Getting	Started
Ottime	Juncu

RegEx Builder is a very simple application. If you are reading this help file, then you have already figured out how to run the installer so I won't cover that here.

Portable Use

RegEx Builder is almost a portable application. The only thing that keeps it from being truly portable is that it requires the .NET framework. It does not use the registry. All you need is the application executable. So, you could run this from a thumb-drive as long as the host system has the .NET 2.0 framework installed.

Canto	atima	I In
Conta	cung	US

The best way to contact me regarding RegEx Builder is to post a comment on the <u>RegEx Builder Product</u> page on my website. If you'd like to post a review of the product, please do so on <u>Download.com</u> or <u>SoftPedia.com</u>. I'd really appreciate that.

If you really need to contact me in person, you can <u>email me</u> directly.

Introduction

RegEx Builder helps you build regular expressions. I built this tool for use in my C# development, but you can use these expressions with any regular expression engine that understands standard regular expressions. Use RegEx Builder to create your expression, then copy the expression to where-ever you need to use it.

RegEx Builder interprets your expression as you build it. This eliminates the old edit-compile-test loop that sometimes seems endless when trying to build a complex expression. With RegEx Builder you simply supply one or more examples of the text you want to match in the target text window. Then start building your expression. RegEx Builder will evaluate your expression as you type it and show you which text in the target text is matched by highlighting all matching occurrences.

There is a lot of really good information available on regular expressions on the web. I am not going to try to replicate any of it here. RegEx Builder provides a help button drop down menu with menu items that should cover almost everything you could ever want to know about regular expressions.

The Quick Start Guide section explains in just a few steps how to start using RegEx Builder.

The Reference section contains descriptions of all the features of RegEx Builder.

The FAQ contains common questions and answers.

RegEx Builder is very easy to use. Just follow these simple steps:

- 1. Startup RegEx Builder.
- 2. Copy, or type, examples of the text patterns that you want to match into the Target Text box. Try to think of as many combinations as you can of the pattern you would like to match. You can also include examples of text that you do not want to match to ensure that you don't accidentally build an expression that matches unexpected text.
- 3. Begin building your expression in the Regular Expression box. You can type expression constructs directly, or use the right-click menu in the Regular Expression box to easily insert common regular expression constructs into your expression. Continue building your expression until your are satisfied that it matches your target text. If the background color of the expression is red, then the expression is not valid and an appropriate error message will be displayed just below the regular expression box. Since RegEx Builder evaluates while you type you will see errors indicated while you have incomplete parts of the expression. This is normal.
- 4. Once your expression is complete you can simply copy it to the clipboard and paste it into your code. If you are using C#, you can use the use the "Copy C# Constructor to Clipboard" menu item on the right-click menu to create a complete constructor ready to paste into your C# code.

That's it. That's all there is to it. You're done...

More details can be found in the Reference section of this help manual.

🛛 🕅 RegEx Builder	
Eile Copy Insert Examples Help	
Target Text (paste text you want to text into this box)	
Today is 11/26/2008 last month was 10/26/200	D8.
Regular supression being tested (can be calit onto reultiple lines)	
Expression is valid	
Expression is valid CultureInvariant ExplicitCapture RightToLeft	☐ IgnorePatternWhites
Expression is valid CultureInvariant ExplicitCapture RightToLeft ECMAScript Multiline Singleline	☐ IgnorePatternWhites ☐ IgnoreCase
Expression is valid CultureInvariant ExplicitCapture RightToLeft ECMAScript Multiline Singleline Matches (select match to highlight above)	 IgnorePatternWhites IgnoreCase James (select name to highlight above)
Expression is valid CultureInvariant ExplicitCapture RightToLeft ECMAScript Multiline Singleline Matches (select match to highlight above) M. I L. Matched String	IgnorePatternWhites IgnoreCase Iames (select name to highlight above) Name Value
Expression is valid CultureInvariant ExplicitCapture RightToLeft CUltureInvariant Multiline Singleline Matches (select match to highlight above) M. I L. Matched String 0 9 10 11/26/2008	IgnorePatternWhites IgnoreCase James (select name to highlight above) Name Value month 11
Expression is valid CultureInvariant ExplicitCapture ECMAScript Multiline Matches (select match to highlight above) M. I N. I N I I Natched String 0 9 1 35 10 10/26/2008	IgnorePatternWhites IgnoreCase Iames (select name to highlight above) Name Value month 11 day 26

1. Main Menu

Contains the File, Copy, Insert, Examples, and Help menus.

2. Target Text Box

The Target Text box is where you want to place any text that you want to match, or want to make sure you don't match. The simplest thing to do is copy and paste from another source. You can also just type example text.

3. Regular Expression Box

Construct your regular expression here. Type your regular expression by hand or use the extensive right-click <u>context menu</u> to place regular expression constructs into your expression. To add a construct, place the cursor where you want the construct, then right-click to pull up the context menu. Your selected construct will be inserted into the expression at the cursor location.

A status line just below the regular expression box tells you the current status of the regular expression. If there are errors in the expression, this status line will show you the error message as reported by the regular expression parser.

4. Microsoft Specific Expression Options

Each of these check boxes enables, if checked, or disables, if not checked, specific Microsoft options for the .NET regular expression engine. If you are using your regular expression in a non-.NET environment, then you will want to leave these all unchecked. The tool-tips for these check boxes contain extensive explanations. More extensive help is available in the MSDN documentation.

5. Matches Table

This table shows details about each match made to the Target Text. Selecting an entry in this table will cause the corresponding text in the Target Text box to be highlighted.

6. Names Table

If your regular expression contains named sub-expression, this table shows the detail of those names for each match. Selecting an entry in this table will cause the corresponding text in the Target Text box to be highlighted.

7. Target Text Splitter

Moved this bar with the mouse to adjust the size of the target text panel. Double-click on the spitter bar to auto-size the panel to show all lines of the target text.

8. Regular Expression Splitter

Moved this bar with the mouse to adjust the size of the regular expression panel. Double-click on the spitter bar to auto-size the panel to show all lines of the regular expression.

9. Match Panel Splitter

Move this bar with the mouse to adjust the size of the matches and namematches tables. When you right-click in the the regular expression box (or use the **Insert** menu item on the main menu bar) you are presented with an extensive cascading menu that allows you to insert many regular expression constructs directly into the expression. This means that you don't need to commit to memory all of the nuances of regular expression syntax.

The top most menu (shown below) contains several sub-menus that present different classes of regular expression constructs. Each is explained briefly.

<u>Character Classes</u>

Go here to select from the different character classes that you need to match. This is usually the place you want to start. Most of the other sub-menus build of of the selection you will make here.

• <u>Quantifiers</u>

The choice in this submenu help you decide how many instances of a character class or literal you want to match.

• Atomic Zero-width Assertions

These constructs do not match any actual characters, but match assertions in the text. For instance: match-beginning-of-line, match-word-boundary, and match-end-of-line are all zer-width assertions.

• <u>Grouping Constructs</u>

Advanced grouping constructs. Please consult MSDN or online references for details on each of these constructs.

Backreference Constructs

Advanced backreferencing constructs. Please consult MSDN or online references for details on each of these constructs.

• <u>Alternation Constructs</u>

Advanced alternation constructs. Please consult MSDN or online references for details on each of these constructs.

• Miscellaneous Constructs

The only item available on this submenu is a regular expression comment. This allows for inserting comments into your regular expression to help clarify the expression.

• Common Expressions

Presents a list of common expressions which can be inserted into the regular expression. This is a dynamic menu built from the regular expression files found in the *Expresisons* subdirectory of the application installation location. You can build your own library of expressions by saving expression files into this directory. Upon application restart any added expressions will be available in this menu.

• Insert Example Expression

Just like the Common Expressions menu, but based on files found in the *Examples* sub directory of the application installation location.

• Any character (.)

Matches any character except a new-line. This is the most basic match. It matches anything but new-line characters. Use this when you just need to match anything.

• Word character (\w)

Matches any character that can be used to make words.

• Non-word character (\W)

The opposite of \w. Matches any non-word character.

• White-space (\s)

Matches any white-space character. Spaces, tabs are examples of white-space.

• Non-white-space (\S) Matches any non-white-space character.

• **Decimal digit (\d)** Matches any decimal digit character.

• Non-decimal digit (\D)

Matches non-digit characters.

• Positive Character Group ([characters])

Matches any of the specified characters. For example, [abcd] will match any of a, b, c, or d.

• Negative Character Group ([^characters]) Matches any character that is not part of the specified list of characters. For example, [^aeiou] will match consonants, but not vowels.

• Character Range Group ([char1-char2])

Matches any character within the range specified. For example, [0-9] matches any decimal digit (like \d). Ranges can be combined, [0-9a-fA-F] will match any hexadecimal digit, including upper and lower case letters.

- Visual Studio 2003 / .NET Framework 1.1
- Visual Studio 2005 / .NET Framework 2.0
- <u>.NET Framework 3.0</u>
- Visual Studio 2008 / .NET Framework 3.0

• Zero or more (*)

This quantifier tells the expression to match zero or more instances of the proceeding term. For example "0." will match zero or more '0' characters.

• One or more (+)

This quantifier tells the expression to match one or more instances of the proceeding term. For example "0+" will match one or more '0' characters.

• Zero or one (?)

This quantifier tells the expression to match zero or one instance of the proceeding term. For example "0?" will match zero or one '0' characters.

• Exactly n ({n})

Matches exactly n instances of the proceeding term. For example " $d{5}$ " will match a 5 digit US zip code, and " $d{5}-d{4}$ " will match a 5-4 US zip code.

• At least n ({n,})

Matches at least n instances of the proceeding term.

• At least n, but no more than m ({n,m})

Matches at least n, but no more than m instances of the proceeding term.

• Make it lazy (<quantifier>?)

Adding a '?' to the quantifier makes the quantifier lazy. This means that the engine will accept the first minimal match, rather than the largest possible match. For example if the test string is "012345" and the regular expression is "\d{2,3}", then the matches will be "012" and "345". If we add the lazy qualifier, "\d{2,3}?", then the matches become "01", "23", and "45".

For detailed information on these constructs see the online reference at MSDN:

• Visual Studio 2003 / .NET Framework 1.1

- <u>Visual Studio 2005 / .NET Framework 2.0</u>
- <u>.NET Framework 3.0</u>
- <u>Visual Studio 2008 / .NET Framework 3.0</u>

• Beginning of line (^)

Matches the beginning of a line. Use this when you want to make sure that you expression only matches strings at the beginning of a line. This must be the first character in the expression.

• End of line (\$)

Matches the end of a line. Use this when you want to make sure that you expression only matches strings at the end of a line. This must be the last character in the expression. You can use both the ^ and \$ to ensure that you expression matches only complete lines.

• Beginning of string, ignore multiline (\A)

Specifies that the match must occur at the beginning of the string (ignores the Multiline option).

• End of string, but before \n, ignoring multiline (\Z) Specifies that the match must occur at the end of the string or before \n at the end of the string (ignores the Multiline option).

• End of string, ignoring multiline (\z) Specifies that the match must occur at the end of the string (ignores the Multiline option).

• At end of last match (\G)

Specifies that the match must occur at the point where the previous match ended. When used with Match.NextMatch(), this ensures that matches are all contiguous.

- Word boundary, between \w and \W (\b) Matches expression on word boundaries. For example: (ve)\b will match the "ve" in "live", but not the "ve" in "liver".
- Not on word boundary (\B)

Matches expressions on non-word boundaries. For example, (ve)\B will match the "ve" in "liver", but not the "ve" in "live".

- Visual Studio 2003 / .NET Framework 1.1
- Visual Studio 2005 / .NET Framework 2.0
- <u>.NET Framework 3.0</u>
- Visual Studio 2008 / .NET Framework 3.0

- Subexpression ((subexpr))
- Named group ((?<name>subexpr))
- Balancing group definition ((?<name1-name2>subexpr))
- Non-capturing Group ((?:subexpr))
- Zero-width positive look-ahead assertion ((?=subexpr))
- Zero-width negative look-ahead assertion ((?!subexpr))
- Zero-width positive look-behind assertion ((?<=subexpr))
- Zero-width negative look-behind assertion ((?<!subexpr))
- Non-backtracking subexpression ((?>subexpr))

- Visual Studio 2003 / .NET Framework 1.1
- Visual Studio 2005 / .NET Framework 2.0
- <u>.NET Framework 3.0</u>
- Visual Studio 2008 / .NET Framework 3.0

- Immediate backreference (\1)
- Backreference (\<num>)
- Named backreference (\|<name>)

- Visual Studio 2003 / .NET Framework 1.1
- <u>Visual Studio 2005 / .NET Framework 2.0</u>
- <u>.NET Framework 3.0</u>
- <u>Visual Studio 2008 / .NET Framework 3.0</u>

- Or (|)
- Yes/No Expression match ((?(expr)yes|no))
- Yes/No Named match ((?(name)yes|no))

- Visual Studio 2003 / .NET Framework 1.1
- <u>Visual Studio 2005 / .NET Framework 2.0</u>
- <u>.NET Framework 3.0</u>
- <u>Visual Studio 2008 / .NET Framework 3.0</u>

1. New

Creates a new Regular Expression by clearing the Target Text, Regular Expression, and Option check boxes.

2. **Save**

Saves the current Target Text, Regular Expression, and Options to an XML file. If the current data is not associated with a filename, then you will be prompted for one, otherwise the current filename is used.

3. Save As...

Saves the current Target Text, Regular Expression, and Options to an XML file, prompting for a new file name first.

4. **Open...**

Open a saved regular expression from a file. Restores the Target Text, Regular Expression, and Options. If there is unsaved work, you will be prompted to save it first.

5. **Exit**

Exits the application, prompting to save any unsaved work.

1. Copy RE to Clipboard

Copies the Regular Expression to the Windows Clipboard

2. Copy C# Constructor to Clipboard

Creates a complete C# Constructor and copies it to the clipboard. You can then paste the constructor directly into your C# source code.

Copy

Insert

Previous Top Next

See <u>Insert/Context menu</u>.

Examples

This is a dynamic menu. Each item in this menu corresponds to a file in the Examples directory found at the installation location for RegEx Builder. When you select an item from this menu, the corresponding example will be loaded into the application. The target text and regular expression will be replaced. If you have unsaved work, you will be prompted to save it first.

You can add you own examples by saving or copying regular expression files into the Examples directory. After restarting the applications you new examples will be in the menu.

- 1. **RegEx Builder Help** Opens this help file.
- 2. **Regular Expressions on Wikipedia** Opens a browser to the Wikipedia page on Regular Expressions.
- 3. Google "Regular Expressions"

Opens a browser to he Google Search results page for "Regular Expressions"

4. Regular Expressions on MSDN

Opens a browser to the MSDN page on Regular Expressions.

• Why is my regular expression box red? The Regular expression box turns red when the regular expression cannot be parsed due to an error in the expression. This will happen quite often while typing expression constructs. Continue editing your expression. When it is valid, the background will not be red.

• How do I know what is matched?

Any text in the Target Text box that is matched by your expression will be highlighted. If no text matches your expression, nothing will be highlighted. This is how you know that your expression isn't working. Any matching text will also be shown in the Matches table.

• I can't remember the syntax for a particular construct?

Neither can I. That's why there is an extensive context menu in the regular expression edit box. Position you cursor where you want the construct, then right-click and find the desired construct in the menu. Select it and it will be pasted into you expression.

• I already know everything there is to know about building Regular Expressions, why do I need this?

Good for you. You don't need this.

FAQ

How To Buy

You can't buy RegEx Builder, but you can make a donation to fund further development. To make a donation, open RegEx Builder, click on the About button, then click on the Donate button. Thanks!

Version 1.0.0.6

• Fixed control layout problem when re-sizing application window.

Version 1.0.0.5

• Initial Release.