to add #include "RegExpr.hpp" where needed.
* Don't forget to replace all '\' in regular expressions with '\\' or redefined [EscChar](#TRegExpr_Interface_EscChar) const.

Q.

**Why many r.e. (including r.e. from TRegExpr help and demo) work wrong in Borland C++ Builder?**

A.

Please, reread answer to previous question ;) Symbol '\' has special treting in C++, so You have to 'escape' it (as described in prev.answer). But if You don't like r.e. like '\\w+\\\\\\w+\\.\\w+' You can redefine constant EscChar (RegExpr.pas), for example EscChar='/' - then r.e. will be '/w+\/w+/./w+', sligtly unusual but more readable..

Q.

**Why does TRegExpr return more then one line?**

For example, r.e. <font .\*> returns the first <font, then **the rest of the file** including last </html>...

A.

For backward compatibility, [modifier /s](#regexp_syntax_modifier_s) is 'On' by default.

Switch it Off and '.' will match any but [Line separators](#regexp_syntax_syntax_line_separators) - as you wish.

BTW I suggest you '<font ([^\n>]\*)>', in Match[1] will be URL.

Q.

**Why does TRegExpr return more then I expect?**

For example r.e. '<p>(.+)</p>' applyed to string '<p>a</p><p>b</p>' returns 'a</p><p>b' but not 'a' as I expected.

A.

By default all operators works in 'greedy' mode, so they match as more as it possible.

If You want 'non-greedy' mode You can use 'non-greedy' operators like '+?' and so on (new in v. 0.940) or switch all operators into 'non-greedy' mode with help of modifier 'g' (use appropriate TRegExpr properties or constractions like '?(-g)' in r.e.).

Q.

**How to parse sources like HTML with help of TRegExpr**

A.

Sorry folks, but it's nearly impossible!

Of course, You can easily use TRegExpr for extracting some information from HTML, as shown in my examples, but if You want accurate parsing You have to use real parser, not r.e.!

You can read full explanation in Tom Christiansen and Nathan Torkington 'Perl Cookbook', for example. In short - there are many constractions that can be easy parsed by real parser but cannot at all by r.e., and real parser is MUCH faster do the parsing, because r.e. doesn't simply scan input stream, it performes optimization search that can take a lot of time.

Q.

**Is there a way to get multiple matchs of a pattern on TRegExpr?**

A.

You can make loop and iterate match by match with ExecNext method.

It cannot be done more easily becase of Dalphi isn't interpretator as Perl (and it's benefit - interpretators work very slow!).

If You want some example, please take a look at TRegExpr.Replace method implementation. or at the examples in [HyperLinksDecorator.pas](#HyperLinksDecorator)

Q.

**I am checking user input. Why does TRegExpr return 'True' for wrong input strings?**

A.

In many cases TRegExpr users forget that regular expression is for SEARCH in input string. So, if You want to make user to enter only 4 digits and using for it '\d{4,4}' expression, You can skip wrong user input like '12345' or 'any letters 1234 and anything else'. You have to add checking for line start and line end to ensure there are not anything else around: '^\d{4,4}$'.

Q.

**Why does non-greedy iterators sometimes work as in greedy mode?**

For example, the r.e. 'a+?,b+?' applied to string 'aaa,bbb' matches 'aaa,b', but should it not match 'a,b' because of non-greediness of first iterator?

A.

This is the limitation of used by TRegExpr (and Perl's and many Unix's regular expressions) mathematics - r.e. performe only 'simple' search optimization, and do not try to do **the best optimization**. In some cases it's bad, but in common it's rather advantage then limitation - because of perfomance and predictability reasons.

The main rule - r.e. first of all try to match from current place and only if it's completely impossible move forward by one char and try again from that place. So, if You use 'a,b+?' it match 'a,b', but in case of 'a+?,b+?' it's '**not recommended**' (due to non-greediness) **but possible** to match more then one 'a', so TRegExpr do it and at last obtaines correct (but non optimum) match. TRegExpr like Perl's or Unix's r.e. doesn't attempt to move forward and check - would it be 'better' match. Moreover, it cannot be compared in terms 'more or less good match' at all..

Please, read '[Syntax](#RegExp_Syntax_mechanism)' section of help for more explanation.

# Author

Andrey V. Sorokin

Saint Petersburg, Russia

[anso@mail.ru](mailto:anso@mail.ru), anso@paycash.ru

[http://RegExpStudio.com](http://RegExpStudio.com/) (http://anso.da.ru)

**Please, if You think You found a bug or have any questions about TRegExpr, download latest TRegExpr version from my** [**home page**](http://RegExpStudio.com/TRegExpr/TRegExpr.html) **and read the** [**FAQ**](#faq) **before sending e-mail to me!**

No doubt I have long '[Disclaimer](#Disclaimer)' text for the software (just curious who's reading such a things, except lawers 8-)).

---------------------------------------------------------------

Gratitudes

---------------------------------------------------------------

All the documentation was made with [Help&Manual](http://www.helpandmanual.com) - the best help authoring tool with native Delphi integration!

[](http://www.helpandmanual.com)

Many features suggested and a lot of bugs founded (and even fixed) by TRegExpr's contributors.

I cannot list here all of them (actually I kept listing only on very early stage of development), but I do appreciate all

bug-reports, features suggestions and questions that I am receiving from You.

* Guido Muehlwitz - found and fixed ugly bug in big string processing
* Stephan Klimek - testing in CPPB and suggesting/implementing many features
* Steve Mudford - implemented Offset parameter
* Martin Baur ([www.mindpower.com](http://www.mindpower.com)) - German help, usefull suggetions, free hosting for the project
* Yury Finkel - implemented UniCode support, found and fixed some bugs
* Ralf Junker - Implemented some features, many optimization suggestions
* Simeon Lilov - Bulgarian help
* Filip Jirsбk and Matthew Winter - help in Implementation non-greedy mode
* Kit Eason many examples for introduction help section
* Juergen Schroth - bug hunting and usefull suggestions
* Martin Ledoux - French help
* Diego Calp, Argentina -Spanish help

And many others - for big work of bug hunting !

I am still looking for person who can help me to translate this documentation into other languages or correct existed translations (some of them is for older versions only)

# Demos

## Demos

First of all I recommend You to read [articles with usage illustrations](#Articles).

Please, note that there are localized demos available (with comments in source code on national languages). This localized versions distributed in localized full TRegExpr packages, and in separate localized documentation packages (when You unpack this documentation package in TRegExpr directory the localized demos overwrite English ones).

**Demos\TRegExprRoutines**

very simple examples, see comments inside the unit

**Demos\TRegExprClass**

slightly more complex examples, see comments inside the unit

**Demos\Text2HTML**

see [description](#Text2Html)

If You don't familiar with regular expression, please, take a look at the [r.e.syntax](#regexp_syntax) topic.

TRegExpr interface described in [TRegExpr interface](#tregexpr_interface).

**Note**

Some of demo-projects use extended VCL properties which exists only in Delphi 4 or higher. While compiling in Delphi 3 or Delphi 2 you'll receive some error messages about unknown properties. You may ignore it - this properties is needed only for resizing and justification of components then form change it's size.

## Text2HTML

Very simple utility, that helps publish plain text as HTML

Uses unit [HyperLinksDecorator](#HyperLinksDecorator) that is based on TRegExpr.

Specially written as a demonstration of TRegExpr usage.

## Unit HyperLinksDecorator

This unit contains functions to decorate hyper-links (see [Text2Html](#Text2Html) demo-project for usage example).

For example, replaces '*www.RegExpStudio.com*' with '*<a href="http://www.RegExpStudio.com">www.RegExpStudio.com</a>*' or '*anso@mail.ru*' with '*<a href="mailto:anso@mail.ru">anso@mail.ru</a>*'.

**function DecorateURLs**

Finds and replaces hyper links like 'http://...' or 'ftp://..' as well as links without protocol, but start with 'www.' If You want to decorate emails as well, You have to use function [DecorateEMails](#HyperLinksDecorator_decorateemails) instead.

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

**Description**

Returns input text AText with decorated hyper links.

AFlags describes, which parts of hyper-link must be included into VISIBLE part of the link:

For example, if [durlAddr] then hyper link '*www.RegExpStudio.com*/contacts.htm' will be decorated as '<a href="http://*www.RegExpStudio.com*/contacts.htm">*www.RegExpStudio.com*</a>'

**type**

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

TDecorateURLsFlagSet = **set** **of** TDecorateURLsFlags;

**Description**

These are the possible values:

**Value** **Meaning**

durlProto Protocol (like 'ftp://' or 'http://')

durlAddr TCP address or domain name (like '*RegExpStudio.com*')

durlPort Port number if specified (like ':8080')

durlPath Path to document (like 'index.html')

durlBMark Book mark (like '#mark')

durlParam URL params (like '?ID=2&User=13')

**function DecorateEMails**

Replaces all syntax correct e-mails with '<a href="mailto:ADDR">ADDR</a>'. For example, replaces 'anso@mail.ru' with '<a href="mailto:anso@mail.ru">anso@mail.ru</a>'.

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

**Description**

Returns input text AText with decorated e-mails

# Articles

## Usage illustrations

* [Text processing from bird's eye view](#Article_Bird_Eye_View)
* [MrDecorator](#Article_MrDecorator)

## Text processing from bird's eye view

Do You want to write program for extracting weather forecast or currency rates or e-mails or whatsoever You want from HTML-pages, e-mails or other unformatted source? Or do You need to import data into Your database from old DB's ugly export format? Or You want just ensure that the e-mail user entered is syntaxically correct one?

There are two ways.

The traditional one - You must make full featured text parser. This is an awful peace of work!

For example, try to implement rules how to recognize e-mail address - simple code like

p := Pos ('@', email);

**if** (p > 1) **and** (p < length (email))

**then** ...

don't filter many common errors, for example, users frequently forget enter domain-part of e-mail, You'll need much more complex code (just read the big article "Extended E-mail Address Verification and Correction" on www.Delphi3000.com). Just think about writing and debugging this code.

The second way - look at the text from bird's eye view with help of regular expressions engine. You don't write the check processing routine, You just describe how regexp engine must do it for You. Your application will be implemented very fast and will be robust and easy to change!

Unfortunately, Delphi component palette contains no TRegularExpression component. But there are some third-party implementations (I think You already know at least one 8-)).

Example 1

How to chech e-mail address syntax.

Just write

**if** ExecRegExpr ('[\w\d\-\.]+@[\w\d\-]+(\.[\w\d\-]+)+', email)

**then** ... gotcha! e-mail **is** valid ...

Do not forget to add TRegExpr into uses section of the unit.

Example 2

How to extract phone numbers from unformatted text (web-pages, e-mails, etc).

For example, we need only St-Petersburg (Russia) phones (city code 812).

**procedure** ExtractPhones (**const** AText : **string**; APhones : TStrings);

**begin**

**with** TRegExpr.Create **do** **try**

Expression := '(\+\d \*)?(\((\d+)\) \*)?(\d+(-\d\*)\*)';

**if** Exec (AText) **then**

**REPEAT**

**if** Match [3] = '812'

**then** APhones.Add (Match [4])

**UNTIL** **not** ExecNext;

**finally** Free;

**end**;

**end**;

For the input text

***"Hi !***

***Please call me at work (812)123-4567 or at home +7 (812) 12-345-67***

***truly yours .."***

this procedure returns

**APhones[0]='123-4567'**

**APhones[1]='12-345-67'**

Example 3

Extracting currency rate from Russian Bank web page.

Create new project and place at the main form TBitBtn, TLabel and TNMHTTP components.

Add following code as BitBtn1 OnClick event handler (don't mind Russian letter - they need for Russian web-page parsing):

**procedure** TForm1.BitBtn1Click(Sender: TObject);

**const**

Template = '(?i)Ioeoeaeuiue eo?n OA ii aieea?o'

+ '.\*Aaoa\s\*Eo?n\s\*Eo?n iie.\s\*Eo?n i?ia. [^<\d]\*'

+ '(\d?\d)/(\d?\d)/(\d\d)\s\*[\d.]+\s\*([\d.]+)';

**begin**

NMHTTP1.Get ('http://win.www.citycat.ru/finance/finmarket/\_CBR/');

**with** TRegExpr.Create **do** **try**

Expression := Template;

**if** Exec (NMHTTP1.Body) **then** **begin**

Label1.Caption := Format ('Russian rouble rate %s.%s.%s: %s',

[Match [2], Match [1], Match [3], Match [4]]);

**end**;

**finally** Free;

**end**;

**end**;

Now, then You click at the BitBtn1, programm connects to specified web-server and extract current rate.

Conclusion

"Free Your mind" ((c) The Matrix ;)) and You'll find many other tasks there regular expressions can save You incredible amount of stupid coding work !

P.S. Sorry for terrible english. My native language is Pascal ;)

## MrDecorator

Here we will discuss how to "decorate url's".

I mean, what if You want to show some plain-text on the HTML-page. The mostly common example - web-forum (BBS board). The user enters the message, for example "the answer You can find at www.RegExpStudio.com" and it must be shown on web-page as text with HTML-link, i.e. converted to "the answer You can find at www.RegExpStudio.com"

There are two ways.

The traditional one - You must make full featured text parser. This is an awful amount of tedious work ! For example, try to implement rules for URL search ;)

The second - look at the text from bird's eye view with help of regular expressions engine. Your application will be implemented very fast and will be robust and easy to support !

Unfortunately, Delphi component palette contains no TRegularExpression component. But there are some third-party implementations (I think You already know at least one 8-))..

The complete source code, ready to run, available in TRegExpr Demos ([HyperLinksDecorator unit](#HyperLinksDecorator))

**uses**

RegExpr; *// Do not forget this line. Actually this is how TRegExpr 'Install' - the*

*// only thing You must do - include RegExpr into uses section.*

**type**

TDecorateURLsFlags = (

*// describes, which parts of hyper-link must be included*

*// into VISIBLE part of the link:*

durlProto, *// Protocol (like 'ftp://' or 'http://')*

durlAddr, *// TCP address or domain name (like 'anso.da.ru')*

durlPort, *// Port number if specified (like ':8080')*

durlPath, *// Path to document (like 'index.htm')*

durlBMark, *// Book mark (like '#mark')*

durlParam *// URL params (like '?ID=2&User=13')*

);

TDecorateURLsFlagSet = **set** **of** TDecorateURLsFlags;

**function** DecorateURLs (

*// can find hyper links like 'http://...' or 'ftp://..'*

*// as well as links without protocol, but start with 'www.'*

**const** AText : **string**;

*// Input text to find hyper-links*

AFlags : TDecorateURLsFlagSet = [durlAddr, durlPath]

*// Which part of hyper-links found must be included into visible*

*// part of URL, for example if [durlAddr] then hyper link*

*// 'http://anso.da.ru/index.htm' will be decorated as*

*// '<a href="http://anso.da.ru/index.htm">anso.da.ru</a>'*

) : **string**;

*// Returns input text with decorated hyper links*

**const**

URLTemplate =

'(?i)'

+ '('

+ '(FTP|HTTP)://' *// Protocol*

+ '|www\.)' *// trick to catch links without*

*// protocol - by detecting of starting 'www.'*

+ '([\w\d\-]+(\.[\w\d\-]+)+)' *// TCP addr or domain name*

+ '(:\d\d?\d?\d?\d?)?' *// port number*

+ '(((/[%+\w\d\-\\\.]\*)+)\*)' *// unix path*

+ '(\?[^\s=&]+=[^\s=&]+(&[^\s=&]+=[^\s=&]+)\*)?'

*// request (GET) params*

+ '(#[\w\d\-%+]+)?'; *// bookmark*

**var**

PrevPos : integer;

s, Proto, Addr, HRef : **string**;

**begin**

Result := '';

PrevPos := 1;

**with** TRegExpr.Create **do** **try**

Expression := URLTemplate;

**if** Exec (AText) **then**

**REPEAT**

s := '';

**if** AnsiCompareText (Match [1], 'www.') = 0 **then** **begin**

Proto := 'http://';

Addr := Match [1] + Match [3];

HRef := Proto + Match [0];

**end**

**else** **begin**

Proto := Match [1];

Addr := Match [3];

HRef := Match [0];

**end**;

**if** durlProto **in** AFlags

**then** s := s + Proto;

**if** durlAddr **in** AFlags

**then** s := s + Addr;

**if** durlPort **in** AFlags

**then** s := s + Match [5];

**if** durlPath **in** AFlags

**then** s := s + Match [6];

**if** durlParam **in** AFlags

**then** s := s + Match [9];

**if** durlBMark **in** AFlags

**then** s := s + Match [11];

Result := Result + System.Copy (AText, PrevPos,

MatchPos [0] - PrevPos) + '<a href="' + HRef + '">' + s + '</a>';

PrevPos := MatchPos [0] + MatchLen [0];

**UNTIL** **not** ExecNext;

Result := Result + System.Copy (AText, PrevPos, MaxInt); *// Tail*

**finally** Free;

**end**;

**end**; *{ of function DecorateURLs*

*--------------------------------------------------------------}*

Note, that You can easely extract any part of URL (see AFlags parameter).

Conclusion

"Free Your mind" ((c) The Matrix ;)) and You'll find many other tasks there regular expressions can save You incredible part of stupid coding work !

P.S. Sorry for terrible english. My native language is Pascal ;)

Keyword index

**No index entries found.**