



Version 2.13.1 — 16 October 2023

Published by Just Great Software Co. Ltd.

Copyright © 2009–2023 Jan Goyvaerts. All rights reserved.

“RegexMagic” and “Just Great Software” are trademarks of Jan Goyvaerts

# Table of Contents

## How to Use RegexMagic ..... 1

1. Introducing RegexMagic .....	3
2. Getting Started with RegexMagic .....	4
Creating Regular Expressions with RegexMagic.....	5
3. How to Add Sample Text .....	6
4. How to Create Regular Expressions.....	7
5. How to Create Capturing Groups and Replacement Text .....	9
6. How to Use Your Regular Expressions.....	10

## RegexMagic Examples ..... 11

1. RegexMagic Examples .....	13
2. Pattern: Unicode characters .....	15
3. Pattern: Basic characters .....	16
4. Pattern: Character masks .....	17
5. Pattern: List of literal text.....	18
6. Pattern: Literal text.....	19
7. Pattern: Literal bytes.....	20
8. Pattern: Control characters .....	21
9. Pattern: Number .....	22
10. Pattern: Integer .....	25
11. Pattern: Date and time.....	28
12. Pattern: Email address .....	30
13. Pattern: URL .....	32
14. Pattern: Country .....	34
15. Pattern: State or Province .....	36
16. Pattern: Currency.....	38
17. Pattern: Credit card number .....	40
18. Pattern: National ID .....	43
19. Pattern: VAT number.....	44
20. Pattern: IPv4 Address.....	47
21. Pattern: GUID .....	50
22. Pattern: Regular expression.....	52
23. Pattern: Pattern used by another field.....	53
24. Pattern: Text matched by another field .....	55
25. Pattern: Match anything .....	57
26. Combining Patterns Using Multiple Fields .....	59
27. Repeating Combined Fields Using a Sequence Field.....	61
28. Matching Unrelated Items Using Alternation.....	64
29. Matching Complex Unrelated Items Using Alternation of Sequences .....	66
30. Capturing Groups.....	69
31. Replacing Text Match By Fields with Text Matched by Other Fields.....	71
32. Typing in Your Own Replacement Text .....	73
33. Matching Version Numbers .....	75

34. \$VAR and \${VAR} Variable Names .....	78
35. Matching Something Within Something Else.....	80
36. Matching Comments in Java Code .....	84
37. Increment ImageIndex Constants .....	88

## **RegexMagic Reference .....93**

1. RegexMagic Assistant.....	95
2. Samples Panel Reference.....	96
3. Match Panel Reference .....	101
4. Pattern: Pattern used by another field.....	108
5. Pattern: Text matched by another field.....	109
6. Pattern: Match anything.....	110
7. Pattern: Unicode characters .....	111
8. Pattern: Basic characters.....	113
9. Pattern: Character masks.....	115
10. Pattern: List of literal text.....	116
11. Pattern: Literal text.....	118
12. Pattern: Literal bytes .....	119
13. Pattern: Control characters .....	120
14. Pattern: Number.....	121
15. Pattern: Integer .....	125
16. Pattern: Date and time.....	127
17. Pattern: Email address .....	131
18. Pattern: URL .....	133
19. Pattern: Country .....	138
20. Pattern: State or Province .....	140
21. Pattern: Currency.....	141
22. Pattern: Credit card number .....	142
23. Pattern: National ID .....	144
24. Pattern: VAT number.....	145
25. Pattern: IPv4 Address .....	146
26. Pattern: GUID .....	148
27. Pattern: Regular expression.....	149
28. Action Panel Reference .....	151
29. Regex Panel Reference .....	157
30. Copying and Pasting Regular Expressions .....	161
31. Compare Applications .....	164
32. Use Panel Reference .....	168
33. Available Functions in Source Code Snippets.....	171
34. Source Code Template Editor.....	174
35. Library Panel Reference .....	182
36. GREP Panel Reference .....	184
37. Share Experiences and Get Help on The User Forums .....	187
38. Forum RSS Feeds.....	192
39. Change RegexMagic's Appearance.....	193
40. Adjust RegexMagic to Your Preferences.....	197
41. Text Layout Configuration .....	202
42. Text Cursor Configuration.....	207
43. Integrate RegexMagic with Searching, Editing and Coding Tools.....	210

44. Basic Integration with RegexMagic .....	212
45. Integrating RegexMagic Using COM Automation .....	215
46. Application Identifiers .....	222

**Part 1**

# **How to Use RegExMagic**



# 1. Introducing RegexMagic

RegexMagic makes creating regular expressions easier than ever. While other regex tools merely make it easier to work with regular expressions, with RegexMagic you don't have to deal with the regular expression syntax at all. RegexMagic generates complete regular expressions to your specifications.

First, you provide RegexMagic with some samples of the text you want your regular expression to match. RegexMagic can automatically detect what sort of pattern your text looks like. Numbers, dates, and email addresses are just a few examples of the wide range of patterns that RegexMagic supports. By marking different parts of your samples, you can create regular expressions that combine multiple patterns to match exactly what you want. RegexMagic's patterns provide many options, so you can make your regular expression as loose or as strict as you want.

Best of all, RegexMagic supports nearly all popular regular expression flavors. Select your flavor, and RegexMagic makes sure to generate a regular expression that works with it. RegexMagic can even generate snippets in many programming languages that you can copy and paste directly into your source code to implement your regular expression.

RegexMagic itself takes very little time to learn. If you take about 10 minutes to read the getting started sections and work through a few examples, you'll have a good idea of what you can do with RegexMagic and how to use its capabilities. You'll be creating regular expressions in no time.

## 2. Getting Started with RegexMagic

RegexMagic enables you to create regular expressions without having to learn the rather cryptic regular expression syntax. Best of all, RegexMagic itself takes very little time to learn. If you take about 10 minutes to read the getting started topic and work through a few examples, you'll have a good idea of what you can do with RegexMagic and how to use its capabilities. While RegexMagic doesn't automatically generate regular expressions on magic alone, it sure makes things a lot easier by allowing you to work with your data instead of the terse regex syntax. You'll be creating regular expressions in no time.

### Overview of RegexMagic

RegexMagic consists of nine separate panels: Assistant, Samples, Match, Action, Regex, Use, Library, GREP, and Forum. The toolbar at the top has a View menu that provides access to all the panels. It also provides a number of different layouts. If you don't like any of the layouts, you can rearrange the panels by dragging and dropping their tabs or caption bars with the mouse.

The Assistant panel displays helpful hints as well as error messages while you work with RegexMagic. You'll want to keep an eye on this panel all the time. If you close it and an error occurs, it will automatically show itself until you dismiss the error.

The Samples panel, Match panel, and Action panel hold all the settings for what is called a RegexMagic Formula. A RegexMagic Formula is simply all the information that RegexMagic needs to generate a complete regular expression, and optionally a replacement text. You'll be spending most of your time with RegexMagic in these three panels. The toolbar at the top has New, Open, Save, Undo, and Redo buttons that work on RegexMagic Formulas or the contents of these three panels as a whole.

The Regex panel shows the regular expression generated by the RegexMagic Formula that you've created. First, select the application or programming language you'll use the regular expression with. After generating the regex, you can click the Copy button to copy the regular expression formatted as a string for a particular programming language.

The Use panel generates complete source code snippets performing a variety of functions in most of the world's popular programming languages that you can readily paste into your own source code. That's often a lot handier than just copying a string with a regex. All the source code templates are fully editable, and you can even create your own.

The Library panel is where you can store your RegexMagic Formulas for later reuse. A RegexMagic Library is a single file that holds as many RegexMagic Formulas as you care to add to it. RegexMagic ships with a sample library that includes all of the examples described in this help file. The help file topic for each sample mentions which formula to select in the library.

The GREP panel uses your regular expression to search through files and folders, just like the traditional `grep` utility. This is a quick way of testing your regular expression on a larger set of files than is practical with the Samples panel. It also allows you to run actual searches.

Finally, the Forum panel is where you can discuss RegexMagic and regular expressions with other RegexMagic users. Since the forum is built right into RegexMagic itself, you can easily attach RegexMagic Formulas to your messages.

## Creating Regular Expressions with RegexMagic

In RegexMagic, you create a regular expression and optional replacement text by building up a RegexMagic Formula using the Samples, Match, and Action panels. The settings on these three panels are all RegexMagic needs to generate a regular expression and a replacement text. Combined, these settings are called a **RegexMagic Formula**.

Start with adding samples of the text that your regular expression should match to the Samples panel. Don't forget to also add samples of the text that your regular expression should not match. Use the toolbar and list at the left hand side of the panel to add as many samples as you like. You can load text files or paste in text from the clipboard. You can edit the text of the sample you've selected in that list in the edit control to the right of it. Read the how to topic for details.

With the samples in place, it's time build up your regular expression. The best way to learn how to create regular expressions with RegexMagic is to work through a few examples. The email pattern example is a simple example. The example for SKU codes shows how you can create regular expressions that match anything you want with RegexMagic, even things for which there's no cookie-cutter pattern. Read the how to topic for details.

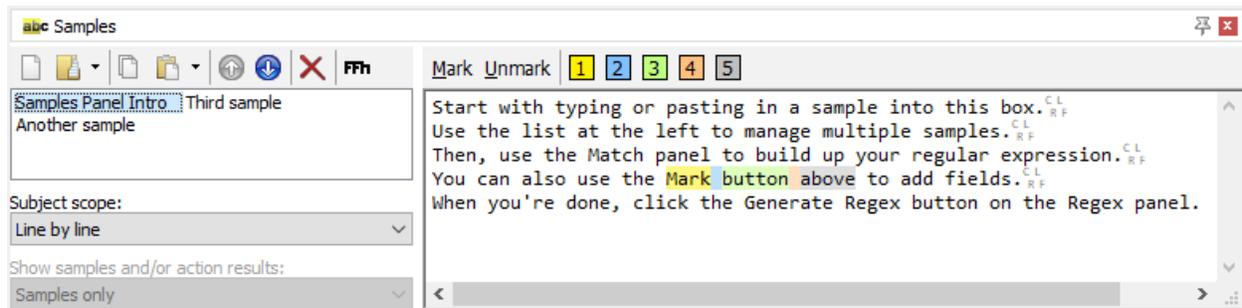
If you want to extract parts of the text matched by the regular expression, or if you want to use the regular expression in a search-and-replace, read the how to create capturing groups and replacement text topic to learn how to use RegexMagic's Action panel.

Finally, all that's left is to generate the regular expression and to put it to use. The how to use your regular expression topic explains the best ways of copying your regular expression from RegexMagic into the application or source code you want to use it with.

When you're done, you can use the Save button on the top toolbar to save your RegexMagic Formula into a file of its own. If you want to store many different formulas, it may be handier to add your formulas to a library instead. Then all your formulas will reside in a single file that you can easily work with on the Library panel.

### 3. How to Add Sample Text

The first thing to do when setting out to create a regular expression with RegexMagic is to provide sample text on the Samples panel. You should provide samples of text that your regular expression should match, as well as samples of text that your regular expression should not match.



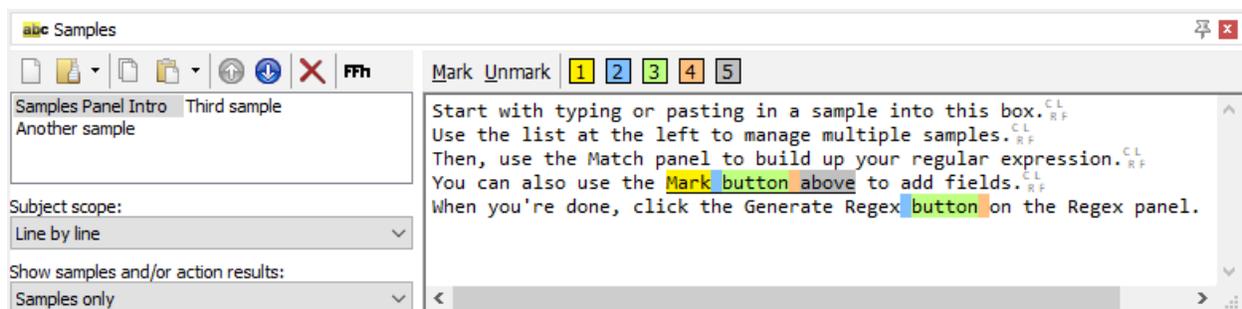
Use the toolbar in the top left corner of the Samples panel to add blank samples, load samples from text files, and paste samples from the clipboard. You can add as many samples to the list as you like. Click on a sample in the list to see and edit its text.

Regular expressions are often applied to short bits of text, such as the text typed into a single line edit control, or a line read from a text file. In situations where your regex will be applied to no more than one line of text at a time, you can set the “subject scope” to “line by line”. RegexMagic then treats each line in your sample text as a separate sample. This way you need to add only one sample to the list on the Samples panel.

The next step is to mark the parts of the samples that your regular expression should match. The How to Create Regular Expressions topic explains this. The screen shot above shows 5 marked fields.

### Checking the Generated Regular Expression

After you’ve generated your regular expression, you’ll come back to the Samples panel to check which text your regular expression actually matches. As the screens shot below shows, the highlighting changes when you generate the regular expression. It indicates the text actually matched by each field.



The five fields that were marked are now highlighted with more intense colors. Intense colors indicate that the regular expression actually matched that part of the text, and the color indicates which field actually matched the text. Muted colors as in the screen shot at the top of this page indicate text that you marked as a certain field, but that was not matched by the regular expression at all.

## 4. How to Create Regular Expressions

In RegexMagic, a regular expression is built from **fields**. A field in RegexMagic is a part of the regular expression that either consists of other fields, or that matches a **pattern**. A pattern describes a basic piece of text for which RegexMagic can generate a regular expression. There are patterns for characters, numbers, dates, email addresses, and many more.

The simplest regular expression you can create in RegexMagic has just one field with one of RegexMagic's patterns. Start off with by clicking the New Formula button on the toolbar at the top to clear out any leftover settings. Then you can create such a regular expression and then clicking the  button on the Match panel. Then use the "pattern to match field" drop-down list to select the pattern you want to base your regular expressions, set the pattern's options, and generate the regex. The email pattern example is a simple example that does this.

The other way to create a regex with a single pattern, after starting with a new formula, is to add and then select the text that the regular expression should match on the Samples panel. Then click the Mark button on the Samples panel. The new field will show up on the Match panel. RegexMagic automatically selects the pattern that best fits the text you marked. Selecting an email address, clicking the Mark button, and then the Generate button on the Regex panel is all you need to do to get a regex that matches any email address. You can find an example of building a longer regex this way in the example explaining the "pattern used by another field" pattern.

By using multiple fields, each of which can have a different pattern, you can create regular expressions that match anything you want. The most basic way to combine fields is to simply add multiple fields, one after the other, to create a regex that matches one pattern after the other. The example on SKU codes combines several basic patterns to match a specialized identification code.

When using multiple fields, you may need to repeat several of those fields as a group. You can do that by putting those fields into a sequence field, and then setting the repetition options for that sequence field. See the example explaining how to repeat combined fields using a sequence field has all the details.

Sometimes, you want to make a regex match one thing or another thing, or one of several things. In RegexMagic, you can do this by selecting "alternation" in the "kind of field" drop-down list for a field. Then you can add a field for each alternative under that field. Matching unrelated items using alternation is a basic example. Matching complex unrelated items using alternation of sequences is a more complex example.

### Generating the Regular Expression

If you want to generate a replacement text so you can do a search-and-replace with your regular expression, follow the topic on creating a replacement text. When you've done that, or if you don't want a replacement text, you're ready to generate the regular expression.

Switch to the Regex panel and select the application or programming language you'll use your regex with from the drop-down list on the toolbar on the Regex panel. This automatically selects the correct regular expression flavor. When selecting a programming language, this also selects the correct string style for the Copy button and the correct source code template for the Use panel. If your application needs to work with multiple applications or multiple versions of the same application, select the primary application or version in the drop-down list and click the Compare button to select all your target applications for comparison.

If your application supports free-spacing regular expressions, push down the  button to get a more readable regular expression. Turn off the  button if you want the whole regex on one line. If your application doesn't have toggle buttons for matching modes such as "case insensitive" or "dot matches line breaks", push down the  button to tell RegexpMagic to use mode modifiers instead. If the  button and/or  button buttons are grayed out, that means the regular expression flavor used by your application doesn't support free-spacing or mode modifiers.

If you're planning to generate a source code snippet on the Use panel, you don't need to worry about the "modifiers" button. The source code snippet will automatically set the correct options, using mode modifiers or another mechanism offered by your programming language or regex library. You only need to toggle the "free-spacing" button depending on how verbose you want your source code to be.

Finally, click the Generate button to generate your regular expression. The Generate button will remain pushed down. If you make changes to the Match panel, your regular expression is automatically updated. While all examples in this help file tell you to push the Generate button at the end, you can actually push it at the start, if you want the Samples panel to highlight your regex match in progress.

You can now check the regular expression matches on the Samples panel, or put your regular expression to good use.

## 5. How to Create Capturing Groups and Replacement Text

In a regular expression, a capturing group isolates a part of the text matched by the regular expression. That allows you to grab part of the regular expression match, or to reuse that part in the replacement text when doing a search-and-replace. In RegexMagic, you can use the Action panel to create capturing groups that you'll use for your own purposes. The example on capturing groups shows you how.

Capturing groups can also be used in search-and-replace actions. By using backreferences to capturing groups in the replacement text, you can reinsert parts of the regex match into the replacement. In RegexMagic, you can work that way with capturing groups, or you can just tell RegexMagic which fields you want to replace, and let RegexMagic figure out which capturing groups and backreferences to use. The replace fields example shows that.

To generate the replacement text, simply generate the regular expression. The replacement text is automatically generated with it.

When the replacement text is generated, you can see the results of the search-and-replace on the Samples panel if you select "samples and replacements" in the "show samples and/or replacements" drop-down list.

## 6. How to Use Your Regular Expressions

Just as RegexMagic makes it easy to create regular expressions, RegexMagic makes it easy to put your regular expressions to use. If you want to search through files and folders, simply switch to the GREP panel in RegexMagic, specify the folder and file masks, and search.

To use your regular expression in another application, click the Copy button on the Regex panel and copy the regular expression “as is”. This copies the regular expression as it appears on the Regex panel in RegexMagic, which is how you should paste it into the Search box of an application that supports regular expressions. Also make note of the “required options”, if any, that you see on the Regex panel. You’ll need to set the same options in the application you’re pasting your regex into. If you don’t, the regex will not work as intended. You can click the Modifiers button (purple tick mark) on the Regex panel if you want a self-contained regex that does not depend on any options. The Modifiers button is disabled for applications that don’t support mode modifiers.

If you want to use your regular expression in the source code of a script or application you’re developing, the best way is to generate a source code snippet on the Use panel. When you select your programming language on the Use panel, RegexMagic automatically selects the correct regex flavor for that language on the Regex panel. The generated source code snippet will have your regular expression properly formatted as a literal string or regex. The source code snippet automatically sets the “dot matches line breaks”, “case insensitive”, and “^ and \$ match at line breaks” options as needed. If your programming language or the function you’re using doesn’t have flags or properties to set those options, RegexMagic automatically adds mode modifiers to the regex, even if you don’t turn on that button on the Regex panel.

Essentially, the Use panel does everything needed to generate a code snippet with the exact same regex that you have in RegexMagic. The most common reason for complaints that “it works in [some regex tool] but not in [some programming language]” and vice versa is that people don’t copy and paste the regex correctly, or forget to set the correct options. Save yourself the hassle and use the Use panel.

If you don’t like RegexMagic’s source code snippets because they don’t fit your coding style, click the Edit button on the toolbar on the Use panel. That button opens RegexMagic’s source code template editor. It allows you to edit all the source code templates to your heart’s content, and even create your own.

If you really want to copy and paste just the regular expression into your source code, click the Copy button and select the string style for your programming language. This copies your regex quoted as a string or literal regex, with the appropriate characters escaped. Note that selecting a string style in the Copy menu does not automatically change the regex flavor. If the regex flavor is set to JavaScript and you click Copy as Python Script, you’ll get a regular expression using JavaScript’s regex syntax quoted as a Python raw string. While that may be useful if a Python server side script needs to pass a regex to a JavaScript client side script, make sure to select the proper regex flavor before copying your regex.

**Part 2**

# **RegexMagic Examples**



# 1. RegexMagic Examples

The best way to learn the ins and outs of RegexMagic is to work through as many examples as you have time for. All examples have full step-by-step instructions. You can do them in any order. The only prerequisite knowledge is given in the getting started topic. But your learning curve will be shallower if you do the examples in order.

## RegexMagic Patterns

Any regular expression you build in RegexMagic will use one or more patterns. In fact, you have to use patterns for all the text that your regular expression should match. A pattern in RegexMagic is a description of a certain piece of text that RegexMagic can generate a regular expression for. Patterns can be simple, matching only certain characters, or quite complex, matching things such as entire email addresses or URLs.

### Regular Expressions Generated from a Single Pattern

The simplest regular expression you can create in RegexMagic consists of just one field that uses one of these patterns. All the examples below create a regular expression using one of the different patterns that RegexMagic provides. These examples are a good way to get started with RegexMagic. They make you familiar with the kinds of patterns that RegexMagic supports. These patterns are the building blocks for regular expressions that match text for which RegexMagic does not provide a ready-made pattern.

- Unicode characters
- Basic characters
- Character masks
- List of literal text
- Literal text
- Number
- Integer
- Date and time
- Email address
- URL
- Country
- Currency
- Credit card number
- National ID
- VAT number
- IPv4 addresses
- GUID
- Regular expression

### Special Patterns Used with Other Patterns

A few of RegexMagic's patterns can only be used (sensibly) in a regular expression that consists of multiple fields. The following examples show each of those patterns.

- Pattern used by another field
- Text matched by another field
- Match anything

## Combining Multiple Patterns

Once you're familiar with the way RegexMagic's patterns work, you can move on to creating regular expressions that use multiple fields to combine different patterns together. This way you can create regular expressions that match pretty much anything you want (within the capabilities of the regular expression syntax).

- Combining patterns using multiple fields
- Repeating combined fields using a sequence field
- Matching unrelated items using alternation
- Matching complex unrelated items using alternation of sequences

## Action Examples

On the Action panel, you can tell RegexMagic that you want to extract part of the text matched by the regular expression, or to use the regular expression in a search-and-replace.

- Capturing groups
- Replace fields
- Replacement text

## Real World Examples

The above examples are all engineered to demonstrate RegexMagic's capabilities in a piecemeal fashion, so you can learn RegexMagic by using it. The examples below put everything together to create regular expressions to solve actual problems, as you will do when using RegexMagic day to day.

- Version numbers
- Variable names
- Matching something within something else
- Java comments
- Incrementing ImageIndex constants

## 2. Pattern: Unicode characters

“Unicode characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of Unicode characters. The repetition settings for the field determine how many characters the field can or must match.

This example shows how you can use the “Unicode characters” pattern to match a currency symbol. You can find this example as “Pattern: Unicode characters” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

Some of the world's major currency symbols are \$, €, ¥, and £.

3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “Unicode characters”.

Individual characters:

Case insensitive

Match all characters except the selected ones

Unicode categories:

<input type="checkbox"/> uppercase letters	<input type="checkbox"/> decimal digits	<input type="checkbox"/> final punctuation	<input type="checkbox"/> paragraph separators
<input type="checkbox"/> lowercase letters	<input type="checkbox"/> letter numbers	<input type="checkbox"/> other punctuation	<input type="checkbox"/> control characters
<input type="checkbox"/> title case letters	<input type="checkbox"/> other numbers	<input type="checkbox"/> math symbols	<input type="checkbox"/> formatting characters
<input type="checkbox"/> modifier letters	<input type="checkbox"/> connector punctuation	<input checked="" type="checkbox"/> currency symbols	<input type="checkbox"/> surrogates
<input type="checkbox"/> letters without case	<input type="checkbox"/> dash punctuation	<input type="checkbox"/> modifier symbols	<input type="checkbox"/> private use characters
<input type="checkbox"/> non-spacing marks	<input type="checkbox"/> open punctuation	<input type="checkbox"/> other symbols	<input type="checkbox"/> unassigned code points
<input type="checkbox"/> spacing combining marks	<input type="checkbox"/> close punctuation	<input type="checkbox"/> space separators	
<input type="checkbox"/> enclosing marks	<input type="checkbox"/> initial punctuation	<input type="checkbox"/> line separators	

6. In the “Unicode categories” list, tick the “currency symbols”.
7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression which matches a single character that is considered to be a currency symbol by the Unicode standard:

```
\p{Sc}
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

8. The Samples panel now shows the regular expression finds 4 different matches in our sample text:

Some of the world's major currency symbols are \$, €, ¥, and £.

### 3. Pattern: Basic characters

“Basic characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of ASCII characters. The repetition settings for the field determine how many characters the field can or must match.

This example shows how you can use the “basic characters” pattern to match an identifier in a hypothetical programming language. This language allows identifiers that consist of any combination of letters, digits, dollar signs, and underscores. You can find this example as “Pattern: basic characters” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:
 

```
$first_and_second = $first + $second
```
3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
4. Click the  button to add field .
5. Tick the “unlimited” checkbox to allow field  to match one or more characters.
6. In the “pattern to match field” drop-down list, select “basic characters”.

Individual characters:

\$\_

Lowercase letters

Uppercase letters

Match all characters except the selected ones

Digits

Punctuation and symbols

Line breaks

Whitespace

Case insensitive

7. Tick the “lowercase letters”, “uppercase letters”, and “digits” checkboxes to allow the pattern to match all ASCII letters and digits.
8. Type a dollar sign and an underscore into the “individual characters” box to allow these two characters as well, without allowing any other punctuation.
9. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
[$0-9A-Z_a-z]+
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

10. The Samples panel now shows the regular expression finds 3 different identifiers in our sample text:

```
$first_and_second = $first + $second
```

## 4. Pattern: Character masks

“Character masks” is one of the patterns that you can select on the Match panel. With this pattern you can easily make a field match a certain combination of letters, digits, and punctuation. This pattern uses text masks similar to the masks used by masked edit controls in various development tools to force the user to enter a certain combination of letters, digits, and/or punctuation.

This example shows how you can use the “character masks” pattern to easily match an SKU code that consists of 6 letters identifying the supplier, a hyphen, and 3 digits indicating the product number. You can find this example as “Pattern: character masks” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

JGSOFT-001

3. On the Match panel, set “begin regex match at” to “start of text”, and set “end regex match at” to “end of text”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “character masks”.



6. Enter this character mask:

LLLLLL-999

7. Set the “field validation mode” to “average”.
8. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

`\A[a-z]{6}-[0-9]{3}\z`

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

9. The Samples panel now shows the regular expression correctly matches the SKU code.:

JGSOFT-001

## 5. Pattern: List of literal text

“List of literal text” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a piece of text from a list that you provide. If you repeat the field, a different item from the list may be matched with each repetition.

This example shows how you can use the “list of literal text” pattern to easily match one of several words. You can find this example as “Pattern: list of literal text” in the RegexpMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

Mary, Sue, and Betty had a little lamb.

3. On the Match panel, set “begin regex match at” to “start of word”, and set “end regex match at” to “end of word”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “list of literal text”.

Case insensitive     Delimit list with page breaks     Match any text except the sp     Can span across lines

Mary<sup>CL</sup><sub>RF</sub>  
 Sue<sup>CL</sup><sub>RF</sub>  
 Betty

6. Enter these three words, one per line:

Mary  
 Sue  
 Betty

7. Set the “field validation mode” to “average”.
8. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b(?:Betty|Mary|Sue)\b
```

**Required options:** Case sensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

9. The Samples panel now shows that this regular expression correctly matches the three names in three separate regex matches:

Mary, Sue, and Betty had a little lamb.

## 6. Pattern: Literal text

“Literal text” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a specific piece of text exactly as you type it in.

This example shows how you can use the “literal text” pattern to make your regular expression match some text literally. You can find this example as “Pattern: literal text” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

RegexMagic is a `*great*` tool for creating regexes!

3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “literal text”.

Case insensitive       Match any text except the specified text       Can span across lines

```
*great*
```

6. Enter this text:

`*great*`

7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression which is the literal text we provided with all metacharacters escaped:

```
\*great\*
```

**Required options:** Case sensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; `^$` don’t match at line breaks; Numbered capture.

8. The Samples panel now shows that this regular expression correctly matches the literal text we typed in:

RegexMagic is a `*great*` tool for creating regexes!

## 7. Pattern: Literal bytes

“Literal bytes” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a specific piece of text exactly as you type it in.

This example shows how you can use the “literal bytes” pattern to make your regular expression match some text literally. You can find this example as “Pattern: literal bytes” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, click the New button to add a new sample and click the FFh button to switch the sample to hexadecimal mode. Then paste this block of bytes into the hex editor:

```
00 10 2A 3B 4C 5D 6E 7F 8A FF 00 10 2A 7F 8A FF
```

3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “literal bytes”.

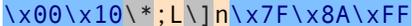
Match any bytes except the specified by  Can span across lines

00000000	 00 10 2A 3B 4C 5D 6E 7F 8A FF	 .*;L]n Šÿ
----------	---	---

6. Enter these bytes in the hexadecimal section:

```
00 10 2A 3B 4C 5D 6E 7F 8A FF
```

7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression. It is the sequence of bytes we entered with printable ASCII characters as literals (escaped if they are metacharacters) and all other bytes as hexadecimal escapes.

```
 \x00\x10\*;L\]n\x7F\x8A\xff
```

**Required options:** Case sensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

## 8. Pattern: Control characters

“Control characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of ASCII characters. The repetition settings for the field determine how many characters the field can or must match.

This example shows how you can use the “control characters” pattern to match an identifier in a hypothetical programming language. This language allows identifiers that consist of any combination of letters, digits, dollar signs, and underscores. You can find this example as “Pattern: control characters” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, click the New button to add a new sample and click the FFh button to switch the sample to hexadecimal mode. Then paste this block of bytes into the hex editor:

```
41 53 43 49 49 20 63 6F 6E 74 72 6F 6C 20 63 68
61 72 61 63 74 65 72 73 3A 00 01 02 03 04 05 06
07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16
17 18 19 1A 1B 1C 1D 1E 1F
```

3. Select byte 09 in the sample and click the Mark button. RegexMagic automatically adds a field using the control characters pattern with character 0x09 selected.
4. Select byte 0B in the sample and click button  above the sample. RegexMagic adds control character 0x0B to the pattern.

Control characters:

<input type="checkbox"/> 00 Null	<input type="checkbox"/> 08 Backspace	<input type="checkbox"/> 10 Data link escape	<input type="checkbox"/> 18 Cancel
<input type="checkbox"/> 01 Start of heading	<input checked="" type="checkbox"/> 09 Character tabulation	<input type="checkbox"/> 11 Device control one	<input type="checkbox"/> 19 End of medium
<input type="checkbox"/> 02 Start of text	<input type="checkbox"/> 0A Line feed (LF)	<input type="checkbox"/> 12 Device control two	<input type="checkbox"/> 1A Substitute
<input type="checkbox"/> 03 End of text	<input checked="" type="checkbox"/> 0B Line tabulation	<input type="checkbox"/> 13 Device control three	<input type="checkbox"/> 1B Escape
<input type="checkbox"/> 04 End of transmission	<input type="checkbox"/> 0C Form feed (FF)	<input type="checkbox"/> 14 Device control four	<input type="checkbox"/> 1C Information separator four
<input type="checkbox"/> 05 Enquiry	<input type="checkbox"/> 0D Carriage return (CR)	<input type="checkbox"/> 15 Negative acknowledge	<input type="checkbox"/> 1D Information separator three
<input type="checkbox"/> 06 Acknowledge	<input type="checkbox"/> 0E Shift out	<input type="checkbox"/> 16 Synchronous idle	<input type="checkbox"/> 1E Information separator two
<input type="checkbox"/> 07 Bell	<input type="checkbox"/> 0F Shift in	<input type="checkbox"/> 17 End of transmission block	<input type="checkbox"/> 1F Information separator one

Match all characters except the selected ones

5. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
[\\t\\v]
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

## 9. Pattern: Number

“Number” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a decimal number that may or may not have decimals, thousand separators, currency symbols, signs, exponents, etc.

This example shows how you can use the “number” pattern to make your regular expression match a pH values. pH values range from 1.0 to 14.0. For this example, we’ll allow one optional decimal. You can find this example as “Pattern: number (basic)” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

Valid pH values:

1  
1.0  
2.9  
4  
7.2  
10.4  
14  
14.0

Invalid pH values:

-7.2  
-4  
.8  
0.9  
14.1  
15.1

3. Set the subject scope to “line by line” to tell RegexMagic to treat each line in our sample as if it were a separate sample. This way we don’t have to provide each sample number separately.
4. On the Match panel, set “begin regex match at” to “start of text”, and set “end regex match at” to “end of text”. We want to create a regex that only matches strings that consist of nothing but a pH number.
5. Click the  button to add field .
6. In the “pattern to match field” drop-down list, select “number”.

Minimum value of integer part: <input type="text" value="1"/>	Maximum value of integer part: <input type="text" value="14"/>	<input checked="" type="checkbox"/> Limit integer part
Decimal separator: <input type="text" value="Period"/>	Min. and max. number of decimals <input type="text" value="0"/> <input type="text" value="1"/>	<input type="checkbox"/> Allow plus sign
Thousand separator: <input type="text" value="None"/>	Currency sign/code position: <input type="text" value="Before only"/>	<input type="checkbox"/> Allow minus sign
Currency sign: <input type="text" value="None"/>	Currency codes: <input type="text"/>	<input type="checkbox"/> Allow parentheses
		<input type="checkbox"/> Sign is required
		<input type="checkbox"/> Whitespace allowed after sign
		<input type="checkbox"/> Thousand separators are required
		<input type="checkbox"/> Allow leading zeros
		<input checked="" type="checkbox"/> Require integer part
		<input type="checkbox"/> Allow exponent
		<input type="checkbox"/> Currency sign or code required

7. Configure the number pattern as in the screen shot: integer range limited from 1 to 14, period as the decimal separator, and one optional decimal.

8. Set the “field validation mode” to “strict”. This is necessary to make RegexpMagic limit the integer range to exactly 1 and 14.
9. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\A(?:1[0-4]|[1-9])(?:\.[0-9])?\z
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

10. The Samples panel now shows which numbers our regular expression matches:

Valid pH values:

```
1
1.0
2.9
4
7.2
10.4
14
14.0
```

Invalid pH values:

```
-7.2
-4
.8
0.9
14.1
15.1
```

## Number with Literal Text Alternatives

The regular expression we just created isn’t perfect. 14.1 is not a valid pH value. The problem is that the “number” pattern in RegexpMagic only has options to limit the range of the integer part, and to limit the number of decimal digits. It doesn’t have an option to limit the decimal part to specific values, so all we could do was to configure the pattern to match numbers between 1 and 14 with one decimal, resulting in a range from 1.0 to 14.9.

We can solve this problem by limiting the number pattern to a range from 1.0 to 13.9. We can then add alternatives to make our regular expression match the 14 and 14.0 values. The “Pattern: number (improved)” example in the RegexpMagic library has the final result. These steps create the same if you continue from the above steps:

12. Change the “maximum value of integer part” from 14 to 13.
13. Select “alternation” in the “kind of field” drop-down list for field **1**. When you do this, RegexpMagic adds field **2** as the first alternative under field **1**, and sets the new field to use the “number” pattern that we configured previously.
14. Click the Add Last Sub-Field button  to add field **3** as the last field under field **1**. The new field defaults to the same “number” pattern.
15. Though we could use the “number” pattern to match “14”, it’s much easier to do this with the “literal text” pattern. The “select field” drop-down list should still have field **3** selected from the previous step. In the “pattern to match field” drop-down list, select “literal text”.

16. Enter 14 in the text box for the “literal text” pattern for field **3**.
17. Click the Add Next Field button  to add field **4** after field **3**. The new field defaults to the same “literal text” pattern.
18. Enter 14.0 in the text box for the “literal text” pattern for field **4**.
19. Make sure the Generate button is still pressed on the Regex panel and you’ll get this regular expression:

```
\A(?: (?: 1[0-3] | [1-9]) (?: \. [0-9])? | 14 | 14\.\0)\z
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

20. The Samples panel now shows that our regular expression correctly matches pH values from 1.0 to 14.0:

Valid pH values:

```
1
1.0
2.9
4
7.2
10.4
14
14.0
```

Invalid pH values:

```
-7.2
-4
.8
0.9
14.1
15.1
```

You’ll notice that three different colors now appear on the Samples panel. The yellow field **1** highlighting we had previously is gone. Field **1** is now an alternation field. Alternation fields don’t match any text directly, but simply alternate between several fields. In our formula, field **1** has three alternatives. Field **2** matches a number between 1.0 and 13.9. Field **3** matches “14”. Field **4** matches “14.0”. The highlighting on the Samples panel indicates which field matched which text.

## 10. Pattern: Integer

“Integer” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an integer number in decimal, hexadecimal, octal, and/or binary notation.

This example shows how you can use the “integer” pattern to look for an integer between 128 and 255 in either decimal or hexadecimal notation in a C-style (C#, Java, etc.) source code file. You can find this example as “Pattern: integer (match)” in the RegexMagic library. You can also watch a video of this example on RegexMagic’s website.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
128 + 0x80 = 256;
90 + 0x90 = 0xB4;
200 + 0x200 = 712;
```

3. On the Match panel, set “begin regex match at” to “start of word”, and set “end regex match at” to “end of word”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “integer”.

Allowed integer formats:

<input checked="" type="checkbox"/> Decimal 987	<input type="checkbox"/> Hexadecimal 3DBh	<input type="checkbox"/> Octal 1733	<input type="checkbox"/> Binary 1111011011
<input type="checkbox"/> Decimal 987d	<input checked="" type="checkbox"/> Hexadecimal 0x3DB	<input type="checkbox"/> Octal 1733o	<input type="checkbox"/> Binary 1111011011b
<input type="checkbox"/> Hexadecimal 3DB	<input type="checkbox"/> Hexadecimal \$3DB	<input type="checkbox"/> Octal 01733	

Limit the integers to these ranges:  Allow leading zeros

128..255

6. In the list of allowed integer formats, check “decimal 987” and “hexadecimal 0x3DB”.
7. Turn on “limit the integers to these ranges”.
8. Enter 128..255 as the range to limit the integers to.
9. Set the “field validation mode” to “strict”. This is necessary to make RegexMagic limit the integer range to exactly 128 and 255.
10. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b(?:25[0-5]||2[0-4][0-9]||1[3-9][0-9]||12[89]||0x[89a-f][0-9a-f])\b
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

11. The Samples panel now highlights the integers between 128 and 255 in decimal and hexadecimal notation:

```
128 + 0x80 = 256;
```

```
90 + 0x90 = 0xB4;
200 + 0x200 = 712;
```

## Marking Samples

Instead of using the Match panel to specify the details of the integer pattern as we did in the above example, we can use the Samples panel to mark the integers we're interested in, and let RegexMagic's auto-detection do the work. For this second example, we'll generate a regular expression that matches any decimal integer between 42 (the answer to everything) and 2552 (the year on the Buddhist calendar that saw the release of RegexMagic 1.0).

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
1
16
42
174
2552
8088
80386
```

3. Select the text you just pasted in, and click the Unmark button on the Samples panel. This removes all fields from the Match panel that may be left over from a previous regular expression.
4. Select the number 42 in the sample text and click the Mark button.
5. Select the number 2552 in the sample text and click the **1** button. It's important to use this button instead of the Mark button, because we want to provide an additional sample for the same field instead of adding a new field.
6. Switch to the Match panel. You'll notice RegexMagic correctly auto-detected the integer pattern with the range 42..2552.
7. Click the "limit the integers to these ranges" checkbox to tell RegexMagic we want to use the minimum and maximum values it auto-detected.
8. Set the "field validation mode" to "strict". This is necessary to make RegexMagic limit the integer between the minimum and maximum values exactly, instead of merely restricting the number of digits.
9. On the Regex panel, select "C# (.NET 2.0–7.0)" as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you'll get this regular expression:

```
\b(?:255[0-2]|25[0-4][0-9]|2[0-4][0-9]{2}|1[0-9]{3}|[1-9][0-9]{2}|[5-9][0-9]
|4[2-9])\b
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

10. The Samples panel now highlights the integers between 42 and 2552:

```
1
16
42
174
```

2552  
8088  
80386

## 11. Pattern: Date and time

“Date and time” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a date, a time, or any part of or any combination of date and time indicators.

This example shows how you can use the “date and time” pattern to look for a date in various formats. You can find this example as “Pattern: date and time” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
8 June 2009
2009-06-08
```

3. On the Match panel, set “begin regex match at” to “start of word”, and set “end regex match at” to “end of word”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “date and time”.

Date separators: Slash, hyphen or dot	Time separators: Colon or dot	AM/PM indicators: AM;PM
Names of months: Jan, January; Feb, February; Mar, March; Apr, April; May, May; Jun, June; Jul, July; Aug, August; Sep, September; Oct, October; Nov, November		
Names of days: Sun, Sunday; Mon, Monday; Tue, Tuesday; Wed, Wednesday; Thu, Thursday; Fri, Friday; Sat, Saturday;		
<input type="checkbox"/> Limit date values	Beginning date: Monday , October 16, 2023	Ending date: Monday , October 16, 2023
<input type="checkbox"/> Limit time values	Beginning time: 12:00:00 AM	Ending time: 1:00:00 AM
Leading zeros: Optional leading zeros		
Allowed date and time formats: 		

6. Enter the names of the months using semicolons to delimit different months, and commas to delimit different spellings of the same month:  
Jan, January; Feb, February; Mar, March; Apr, April; May, May; Jun, June; Jul, July; Aug, August; Sep, September; Oct, October; Nov, November; Dec, December
7. Select “optional leading zeros” in the “leading zeros” drop-down list.
8. Enter the date formats we want to allow, one per line. Right-click the “allowed date and time formats” box to select the format specifiers from a menu.

```
d M Y
Y/m/d
```

9. Set the “field validation mode” to “average”.
10. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```

\b
# 1. Date and time
(?:
# d M Y
(?:3[01]|[12][0-9]|0?[1-9])
[\t ]+(?:Jan|January|Feb|February|Mar|March|Apr|April|May|May|Jun|June|Jul|July|Aug|August|Sep|September|Oct|October|Nov|November|Dec|December)
[\t ]+[0-9]{4}
# Y/m/d
[0-9]{4}[/.-](?:1[0-2]|0?[1-9])[/.-](?:3[01]|[12][0-9]|0?[1-9])
)
\b

```

**Required options:** Case insensitive; Free-spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

11. The Samples panel now highlights the dates our regex matches:

```

8 June 2009
2009-06-08

```

## 12. Pattern: Email address

“Email address” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an email address. You can set the pattern to allow any email address, or only email addresses for specific users or specific domains.

This example shows how you can use the “email” pattern to look for email addresses. You can find this example as “Pattern: email (any)” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
support@regexmagic.com
sales@regexmagic.com
```

3. On the Match panel, set “begin regex match at” to “start of word”, and set “end regex match at” to “end of word”.
4. Click the  button to add field .
5. In the “pattern to match field” drop-down list, select “email”.

User name:	Domain name:	Mailto: prefix:
<input type="text" value="Allow any user name"/>	<input type="text" value="Allow any domain name"/>	<input type="text" value="No prefix"/>

6. Set the “field validation mode” to “average”.
7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b[!#$%&' *+ . / 0-9=? _ ` a-z { | } ~ ^ - ] + @ [ . 0-9 a - z - ] + \ . [ a - z ] { 2 , 6 3 } \b
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

8. The Samples panel now highlights the email addresses our regex matches:

```
support@regexmagic.com
sales@regexmagic.com
```

### Specific Email Addresses

Generating a regex that matches any email address is trivial with RegexMagic. Then again, picking such a regex from an online library of regular expressions is trivial too. What you can do with RegexMagic’s email pattern that you can’t do with copy-and-paste from the web is to generate a regex that matches specific email addresses.

This second example continues from the one above. It shows how you can use the “email” pattern to look for specific email addresses. You can find this example as “Pattern: email (specific)” in the RegexMagic library.

10. On the Match panel, set the “user name” option for the email pattern to “specific user names only”. Enter `support` as the user name we want.
11. Set the “domain name” option for the email pattern to “any domain on specific TLDs”. Enter `com` as the top-level domain (TLD) that we want.

User name:	Domain name:	Mailto: prefix:
Specific user names only	Any domain on specific TLDs	No prefix
support	com	

12. Make sure the Generate button is still pressed on the Regex panel and you’ll get this regular expression:

```
\bsupport@[.0-9a-z-]+\.\com\b
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

13. The Samples panel now shows that our regular expression only matches email addresses on the `.com` domain that have “support” as the user name:

```
support@regexamagic.com
sales@regexamagic.com
```

## 13. Pattern: URL

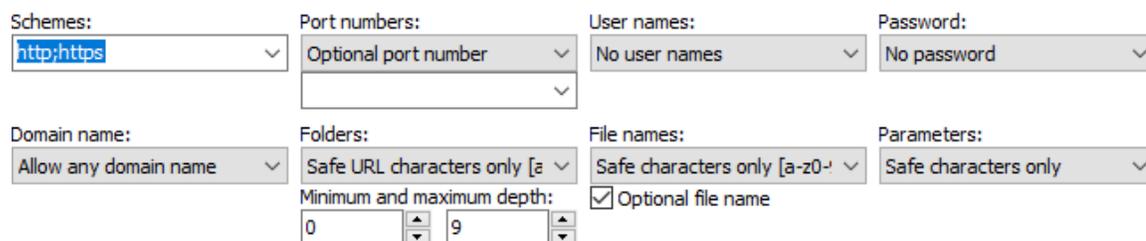
“URL” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an internet address as you would enter into a web browser’s address bar. You can set the pattern to allow any URL, or to require specific schemes, ports, domains, file paths, parameters, etc.

This example shows how you can use the “URL” pattern to look for URLs. You can find this example as “Pattern: URL” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
regexmagic.com
www.regexmagic.com
http://regexmagic.com
http://www.regexmagic.com
http://www.regexmagic.com/
http://www.regexmagic.com/index.html
http://www.regexmagic.com/index.html?source=library
support@regexmagic.com
```

3. On the Match panel, click the  button to add field .
4. In the “pattern to match field” drop-down list, select “URL”.



5. Set the options for the URL pattern as shown in the screen shot.
6. Set the “field validation mode” to “average”.
7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\bhttps?://[.0-9a-z-]+\.[a-z]{2,63}(::[0-9]{1,5})?(?:/([!$'()*+,.0-9_a-z-]
+){0,9}(?:/[!$'()*+,.0-9_a-z-]*)?(?:\?([!$&'()*+,.0-9=_a-z-]*)?)?
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

8. The Samples panel now highlights the URLs our regex matches:

```
regexmagic.com
www.regexmagic.com
http://regexmagic.com
http://www.regexmagic.com
http://www.regexmagic.com/
```

<http://www.regexmagic.com/index.html>  
<http://www.regexmagic.com/index.html?source=library>  
[support@regexmagic.com](mailto:support@regexmagic.com)

## 14. Pattern: Country

“Country” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a country code or country name as defined in the ISO 3166 standard. You can restrict the field to the codes or names of specific countries.

Suppose you have a database of orders for a company that sells worldwide. The database stores a two-letter ISO country code for each order, indicating the customer’s country. You want to retrieve all orders from customers based in the European Union. You want to create a regular expression that matches the country codes of all the EU countries. You can find this example as “Pattern: country” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
US
CA
GB
DE
FR
```

3. On the Match panel, select GB in the sample text and click the Mark button. RegexMagic automatically adds a field using the country pattern. The option “two-letter codes” is automatically selected along with United Kingdom in the list of countries.
4. Tick European Union in the list of country groups. This automatically selects all 26 EU member states. The NATO and OECD boxes are filled in to indicate that some but not all of the countries in these groups are selected. Some EU member states are also members of NATO and the OECD.

<input checked="" type="checkbox"/> Two-letter codes	<input type="checkbox"/> English names	<input type="checkbox"/> French names	<input type="checkbox"/> Make suffixes optional
<b>Country groups:</b>	<b>Countries:</b>		
<input type="checkbox"/> African Union <input type="checkbox"/> Arab League <input type="checkbox"/> ASEAN <input type="checkbox"/> CIS <input checked="" type="checkbox"/> European Union <input checked="" type="checkbox"/> NATO <input checked="" type="checkbox"/> OECD <input type="checkbox"/> Org of American States <input type="checkbox"/> SAARC <input type="checkbox"/> Others	<input type="checkbox"/> AFGHANISTAN <input type="checkbox"/> ÅLAND ISLANDS <input type="checkbox"/> ALBANIA <input type="checkbox"/> ALGERIA <input type="checkbox"/> AMERICAN SAMOA <input type="checkbox"/> ANDORRA <input type="checkbox"/> ANGOLA <input type="checkbox"/> ANGUILLA <input type="checkbox"/> ANTARCTICA <input type="checkbox"/> ANTIGUA AND BARBUDA <input type="checkbox"/> ARGENTINA <input type="checkbox"/> ARMENIA <input type="checkbox"/> ARUBA <input type="checkbox"/> AUSTRALIA <input checked="" type="checkbox"/> AUSTRIA	<input type="checkbox"/> AZERBAIJAN <input type="checkbox"/> BAHAMAS <input type="checkbox"/> BAHRAIN <input type="checkbox"/> BANGLADESH <input type="checkbox"/> BARBADOS <input type="checkbox"/> BELARUS <input checked="" type="checkbox"/> BELGIUM <input type="checkbox"/> BELIZE <input type="checkbox"/> BENIN <input type="checkbox"/> BERMUDA <input type="checkbox"/> BHUTAN <input type="checkbox"/> BOLIVIA, PLURINATIONAL ST <input type="checkbox"/> BONAIRE, SINT EUSTATIUS A <input type="checkbox"/> BOSNIA AND HERZEGOVINA <input type="checkbox"/> BOTSWANA	<input type="checkbox"/> BOUVET ISLAND <input type="checkbox"/> BRAZIL <input type="checkbox"/> BRITISH INDIAN OCEAN TERF <input type="checkbox"/> BRUNEI DARUSSALAM <input checked="" type="checkbox"/> BULGARIA <input type="checkbox"/> BURKINA FASO <input type="checkbox"/> BURUNDI <input type="checkbox"/> CAMBODIA <input type="checkbox"/> CAMEROON <input type="checkbox"/> CANADA <input type="checkbox"/> CAPE VERDE <input type="checkbox"/> CAYMAN ISLANDS <input type="checkbox"/> CENTRAL AFRICAN REPUBLIC <input type="checkbox"/> CHAD <input type="checkbox"/> CHILE

5. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
AT|B[EG]|C[YZ]|D[EK]|E[ES]|F[IR]|G[BR]|H[RU]|I[ET]|L[TUV]|MT|NL|P[LT]|RO|S[EIK]
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

- The Samples panel now highlights the country codes that belong to EU member countries, but not those that belong to other countries:

```
US
CA
GB
DE
FR
```

- On the Use panel, select a database such as MySQL, Oracle, or PostgreSQL in the Language drop-down list.
- In the Function drop-down list, choose “select rows in which a column is entirely matched by the regex”.
- Enter the name of table you're selecting rows from and the name of the column with the country code to generate a SQL statement for selecting all EU orders.

## Country Names with Alternatives

The ISO 3166 standard lists one name in English and one name in French for each country. These are the country names that are built into the “country” pattern in RegexMagic. But many countries are known by various names. Some of the choices made by the ISO committee are very controversial. This example shows how you can generate a regular expression that matches all ISO 3166 names in addition to a list of your own alternative names. You could use this regex to validate country names printed on shipping labels, without forcing the user to adhere to the ISO standard. E.g. you could allow USA and UNITED STATES OF AMERICA in addition to UNITED STATES as specified by ISO. You can find this example as “Pattern: country (alternatives)” in the RegexMagic library.

- Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
- Click the  button to add field .
- In the “pattern to match field” drop-down list, select “country”.
- Right-click on the list of countries and select Check All.
- Tick “English names”, “French names”, and “make suffixes optional” to allow all ISO country names.
- If you generate the regular expression now, you'll get a very long list of country names delimited by vertical bars (the alternation operator in regular expressions).
- Select “alternation” in the “kind of field” drop-down list for field . This automatically moves the country pattern we just created into a new field  that becomes the first alternative of field .
- Click the Add Last Sub-Field button  to add field  as the last field under field .
- In the “pattern to match field” drop-down list, select “list of fixed values”.
- Enter the list of additional country names, one on each line. You can add as many as you like. E.g. enter USA, press Enter to start a new line, and enter UNITED STATES OF AMERICA.
- If you generate the regular expression again, you'll notice RegexMagic adds the additional country names to the end of the list.

## 15. Pattern: State or Province

“State or province” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match the name or code of a state or province from a list of specific states or all states from specific countries.

You could use this pattern to check whether a valid US state name or abbreviation was entered into the state field on an order form. RegexMagic makes it easy to restrict the pattern to only those states that you can actually ship to. In this example, we’ll exclude APO addresses.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
Alabama
AL
Alberta
AB
```

3. Set the subject scope to “line by line”.
4. On the Match panel, select `Alabama` in the sample text.
5. Click the Mark button. RegexMagic automatically adds a field using the state pattern. The option “full names” is automatically selected along with “US States” in the list of countries and Alabama in the list of states.
6. Select `AL` in the sample text.
7. Click the  button. It’s important to use this button instead of the Mark button, because we want to provide an additional sample for the same field instead of adding a new field. RegexMagic automatically detects we also want state codes and ticks the “two-letter codes” checkbox.
8. Right-click on the list of states and select Check All. This adds all US states to the pattern.
9. At the end of the list, uncheck the three Armed Forces “states”.

<input checked="" type="checkbox"/> Two-letter codes	<input checked="" type="checkbox"/> Full names																																													
<b>Countries:</b>	<b>States and Provinces:</b>																																													
<input type="checkbox"/> Canadian Provinces <input checked="" type="checkbox"/> US States	<table border="1"> <tr><td><input checked="" type="checkbox"/> Alabama</td><td><input checked="" type="checkbox"/> Iowa</td><td><input checked="" type="checkbox"/> New Jersey</td></tr> <tr><td><input checked="" type="checkbox"/> Alaska</td><td><input checked="" type="checkbox"/> Kansas</td><td><input checked="" type="checkbox"/> New Mexico</td></tr> <tr><td><input checked="" type="checkbox"/> Arizona</td><td><input checked="" type="checkbox"/> Kentucky</td><td><input checked="" type="checkbox"/> New York</td></tr> <tr><td><input checked="" type="checkbox"/> Arkansas</td><td><input checked="" type="checkbox"/> Louisiana</td><td><input checked="" type="checkbox"/> North Carolina</td></tr> <tr><td><input checked="" type="checkbox"/> California</td><td><input checked="" type="checkbox"/> Maine</td><td><input checked="" type="checkbox"/> North Dakota</td></tr> <tr><td><input checked="" type="checkbox"/> Colorado</td><td><input checked="" type="checkbox"/> Maryland</td><td><input checked="" type="checkbox"/> Ohio</td></tr> <tr><td><input checked="" type="checkbox"/> Connecticut</td><td><input checked="" type="checkbox"/> Massachusetts</td><td><input checked="" type="checkbox"/> Oklahoma</td></tr> <tr><td><input checked="" type="checkbox"/> Delaware</td><td><input checked="" type="checkbox"/> Michigan</td><td><input checked="" type="checkbox"/> Oregon</td></tr> <tr><td><input checked="" type="checkbox"/> District of Columbia</td><td><input checked="" type="checkbox"/> Minnesota</td><td><input checked="" type="checkbox"/> Pennsylvania</td></tr> <tr><td><input checked="" type="checkbox"/> Florida</td><td><input checked="" type="checkbox"/> Mississippi</td><td><input checked="" type="checkbox"/> Rhode Island</td></tr> <tr><td><input checked="" type="checkbox"/> Georgia</td><td><input checked="" type="checkbox"/> Missouri</td><td><input checked="" type="checkbox"/> South Carolina</td></tr> <tr><td><input checked="" type="checkbox"/> Hawaii</td><td><input checked="" type="checkbox"/> Montana</td><td><input checked="" type="checkbox"/> South Dakota</td></tr> <tr><td><input checked="" type="checkbox"/> Idaho</td><td><input checked="" type="checkbox"/> Nebraska</td><td><input checked="" type="checkbox"/> Tennessee</td></tr> <tr><td><input checked="" type="checkbox"/> Illinois</td><td><input checked="" type="checkbox"/> Nevada</td><td><input checked="" type="checkbox"/> Texas</td></tr> <tr><td><input checked="" type="checkbox"/> Indiana</td><td><input checked="" type="checkbox"/> New Hampshire</td><td><input checked="" type="checkbox"/> Utah</td></tr> </table>	<input checked="" type="checkbox"/> Alabama	<input checked="" type="checkbox"/> Iowa	<input checked="" type="checkbox"/> New Jersey	<input checked="" type="checkbox"/> Alaska	<input checked="" type="checkbox"/> Kansas	<input checked="" type="checkbox"/> New Mexico	<input checked="" type="checkbox"/> Arizona	<input checked="" type="checkbox"/> Kentucky	<input checked="" type="checkbox"/> New York	<input checked="" type="checkbox"/> Arkansas	<input checked="" type="checkbox"/> Louisiana	<input checked="" type="checkbox"/> North Carolina	<input checked="" type="checkbox"/> California	<input checked="" type="checkbox"/> Maine	<input checked="" type="checkbox"/> North Dakota	<input checked="" type="checkbox"/> Colorado	<input checked="" type="checkbox"/> Maryland	<input checked="" type="checkbox"/> Ohio	<input checked="" type="checkbox"/> Connecticut	<input checked="" type="checkbox"/> Massachusetts	<input checked="" type="checkbox"/> Oklahoma	<input checked="" type="checkbox"/> Delaware	<input checked="" type="checkbox"/> Michigan	<input checked="" type="checkbox"/> Oregon	<input checked="" type="checkbox"/> District of Columbia	<input checked="" type="checkbox"/> Minnesota	<input checked="" type="checkbox"/> Pennsylvania	<input checked="" type="checkbox"/> Florida	<input checked="" type="checkbox"/> Mississippi	<input checked="" type="checkbox"/> Rhode Island	<input checked="" type="checkbox"/> Georgia	<input checked="" type="checkbox"/> Missouri	<input checked="" type="checkbox"/> South Carolina	<input checked="" type="checkbox"/> Hawaii	<input checked="" type="checkbox"/> Montana	<input checked="" type="checkbox"/> South Dakota	<input checked="" type="checkbox"/> Idaho	<input checked="" type="checkbox"/> Nebraska	<input checked="" type="checkbox"/> Tennessee	<input checked="" type="checkbox"/> Illinois	<input checked="" type="checkbox"/> Nevada	<input checked="" type="checkbox"/> Texas	<input checked="" type="checkbox"/> Indiana	<input checked="" type="checkbox"/> New Hampshire	<input checked="" type="checkbox"/> Utah
<input checked="" type="checkbox"/> Alabama	<input checked="" type="checkbox"/> Iowa	<input checked="" type="checkbox"/> New Jersey																																												
<input checked="" type="checkbox"/> Alaska	<input checked="" type="checkbox"/> Kansas	<input checked="" type="checkbox"/> New Mexico																																												
<input checked="" type="checkbox"/> Arizona	<input checked="" type="checkbox"/> Kentucky	<input checked="" type="checkbox"/> New York																																												
<input checked="" type="checkbox"/> Arkansas	<input checked="" type="checkbox"/> Louisiana	<input checked="" type="checkbox"/> North Carolina																																												
<input checked="" type="checkbox"/> California	<input checked="" type="checkbox"/> Maine	<input checked="" type="checkbox"/> North Dakota																																												
<input checked="" type="checkbox"/> Colorado	<input checked="" type="checkbox"/> Maryland	<input checked="" type="checkbox"/> Ohio																																												
<input checked="" type="checkbox"/> Connecticut	<input checked="" type="checkbox"/> Massachusetts	<input checked="" type="checkbox"/> Oklahoma																																												
<input checked="" type="checkbox"/> Delaware	<input checked="" type="checkbox"/> Michigan	<input checked="" type="checkbox"/> Oregon																																												
<input checked="" type="checkbox"/> District of Columbia	<input checked="" type="checkbox"/> Minnesota	<input checked="" type="checkbox"/> Pennsylvania																																												
<input checked="" type="checkbox"/> Florida	<input checked="" type="checkbox"/> Mississippi	<input checked="" type="checkbox"/> Rhode Island																																												
<input checked="" type="checkbox"/> Georgia	<input checked="" type="checkbox"/> Missouri	<input checked="" type="checkbox"/> South Carolina																																												
<input checked="" type="checkbox"/> Hawaii	<input checked="" type="checkbox"/> Montana	<input checked="" type="checkbox"/> South Dakota																																												
<input checked="" type="checkbox"/> Idaho	<input checked="" type="checkbox"/> Nebraska	<input checked="" type="checkbox"/> Tennessee																																												
<input checked="" type="checkbox"/> Illinois	<input checked="" type="checkbox"/> Nevada	<input checked="" type="checkbox"/> Texas																																												
<input checked="" type="checkbox"/> Indiana	<input checked="" type="checkbox"/> New Hampshire	<input checked="" type="checkbox"/> Utah																																												

10. On the Match panel, set “begin regex match at” to “start of text”, and set “end regex match at” to “end of text”.
11. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```

\A(?:Ala(?:?:bam|sk)a|Arizona|Arkansas|California|Colorado|Connecticut|De
laware|District of
Columbia|Florida|Georgia|Hawaii|Idaho|Illinois|Indiana|Iowa|Kansas|Kentucky
|Louisiana|Maine|Maryland|Massachusetts|Michigan|Minnesota|Miss(?:?:issipp
four)i|Montana|Nebraska|Nevada|New(?:?:Hampshire|Jersey|Mexico|York)|North
(?:?:Carolin|Dakot)a|Ohio|Oklahoma|Oregon|Pennsylvania|Rhode Island|South
(?:?:Carolin|Dakot)a|Tennessee|Texas|Utah|Vermont|Virginia|Washington|Wes
t
Virginia|Wisconsin|Wyoming|A[KLRZ]|C[AOT]|D[CE]|FL|GA|HI|I[ADLN]|K[SY]|LA|M
[ADEINOST]|N[CDEHJMVY]|O[HKR]|PA|RI|S[CD]|T[NX]|UT|V[AT]|W[AIVY])\Z

```

**Required options:** Case insensitive; Exact spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

12. The Samples panel now highlights the US states, but not states or provinces from other countries:

```

Alabama
AL
Alberta
AB

```

## States or Provinces from Other Countries

If the RegexMagic pattern for states and provinces does not support your country, you can use the “list of literal text” pattern. Type or paste in the list of states or provinces for your country and the pattern will match any one of them.

## 16. Pattern: Currency

“Currency” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a currency code as defined in the ISO 4217 standard. You can restrict the field to specific currencies.

Suppose you have a database of orders for a company that accepts various currencies. The database stores a three-letter ISO currency code for each order. You want to retrieve all orders in euro (and British pounds) from customers based in the European Union. You want to create a regular expression that matches these two currency codes. You can find this example as “Pattern: currency” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
USD
CAD
EUR
GBP
```

3. On the Match panel, select EUR in the sample text and click the Mark button. RegexMagic automatically adds a field using the currency pattern with “EUR Euro” selected in the list of currencies.
4. Select GBP in the sample text and click button . RegexMagic automatically selects “GBP Pound sterling” in the list of currencies.

Show special currency codes       Show obsolete currencies

Currencies:

<input type="checkbox"/> AED United Arab Emirates dirh	<input type="checkbox"/> BIF Burundian franc	<input type="checkbox"/> COP Colombian peso	<input type="checkbox"/> FJD Fiji dollar
<input type="checkbox"/> AFN Afghan afghani	<input type="checkbox"/> BMD Bermudian dollar	<input type="checkbox"/> COU Unidad de Valor Real	<input type="checkbox"/> FKP Falkland Islands pound
<input type="checkbox"/> ALL Albanian lek	<input type="checkbox"/> BND Brunei dollar	<input type="checkbox"/> CRC Costa Rican colon	<input checked="" type="checkbox"/> GBP Pound sterling
<input type="checkbox"/> AMD Armenian dram	<input type="checkbox"/> BOB Boliviano	<input type="checkbox"/> CUC Cuban convertible peso	<input type="checkbox"/> GEL Georgian lari
<input type="checkbox"/> ANG Netherlands Antillean guilder	<input type="checkbox"/> BRL Brazilian real	<input type="checkbox"/> CUP Cuban peso	<input type="checkbox"/> GHS Ghanaian cedi
<input type="checkbox"/> AOA Angolan kwanza	<input type="checkbox"/> BSD Bahamian dollar	<input type="checkbox"/> CVE Cape Verde escudo	<input type="checkbox"/> GIP Gibraltar pound
<input type="checkbox"/> ARS Argentine peso	<input type="checkbox"/> BTN Bhutanese ngultrum	<input type="checkbox"/> CZK Czech koruna	<input type="checkbox"/> GMD Gambian dalasi
<input type="checkbox"/> AUD Australian dollar	<input type="checkbox"/> BWP Botswana pula	<input type="checkbox"/> DJF Djiboutian franc	<input type="checkbox"/> GNF Guinean franc
<input type="checkbox"/> AWG Aruban florin	<input type="checkbox"/> BYR Belarusian ruble	<input type="checkbox"/> DKK Danish krone	<input type="checkbox"/> GTQ Guatemalan quetzal
<input type="checkbox"/> AZN Azerbaijani manat	<input type="checkbox"/> BZD Belize dollar	<input type="checkbox"/> DOP Dominican peso	<input type="checkbox"/> GYD Guyanese dollar
<input type="checkbox"/> BAM Bosnia and Herzegovina	<input type="checkbox"/> CAD Canadian dollar	<input type="checkbox"/> DZD Algerian dinar	<input type="checkbox"/> HKD Hong Kong dollar
<input type="checkbox"/> BBD Barbados dollar	<input type="checkbox"/> CDF Congolese franc	<input type="checkbox"/> EGP Egyptian pound	<input type="checkbox"/> HNL Honduran lempira
<input type="checkbox"/> BDT Bangladeshi taka	<input type="checkbox"/> CHF Swiss franc	<input type="checkbox"/> ERN Eritrean nakfa	<input type="checkbox"/> HRK Croatian kuna
<input type="checkbox"/> BGN Bulgarian lev	<input type="checkbox"/> CLP Chilean peso	<input type="checkbox"/> ETB Ethiopian birr	<input type="checkbox"/> HTG Haitian gourde
<input type="checkbox"/> BHD Bahraini dinar	<input type="checkbox"/> CNY Renminbi	<input checked="" type="checkbox"/> EUR Euro	<input type="checkbox"/> HUF Hungarian forint

5. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
EUR|GBP
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

6. On the Use panel, select a database such as MySQL, Oracle, or PostgreSQL in the Language drop-down list.

7. In the Function drop-down list, choose “select rows in which a column is entirely matched by the regex”.
8. Enter the name of table you’re selecting rows from and the name of the column with the currency code to generate a SQL statement for selecting all orders in euro and pounds.

## 17. Pattern: Credit card number

“Credit card number” is one of the patterns that you can select on the Match panel. Use this pattern to match a series of digits that looks like a credit card number from one or more of the world’s major credit card issuers.

This example shows how you can use the “credit card” pattern to look for credit card numbers. You can find this example as “Pattern: credit card number” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample that has a block of valid card numbers and a block of invalid card numbers for each of the major credit card brands:

```
Visa
4123456789012
4123456789012345
412345678901
41234567890123
412345678901234
41234567890123456
MasterCard
5123456789012345
512345678901234
51234567890123455
Discover
6011123456789012
6512345678901234
601112345678901
60111234567890123
American Express
341234567890123
371234567890123
34123456789012
301234567890123
3712345678901234
Diner's Club
30012345678901
30512345678901
36012345678901
36912345678901
38012345678901
38912345678901
30612345678901
37012345678901
JCB
213112345678901
180012345678901
3512345678901234
21311234567890
18001234567890
2131123456789012
1800123456789012
35123456789012
Random
```

123456789012345

- On the Match panel, click the  button to add field .
- In the “pattern to match field” drop-down list, select “credit card number”.

<input checked="" type="checkbox"/> Visa	<input checked="" type="checkbox"/> Diners Club	<input checked="" type="radio"/> No spaces or dashes
<input checked="" type="checkbox"/> MasterCard	<input checked="" type="checkbox"/> Discover	<input type="radio"/> Spaces and dashes to group digits
<input checked="" type="checkbox"/> American Express	<input checked="" type="checkbox"/> JCB	<input type="radio"/> Spaces and dashes anywhere

- Tick all the checkboxes for the credit card brands.
- Set the “field validation mode” to “average”.
- On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b(?:4[0-9]{12}(?:[0-9]{3})?|(?:5[1-5][0-9]{2}|2720|27[01][0-9]|2[3-6][0-9]{2}|22[3-9][0-9]|222[1-9])[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|68)[0-9]{0-9})[0-9]{11}|6(?:011|5[0-9]{2})[0-9]{12}|(?:2131|1800|35[0-9]{3})[0-9]{11})\b
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

- The Samples panel now highlights the valid credit card numbers matched by our regular expression:

```

Visa
4123456789012
4123456789012345
412345678901
41234567890123
412345678901234
41234567890123456
MasterCard
5123456789012345
512345678901234
51234567890123455
Discover
6011123456789012
6512345678901234
601112345678901
60111234567890123
American Express
341234567890123
371234567890123
34123456789012
301234567890123
3712345678901234
Diner's Club
30012345678901
30512345678901
36012345678901
36912345678901
38012345678901
38912345678901
30612345678901
37012345678901

```

JCB

213112345678901

180012345678901

3512345678901234

21311234567890

18001234567890

2131123456789012

1800123456789012

35123456789012

Random

123456789012345

## 18. Pattern: National ID

“National ID” is one of the patterns that you can select on the Match panel. Use it to make a field match a personal identification code used in a specific country such as social security numbers, identity numbers, citizen numbers, tax numbers, license plate numbers, etc.

This example shows how you can use the “national ID” pattern to look for US social security numbers. You can find this example as “Pattern: national ID” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample that has a block of valid and a block of invalid US social security numbers:

```
Valid:
123-12-1234
078-05-1120
Invalid:
123121234
1234-12-1234
000-00-0000
```

3. On the Match panel, click the  button to add field .
4. In the “pattern to match field” drop-down list, select “national ID”.

Kind of ID:

5. In the “kind of ID” drop-down list, select “US social security number”.
6. Set the “field validation mode” to “strict”, to make sure the “national ID” pattern generates a regex that excludes invalid social security numbers such as those beginning with 000 or 666.
7. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b(?:000|666)[012345678][0-9]{2}-(?:00)[0-9]{2}-(?:0000)[0-9]{4}\b
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

8. The Samples panel now highlights the valid social security numbers matched by our regular expression:

```
Valid:
123-12-1234
078-05-1120
Invalid:
123121234
1234-12-1234
000-00-0000
```

## 19. Pattern: VAT number

“VAT number” is one of the patterns that you can select on the Match panel. Use it to make a field match a value added tax number from one or more European countries.

This example shows how you can use the “VAT number” pattern to check for VAT numbers from the Benelux countries. You can find this example as “Pattern: VAT number” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample that has a block of valid and a block of invalid VAT numbers:

```
ATU12345678
BE123456789
BE0123456789
BG123456789
BG0123456789
CY12345678L
CZ12345678
CZ123456789
CZ1234567890
DE123456789
DK12345678
EE123456789
GR123456789
EL123456789
ES012345678
ESX1234567Y
FI12345678
FR12123456789
FRAB123456789
GB123456789123
GB123456789AB1
HU12345678
IE1S12345L
IT12345678901
LU12345678
LT123456789
LT123456789012
LV12345678901
MT12345678
NL123456789B01
PL1234567890
PT123456789
RO12345678
R0123456789
R01234567890
SE123456789012
SI12345678
SK1234567890
```

3. On the Match panel, click the  button to add field .

- In the “pattern to match field” drop-down list, select “VAT number”.

Countries:

<input type="checkbox"/> AT Austria	<input type="checkbox"/> DK Denmark	<input type="checkbox"/> GB United Kingdom	<input checked="" type="checkbox"/> LU Luxembourg	<input type="checkbox"/> RO Romania
<input checked="" type="checkbox"/> BE Belgium	<input type="checkbox"/> EE Estonia	<input type="checkbox"/> HR Croatia	<input type="checkbox"/> LV Latvia	<input type="checkbox"/> SE Sweden
<input type="checkbox"/> BG Bulgaria	<input type="checkbox"/> EL Greece	<input type="checkbox"/> HU Hungary	<input type="checkbox"/> MT Malta	<input type="checkbox"/> SI Slovenia
<input type="checkbox"/> CY Cyprus	<input type="checkbox"/> ES Spain	<input type="checkbox"/> IE Ireland	<input checked="" type="checkbox"/> NL Netherlands	<input type="checkbox"/> SK Slovakia
<input type="checkbox"/> CZ Czech Republic	<input type="checkbox"/> FI Finland	<input type="checkbox"/> IT Italy	<input type="checkbox"/> PL Poland	<input type="checkbox"/> XI Northern Ireland
<input type="checkbox"/> DE Germany	<input type="checkbox"/> FR France	<input type="checkbox"/> LT Lithuania	<input type="checkbox"/> PT Portugal	

Country code:  Grouping characters:

- In the list of countries, tick Belgium, The Netherlands, and Luxembourg.
- In the “country code” drop-down list, select “require country code”.
- In the “grouping characters” drop-down list, select “none”.
- Set the “field validation mode” to “strict”.
- On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\b(?:LU[0-9]{8}|BE0[0-9]{9}|NL[0-9]{9}B[0-9]{2})\b
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

- The Samples panel now highlights the valid social security numbers matched by our regular expression:

```
ATU12345678
BE123456789
BE0123456789
BG123456789
BG0123456789
CY12345678L
CZ12345678
CZ123456789
CZ1234567890
DE123456789
DK12345678
EE123456789
GR123456789
EL123456789
ES012345678
ESX1234567Y
FI12345678
FR12123456789
FRAB123456789
GB123456789123
GB123456789AB1
HU12345678
IE1S12345L
IT12345678901
LU12345678
LT123456789
LT123456789012
LV12345678901
MT12345678
```

NL123456789B01  
PL1234567890  
PT123456789  
R012345678  
R0123456789  
R01234567890  
SE123456789012  
SI12345678  
SK1234567890

## 20. Pattern: IPv4 Address

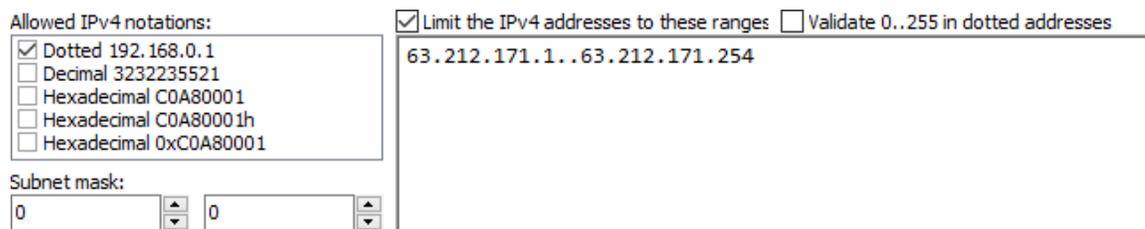
“IPv4 address” is one of the patterns that you can select on the Match panel. Use it to make a field match a internet address such as `192.168.0.1`.

The documentation for Google Analytics explains how you can use a regular expression in Google Analytics to filter on IP addresses. We’ll follow the same example here, but explain how to generate the regular expression with RegexMagic.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
10.1.1.1
10.255.255.254
172.0.0.1
172.16.0.1
172.31.255.254
172.255.255.254
192.168.1.1
192.168.255.254
63.212.171.1
63.212.171.254
8.8.8.8
8.8.4.4
```

3. Set the subject scope to “line by line”.
4. On the Match panel, set “begin regex match at” to “start of text”, and set “end regex match at” to “end of text”.
5. Click the  button to add field .
6. In the “pattern to match field” drop-down list, select “IPv4 address”. By default, this pattern matches any IPv4 address in dotted notation.



Allowed IPv4 notations:

- Dotted 192.168.0.1
- Decimal 3232235521
- Hexadecimal COA80001
- Hexadecimal COA80001h
- Hexadecimal 0xCOA80001

Subnet mask:

Limit the IPv4 addresses to these ranges  Validate 0..255 in dotted addresses

63.212.171.1..63.212.171.254

7. Turn on “limit the IPv4 addresses to these ranges”.
8. Enter `63.212.171.1..63.212.171.254` as the range to limit the IP addresses to.
9. Set the “field validation mode” to “strict”. This is necessary to make RegexMagic limit the range of IP addresses to exactly what you specified.
10. On the Regex panel, select “POSIX ERE” as your application. This is the regular expression flavor used by Google Analytics. It is limited in features, but offers all you need to match IP address ranges.
11. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
^63\.212\.171\.([25[0-4]|2[0-4][0-9]|1[0-9]{2}|[1-9][0-9]?)$
```

**Required options:** ^\$ don't match at line breaks; dot matches line breaks.

**Unused options:** Case sensitive.

12. The Samples panel now highlights the IP addresses in the range that we specified:

```
10.1.1.1
10.255.255.254
172.0.0.1
172.16.0.1
172.31.255.254
172.255.255.254
192.168.1.1
192.168.255.254
63.212.171.1
63.212.171.254
8.8.8.8
8.8.4.4
```

## Keeping The Regex Short Enough

The regex engine used by Google Analytics does not support regular expressions longer than 255 characters. The POSIX ERE specification allows this limitation but does not require it. RegexMagic always tries to generate regular expressions that are as short and simple as possible. But it will allow the regular expression to be as long as it needs to be to make it do what you specified. If the resulting regex is too long, RegexMagic does provide several options to make it shorter if you allow a change in specifications.

To keep your regex short, turn off the option “validate 0..255 in dotted addresses” and specify ranges that span the whole 0..255 range as much as possible. So specify 63.212.171.0..63.212.171.255 or 63.212.171.1/24 instead of 63.212.171.1..63.212.171.254. When you do this, RegexMagic generates `[0-9]{1,3}` rather than `(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])` to match a number between 0 and 255. Though `[0-9]{1,3}` could match numbers such as 000 or 999 that don't belong in IPv4 addresses, this isn't a problem when filtering web logs. The logs don't contain invalid IP addresses. There's no need to make the regex longer in order to exclude them. The regex from our example then becomes:

```
^63\.212\.171\.[0-9]{1,3}$
```

**Required options:** ^\$ don't match at line breaks; dot matches line breaks.

**Unused options:** Case sensitive.

The RegexMagic pattern for IPv4 addresses allows you to specify as many IP ranges as you want delimited with semicolon. RegexMagic will roll all the ranges into one big regex. You can match any private IPv4 address by setting the range to 10.0.0.0/8;172.16.0.0/12;192.168.0.0/16. RegexMagic combines these 3 ranges into one compact regex that still strictly matches all 3:

```
^(10\.[0-9]{1,3}|172\.[3[01]|2[0-9]|1[6-9])|192\.168)\.[0-9]{1,3}\.[0-9]{1,3}$
```

**Required options:** ^\$ don't match at line breaks; dot matches line breaks.

**Unused options:** Case sensitive.

While Google Analytics can only handle regular expressions up to 255 characters, it does allow you to use multiple filters, all with their own regular expression. So if you want to filter on multiple IP address ranges and the regex RegexMagic generates for all those ranges is too long, have RegexMagic generate one regex for each range, and use those regexes in separate filters in Google Analytics.

Setting the “field validation mode” to “strict” as this example does is appropriate when a regular expression is all you can use to restrict the range of IP addresses. But in other situations, such as when developing your own software, you may be better off with a simpler regex that just matches any combination of 4 numbers with dots between them, and filter the IP addresses in procedural code. If you’ll be processing the matched addresses in procedural code anyway, you’ll likely get better performance from a simple regex with a few extra checks in procedural code.

If you set the “field validation mode” to “average”, RegexMagic will generate a regular expression that is much shorter. The regex still requires a match in the form of 1.2.3.4. But if you set the pattern to restrict the address to certain ranges, the regex may match some addresses beyond the ranges you specified. For example, using “average” field validation with the range set to 10.0.0.0/8;172.16.0.0/12;192.168.0.0/16, the resulting regex will match any IPv4 address that begins with 10, 172, or 192. That includes some addresses that aren’t private IPv4 addresses. But you can see that the “average” regex is simpler than the previous “strict” regex:

```
^(10|172|192)\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$
```

**Required options:** ^\$ don’t match at line breaks; dot matches line breaks.

**Unused options:** Case sensitive.

The more complex your combination of IP ranges, the bigger the difference between “average” and “strict” modes. If “average” isn’t simple enough, “loose” mode grabs any IPv4 address, as if both “limit the IPv4 addresses to these ranges” and “validate 0..255 in dotted addresses” were turned off:

```
^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$
```

**Required options:** ^\$ don’t match at line breaks; dot matches line breaks.

**Unused options:** Case sensitive.

## 21. Pattern: GUID

“GUID” is one of the patterns that you can select on the Match panel. Use it to make a field match a globally unique identifier such as {7E9081B5-9A6D-4CC1-A8C3-47F69FB4198D}.

This example shows how you can use the “GUID” pattern to validate some text as being a proper GUID formatted with braces and hyphens.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample that has a block of valid and a block of invalid GUIDs:

```
{D1A5279D-B27D-4CD4-A05E-EFDD53D08E8D}
{594ABD08-C02C-4827-8ABF-A128006848A3}
{B59511BD6A5F4DF09ECF562A108D8A2E}
69593D62-71EA-4548-85E4-04FC71357423
677E2553DD4D43B09DA77414DB1EB8EA
{99367d71-08ad-4aec-8e73-55ae151614f9}
{5ba3bba3-729a-4717-88c1-b7c4b7ba80db}
{7e9081b59a6d4cc1a8c347f69fb4198d}
0c74f13f-fa83-4c48-9b33-68921dd72463
b4b2fb69c6244e5eb0698e0c6ec66618
DE011585-7317-4C0D-A9AF-0AF2C5E61DB8}
{E8038A99-A409-490B-B11A-159A3496426D}
{D44EF4F4-280B47E5-91C7-261222A59621}
{5EDEB36C-9006-467A8D04-AFB6F62CD7D2}
{D995AC86-5697-4683-A34F2B778BD837ED}
{283B67B2-430F-4E6F-97E6-19041992-C1B0}
2F2B15A5-2615-4748-BDABA124210F15EC
07D267AB-7126-45EB8FFC-51994D05F0B2
```

3. Select the text you just pasted in, and click the Unmark button on the Samples panel. This removes all fields from the Match panel that may be left over from a previous regular expression.
4. Set the subject scope to “line by line”.
5. Select the first GUID {D1A5279D-B27D-4CD4-A05E-EFDD53D08E8D} in the sample text.
6. Click the Mark button. RegexMagic automatically adds a field using the GUID pattern. It automatically detects the GUID has braces, hyphens, and is uppercase and sets those options.
7. In the block with lowercase GUIDs in the sample, select the first GUID {99367d71-08ad-4aec-8e73-55ae151614f9}.
8. Click the  button. It’s important to use this button instead of the Mark button, because we want to provide an additional sample for the same field instead of adding a new field. RegexMagic automatically determines that the GUID pattern is still the proper pattern for field . It also determines that both GUIDs have braces and hyphens, and that one is uppercase and the other lowercase. RegexMagic changes the pattern to be case insensitive.
9. On the Match panel, set “begin regex match at” to “start of text”, and set “end regex match at” to “end of text”.
10. Set the “field validation mode” to “strict”.
11. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\A\{[0-9A-F]{8}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{12}\}\z
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

12. The Samples panel now highlights all the valid GUIDs matched by our regular expression:

```
{D1A5279D-B27D-4CD4-A05E-EFDD53D08E8D}
{594ABD08-C02C-4827-8ABF-A128006848A3}
{B59511BD6A5F4DF09ECF562A108D8A2E}
69593D62-71EA-4548-85E4-04FC71357423
677E2553DD4D43B09DA77414DB1EB8EA
{99367d71-08ad-4aec-8e73-55ae151614f9}
{5ba3bba3-729a-4717-88c1-b7c4b7ba80db}
{7e9081b59a6d4cc1a8c347f69fb4198d}
0c74f13f-fa83-4c48-9b33-68921dd72463
b4b2fb69c6244e5eb0698e0c6ec66618
DE011585-7317-4C0D-A9AF-0AF2C5E61DB8}
{E8038A99-A409-490B-B11A-159A3496426D}
{D44EF4F4-280B47E5-91C7-261222A59621}
{5EDEB36C-9006-467A8D04-AFB6F62CD7D2}
{D995AC86-5697-4683-A34F2B778BD837ED}
{283B67B2-430F-4E6F-97E6-19041992-C1B0}
2F2B15A5-2615-4748-BDABA124210F15EC
07D267AB-7126-45EB8FFC-51994D05F0B2
```

## Matching Specific GUIDs

The GUID pattern can only be used to match arbitrary GUIDs. If you want to match specific GUIDs, use the “literal text” pattern to match a single GUID, or the “list of literal text” pattern to match one from a list of GUIDs.

Continuing from the above example, to get a regex that matches only those two GUIDs that we marked, select “list of literal text” in the “pattern to match field” drop-down list. RegexMagic automatically adds the two GUIDs to the pattern. Turn on the “case insensitive” checkbox to make the regex case insensitive.

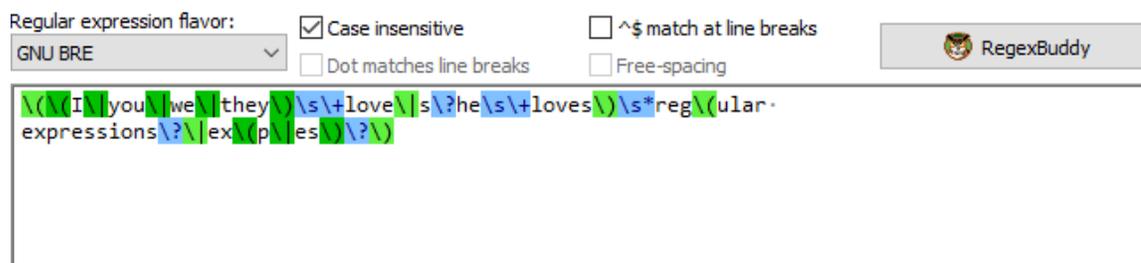
## 22. Pattern: Regular expression

“Regular expression” is one of the patterns that you can select on the Match panel. With this pattern you can insert actual regular expression code into your regular expression.

One handy aspect of the “regular expression” pattern is that it accepts a regular expression in any of the regular expression flavors that RegexMagic supports. This flavor can be a different one than the regex flavor used by the application you have selected on the Regex panel. RegexMagic automatically converts the regular expression from the flavor you specified in the pattern to the flavor selected on the Regex panel.

In this example, we’ll use this ability to take a regular expression that you might have found in old source code that was created for a legacy regular expression flavor and convert it to a modern flavor. You can find this example as “Pattern: regular expression” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. Set both “begin regex match at” and “end regex match at” to “anywhere”. This makes sure RegexMagic does not add any anchors to our regular expression.
3. Click the  button to add field .
4. In the “pattern to match field” drop-down list, select “regular expression”.



5. In the “regular expression flavor” drop-down list, select GNU BRE.
6. Tick the “case insensitive” checkbox.
7. Paste in this regular expression:

```
\((I|you|we|they)\s+love\s|he\s+loves)\s*reg(ular expressions)?|ex(ples)?\)
```

8. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and RegexMagic converts the regex from the GNU BRE flavor to the JGsoft flavor:

```
((I|you|we|they)[\t\n\v\f\r ]+love\s|he[\t\n\v\f\r ]+loves)[\t\n\v\f\r ]*reg(ular expressions)?|ex(ples)?)
```

**Required options:** Case insensitive; Exact spacing.

**Unused options:** Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

## 23. Pattern: Pattern used by another field

“Pattern used by another field” is one of the patterns that you can select on the Match panel. If two or more fields need to match the same kind of pattern, but not match exactly the same text, you need to specify the pattern only for the first field that needs it. For the other fields, you can select “pattern used by another field” and reference the first field.

For this example, we’ll try to match a comma-delimited list of 3 numbers. The 3 numbers can be different, but they must all range from -999.999 to +999.999. The decimal digits are mandatory, the sign is optional.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

12.374, -17.872, 519.878

3. On the Match panel, set “begin regex match at” to “start of line”, and set “end regex match at” to “end of line”.
4. Again on the Samples panel, select the first number, and click the Mark button. This marks the first number as field **1**.
5. Select the first comma, and click the Mark button to mark it as field **2**.
6. Select the second number, and mark it as field **3**.
7. Mark the second comma as field **4**.
8. Mark the third number as field **5**. The whole sample has been marked now:

12.374, -17.872, 519.878

9. On the Match panel, use the “select field” drop-down list to select field **1**. RegexMagic has correctly detected the “number” pattern for this field. But the settings for the pattern aren’t the ones we want. One sample is not enough for RegexMagic to extrapolate from.
10. Set the minimum and maximum values of the integer part to -999 and 999.
11. Tick “limit integer part”, “allow plus sign”, “allow minus sign”, and “require integer part”. Clear all the other checkboxes. The pattern should now look like this:

Minimum value of integer part: -999	Maximum value of integer part: 999	<input checked="" type="checkbox"/> Limit integer part
Decimal separator: Period	Min. and max. number of decimals 3	<input checked="" type="checkbox"/> Allow plus sign
Thousand separator: None	Currency sign/code position: Before only	<input checked="" type="checkbox"/> Allow minus sign
Currency sign: None	Currency codes:	<input type="checkbox"/> Allow parentheses
		<input type="checkbox"/> Sign is required
		<input type="checkbox"/> Whitespace allowed after sign
		<input type="checkbox"/> Thousand separators are required
		<input type="checkbox"/> Allow leading zeros
		<input checked="" type="checkbox"/> Require integer part
		<input type="checkbox"/> Allow exponent
		<input type="checkbox"/> Currency sign or code required

12. Use the “select field” drop-down list again to select field **3**. RegexMagic detected the “number” pattern for this field too, but with different options.
13. In the “pattern to match field” drop-down list, select “pattern used by another field”.

Use the pattern from the field:

**1** Number

14. Use the “select field” drop-down list again to select field **5**.
15. In the “pattern to match field” drop-down list, select “pattern used by another field” once more.
16. Set the “field validation mode” to “strict”, to make sure the number pattern uses all the settings we made.
17. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```

^
# 1. Number
[+-]?(?:[1-9][0-9]{1,2}|[0-9])\.[0-9]{3}
# 2. Literal text
,
# 3. Same as field 1: Number
[+-]?(?:[1-9][0-9]{1,2}|[0-9])\.[0-9]{3}
# 4. Literal text
,
# 5. Same as field 1: Number
[+-]?(?:[1-9][0-9]{1,2}|[0-9])\.[0-9]{3}
$

```

**Required options:** Free-spacing; ^\$ match at line breaks.

**Unused options:** Case sensitive; Dot doesn’t match line breaks; Numbered capture.

18. The Samples panel now confirms our regular expression matches the 3 comma-delimited numbers:

```
12.374,-17.872,519.878
```

## 24. Pattern: Text matched by another field

“Text matched by another field” is one of the patterns that you can select on the Match panel. This pattern requires a field to match the exact same text that was matched by a previous field in the regular expression.

For this example, we’ll try to match a comma-delimited list of 3 numbers. The 3 numbers must be exactly the same. They can range from -999,999 to +999,999. The decimal digits are mandatory, the sign is optional.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
12.374,12.374,12.374
-17.872,-17.872,-17.872
519.878,519.878,519.878
12.374,-17.872,519.878
```

3. On the Match panel, set “begin regex match at” to “start of line”, and set “end regex match at” to “end of line”.
4. Again on the Samples panel, select the first number, and click the Mark button. This marks the first number as field **1**.
5. Select the first comma, and click the Mark button to mark it as field **2**.
6. Select the second number, and mark it as field **3**.
7. Mark the second comma as field **4**.
8. Mark the third number as field **5**. The first sample has been marked now:

```
12.374,12.374,12.374
-17.872,-17.872,-17.872
519.878,519.878,519.878
12.374,-17.872,519.878
```

9. On the Match panel, use the “select field” drop-down list to select field **1**. RegexMagic has correctly detected the “number” pattern for this field. But the settings for the pattern aren’t the ones we want. One sample is not enough for RegexMagic to extrapolate from.
10. Set the minimum and maximum values of the integer part to -999 and 999.
11. Tick “limit integer part”, “allow plus sign”, “allow minus sign”, and “require integer part”. Clear all the other checkboxes. The pattern should now look like this:

Minimum value of integer part: -999	Maximum value of integer part: 999	<input checked="" type="checkbox"/> Limit integer part
Decimal separator: Period	Min. and max. number of decimals 3 3	<input checked="" type="checkbox"/> Allow plus sign
Thousand separator: None	Currency sign/code position: Before only	<input checked="" type="checkbox"/> Allow minus sign
Currency sign: None	Currency codes:	<input type="checkbox"/> Allow parentheses
		<input type="checkbox"/> Sign is required
		<input type="checkbox"/> Whitespace allowed after sign
		<input type="checkbox"/> Thousand separators are required
		<input type="checkbox"/> Allow leading zeros
		<input checked="" type="checkbox"/> Require integer part
		<input type="checkbox"/> Allow exponent
		<input type="checkbox"/> Currency sign or code required

12. Use the “select field” drop-down list again to select field **3**. RegexMagic detected the “number” pattern for this field too, which isn’t what we want.

13. In the “pattern to match field” drop-down list, select “text matched by another field”.

Match the text matched by the field:  
  Case insensitive

14. Use the “select field” drop-down list again to select field **5**.  
 15. In the “pattern to match field” drop-down list, select “text matched by another field” once more.  
 16. Set the “field validation mode” to “strict”, to make sure the number pattern uses all the settings we made.  
 17. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```

^
# 1. field1: Number
(?<field1>[+-]?(?:[1-9][0-9]{1,2}|[0-9])\.[0-9]{3})
# 2. Literal text
,
# 3. Same text as field 1
\k<field1>
# 4. Literal text
,
# 5. Same text as field 1
\k<field1>
$

```

**Required options:** Case sensitive; Free-spacing; ^\$ match at line breaks.

**Unused options:** Dot doesn’t match line breaks; Numbered capture.

18. The Samples panel now confirms our regular expression matches the 3 comma-delimited numbers, but only if the 3 numbers are the same:

```

12.374,12.374,12.374
-17.872,-17.872,-17.872
519.878,519.878,519.878
12.374,-17.872,519.878

```

## 25. Pattern: Match anything

“Match anything” is one of the patterns that you can select on the Match panel. This pattern allows a field to match any text, except perhaps for certain characters that may occur in the next field. The repetition settings for the field determine how many characters the field can or must match.

This pattern is actually the first pattern in the list of patterns in RegexpMagic because it is the most generic pattern. But it’s the last example in this help file because in most cases you get a more accurate and better performing regular expression by using a more specific pattern. It is very rare that you want to allow part of your regex to match truly any text. Usually there are exceptions. If you do want to use the “match anything pattern”, pay attention to the “match anything except” choice. This example illustrates two of those choices.

For this example, we’ll try to match a pair of HTML bold tags, with any text between them, but no other HTML tags between them.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
This is some <b>bold</b> text.
This one <b>also</b>.
<b>Whatever</b>.
Mixing <b>bold and <i>italic</i></b> together.
```

3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
4. Again on the Samples panel, select the first <b> tag, and click the Mark button. This marks the first opening bold tag as field **1**.
5. Select the word **bold** adjacent to the tag you just marked, and click the Mark button to mark it as field **2**.
6. Select the first </b> tag. Click the Mark button to mark the first closing bold tag as field **3**. The first sample has been marked now:

```
This is some <b>bold</b> text.
This one <b>also</b>.
<b>Whatever</b>.
Mixing <b>bold and <i>italic</i></b> together.
```

7. On the Match panel, use the “select field” drop-down list to select field **2**. Since we only marked one piece of text for this field that doesn’t fit any particular pattern, RegexpMagic selected the “literal text” pattern for this field, making it match the word “bold” that we marked.
8. In the “pattern to match field” drop-down list, select “match anything”.
9. In the “match anything except” drop-down list, select “first character of the next field”. The next field is field **3**, which matches the literal text </b>. Thus, excluding the first character of the next field makes field **2** match any character except an opening angle bracket. This meets our requirement of excluding nested HTML tags.

Match anything except:  Character to escape delimiters:   Can span across lines

10. In the list of fields in the regular expression at the top of the Match panel, next to field **2**, enter 1 under “repeat this field” and tick the “unlimited” checkbox. Since we only marked one sample that is

4 characters long, RegexMagic set this field to allow exactly 4 characters. Now we've set it to allow any number of characters.

11. Use the “select field” drop-down list to select field **1**.
12. Turn on “case insensitive” among the options for the literal text pattern for this field.
13. Repeat the previous two steps for field **3**.
14. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you'll get this regular expression:

```
<b>[^\n\r<]+</b>
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

15. The Samples panel now confirms our regular expression matches a pair of bold tags with any text between them, except HTML tags:

```
This is some <b>bold</b> text.
This one <b>also</b>.
<b>Whatever</b>.
Mixing <b>bold and <i>italic</i></b> together.
```

With a few simple changes, we can generate another regex that matches a pair of HTML bold tags that allows any text in between, including any HTML tags except the closing bold tag.

16. On the Match panel, set “how to repeat this field” to “as few times as possible” for field **2**. This is necessary to ensure that the regex match will end at the first closing tag after the opening tag rather than at the last closing tag in the file.
17. Use the “select field” drop-down list to select field **2** if not already selected.
18. In the “match anything except” drop-down list, select “text matched by the next field”.
19. Regenerate the regex and you'll get:

```
<b>(?.+?</b>)
```

**Required options:** Case insensitive; Dot doesn't match line breaks.

**Unused options:** Exact spacing; ^\$ don't match at line breaks; Numbered capture.

20. The Samples panel now confirms our regular expression matches a pair of bold tags with any text between them, including HTML tags, except </b>:

```
This is some <b>bold</b> text.
This one <b>also</b>.
<b>Whatever</b>.
Mixing <b>bold and <i>italic</i></b> together.
```

## 26. Combining Patterns Using Multiple Fields

This example shows how you can easily generate a regular expression to handle all kinds of (weird) identification codes by combining several fields into a regular expression. You can find this example as “Fields: multiple” in the RegexMagic library.

We want to create a regular expression that checks whether a particular string is a valid SKU code. SKU codes are used in inventory management to uniquely identify products. The SKU codes in this example are completely made up. It provides redundant human-readable information that makes it easier to prevent mistakes than an opaque ID that only consists of digits.

The format for the SKU codes in our example is `000000SUPPLIER/PRODUCTMMDDYY`. A 6-digit unique product number, a supplier name, a slash, a product name (can contain digits), and 6 digits representing a date. The supplier and product names can each be between 1 and 16 characters in length.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
018719ACME/WMD001080165
028912MS/WIN95082495
032789JGSOFT/EPP6070106
```

3. Set the subject scope to “line by line”. Our sample text has 3 lines, each with one SKU number. Because we want a regex that allows only one SKU code per string being validated, we need to tell RegexMagic to pretend that we have 3 samples (1 line each) rather than 1 sample with 3 lines.
4. Click the  button to add field .
5. Set both “repeat this field” spinner controls to 6 to require this field to match 6 characters.
6. In the “pattern to match field” drop-down list, select “basic characters”. We use this pattern instead of the “integer” or “numbers” pattern because we want to match exactly 6 digits.
7. Tick the “digits” checkbox in the settings for the “basic characters” pattern.
8. Click the Add Next Field button  to add field .
9. Set the “repeat this field” spinner controls for field  to 1 and 16 in order to allow between 1 and 16 characters for this field.
10. RegexMagic copied the pattern from field  to field . All we need to do is to untick “digits” and to tick “uppercase letters” instead.
11. On the Samples panel, select the slash after ACME, and click the Mark button. That’s all it takes to add field  and make it match a slash.
12. Click the Add Next Field button  to add field .
13. Set the “repeat this field” spinner controls for field  to 1 and 16 in order to allow between 1 and 16 characters for this field.
14. In the “pattern to match field” drop-down list, select “basic characters”.
15. Tick the “digits” and “uppercase letters” checkboxes in the settings for the “basic characters” pattern.
16. Click the Add Next Field button  to add field .
17. In the “pattern to match field” drop-down list, select “date and time”.
18. Select “leading zeros required” in the “leading zeros” drop-down list.
19. Enter `mdy` into the “allowed date and time formats” box.



## 27. Repeating Combined Fields Using a Sequence Field

This example demonstrates how you can make part of your regular expression repeat itself to match text of variable length, even if the repeated part is complex (meaning there's no single RegexMagic pattern to match it). You can find this example as “Fields: sequence” in the RegexMagic library.

For this example, we'll create a regex that verifies if a string holds a special kind of code. The code is delimited with angle brackets. Between those are 3 to 6 numbers ranging from 127 to 67301. The numbers are delimited by colons.

The description in the preceding paragraph is what you might read in a specification. Regular expressions can't be written that way. Regular expressions work from left to right. Rewriting the spec from left to right, we have an opening angle bracket, followed by a number between 127 and 64301, followed by 2 to 5 times a colon and a number between 127 and 64301, followed by a closing angle bracket. While our rewritten spec is long-winded and perhaps less clear, writing it out or thinking it through this way makes it much easier to build the regular expression.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. Set both “begin regex match at” and “end regex match at” to “anywhere”.
3. Set the “field validation mode” to “strict”. We need to do this to make the integer pattern use the minimum and maximum values we specify.
4. On the Samples panel, paste in one new sample:

```
<127:7898:1983>
<1234:789:18988:33891:9819>
<333:4444:55555:67301:1289:1081>
```

5. Select the first < in the sample.
6. Click the Mark button above the sample to mark the angle bracket as field **1**. RegexMagic automatically detects the correct “literal text” pattern for this field.
7. Select the number 127 right after the first angle bracket in the sample.
8. Click the Mark button to mark the number as field **2**. RegexMagic detects the “integer” pattern that we want, but not with all the right options as those can't be guessed from marking just one number.
9. On the Match panel, tick the “limit values” checkboxes among the settings for the integer pattern.
10. In the edit control below that checkbox, enter “127..67301”.
11. Back on the Samples panel, select : right after the number 127.
12. Click the Mark button to add field **3**. RegexMagic detects it as matching a literal colon.
13. Since our regex needs to match the colon 2 to 5 times together with another number, we stop marking fields on the Samples panel. To start the group, switch to the Match panel. Next to field **3**, in the “kind of field” drop-down list, select “sequence”. This tells RegexMagic we want to repeat multiple fields as one unit. The old field **3** that matches a literal colon is moved into the sequence as field **4**.
14. Set the “repeat this field” spinner controls for field **3** to 2 and 5.
15. Click the Add Last Sub-Field button **5** to add field **5** as the second field in the sequence under field **3**.
16. In the “pattern to match field” drop-down list, select “pattern used by another field”.
17. In the “use pattern from the field” drop-down list, select field **2**. This makes field **5** match an integer between 127 and 67301 just as field **2** does, though not necessarily the same integer.
18. Click on the colored rectangle for field **3** to make the field buttons work on that field.

19. Click the Add Next Field button  to add field  after field , without making it a part of the sequence under field . Make sure not to confuse this button with the Add Last Sub-Field button  which would make the new field part of the sequence.
20. In the “pattern to match field” drop-down list, select “literal text”.
21. Enter a > as the literal text that field  should match.

Fields in the regular expression

Begin regex match at: Anywhere End regex match at: Anywhere

Field ID	Kind of field:	Repeat:	Unlimited	How to repeat this field:
	Pattern (Literal text)	1	<input type="checkbox"/>	As many times as possible
	Pattern (Integer)	1	<input type="checkbox"/>	As many times as possible
	Sequence	2	<input type="checkbox"/>	As many times as possible
	Pattern (Literal text)	1	<input type="checkbox"/>	As many times as possible
	Pattern (Pattern used by another field)	1	<input type="checkbox"/>	As many times as possible
	Pattern (Literal text)	1	<input type="checkbox"/>	As many times as possible

22. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
# 1. Literal text
<
# 2. Integer
(?:6730[01]|67[0-2][0-9]{2}|6[0-6][0-9]{3}|[1-5][0-9]{4}|[1-9][0-9]{3}|[2-9][0-9]{2}|1[3-9][0-9]|12[7-9])
# 3. Fields 4 to 5 in sequence
(?:
# 4. Literal text
:
# 5. Same as field 2: Integer
(?:6730[01]|67[0-2][0-9]{2}|6[0-6][0-9]{3}|[1-5][0-9]{4}|[1-9][0-9]{3}|[2-9][0-9]{2}|1[3-9][0-9]|12[7-9])
){2,5}
# 6. Literal text
>
```

**Required options:** Free-spacing.

**Unused options:** Case sensitive; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Numbered capture.

23. The Samples panel now indicates our regex matches the code numbers. You’ll see that in each code number, all the colons are highlighted as field  and the integers after them as field . No text is highlighted as field . That’s because the highlighting indicates which pattern matched which text. Field  doesn’t match any text on its own. Its only purpose is to repeat fields  and  together.

```
<127:7898:1983>  
<1234:789:18988:33891:9819>  
<333:4444:55555:67301:1289:1081>
```

## 28. Matching Unrelated Items Using Alternation

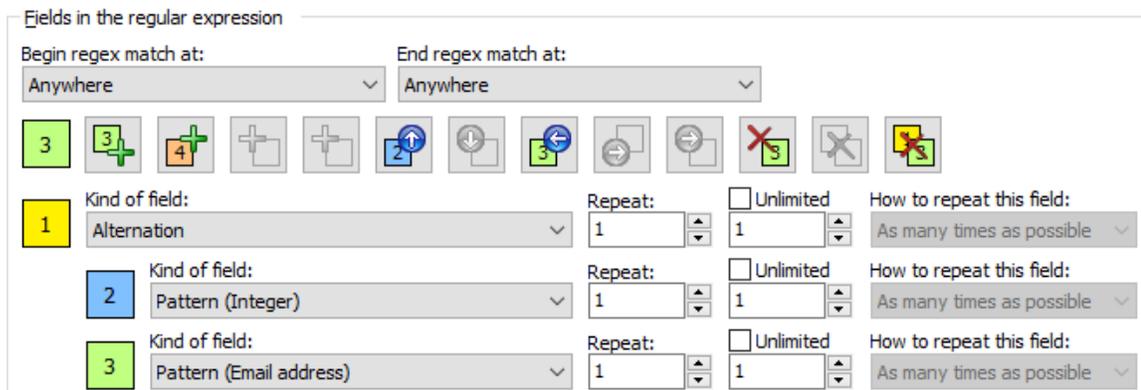
This example demonstrates how you can generate a regular expression that matches two or more unrelated bits of text in a file, regardless of where or how many times each bit of text occurs in the file. You can find this example as "Fields: alternation single" in the RegexMagic library.

For this example, we'll create a regex to find all numbers and all email addresses in a file. To do this, we create a regex that matches a number or an email address. When you repeat this regex using the "find all" command in an application or programming languages, you will get a list of all the numbers and email addresses.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. Set both "begin regex match at" and "end regex match at" to "anywhere".
3. Set the "field validation mode" to "average".
4. On the Samples panel, paste in one new sample:

```
My favorite number is 42.
You can email me at joe@fortytwo.com.
Other nice numbers are 17, 382, and 794.
joe@fortytwo@gmail.com is my alternative email address.
```

5. Select the number "42" in the sample.
6. Click the Mark button above the sample to mark "42" as field **1**. RegexMagic automatically detects the correct "integer" pattern for this field.
7. Select the email address "joe@fortytwo.com" in the sample.
8. Click the Mark button to mark the email address as a field. RegexMagic knows that because the email address is not adjacent to the number, the only way a regex could match both is by using alternation. Thus, RegexMagic adds a new field **1** with "kind of field" set to "alternation". The field with the integer pattern becomes field **2**, which is the first alternative under field **1**. The new field **3** is added as the second alternative under field **1**. RegexMagic automatically detects the correct "email address" pattern for this field.



9. On the Regex panel, select "C# (.NET 2.0–7.0)" as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you'll get this regular expression:

*# 1. One of the fields 2 to 3*

**# 2. Integer**`[0-9]+`

|

**# 3. Email address**`[!#$%&'*+./0-9=?_`a-z{|}~^-]+@[.0-9a-z-]+\.[a-z]{2,63}`

**Required options:** Case insensitive; Free-spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

10. Because the Samples panel always highlights all regex matches like a “find all” command would, it now shows that our regex matches all the numbers and email addresses:

My favorite number is 42.  
 You can email me at joe@fortytwo.com.  
 Other nice numbers are 17, 382, and 794.  
 joe@fortytwo@gmail.com is my alternative email address.

11. If you want to get the same results when programming, select one of the functions “get a list of all regex matches in a string” or “use regex object to get a list of all regex matches in a string” on the Use panel. Most of the source code templates that ship with RegexMagic have one of these functions. Some may use the term “array” instead of “list”.

The generated regular expression uses alternation to combine the patterns for all the fields into one regular expression. The result is that this regular expression will match either a number, or an email address, regardless of context. To find all numbers and email addresses in a file, apply this regular expression repeatedly to the same text using a “find all” command in your application or programming language.

## 29. Matching Complex Unrelated Items Using Alternation of Sequences

This example demonstrates how you can generate a regular expression that matches two or more unrelated bits of text in a file, regardless of where or how many times each bit of text occurs in the file. The bits of text are complex, meaning multiple RegexMagic patterns must be placed in sequence to match them. You can find this example as “Fields: alternation sequence” in the RegexMagic library.

For this example, we’ll create a regex to find all numbers and all email addresses in a file, with the numbers between square brackets and the email addresses between angle brackets. The regex matches should include the delimiters. To do this, we create a regex that matches a number between square brackets or an email address between angle brackets. When you repeat this regex using the “find all” command in an application or programming language, you will get a list of all the numbers and email addresses with their delimiters.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. Set both “begin regex match at” and “end regex match at” to “anywhere”.
3. Set the “field validation mode” to “average”.
4. On the Samples panel, paste in one new sample:

```
My favorite number is [42].
You can email me at <joe@fortytwo.com>.
Other nice numbers are [17], [382], and [794].
<joefortytwo@gmail.com> is my alternative email address.
```

5. Select the first [ in the sample, immediately before the number 42 on the first line.
6. Click the Mark button above the sample to mark the opening square bracket as field **1**. RegexMagic automatically detects the correct “literal text” pattern for this field.
7. Select the number “42” in the sample.
8. Click the Mark button to mark “42” as field **2**. RegexMagic automatically detects the correct “integer” pattern for this field.
9. Select the ] right after the number 42 that we just marked.
10. Click the Mark button to mark the closing square bracket as field **3**. Again we get the “literal text” pattern we want.
11. Select the first < in the sample, immediately before “joe@fortytwo.com”.
12. Click the Mark button to mark the opening angle bracket as a field. RegexMagic knows that because the angle bracket is not adjacent to the previous three fields we just marked, the only way a regex could match both is by using alternation. Thus, RegexMagic adds a new field **4** with “kind of field” set to “alternation”. The first alternative is a new field **2**, with “kind of field” set to “sequence”. The 3 fields we marked previously are placed under the sequence field, with their field numbers changed to **3**, **4**, and **5**. The new field to match the angle bracket is field **6**, which is added as the second alternative under field **4**.
13. Select the email address “joe@fortytwo.com” in the sample.
14. Click the Mark button to add a field for the email address. Since we’re marking this immediately after field **6**, RegexMagic assumes that we always want the email address to follow immediately after the angle bracket. To accomplish this, RegexMagic adds a new field **6**, with “kind of field” set to “sequence”. The field for the angle bracket becomes field **7** as the first field in the sequence. Field **8** is the new field for the email address.



**Required options:** Case insensitive; Free-spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

20. Because the Samples panel always highlights all regex matches like a “find all” command would, it now shows that our regex matches all the numbers and email addresses with their delimiters:

```
My favorite number is [42].
You can email me at <joe@fortytwo.com>.
Other nice numbers are [17], [382], and [794].
<joefortytwo@gmail.com> is my alternative email address.
```

21. If you want to get the same results when programming, select one of the functions “get a list of all regex matches in a string” or “use regex object to get a list of all regex matches in a string” on the Use panel. Most of the source code templates that ship with RegexMagic have one of these functions. Some may use the term “array” instead of “list”.

## 30. Capturing Groups

This example demonstrates how you can add capturing groups to a regular expression to figure out which part of the regular expression found the match. You can find this example as “Capturing groups” in the RegexMagic library.

For this example, we’ll continue with the regular expression created in the example about matching unrelated items using alternation. That regular expression matches a number or an email address. Using this regex with a “find all” command, we can get a list of all numbers and email addresses.

Now we want to use this regex to iterate over all the numbers and email addresses in a file, and we want to separate the numbers and the email addresses, without having to use a second regular expression to check whether the match found by our regular expression is a number or an email address. We can achieve this by placing capturing groups around the parts of the regex that match the number and the email address. Since our regex matches only one number or one email address at a time, only one of the capturing groups will actually capture any text with each regex match. If the group for the number captured text, we know we have a number. If not, the group for the email address will have captured the email address.

You could actually achieve this with just one capturing group for the number. When the group for the number doesn’t capture anything, retrieving the overall regex match gives the email address. But in this example we’ll create two groups just for practice.

1. On the Library panel, load RegexMagic.rml if it isn’t loaded already.
2. In the list of RegexMagic formulas in the library, select “Fields: alternation single”.
3. Click the Use button to populate the Samples, Match, and Action panels with the settings from the RegexMagic formula we just loaded. These are the settings we made in the example about matching unrelated items using alternation.
4. On the Action panel, click the New button to add a capturing group.
5. Enter “number” as the group’s name. You should give your capturing groups a name even if your regex flavor does not support named capture.
6. Select “**2** integer” in the “field” drop-down list. We now have a capturing group that will give us the text matched by field **2**, if any.

Capture parts of the regular expression match

Capturing groups:

number	New	Delete	Delete All
--------	-----	--------	------------

Group name: number

Actual backreferences: none

What to capture: Text matched by a field

Field: **2** number: Integer

7. Click the New button on the Action panel again to add the second capturing group.
8. Enter “email” as the group’s name.
9. Select “**3** email address” in the “field” drop-down list.
10. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get the regular expression below. Notice the difference between this regex and the one we got in the example about matching unrelated items using alternation. Two named capturing groups have been added, and comments for each field indicate the capturing group names.

```
# 1. One of the fields 2 to 3
# 2. number: Integer
(?<number>[0-9]+)
|
# 3. email: Email address
(?<email>[!#$%&'*+./0-9=?_`a-z{|}~^-]+@[.0-9a-z-]+\.[a-z]{2,63}+)
```

**Required options:** Case insensitive; Free-spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Greedy quantifiers.

11. On the Action panel, you can now check the “actual backreferences”. This box indicates the name or number you'll need to reference the group in your source code.

Capture parts of the regular expression match

Capturing groups:

Group name:  Actual backreferences:

What to capture:  Field:

12. On the Regex panel, select the JavaScript regular expression flavor. With the Generate button still down, you'll get a regex that still has the group names in the comments. The actual capturing groups are unnamed, because JavaScript does not support named capture.

```
(?<number>[0-9]+)|(?<email>[\d!#$%&'*+./=?_`a-z{|}~^-]+@[\.d.a-z-]+\.[a-z]{2,63})
```

**Required options:** Case insensitive.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks.

13. On the Action panel, the actual backreference is now 1 for the group capturing the number, and 2 for the group capturing the email address.

Capture parts of the regular expression match

Capturing groups:

Group name:  Actual backreferences:

What to capture:  Field:

## 31. Replacing Text Match By Fields with Text Matched by Other Fields

This example demonstrates how you can prepare a search-and-replace action with RegexpMagic that swaps different parts of the regular expression match with other parts of the regular expression match. You can find this example as “Action: replace fields” in the RegexpMagic library.

For this example, we’ll continue with the regular expression created in the example about the “pattern used by another field” pattern. That regular expression matches three numbers delimited by commas.

Now we want to use this regex in a search-and-replace that swaps the first and third numbers. This is done by adding capturing groups to the regex and using backreferences to those groups in the replacement text. With RegexpMagic, you only need to tell which fields you want to replace. RegexpMagic automatically adds the necessary capturing groups and automatically generates the replacement text.

1. On the Library panel, load RegexpMagic.rml if it isn’t loaded already.
2. In the list of RegexpMagic formulas in the library, select “Pattern: pattern used by another field”.
3. Click the Use button to populate the Samples, Match, and Action panels with the settings from the RegexpMagic formula we just loaded.
4. On the Action panel, select “replace all fields” in the “action to take” drop-down list.

The screenshot shows the configuration for the 'Action to take' panel. It includes three dropdown menus: 'Action to take' (set to 'Replace all fields'), 'Field to replace' (set to '1 Number'), and 'How to replace' (set to 'Replace with another field'). Below these is a 'Field to replace with' dropdown set to '5 Same as field 1: Number'.

5. Select “1 number” in the “field to replace” drop-down list.
6. Select “replace with another field” in the “how to replace” drop-down list.
7. Select “5 number” in the “field to replace with” drop-down list. This tells RegexpMagic we want to replace the text matched by field 1 with the text matched by field 5.
8. Select “5 number” in the “field to replace” drop-down list.
9. Select “replace with another field” in the “how to replace” drop-down list.
10. Select “1 number” in the “field to replace with” drop-down list. This tells RegexpMagic we want to replace the text matched by field 5 with the text matched by field 1.
11. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get the regular expression below. Notice the difference between this regex and the one we got in the example about the “pattern used by another field” pattern. Three named capturing groups have been added to capture the text matched by field 1, the text matched by fields 2 through 4, and the text matched by field 5.

```

^
# 1. field1: Number
(?<field1>[+-]?(?:[1-9][0-9]{1,2}|[0-9])\.[0-9]{3})
(?<fields2to4>
# 2. Literal text
,

```

```

# 3. Same as field 1: Number
[+-]?(?:[1-9][0-9]{1,2}||[0-9])\. [0-9]{3}
# 4. Literal text
,
)
# 5. field5: Same as field 1: Number
(?:<field5>[+-]?(?:[1-9][0-9]{1,2}||[0-9])\. [0-9]{3})
$

```

**Required options:** Free-spacing; ^\$ match at line breaks.

**Unused options:** Case sensitive; Dot doesn't match line breaks; Numbered capture.

We also get a replacement text that consists of three backreferences. The first backreference reinserts the text matched by field [5](#), the second reinserts fields [2](#) through [4](#), and the last one puts field [1](#) at the end of the replacement.

```

${field5}${fields2to4}${field1}

```

- If you select the JavaScript regex flavor, which does not support named groups, you'll see that RegexpMagic can use numbered capturing groups just as well.

```

^(?:<field1>[+-]?(?:[1-9][0-9]{1,2}||[0-9])\. \d{3})(?:<fields2to4>[+-]?(?:[1-9][0-9]{1,2}||[0-9])\. \d{3},(?:<field5>[+-]?(?:[1-9][0-9]{1,2}||[0-9])\. \d{3})$

```

**Required options:** ^\$ match at line breaks.

**Unused options:** Case sensitive; Dot doesn't match line breaks.

The numbers in the replacement text refer to the capturing groups. In a regular expression, unnamed groups are always numbered from left to right starting at one.

```

$<field5>$<fields2to4>$<field1>

```

## 32. Typing in Your Own Replacement Text

As shown in the “replacing text match by fields with text matched by other fields”, RegexMagic can generate a replacement text along with the regular expression. If you’re already familiar, you may find it quicker to enter the replacement text yourself. This example shows how RegexMagic can use a replacement text that you provide. You can find this example as “Action: replacement text” in the RegexMagic library.

For this example, we’ll continue with the regular expression created in the example about the “pattern used by another field” pattern. That regular expression matches three numbers delimited by commas.

Now we want to use this regex in a search-and-replace that swaps the first and third numbers. This is done by adding capturing groups to the regex and using backreferences to those groups in the replacement text.

1. On the Library panel, load RegexMagic.rml if it isn’t loaded already.
2. In the list of RegexMagic formulas in the library, select “Pattern: pattern used by another field”.
3. Click the Use button to populate the Samples, Match, and Action panels with the settings from the RegexMagic formula we just loaded. The Match panel defines 5 fields. Fields **1**, **3**, and **5** match the three numbers. Fields **2** and **4** match the commas that delimit them.
4. On the Action panel, click the New button to add a capturing group. Set the group to capture field **1**. We’ll use numbered backreferences in our replacement text, so don’t bother naming the group.
5. Click the New button again to add a group to capture field **3**.
6. Click the New button one more time to capture field **5**.
7. Select “replace whole regex match” in the “action to take” drop-down list.

The screenshot shows the 'Action to take' panel in RegexMagic. It has three dropdown menus at the top: 'Action to take' (set to 'Replace whole regex match'), 'How to replace' (set to 'Enter a replacement with backreferenc'), and 'Replacement text flavor' (set to '.NET 1.1-7.0'). Below these is a text area labeled 'Replacement text:' containing the text '\$3, \$2, \$1'.

8. Select “enter a replacement with backreferences” in the “how to replace” drop-down list.
9. Set “replacement text flavor” to “%DEFAULTFLAVORREPLACE”.
10. Now we can enter our replacement text:

```
{3}, {2}, {1}
```

11. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn on free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get the regular expression below. If you compare this regex with the one we got in the example about the “pattern used by another field” pattern, you’ll see that three pairs of parentheses have been added. Those are our three capturing groups.

```
^
# 1. Number
([+-]?([1-9][0-9]{1,2}|[0-9])\.[0-9]{3})
# 2. Literal text
,
```

```

# 3. Same as field 1: Number
([+-]?(?:[1-9][0-9]{1,2}||[0-9])\.[0-9]{3})
# 4. Literal text
,
# 5. Same as field 1: Number
([+-]?(?:[1-9][0-9]{1,2}||[0-9])\.[0-9]{3})
$

```

**Required options:** Free-spacing; ^\$ match at line breaks.

**Unused options:** Case sensitive; Dot doesn't match line breaks; Numbered capture.

The replacement text generated by RegexMagic is identical to the one we provided, because the replacement text flavor (.NET 1.1–7.0) we used on the Action panel is the same as the replacement text flavor used by the application (C# (.NET 2.0–7.0)) we selected on the Regex panel.

```

${3},${2},${1}

```

12. If we select an application on the Regex panel that uses a different replacement text flavor then RegexMagic converts the replacement text from the flavor selected on the Action panel to the flavor used by the application selected on the Regex Panel. If we select Python on the Regex panel, for example, then RegexMagic generates the same regular expression, but a different replacement text:

```

\g<3>,\g<2>,\g<1>

```

Python uses a different syntax for backreferences in the replacement text. RegexMagic knows this and adjusts the replacement string on the Regex panel accordingly.

### 33. Matching Version Numbers

This example shows how you can generate a regular expression to match version numbers as they're often used in the software world. The version number `v1.2.3.4444` consists of a major version number, minor version number, release number, and build number. To make things interesting, we'll allow for the `v` and the build number to be optional. When the build number is omitted, the dot that separates it from the release number must also be omitted. You can find this example as "Real world: version number (match)" in the RegexMagic library. You can also watch a video of this example on RegexMagic's website.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
v6.0.156 for Windows 2000/2003/XP/Vista
Server version 1.1.20
Client Manager version 1.1.24
v12.22.5.1333
```

3. Select the `v` at the start of the sample and click the Mark button. RegexMagic adds field **1** which matches a literal `v`.
4. On the Match panel, set the "repeat this field" spinner controls for field **1** to 0 and 1 in order to make this field optional.
5. Back on the Samples panel, select the `6` immediately after the `v` we just marked and click the Mark button. RegexMagic adds field **2** which matches a decimal integer.
6. Select the dot immediately after the `6` and click the Mark button. RegexMagic adds field **3** which matches a literal dot.
7. Select the `0` immediately after the dot we just marked and click the Mark button. RegexMagic adds field **4** which matches a binary integer. That's not what we want. We could change the pattern directly on the Match panel, or we could mark another sample to give RegexMagic more information for its auto-detection.
8. Select the number `22` in the last line of the sample and click button **4** above the samples. Now RegexMagic changes field **4** to match a decimal integer.
9. Select the dot after the `22` we just marked and click the Mark button. RegexMagic adds field **5** which matches a literal dot.
10. Mark the number `5` in the last line of the sample and click the Mark button. RegexMagic adds field **6** which matches a decimal integer.
11. Mark the dot after the number `5` and click the Mark button. RegexMagic adds field **7** to match the literal dot.
12. Now, we want to make that dot optional, but in combination with the build number that follows it. To make two fields optional as a combination, we have to use a sequence field. On the Match panel, select "sequence" in the "kind of field" drop-down list for field **7**. RegexMagic moves the old field **7** that matches the literal dot as field **8** under the new sequence field **7**.
13. Set the "repeat this field" spinner controls for field **7** to 0 and 1 in order to make this sequence field optional.
14. Select the number `1333` at the end of the sample text and click the Mark button. RegexMagic adds field **9** as the second field under sequence field **7** and configures it to match an integer.
15. On the Regex panel, select "C# (.NET 2.0–7.0)" as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you'll get this regular expression:

```
\bv?[0-9]+\.\.[0-9]+\.\.[0-9]+(?:\.[0-9]+)?\b
```

**Required options:** Case sensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

16. The Samples panel now shows that our regex matches all the version numbers:

```
v6.0.156 for Windows 2000/2003/XP/Vista
Server version 1.1.20
Client Manager version 1.1.24
v12.22.5.1333
```

## Capturing Parts of Version Numbers

Sometimes, it's not enough to just get the whole version number. You need to get the major, minor, release, and build numbers separately. To do that, we need to add capturing groups to the regular expression. This example continues from the previous one. You can find it as "Real world: version number (capture)" in the library.

17. On the Action panel, click the New button to add a capturing group.
18. Enter "major" as the group's name. You should give your capturing groups a name even if your regex flavor does not support named capture.
19. Select "[2] integer" in the "field" drop-down list. We now have a capturing group that will give us the major version number.
20. Click the New button, enter "minor" as the group's name, and select field [4] to add a capturing group for the minor version number.
21. Click the New button, enter "release" as the group's name, and select field [6] to add a capturing group for the release number.
22. Click the New button, enter "build" as the group's name, and select field [9] to add a capturing group for the build number.
23. On the Regex panel, click the Generate button if it isn't down already to get the regular expression with added capturing groups:

```
\bv?(?<major>[0-9]+)\.(?<minor>[0-9]+)\.(?<release>[0-9]+)(?:\.(?<build>[0-9]+))?\b
```

**Required options:** Case sensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

24. The matches highlighted on the Samples panel are still the same:

```
v6.0.156 for Windows 2000/2003/XP/Vista
Server version 1.1.20
Client Manager version 1.1.24
v12.22.5.1333
```

## Matching Specific Version Numbers

Editing either of the above regular expressions to match only specific version numbers is trivial with RegexMagic. On the Match panel, we have defined four fields that use the integer pattern to match any decimal number: **2**, **4**, **6**, and **9**. You can easily set minimum and maximum values for one or more of these fields to match a specific range of version numbers.

## 34. \$VAR and \${VAR} Variable Names

Some programming languages allow variable names to be specified as \$VAR or \${VAR} where VAR is an alphanumeric sequence. This example shows how you can generate a regular expression to match any variable in either notation. You can find it as “Real world: \$var and \${var} variable names” in the RegexMagic library.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
$VARIABLE_NAME
${BETWEEN_BRACES}
$ANOTHER_VARIABLE
$VAR4
${VAR5}
$6
${7}
```

3. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”. You can’t use start/end of word because \$ and } are not word characters.
4. Select the first \$ in the sample and click the Mark button. RegexMagic adds field 1 and detects it should match a literal \$.
5. Select VARIABLE\_NAME and click the Mark button. RegexMagic adds field 2 and also detects it as literal text. We’ll fix that later.
6. Select the second \$ in the sample and click the 1 button above the sample. This tells RegexMagic that the second dollar sign in the sample starts a variable, just like the \$ we already marked does.
7. Select the { and click the Mark button. This tells RegexMagic that two different fields can follow field 1. To make this happen, RegexMagic changes field 2 to be an alternation field. The field matching VARIABLE\_NAME becomes field 3, the first alternative under field 2. The newly marked field for the opening brace is added as field 4, the second alternative under field 2.
8. Select BETWEEN\_BRACES and click the Mark button. This tells RegexMagic that the second alternative under field 2 consists of two consecutive fields (so far): the brace and BETWEEN\_BRACES. To make this happen, RegexMagic changes 4 to be a sequence field. The field to match the brace becomes field 5, and the newly marked BETWEEN\_BRACES becomes field 6.
9. Select the } and click the Mark button. RegexMagic continues the sequence adding field 7.
10. If you were to generate the regex now, it would correctly match the two variables we marked, but not the others. What’s left to do now is to tell RegexMagic that we want to allow any alphanumeric sequence rather than only VARIABLE\_NAME and BETWEEN\_BRACES.
11. On the Match panel, click on the big 3 or select field 3 in the “select field” drop-down list.
12. In the “pattern to match field” drop-down list, select “basic characters”. RegexMagic automatically detects we want the underscore and uppercase letters.
13. Tick the boxes for lowercase letters and digits to allow any alphanumeric sequence.
14. Select field 6, change its pattern to “basic characters”, and tick “digits” and “lowercase letters”.
15. In the tree of fields on the Match panel, set fields 3 and 6 to repeat between 1 and 32 times or whatever lengths your variable names are allowed to have.
16. On the Regex panel, select “C# (.NET 2.0–7.0)” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
\$(?:[0-9A-Z_a-z]{1,32}|\{[0-9A-Z_a-z]{1,32}\})
```

**Unused options:** Case sensitive; Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture.

17. The Samples panel now shows that our regex matches all the version numbers:

```
$VARIABLE_NAME  
${BETWEEN_BRACES}  
$ANOTHER_VARIABLE  
$VAR4  
${VAR5}  
$6  
${7}
```

Essentially what we've done here is to separately mark all the different pieces of the text we're trying to match so that we can use one RegexpMagic Pattern for each of those pieces. Though it takes quite a number of steps, they're all very straightforward. Once you get the hang of how RegexpMagic combines patterns using alternation field and sequence fields you'll be able to create your own regular expressions this way very quickly.

## 35. Matching Something Within Something Else

A common class of problems that people try to solve with regular expressions is to find all occurrences of a certain pattern, but only within the occurrences of another pattern. This example illustrates this with a block of HTML that contains div tags and paragraph tags within those div tags as well as outside the div tags. We want to match all the paragraphs within the div tags, but not those outside the div tags.

A problem like this is best solved using two regular expressions. Use one regular expression to match the div tags. Use a second regular expression to match the paragraph tags within the matches of the first regular expression. A bit of procedural code glues everything together.

RegexMagic can generate only one regular expression at a time. So we'll tackle this problem in two steps. We'll create the two regexes separately and have RegexMagic generate a source code snippet for each. We'll combine the two code snippets ourselves.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
<h1>Matching Something Within Something Else</h1>
<p>Introduction</p>
<div>
<p>We want <i>this</i> paragraph.</p>
<table><tr><td>We</td><td>don't</td><td>want</td><td>tables</td></tr></table>
<p>We want this one too.</p>
</div>
<p>We don't want this one.</p>
<p>Nor this one.</p>
<div>
<p>Another one we want.</p>
<p>:-)</p>
</div>
<p>The end.</p>
```

3. Set the subject scope to “whole sample”.
4. On the Match panel, set both “begin regex match at” and “end regex match at” to “anywhere”.
5. Select the first occurrence of `<div>` in the sample.
6. Click the Mark button. RegexMagic automatically adds a literal text pattern that matches the text we marked.
7. In the settings for the “literal text” pattern, tick the “case insensitive” checkbox.
8. Select everything between the `<div>` we just marked and the first `</div>` that follows it, including all the line breaks. One way to do this is to put the cursor at the end of the line `<div>` and then press Shift+Right and then Shift+Down until the cursor is before the `</div>`.
9. Click the Mark button to make the whole contents of the div tag the second field.
10. In the “pattern to match field” drop-down list, select “match anything” for field [2](#).
11. In the “match anything except” drop-down list, select “nothing”. We want field [2](#) to match absolutely anything.
12. Set the left hand “repeat this field” spinner for field [2](#) to zero and tick the “unlimited” checkbox to allow field [2](#) to match any number of characters.
13. Set “how to repeat this field” for field [2](#) to “as few times as possible”. We want the regex to end its match at the first `</div>` tag, not the last one.
14. In the samples, select the first `</div>` tag.

15. Click the Mark button. RegexpMagic automatically adds another “literal text” field to match the closing tag.
16. In the settings for the “literal text” pattern, tick the “case insensitive” checkbox.
17. On the Regexp panel, select “PHP preg 8.0.0–8.1.24” as the application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
<div>\p{Any}*</div>
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn’t match line breaks; ^\$ don’t match at line breaks; Greedy quantifiers.

18. The Samples panel now confirms our regular expression matches all div tags in the sample:

```
<h1>Matching Something Within Something Else</h1>
<p>Introduction</p>
<div>
<p>We want <i>this</i> paragraph.</p>
<table><tr><td>We</td><td>don't</td><td>want</td><td>tables</td></tr></table>
<p>We want this one too.</p>
</div>
<p>We don't want this one.</p>
<p>Nor this one.</p>
<div>
<p>Another one we want.</p>
<p>:-)</p>
</div>
<p>The end.</p>
```

19. On the Use panel, in the Function drop-down list, choose “iterate over all matches in a string”.
20. Set the “subject text” parameter to \$subject and the “result array” parameter to \$div.
21. The Use panel now shows the first part of our code. Copy it into your PHP source code editor:

```
preg_match_all('%<div>\p{Any}*</div>%ui', $subject, $div,
PREG_PATTERN_ORDER);
for ($i = 0; $i < count($div[0]); $i++) {
# Matched text = $div[0][$i];
}
```

22. Now we move on to the second regular expression to match the paragraph tags. The second regex is very similar to the first. Instead of matching the opening and closing pair of a div tag and everything between it, the second regex will match a p tag. We’ll take a shortcut and edit the RegexpMagic formula that we used to generate the first regex instead of starting over.
23. On the Match panel, use the “select field” drop-down list to select field **1**.
24. Replace <div> with <p> as the text this field should match.
25. On the Match panel, use the “select field” drop-down list to select field **3**.
26. Replace </div> with </p> as the text this field should match.
27. The Regexp panel now generates this regular expression:

```
<p>\p{Any}*</p>
```

**Required options:** Case insensitive.

**Unused options:** Exact spacing; Dot doesn't match line breaks; ^\$ don't match at line breaks; Greedy quantifiers.

28. Because we took a shortcut, the Samples panel looks a bit messy. The intense highlighting shows that our regex matches all paragraph tags, including those outside the div tags. That's OK. RegexpMagic handles only one regular expression at the time. We'll combine them in our PHP code to get only the paragraph tags inside the div tags. You can ignore the faint highlighting on the Samples panel. It indicates the text that we marked to match the div tags, which our modified regex no longer does.

```
<h1>Matching Something Within Something Else</h1>
<p>Introduction</p>
<div>
<p>We want <i>this</i> paragraph.</p>
<table><tr><td>We</td><td>don't</td><td>want</td><td>tables</td></tr></table>
<p>We want this one too.</p>
</div>
<p>We don't want this one.</p>
<p>Nor this one.</p>
<div>
<p>Another one we want.</p>
<p>:-)</p>
</div>
<p>The end.</p>
```

29. On the Use panel, select the function “get an array of all regex matches in a string”.
30. Set the “subject text” parameter to `$div[0][$i]`. The variable holding the result in the loop in the first code snippet.
31. Set the “result array” parameter to `$p`.
32. The Use panel now shows the second part of our code:

```
preg_match_all('%<p>\p{Any}*</p>%ui', $div[0][$i], $p,
PREG_PATTERN_ORDER);
$p = $p[0];
```

33. Paste this part into your PHP code editor, inside the loop of the first part:

```
preg_match_all('%<div>\p{Any}*</div>%ui', $subject, $div,
PREG_PATTERN_ORDER);
for ($i = 0; $i < count($div[0]); $i++) {
preg_match_all('%<p>\p{Any}*</p>%ui', $div[0][$i], $p,
PREG_PATTERN_ORDER);
$p = $p[0];
}
```

34. Add one line and change one line of code to create a new variable `$pwithindiv` that will hold all of the paragraph tags within div tags in the string `$subject`:

```
$pwithindiv = array();
preg_match_all('%<div>\p{Any}*</div>%ui', $subject, $div,
PREG_PATTERN_ORDER);
for ($i = 0; $i < count($div[0]); $i++) {
preg_match_all('%<p>\p{Any}*</p>%ui', $div[0][$i], $p,
```

```
PREG_PATTERN_ORDER);  
$pwithindiv = array_merge($pwithindiv, $p[0]);  
}
```

Though this example requires a lot of steps, it's all very straightforward. You simply generate two regexes independently, one to match the outer text, and one to match the inner text. In your source code you combine the two regexes to make the second regex search through only text matched by the first regex.

If you're not developing software, you can use the same method if you're using an advanced grep tool such as PowerGREP that allows you to use more than one regular expression to run your searches. In PowerGREP, set the "action type" to "search". Then set "file sectioning" to "search for sections". Paste the regular expression that matches the outer text (in this example the one for the `div` tags) into the "section search" box. Then paste the regex for the inner text (the one for `p` tags) into the main part of the action in PowerGREP. When you execute this action, PowerGREP use the file sectioning regex to find all the `div` tags, and then use the main regex to find the paragraphs within the `div` tags only. This is no different from what our PHP code does, except that it requires no programming.

## 36. Matching Comments in Java Code

This example shows how you can generate a regular expression to match comments as used in Java and other programming languages. First we create a regex that matches single-line comments. These start with `//` and run until the end of the line.

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:

```
// Single-line comment
/* Multi-line comments
can span multiple lines */
/* Multi-line comments
/* cannot be nested // at all */
Not a comment
/* This comment */ // split in two
// Single-line comments // cannot be /* nested */
/** Documentation comment */
// Done!
```

3. Set the subject scope to “whole sample”.
4. Select the `//` at the start of the sample and click the Mark button. RegexMagic adds field **1** which matches `//` literally.
5. Select the remainder of the line. Begin your selection immediately after the `//` so the space is included. End your selection before the line break at the end of the line.
6. Click the Mark button again. RegexMagic adds field **2**. It also literally matches the text we just marked.
7. We’ll need to change field **2** to match anything until the end of the line. On the Match panel, set “pattern to match field” to “match anything”. Then set “match anything except” to “nothing”.
8. Make sure “can span across lines is turned off”. It should be by default because RegexMagic sees that the sample we marked for field **2** does not include any line breaks.
9. Set the left hand “repeat this field” spinner for field **2** to zero and tick the “unlimited” checkbox to allow field **2** to match any number of characters.
10. On the Regex panel, select “Java 8” as your application, turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you’ll get this regular expression:

```
//.*
```

**Required options:** Dot doesn’t match line breaks; Default line breaks.

**Unused options:** Case sensitive; Exact spacing; `^$` don’t match at line breaks.

11. The Samples panel now shows that our regex matches all single-line comments:

```
// Single-line comment
/* Multi-line comments
can span multiple lines */
/* Multi-line comments
/* cannot be nested // at all */
Not a comment
/* This comment */ // split in two
// Single-line comments // cannot be /* nested */
```

```
/** Documentation comment */
// Done!
```

## Matching Multi-Line Comments Too

The first example matches only single-line comments. It totally ignores multi-line comments. So it does match any // that occurs within a multi-line comments as a single-line comment. We can extend our regular expression to properly deal with both types of comments.

12. On the Samples panel, select the first /\* and click the Mark button. Because we left a gap (the line break) between field 2 and the newly marked field, RegexMagic treats the newly marked /\* as the start of a new alternative that the generated regular expression should be able to match independently from the previous set of fields. This is correct. We want our regex to be able to separately match single-line and multi-line comments. So RegexMagic adds a new alternation field 1 to hold our two alternatives. It adds a new sequence field 2 that holds the original two fields that match the single-line comment. These are now numbered 3 and 4. The second alternative under field 1 is the newly added field 5 which matches the /\* we marked literally.
13. Select all the text between the /\* we just marked and the \*/ that follows it, including the spaces right next to these. Click the Mark button. Because we marked this immediately adjacent to what we marked for the last field, RegexMagic treats it as a continuation of the same alternative. Again correct. We're still building the alternative for the multi-line comment. RegexMagic changes field 5 to be a sequence field with field 6 being the old field 5 matching the /\* and field 7 matching the two lines of text we just marked. We'll adjust field 7 soon.
14. Select the \*/ and click the Mark button. RegexMagic adds field 8 to the sequence. It matches \*/ literally.
15. On the Match panel, select field 7 in the "select field" drop-down list.
16. Set "pattern to match field" to "match anything".
17. Set "match anything except" to "text matched by the next field". This ensures that field 7 can never match the text matched by field 8. The comment must end at the first \*/. This would be important if we added more fields to the sequence after field 8 to match something that follows the comment. If that something is missing, we don't want the regex engine to try to expand the comment until the next \*/ in the file.
18. Set the left hand "repeat this field" spinner for field 7 to zero and tick the "unlimited" checkbox to allow it to match any number of characters.
19. Set "how to repeat this field" to "as few times as possible". We have to do this so that field 7 stops before the first \*/. RegexMagic will refuse to generate the regex if we don't select "as few times as possible" for a field that has "match anything except" set to "text matched by the next field".
20. Click the Generate button on the Regex panel again and you'll get:

```
// .* | /\*(?>(?:\s|.)*?\s*/)
```

**Required options:** Dot doesn't match line breaks; Default line breaks.

**Unused options:** Case sensitive; Exact spacing; ^\$ don't match at line breaks.

21. The Samples panel now shows that our regex matches all single-line and multi-line comments:

```
// Single-line comment
/* Multi-line comments
can span multiple lines */
/* Multi-line comments
/* cannot be nested // at all */
```

```

Not a comment
/* This comment */ // split in two
// Single-line comments // cannot be /* nested */
/** Documentation comment **/
// Done!

```

## Matching Only Multi-Line Comments

If you don't have any single-line comments to deal with, you can easily remove that part from the regular expression.

22. Click on the colored rectangle for field **2** to make the field buttons work on that field.
23. Click the Delete List of Fields button. It's glyph will show **2** to indicate that it deletes sequence field **2** along with fields **3** and **4** that the sequence consists of.

This is all we really need to do. But it leaves some unnecessary fields. This doesn't impact the generated regular expression. But it leaves clutter on the Match panel.

24. Select field **1**.
25. Click the Delete One Field button **X** to delete field **1**.
26. We now have a new field **1** that we don't need either. Click the Delete One Field button **X** again.
27. Click the Generate button on the Regex panel and you'll get:

```
/\*(?>.*?\/)
```

**Required options:** Dot matches line breaks; Default line breaks.

**Unused options:** Case sensitive; Exact spacing; ^\$ don't match at line breaks.

28. The Samples panel now shows that our regex matches only multi-line comments:

```

// Single-line comment
/* Multi-line comments
can span multiple lines */
/* Multi-line comments
// cannot be nested // at all */
Not a comment
/* This comment */ // split in two
// Single-line comments // cannot be /* nested */
/** Documentation comment **/
// Done!

```

If you wanted to create this regex from scratch, you could proceed as follows after pasting in the sample text:

4. On the Samples panel, select the first `/*` and click the Mark button. RegexMagic adds field **1** which matches the `/*` we marked literally.
5. Select all the text between the `/*` we just marked and the `*/` that follows it, including the spaces right next to these. Click the Mark button. RegexMagic adds field **2** matching the two lines of text we just marked. We'll adjust field **2** soon.
6. Select the `*/` and click the Mark button. RegexMagic adds field **3** to the sequence. It matches `*/` literally.
7. On the Match panel, select field **2** in the "select field" drop-down list.

8. Set “pattern to match field” to “match anything”.
9. Set “match anything except” to “text matched by the next field”. This ensures that field **2** can never match the text matched by field **3**. The comment must end at the first `*/`. This would be important if we added more fields after field **3** to match something that follows the comment. If that something is missing, we don’t want the regex engine to try to expand the comment until the next `*/` in the file.
10. Set the left hand “repeat this field” spinner for field **2** to zero and tick the “unlimited” checkbox to allow it to match any number of characters.
11. Set “how to repeat this field” to “as few times as possible”. We have to do this so that field **2** stops before the first `*/`. RegexMagic will refuse to generate the regex if we don’t select “as few times as possible” for a field that has “match anything except” set to “text matched by the next field”.
12. Click the Generate button on the Regex panel and you’ll get exactly the same regular expression:

```
/\*(?>.*?\/)
```

**Required options:** Dot matches line breaks; Default line breaks.

**Unused options:** Case sensitive; Exact spacing; `^$` don’t match at line breaks.

## 37. Increment ImageIndex Constants

When developing applications in Delphi, it is common to have a `TImageList` that holds the various glyphs used by the application. Those glyphs can then be used by other components like `TToolButton` by assigning the number of a glyph to the `ImageIndex` property. When adding new glyphs to the application, the easy solution is to add them to the end of the image list, so existing glyph number remain unchanged. But over time this can make a mess of the image list, making it harder to find existing images in the list that need to be reused by newly added components.

If you want to keep the image list organized, you'll want to insert new glyphs in the middle of the list, next to the glyphs they relate to. To make sure the existing glyphs are still referenced correctly, need a way to increment the `ImageIndex` properties with numbers equal or greater to the number of the glyph being inserted. Fortunately, you can easily generate a regular expression that matches these numbers with `RegexMagic`. Actually incrementing the numbers requires arithmetic. Regular expressions don't do math. But `PowerGREP` can do simple math on `regex` and `group` matches. So can a single line of Delphi code.

Below you'll find two examples, one using `PowerGREP` and another using some quick Delphi code, to increment `ImageIndex` properties like "`ImageIndex = 123`" in DFM files. The example also handles constants declarations like "`imgSomeName = 123`" and property assignments like "`ImageIndex := 123`". You can easily adjust these examples to handle differently named identifiers. You can use these examples as "Real world: increment `ImageIndex` constants" in the `RegexMagic` library.

### Increment ImageIndex Constants Using PowerGREP

First we prepare `PowerGREP` for a new search-and-replace.

1. Start `PowerGREP` 5. Version 4 would also work. The free trial version is sufficient to follow this example.
2. Use the File Selector panel to mark the folder(s) containing your Delphi source code.
3. Enter "`*.pas;*.dfm`" into the "include files" box.
4. Click the Clear button on the Action toolbar.
5. Set "action type" to "search-and-replace".
6. Click the `RegexMagic` button on the Action toolbar. It is important to launch `RegexMagic` from `PowerGREP`. This allows `RegexMagic` to send its entire formula back to `PowerGREP`.

Now we can generate the regular expression with `RegexMagic`.

7. On the Samples panel, paste in one new sample:

```
ImageIndex := 123
ImageIndex = 99
imgSomeName = 256
```

8. Select the first occurrence of `ImageIndex` in the sample.
9. Click the Mark button. `RegexMagic` automatically adds a field using the literal text pattern that matches `ImageIndex`.
10. Select `img` at the start of the third line in the example.

11. Click the Mark button. RegexMagic adds another field to match `img`. It puts this field and the previous field into an alternation field. That's because we left a gap between the two fields that we marked. RegexMagic correctly interprets that we want our regex to be able to match `ImageIndex` and `img` separately.
12. Select `SomeName` adjacent to `img`.
13. Click the Mark button. RegexMagic adds a field to match `SomeName` in sequence with the `img` field. So now we have field [2](#) matching `ImageIndex`, field [4](#) matching `Img` and field [5](#) matching `SomeName`.
14. In the “pattern to match field” drop-down list, select “basic characters” for field [5](#).
15. Tick the “lowercase letters”, “uppercase letters”, and “digits” checkboxes.
16. Set field [5](#) to repeat between 1 and 64 times. Though you could just tick “unlimited”, setting a reasonable upper limit is a good habit. Though not likely in this case, there are situations where this can prevent a runaway regular expression slowing down your application.
17. Select the space immediately after `ImageIndex` on the first line of the example.
18. Click the Mark button. RegexMagic adds field [6](#) to match the space. Because we marked this adjacent but out of sequence to another field, RegexMagic adds this as an alternative under field [1](#) and sets field [1](#) to be repeated twice. We need to fix that.
19. With field [6](#) still selected, click the Move Field Left button. It'll have [6](#) as its glyph. This moves field [6](#) out of the alternation. It is now in sequence with and below field [1](#). This means our regex first matches one of the two alternatives (`ImageIndex` or `imgSomeName`) and the matches the space.
20. In the “pattern to match field” drop-down list, select “basic characters” for field [6](#).
21. Make sure the “whitespace” option is ticked. This way field [6](#) can match spaces or tabs.
22. Set field [6](#) to repeat between 0 and unlimited times.
23. Select `:` in the sample.
24. Click the Mark button. RegexMagic adds field [7](#) to match a literal colon in sequence with field [6](#).
25. Set field [7](#) to repeat between 0 and 1 times. This makes the colon optional, allowing the regex to handle both constant declarations and property assignments.
26. Select the `=` adjacent to the `:` in the sample.
27. Click the Mark button. RegexMagic adds field [8](#) to match a literal equals sign.
28. Select the space immediately after the equals sign we just marked.
29. Click the Mark button. RegexMagic adds field [9](#).
30. In the “pattern to match field” drop-down list, select “basic characters” for field [9](#).
31. Make sure the “whitespace” option is ticked. This way field [9](#) can match spaces or tabs.
32. Set field [9](#) to repeat between 0 and unlimited times.
33. Select `123` in the sample text.
34. Click the Mark button. RegexMagic adds field [10](#). It automatically sets it to match any integer number.
35. We need to restrict field [10](#) to the range of numbers we actually want to increment. Tick the “limit the integers to these ranges” checkbox. In the edit control, replace `123` with a dotted range such as `42..999`. `42` should be the position in the image list where you'll be inserting the glyph. `999` should be any number that is surely higher than the highest existing image index. But don't go overboard with the end of the range as the higher the number the more complex the regex. A sequence of nines makes for a simpler upper end.
36. Set the “field validation mode” on the Match panel to “strict”. This is necessary to make RegexMagic limit the integer range to exactly what you specify.
37. Set “begin regex match at” and “end regex match at” to “start of word” and “end of word”. This ensures the regex does not partially match larger numbers. That would yield incorrect results when we increment the number.
38. On the Action panel, click the New button to add a capturing group.
39. Enter “number” as the group's name.
40. Select “[10](#) integer” in the “field” drop-down list. We now have a capturing group that will give us the value of the `ImageIndex` constant.

41. If you generate the regular expression then the Samples panel shows that our regex correctly matches the 3 numbers and their identifiers:

```
ImageIndex : 123
ImageIndex : 99
imgSomeName : 256
```

Now we can prepare the replacement with RegexMagic.

42. On the Action panel, set “action to take” to “replace one field”.  
 43. Set “field to replace” to “**10** integer”.  
 44. Set “how to replace” to “enter literal replacement text”.  
 45. Enter “%groupnumber:+1%” as the replacement string. RegexMagic doesn’t understand this. But PowerGREP does. It’s a match placeholder that inserts the match of the capturing group named “number” incremented by one.

Now we’re ready to perform the search-and-replace with PowerGREP. Resist the urge to copy and paste the regex from RegexMagic into PowerGREP. Instead click the Send button on the Regex panel in RegexMagic. This button only appears if you launched RegexMagic from PowerGREP (or another application that integrates with RegexMagic).

You can now preview or execute the search-and-replace in PowerGREP. You’ll see this regular expression:

```
\m(?<before10>(? :ImageIndex|img[0-9A-Za-z]{1,64})[\t ]*:?=[\t ]*)(?<number>[1-9][0-9]{2}|[5-9][0-9]|4[2-9])\M
```

**Required options:** Case sensitive; Exact spacing.

**Unused options:** Dot doesn’t match line breaks.

You’ll also see this replacement string:

```
${before10}%groupnumber:+1%
```

RegexMagic added an extra capturing group to the regex and a reference to it to the replacement string. This preserves the matches of fields 1 to 9.

What you don’t see in PowerGREP is that RegexMagic also sent the RegexMagic Formula (all your settings on the Samples, Match, and Action panels) to PowerGREP. PowerGREP does store this. Use the Action|Save or Action|Add to Library to save your PowerGREP action. The RegexMagic Formula will be stored inside the action. When you load the action into PowerGREP at a later time and click the RegexMagic button in PowerGREP then PowerGREP will send the RegexMagic formula back to RegexMagic. This way you can edit the formula in RegexMagic and send a newly generated regex (and updated settings) back to PowerGREP.

In this example that will be very handy. Each time you do this search-and-replace, you’ll want to increment a different range of numbers. By saving the RegexMagic formula, PowerGREP lets you do this quickly by launching RegexMagic from PowerGREP again, changing the range of numbers for field **10**, and sending the result back to PowerGREP.

This only works if you always launch RegexMagic from PowerGREP and always use the Send button to commit your regex. If you copy and paste, PowerGREP won't have the RegexMagic formula, and you won't be able to use RegexMagic to edit the regex later. You can't copy and paste a regex from PowerGREP or any other application into RegexMagic.

## Increment ImageIndex Constants Using Delphi

First, prepare the regular expression using RegexMagic as described in steps 7 through 41 above. Then we can prepare the replacement. Delphi does not have a feature similar to PowerGREP's match placeholders. So instead we'll tell RegexMagic to remove the matched number from the replacement. We'll add the incremented number with some Delphi code.

1. On the Action panel, set "action to take" to "replace one field".
2. Set "field to replace" to "**10** integer".
3. Set "how to replace" to "delete the matched text".
4. On the Regex panel, select TPerlRegEx (not TRegEx) for your version of Delphi. For this example, we'll select "Delphi 10.3–11 (TPerlRegEx)".
5. Turn off free-spacing, and turn off mode modifiers. Click the Generate button, and you'll get this regular expression:

```
[[[:<:]](?<before10>(?:ImageIndex|img[\dA-Za-z]{1,64})[\t ]*?:?=[\t ]*)(?<number>[1-9][0-9]{2}|[5-9][0-9]|4[2-9])[[[:>:]]]
```

**Required options:** Case sensitive; Exact spacing.

**Unused options:** Dot doesn't match line breaks; ^\$ don't match at line breaks; Numbered capture; Greedy quantifiers; Allow zero-length matches.

6. You'll also get this replacement string, which preserves the identifier and operator, but drops the number we matched:

```
`${before10}
```

7. On the Use panel, select the function "use regex object to search and replace through a string". You'll see this code:

```
var
  Regex: TPerlRegEx;
  ResultString: string;
  Regex := TPerlRegEx.Create;
  try
    Regex.RegEx := '[[[:<:]](?<before10>(?:ImageIndex|img[\dA-Za-z]{1,64})[\t ]*?:?=[\t ]*)(?<number>[1-9][0-9]{2}|[5-9][0-9]|4[2-9])[[[:>:]]]';
    Regex.Options := [];
    Regex.State := [];
    Regex.Subject := SubjectString;
    Regex.Replacement := `${before10}`;
    Regex.OnReplace := ComputeReplacement;
    Regex.ReplaceAll;
    ResultString := Regex.Subject;
  finally
    Regex.Free;
```

```

end;
procedure TMyClass.ComputeReplacement(Sender: TObject; var ReplaceWith:
string);
begin
  // You can vary the replacement text for each match on-the-fly
  // ReplaceWith defaults to Regex.Replacement, with backreferences
  substituted
end;

```

The reason we chose TPerlRegEx over TRegEx is that TPerlRegEx.ReplaceAll uses both the Replacement property and the OnReplace event. When our event handler ComputeReplacement is called, the ReplaceWith parameter holds the replacement string with the backreference already substituted. Since we told RegExMagic to only delete the match of field `10` from the replacement, all we need to do is to write one line of code to append the incremented image index:

```

procedure TMyClass.ComputeReplacement(Sender: TObject; var ReplaceWith: string);
begin
  ReplaceWith := ReplaceWith +
  IntToStr(StrToInt(Regex.Groups[Regex.NamedGroup('number')]) + 1)
end;

```

**Part 3**

# **RegexMagic Reference**



# 1. RegexMagic Assistant

The RegexMagic Assistant displays helpful hints as well as error messages while you work with RegexMagic. Press **Alt+1** on the keyboard to make the Assistant visible. Or, right-click on the caption or tab of any panel in RegexMagic, and click on View, Assistant. To hide the assistant, click the X button on its caption. In the default layout, the assistant is permanently visible along the right hand edge of RegexMagic's window.

## Helpful Hints

When you use the mouse, the assistant will explain the purpose of the menu item, button or other control that you point at with the mouse. When you use the Tab key on the keyboard to move the keyboard focus between different controls, the assistant describes the control that just received keyboard focus. If you move the mouse pointer over the assistant, the assistant will also explain the control that has keyboard focus, whether you pressed Tab or clicked on it.

Some of the assistant's hints mention other items that have an effect on or are affected by the control the assistant is describing. These will be underlined in blue, like hyperlinks on a web site. If you click on such a link, the assistant will move keyboard focus to the item the link mentions. Since you can only click on a link when moving the mouse pointer over the assistant, you will only be able click on a link when the assistant is describing the control that has keyboard focus. After you click, the assistant will automatically describe the newly activated control.

## Error Messages

Most applications display error messages on top of the application, blocking your view of the application and whatever the error may be complaining about. Clicking OK brings the application back to life again, but then you have to remember what the problem was before you can fix it.

RegexMagic uses a different approach. When there is a problem, RegexMagic will use the Assistant panel to deliver the message. If you closed the assistant, it will automatically pop up in the place it was last visible.

You can recognize error messages by their bold red headings. Hints have black headings. The assistant will continue to show the error message until you resolve the problem, or dismiss the error by clicking the Dismiss link below the error message. Meanwhile, the assistant will not display hints or descriptions. If the assistant was invisible before the error occurred, dismissing an error automatically hides the assistant. Otherwise, the assistant resumes showing hints.

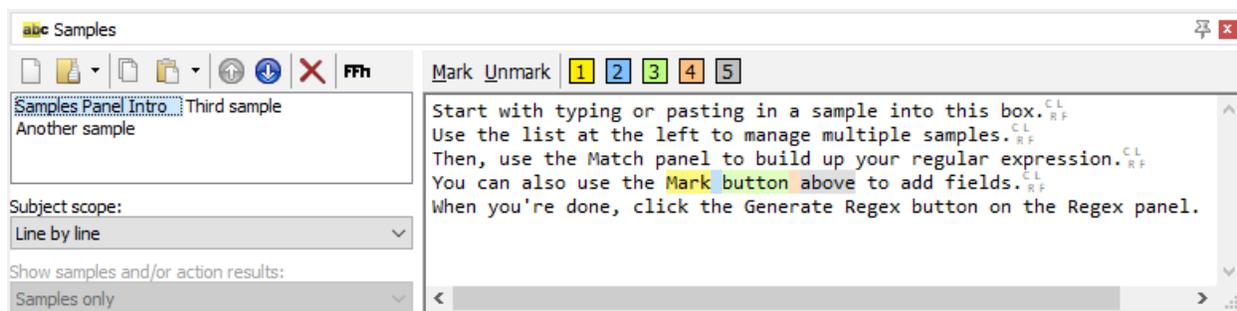
Error messages disappear automatically when you fix the problem. E.g. if clicking the Generate button triggers an error, the error message will automatically disappear if you fix the problem and click the Generate button again. You don't need to dismiss errors, unless you want to see the hints again before trying again.

## 2. Samples Panel Reference

The Samples panel is the place to provide sample text to base your regular expression on, and to test the regular expression against. Together, the Samples, Match, and Action panels define a RegexMagic formula to generate a regular expression, and possibly a replacement text.

In the default layout, the Samples panel shares the top left tab with the Match and Action panels. In all other layouts, the Samples panel has its own space in the top left corner of RegexMagic's window. All layouts allow you to have the Samples panel visible when working with the Match and Action panels, which is a must. You'll want to reference your sample text while defining the patterns to match it.

Press **Alt+2** on the keyboard to move keyboard focus to the Samples panel, and to make it visible if it was hidden. Or, select the Samples item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.



The screen shot shows the Samples panel holding three samples. The first one is selected. Five different fields have been marked in that sample.

### List of Samples

For RegexMagic to be able to do any auto-detection at all, you need to provide at least one sample text. This is the first thing to do when you want to create a new regular expression with RegexMagic. You can provide as many samples as you want, by adding them to the list at the left hand side of the Samples panel. Though you can generate a regular expression with just one sample, it's a good idea to test your regular expression on a representative sample of the input it will have to deal with, before letting it loose on your actual data.

### Hexadecimal Mode

Toggle the selected sample between text mode and hexadecimal mode.

In hexadecimal mode, RegexMagic's regular expression engine matches bytes rather than characters. This is appropriate for testing regular expressions on 8-bit binary data.

## Subject Scope

If you want to test on a large number of short samples, you can load them as a single sample in the list above. Set the subject scope to treat each line or each page in each sample in the list above as a separate subject. RegexMagic will feed the separated subjects one after the other into its regex engine. One subject will never affect another.

- **Whole sample:** Treat each sample in the list as one subject, even if it contains line breaks or page breaks.
- **Page by page:** Split items along page breaks, treating each page in each sample as a separate subject. Press Ctrl+Enter in the edit box to insert a page break, and Backspace to delete it. Page breaks are displayed as a colored line.
- **Line by line:** Treat each line in each sample as a separate subject. Use this option if your actual application never processes more than one line of text at the same time.

## Marking Fields

There are two ways to build up a regular expression in RegexMagic. One way is to add fields on the Match panel. The other way is to mark fields on the Samples panel.

This section describes exactly how RegexMagic operates when you mark fields. If you're new to RegexMagic, you may find this description incomprehensible. Instead, you should try out some of the examples. You'll find that marking samples is quite intuitive with just a little bit of practice.

If you're already familiar with RegexMagic and you want to learn exactly how the buttons on the Samples panel operate, read on. There are two basic ways to mark fields: adding New fields with the Mark button, and providing additional samples for existing fields with the colored buttons that appear once you've used the Mark button. Depending on which text you select and which buttons you click, RegexMagic can modify your regular expression in five ways in response to your commands.

### Marking Adjacent Fields Expands Your Regular Expression

To build up a regular expression by marking fields, select the first bit of text that the regular expression should match, and click the Mark button. The text you selected becomes highlighted in yellow, to indicate it is marked as field **1**. The field is added to the Match panel, where you can change its settings if you want.

To expand your regular expression, select a bit of text immediately after the first field and click the Mark button again. This tells RegexMagic to add field **2** right after the first one. Repeat this to add a third field or as many as you like.

### Marking Additional Samples Refines Patterns

Each time you click the Mark button, a new colored button appears above the sample text. Use these colored buttons when you want to mark additional sample text for an existing field, rather than add new fields as the Mark button does. E.g. if you've clicked the Mark button 3 times to mark one regex match that should consist of 3 fields, you can then use buttons **1**, **2**, and **3** to mark the text that should be matched by the

second regex match. You can continue selecting bits of text and clicking these 3 buttons to mark additional regex matches.

Whenever you mark more text as an existing field, RegexMagic refines the pattern for that field. Depending on the text you're working with, it may be easier to mark more text on the Samples panel to refine the pattern than to manually configure the pattern on the Match panel.

When you use the colored buttons to mark additional samples, it's your responsibility to do so in a consistent manner. If your regex consists of 3 pattern fields, then you always need to mark the fields 1, 2, and 3 in that order. RegexMagic can't adapt your regular expression if you contradict yourself.

## Marking Non-Adjacent Fields Creates Alternation

If you don't mark the fields adjacent and in order, RegexMagic tries to be clever about how it adds the new field to your regular expression. If you select some text and mark it, that becomes field 1. If you then select some non-adjacent text and click the Mark button, the previously marked text becomes field 2, and the newly marked text becomes field 3. Both will be inside an alternation field 1 that has no marked text. The alternation field allows your regular expression to match either the pattern selected for field 2, or the pattern selected for field 3. When using such a regex with a "find all" command, it will find both pieces of text that you marked. If you were to use the Mark button on any more pieces of text not adjacent to anything you've marked, those would become additional alternatives 4, 5, 6, etc. under the alternation field 1.

## Skipping Fields Creates Alternation

If use use the Mark button twice to mark two adjacent bits of text, you get a regex with two pattern fields 1 and 2. Suppose you then use button 1 to mark a bit of text that's not adjacent to the text you just marked. Then you select a bit of text adjacent to the text marked with button 1 and click the Mark button. In this case, the Mark button will not mark the text as field 2. Had that been what you wanted, you should have clicked button 2 instead of the Mark button. The colored buttons add samples to existing fields. The Mark button always adds a new field.

So you're marking a brand new field immediately adjacent to field 1. This tells RegexMagic that your regex should either match field 1 followed by field 2 (the first sample you marked), or it should match field 1 followed by the new field. To accomplish this, RegexMagic rearranges your regular expression. Field 1 stays as it is. Field 2 becomes an alternation field. The original field 2 is renumbered to 3 and becomes the first alternative under field 2. The newly marked text becomes field 4, as the second alternative under field 2.

## Continuing After Skipped Fields Creates Sequences

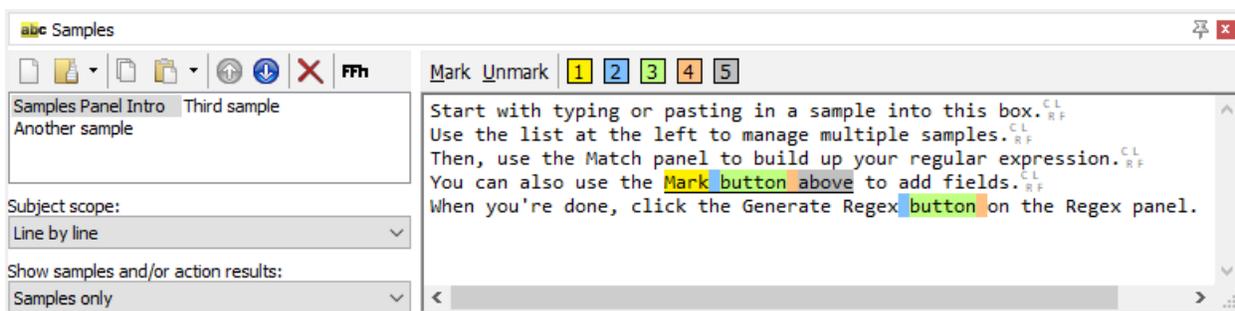
Continuing from the example in the previous section, suppose you select some text immediately adjacent to the text marked as field 4 and click the Mark button. This tells RegexMagic that the regex should match the new field immediately after field 4. But that field is inside the alternation field 2, so RegexMagic can't just stick the new field at the end of your regex. It needs to specify that field 2 consists of two alternatives. The first alternative is field 3, and the second alternative consists of two fields: field 4 followed by the newly marked field.

To do this, RegexMagic changes field **4** into a sequence field, and the old field **4** becomes field **5** and the first field in the sequence. The newly marked field becomes field **6** and the second field in the sequence.

## Check The Results of Your Regular Expression

When you're done marking fields on the Samples panel, you'll continue on the Match panel to specify more accurately how those fields should be matched. Then you'll go to the Action panel to tell RegexMagic to capture or replace certain fields. When you're done with the Samples, Match, and Action panels, you'll generate your regular expression on the Regex panel.

Finally, you'll complete the circle by coming back to the Samples panel. The highlighting will have changed. After generating your regular expression, the highlighting indicates the text actually matched by each field.



The five fields that were marked are now highlighted with more intense colors. Intense colors indicate that the regular expression actually matched that part of the text, and the color indicates which field actually matched the text. Muted colors as in the screen shot at the top of this page indicate text that you marked as a certain field, but that was not matched by the regular expression at all.

The five fields that were marked are also underlined. Underlining indicates that the text was matched by the same field that you marked the text for. Fields **2**, **3**, and **4** are highlighted a second time, without underlining. That indicates the regular expression found a match for those fields in text that was not marked at all. If text is matched by a field different from the one you marked the text for, it will be highlighted using the color of the field that actually matched the text, but the text will be struck out to indicate it was marked for a different field.

## Show Samples and/or Action Results

- **Samples only:** Show the editor for entering the sample text only.
- **Samples and results:** Show both the editor for entering the sample text, and the result of the search-and-replace or split action.
- **Results only:** Show the only the result of the search-and-replace or split action.

## Same Results in RegexMagic and Your Actual Application

Highlighting the matches of the generated regular expression is done with a special regular expression engine that accurately emulates the syntax and behavior of all the applications that RegexMagic can generate regular

expressions for. You can trust that the generated regular expression will find the exact same matches in your actual application as indicated by RegexMagic.

This assumes, of course, that you've correctly transferred the generated regex from RegexMagic to your actual application. The easiest way to do that is via the source code snippets that RegexMagic generates on the Use panel. The generated snippet automatically uses the correct string format for the regex and automatically sets all matching modes. If you copy and paste the regex from the Regex panel then you have to make sure that you also set the matching modes indicated on the Regex panel.

It also assumes that the sample text you're using in RegexMagic is the same text your actual application works on and that the "subject scope" setting corresponds with the way your application handles that text. When the scope is set to "whole sample" line breaks can make things a little tricky. Windows text files normally use CRLF line breaks. Linux and OS X use UNIX-style LF only line breaks. Classic Mac used CR only for line breaks. All editor controls in RegexMagic handle all these line break styles, as well as all other Unicode line breaks even though they are not commonly used in plain text files.

Most regular expression engines are not so smart, however. Many recognize only the line feed as a line break character. This mainly affects the "begin regex match at" and "end regex match at" settings on the Match panel when you set them to "start of line" and "end of line".

In many cases, you'll never notice this. For example, the regex engines in Perl, Python, and Ruby only recognize LF as a line break. Yet, when you read a Windows text file in its entirety into a variable in any of these languages, regexes set to match at the start and end of a line work just fine. The reason is that these languages open files in "text mode" by default. In this mode, when your script runs on Windows, reading from a file automatically converts CRLF line breaks into LF only, and writing to a file automatically does the opposite conversion.

To make it easier to test your regular expression, RegexMagic automatically converts line breaks in your samples to fit the regex engine of the selected application each time you generate the regular expression. If you generate a regex for Perl, Python, or Ruby, then line breaks in your samples are automatically converted to LF only.

In most situations you don't need to worry about this because most regex engines handle line breaks in a way that is consistent with what the overall environment does. One major exception among the applications supported by RegexMagic are the applications and programming languages based on the .NET framework. Since .NET is a Windows development library, you'll most likely be reading text files with Windows line breaks. The .NET classes for reading files do not do any line break conversion. But the .NET Regex class treats only the line feed character as a line break. This means that if you have a string that contains CRLF line breaks, then a regex set to match at the end of a line will not find the correct matches or any matches at all because the end-of-line anchor will try to match between the CR and the LF instead of before the CRLF.

Because the .NET Regex class only supports line feeds and because RegexMagic emulates only regex engines rather than complete programming languages, RegexMagic converts the line break style of your samples to LF only when you generate a regex for .NET. To match up the behavior between RegexMagic and your .NET application, you'll need to do the same conversion in your .NET application by replacing all matches of the literal string "\r" with nothing.

If your regex doesn't need to find matches that span multiple lines, then a less confusing solution may be to make your .NET application read files one line at a time, and pass each line separately to the regex. You can make RegexMagic do the same by selecting "line by line" as the subject scope. This way your subject strings will never contain any line breaks, so you don't need to worry how your regex deals with them.

### 3. Match Panel Reference

The Match panel is the place where you tell RegexMagic which parts of the text you want your regular expression to match, and what that text should look like or which patterns that text should fit. Together, the Samples, Match, and Action panels define a RegexMagic formula to generate a regular expression, and possibly a replacement text.

In the default layout, the Match panel shares the top left tab with the Samples and Action panels. Depending on which panel you last used, that tab will say “Samples, Match” or “Samples, Action”. Click that tab, and then on the Match tab inside it, to activate the Match panel in the default layout. In the side by side layout, the Match and Action panels are combined into one tab labeled “Match, Action”. Clicking that tab shows both panels. In the dual monitor tabbed layout, the Match panel has its own tab in RegexMagic’s main window. In the dual monitor side by side layout, the Match panel is always visible in the main window.

Press **Alt+3** on the keyboard to move keyboard focus to the Match panel, and to make it visible if it was hidden. Or, select the Match item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.

**Match**

Fields in the regular expression

Begin regex match at: Start of line

End regex match at: End of line

Kind of field: Alternation

Repeat: 1

Unlimited

How to repeat this field: As few times as possible

Kind of field: Pattern (Basic characters)

Repeat: 1

Unlimited

How to repeat this field: As many times as possible

Kind of field: Sequence

Repeat: 1

Unlimited

How to repeat this field: As few times as possible

Kind of field: Pattern (Number)

Repeat: 1

Unlimited

How to repeat this field: As many times as possible

Kind of field: Pattern (Date and time)

Repeat: 2

Unlimited

How to repeat this field: As many times as possible

Kind of field: Pattern (National ID)

Repeat: 1

Unlimited

How to repeat this field: As many times as possible

Patterns for the fields in the regular expression

Select field: National ID: US social security number

Pattern to match field: National ID

Kind of ID: US social security number

Overall options for generating the regular expression

Field validation mode: Average

## Build Up Your Regular Expression

There are two ways to build up a regular expression in RegexMagic. One way is to mark fields on the Samples panel. The other way is to add fields on the Match panel using the row of 12 buttons. These 12 buttons operate on or relative to the selected field. The colored rectangle to the left of the 12 buttons indicates the selected field. In the large screen shot above, field 6 is selected. To select another field, click on the colored rectangle to the left of the field's "kind of field" drop-down list. Clicking on field 3, for example, changes the row of buttons to this:



Different buttons are now enabled. Their glyphs have changed to indicate that they now operate on or relative to field 3 instead of field 6. The function of each of the 12 buttons is always the same. The examples given in the explanation assume you have six fields on the Match pattern as shown in the above screen shot. When an explanation for the same button has multiple examples, all the examples start from the original six fields. There are no consecutive examples. It will help to understand the explanations if you have already tried the examples that use pattern fields, sequence fields, and alternation fields.

1. **Add Prior Field:** Adds a new field before the selected field. With field 3 selected,  adds a new field between fields 2 and 3. The new field will get number 3. The old fields 3 through 6 will have their numbers shifted by one to 4 through 7. If you did not yet add any fields then this button appears as . Clicking it adds the first field.
2. **Add Next Field:** Adds a new field after the selected field. If the selected field is an alternation or sequence field, the new field is added after the entire alternation or sequence. With field 3 selected,  adds a new field 6. The new field will be part of the alternation under field 1, just like field 3. The new field will *not* be part of the sequence under field 3. The old field 6 will have its number shifted to 7.
3. **Add First Sub-Field:** Adds the first field to an alternation or sequence. This button is disabled when a pattern field is selected. With field 3 selected,  adds a new field 4. The new field becomes the first field in the sequence under field 3, just like the old field 4. The old fields 4 through 6 will have their numbers incremented by one.
4. **Add Last Sub-Field:** Adds a field to the end of an alternation or sequence. This button is disabled when a pattern field is selected. With field 3 selected,  adds a new field 6. The new field becomes the first field in the sequence under field 3, just like the old field 4. The old fields 4 through 6 will have their numbers incremented by one.
5. **Move Field Up:** Moves the selected field one position upwards. Essentially, the selected field swaps positions with the field before it. If those fields are alternation or sequence fields, the fields under them move along with them. With field 3 selected,  moves the whole sequence 3 through 5 upwards, changing their numbers to 2 through 4. The original field 2 ends up below (but not part of) the sequence with its number changed to 5. With field 6 selected,  moves field 6 up and over field 3. Field 6 has its number changed to 3, and fields 3 to 5 have their numbers changed to 4 through 6. This button is always disabled for field 1. It is also disabled for all fields that are the first field under an alternation or sequence field. In the example, it is disabled for fields 2 and 4.
6. **Move Field Down:** Moves the selected field one position downwards. This button is the exact reverse of the Move Field Up button. The selected field swaps positions with the field after it. If those fields are alternation or sequence fields, the fields under them move along with them. With field 2 selected,  does exactly the same as what  does with field 3 selected. With field 3 selected,  does exactly the same as what  does with field 6 selected. This button is disabled

for all fields that are the last field under an alternation or sequence field. In the example, it is disabled for fields **1**, **5**, and **6**.

7. **Move Field Left:** Moves the selected field out of an alternation or sequence if it is the first or last field in an alternation or sequence. If the selected field is an alternation or sequence itself, then all the fields under the selected field are moved along with it. If the field you moved to the left was the only field in the alternation or sequence, then the alternation or sequence field is automatically deleted. With field **2** selected in the example,  puts field **2** above field **1**. The numbers of these two fields will be swapped. With field **6** selected as in the screen shot,  moves that field to the left so it is no longer part of the alternation under field **1**.
8. **Move Field Right:** Moves the selected field into an alternation or sequence if the field that precedes it at the same level is an alternation or sequence field. In the example, is enabled only when field **6** is selected. Then  moves field **6** into the sequence under field **3** that precedes it. It will be the last field in the sequence and will keep its number. If field **6** were an alternation or sequence field too, then all the fields under it would move along with it.
9. **Move Field Right and Down:** Moves the selected field into an alternation or sequence if the field that follows it is an alternation or sequence field. In the example, is enabled only when field **2** is selected. Then  moves field **2** into the sequence under field **3** that precedes it. It will be the first field in the sequence. The numbers of fields **2** and **3** will be swapped. Fields **4** and **5** will remain a part of the sequence. If field **2** were an alternation or sequence field too, then all the fields under it would move along with it, and their numbers would be shifted accordingly.
10. **Delete One Field:** Deletes a single field. If the selected field is an alternation or sequence field, then the fields under it are *not* deleted. If the selected field is a pattern field and it is the only field inside an alternation or sequence, then the alternation or sequence field is automatically deleted along with its only pattern. When field **3** is selected in the example,  deletes only field **3**. Fields **4** and **5** are automatically moved to the left into the alternation under field **1**. Fields **4** through **6** will have their field numbers decremented by one.
11. **Delete List of Fields:** Deletes an alternation or sequence field along with all fields under it. If the deleted field was the only field inside another alternation or sequence, then the other alternation or sequence is automatically deleted along with it. When field **3** is selected in the example,  deletes fields **3** through **5**. Field **6** will be renumbered to **3**.
12. **Delete All Fields:** Deletes all the fields on the Match panel.  deletes all the fields in the example. Because “begin/end regex match at” are set to “start/end of line”, this would leave you with a regular expression that matches only blank lines.

## Begin Regex Match At

Specify at which position(s) in the text the regular expression match(es) may begin.

- **Anywhere:** Allow the regular expression match to begin anywhere in the text.
- **Start of text:** The regular expression match must begin at the start of the text. The regular expression can never find more than one match in a single input text.
- **Start of line:** If the text consists of multiple lines, the regular expression match can begin at the start of each line. For single-line texts, the match can only begin at the start of the text.
- **Start of word:** The regular expression match can begin at the start of any of the words in the text. A “word” is a string of one or more characters that are letters, digits, and/or underscores.
- **Start of attempt:** The regular expression match must begin at the start of the match attempt which is at the start of the text for the first match and at the end of the previous match for all following matches. All matches found by the regular expression, will be adjacent to each other and to the start of the input text.

To make the regular expression begin with something else, such as a certain word, select “anywhere” as the starting position. Then select the text at which the regular expression should begin in the sample text, and click button **1** above the sample text to mark it as the first field.

## End Regex Match At

Specify at which position(s) in the text the regular expression match(es) must end.

- **Anywhere:** Allow the regular expression match to end anywhere in the text. It’s best to use this setting until you are done adding all the fields to your regular expression. You can better follow your progress if you allow the partly built regular expression to match part of the line or text, even if the final regex has to match the line or text entirely.
- **End of text:** The regular expression match must end at the end of the text. The regular expression can never find more than one match in a single input text.
- **End of line:** If the text consists of multiple lines, the regular expression match must end at the end of a line. For single-line texts, the match must end at the end of the text.
- **End of word:** The regular expression must end at the end of any of the words in the text. A “word” is a string of one or more characters that are letters, digits, and/or underscores.

## Settings for Each Field

### Kind of Field

Select what kind of field you want to add to the regular expression here.

- **Pattern:** Match a special pattern that RegexMagic can generate for you, such as a number in a range, a date, or an email address.
- **Alternation:** Add a list of fields under this field, and match a single one of those.
- **Sequence:** Add a list of fields under this field, and match all of those, one after the other. Use a sequence field inside an alternation field if you want to use multiple fields to specify one of the alternatives.

If you change a pattern field to become an alternation or sequence field, then one field is added to the alternation or sequence. The new field will use the same pattern with the same settings that the changed field used.

If you change an alternation field or sequence field to become a pattern field, then it will use the pattern from the first pattern field inside the alternation or sequence. All fields that were inside the alternation or sequence will be deleted.

### Repeat This Field

Specify how many times this field can or should be repeated.

Use the left hand spinner box to set the minimum number of times the field should be matched. If you set the minimum to zero, the field becomes optional.

Use the right hand spinner box to set the maximum number of times the field can be repeated. If you don't want to specify a specific upper limit, tick the "unlimited" checkbox.

Repeating a **pattern** field allows the pattern to be matched multiple times, and find a different match each time. If your field has a pattern that matches any single letter, and you repeat that field 3 times, then your regular expression can match ABC, XYZ, or any other 3-letter combination.

Repeating an **alternation** repeats the alternation as a whole. A different field from the alternation can match with each repetition. If you have an alternation with 3 fields that match A, B, and C, then repeating that alternation 4 times allows your regular expression to match AAAA, ABCA, BCAA, or any other 4-letter combination of A, B, and/or C.

Repeating a **sequence** repeats the sequence as a whole. If you have a sequence with 3 fields that match A, B, and C, then repeating that sequence 4 times makes your regex search for ABCABCABCABC.

## How to Repeat This Field

Choose in which way you want the field to be repeated.

- **As many times as possible:** Try to repeat the field as many times as possible. Reduce the number of repetitions only when required to allow the remainder of the regular expression to match.
- **As few times as possible:** Repeat the field only the minimum number of times required. Continue repeating the field only when required to allow the remainder of the regular expression to match.

## Patterns for The Fields in The Regular Expression

For each field listed on the Match panel as a "pattern" field, you can select which pattern the field should have. The pattern specifies what kind of text the field should match: a date, a number, an email address, etc. All the patterns provide various options to restrict the acceptable values. E.g. the "number" pattern allows you to limit the numbers your regex will match to a certain range, require or disallow decimals, etc.

In the screen shot of the Match panel, the "basic characters" pattern was selected for field 2, the "number" pattern was selected for field 4, the "date and time" pattern for field 5, and the "national ID" pattern for field 6. Field 5 is selected in the "select field" drop-down list, allowing its pattern to be edited. The "pattern to match field" drop-down list is always present, allowing you to choose a completely different pattern for that field. Below that list you can configure the pattern. For the "national ID" pattern, there is only one drop-down list labeled "kind of ID" that allows you to choose a particular national ID such as a US social security number.

To edit the pattern of another field, use the "select field" drop-down list. Only pattern fields appear in this list. You can also click on the colored squares to the left of the settings for each pattern field to edit that field's pattern. Clicking the colored square of an alternation or sequence selects the pattern of the first pattern field inside the alternation or sequence.

- Pattern used by another field: Use the same pattern as another field, but allow the two fields to match different text, as long as it fits the pattern.
- Text matched by another field: Require this field to match the exact same text already matched by a previous field.
- Match anything: Do not use any pattern to restrict the contents of the field.
- Unicode characters: Restrict this field to certain kinds of Unicode characters.
- Basic characters: Restrict this field to certain ASCII characters.
- Character masks: Require this field to match one of a list of simple character masks.
- List of literal text: Require this field to match one of a list of literal text values.
- Literal text: Make this field always match the same bit of text.
- Number: Match floating point numbers or monetary values.
- Integer: Match integer numbers in decimal, hexadecimal, octal and/or binary notation.
- Date and time: Match a date and/or a time.
- Email address: Match an email address.
- URL: Match an Internet address.
- Country: Country codes and/or names (ISO 3166).
- Currency: Currency codes (ISO 4217).
- Credit card number: Match a credit card number.
- National ID: ID card numbers, social security numbers, license plate numbers, etc. used in particular countries.
- VAT number: Value Added Tax numbers used in the European Union.
- GUID: Globally Unique Identifier.
- Regular expression: Use an arbitrary regular expression to validate this field.

## Overall Options for Generating The Regular Expression

### Field Validation Mode

When creating a regular expression, there is always a trade-off between, on one hand, making the regular expression match exactly what you want but not what you don't want, and on the other hand, keeping the regular expression fast and simple. If you know the data to be valid, you can use a fast and loose regex. If some of the data may be invalid, and you can't use programming logic to filter the regex matches, a strict and complex regex may be appropriate.

In RegexMagic, the field validation mode determines how tightly the generated regular expression will correspond with the patterns that you have configured for each of the fields.

- **Strict:** Restrict the fields to exactly what you have specified. This option generally yields long and complex regular expressions that may exceed the capabilities of more limited regex flavors. Use this option only if the regular expression is the only means to filter the input.
- **Average:** Restrict the fields to the validation patterns that you specified as much as possible, as long as it doesn't require a very long and complex regular expression. Use this mode when the strict regex is too slow or too long.
- **Loose:** Create a straightforward regular expression that checks if the fields roughly correspond with the validation patterns. Only obviously invalid fields will be filtered out by the regular expression. This is generally the best choice when programming, as it will yield a fast and compact regular expression. Additional programming logic will be more efficient at filtering the actual regex matches as needed.

- **None:** Use a single character class for each field. This mode does not really validate any fields, but the character classes will make the regex fail faster on data that doesn't fit the overall structure at all. This is the best option if you know that all your data is valid, and you need the regex to grab it.

The exact effects of these four modes are indicated in the RegexMagic Assistant's hints for each of the various settings you can make for each of the patterns. If an option's does not indicate that it works differently for any of the modes, that means the Strict, Average, and Loose modes all respect the option strictly. Since the None mode always builds just one character class per field, that is not specifically indicated in the hints.

## 4. Pattern: Pattern used by another field

“Pattern used by another field” is one of the patterns that you can select on the Match panel. If two or more fields need to match the same kind of pattern, but not match exactly the same text, you need to specify the pattern only for the first field that needs it. For the other fields, you can select “pattern used by another field” and reference the first field.

Use the pattern from the field:

A dropdown menu with a yellow square containing the number '1' on the left, the text 'Number' in the middle, and a small downward-pointing arrow on the right.

### Use The Pattern from The Field

Select the field of which this field should use the pattern.

## 5. Pattern: Text matched by another field

“Text matched by another field” is one of the patterns that you can select on the Match panel. This pattern requires a field to match the exact same text that was matched by a previous field in the regular expression.

Match the text matched by the field:

Case insensitive

### Match The Text Matched By The Field

Select the field that matched the text that this field should match again.

### Case Insensitive

Turn on to ignore differences between lowercase and uppercase letters in the regex and subject string. The regex “word” will match “Word”, “woRD”, etc.

Turn off to treat uppercase and lowercase letters as different letters. The regex “word” will match “word” but not “Word”.

## 6. Pattern: Match anything

“Match anything” is one of the patterns that you can select on the Match panel. This pattern allows a field to match any text, except perhaps for certain characters that may occur in the next field. The repetition settings for the field determine how many characters the field can or must match.

Match anything except: First character of the next field Character to escape delimiters: Backslash  Can span across lines

### Match Anything Except

Usually, when people say that a field can match “anything”, the field doesn’t truly allow everything. The exact contents are unspecified, but the field does stand in relation to the text around it.

- **Characters allowed by the next field:** Do not allow any of the characters that may appear in the next field.
- **First character of the next field:** Do not allow any of the characters that may occur at the start of the next field.
- **Unescaped first character of the next field:** Do not allow any of the characters that may occur at the start of the next field, except when those characters are preceded by a specific escape character.
- **Text matched by the next field:** Do not allow this field to match any text that can be matched by the next field, while still allowing the field to match characters that may appear in the next field.
- **Nothing:** Really allow any character. This may yield inefficient regular expressions, particularly if you have two fields that allow anything with unlimited repetition.

### Character to Escape Delimiters

If you set “what kind of anything” to “anything including an escaped delimiter”, specify the character to be used to escape the delimiter characters that occur within the field itself.

### Can Span Across Lines

Turn on to allow this field to match more than one line of text.

Turn off to require this field’s match to stay within one line.

## 7. Pattern: Unicode characters

“Unicode characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of Unicode characters. The repetition settings for the field determine how many characters the field can or must match.

Individual characters:

Case insensitive

Match all characters except the selected ones

Unicode categories:

<input type="checkbox"/> uppercase letters	<input type="checkbox"/> decimal digits	<input type="checkbox"/> final punctuation	<input type="checkbox"/> paragraph separators
<input type="checkbox"/> lowercase letters	<input type="checkbox"/> letter numbers	<input type="checkbox"/> other punctuation	<input type="checkbox"/> control characters
<input type="checkbox"/> title case letters	<input type="checkbox"/> other numbers	<input type="checkbox"/> math symbols	<input type="checkbox"/> formatting characters
<input type="checkbox"/> modifier letters	<input type="checkbox"/> connector punctuation	<input checked="" type="checkbox"/> currency symbols	<input type="checkbox"/> surrogates
<input type="checkbox"/> letters without case	<input type="checkbox"/> dash punctuation	<input type="checkbox"/> modifier symbols	<input type="checkbox"/> private use characters
<input type="checkbox"/> non-spacing marks	<input type="checkbox"/> open punctuation	<input type="checkbox"/> other symbols	<input type="checkbox"/> unassigned code points
<input type="checkbox"/> spacing combining marks	<input type="checkbox"/> close punctuation	<input type="checkbox"/> space separators	
<input type="checkbox"/> enclosing marks	<input type="checkbox"/> initial punctuation	<input type="checkbox"/> line separators	

### Individual Characters

Type in a list of characters that you want to allow in this field.

You can only type in characters that don't fit any of the categories that you've turned on below.

### Case Insensitive

Turn on to allow the field to match both the lowercase and uppercase variants of all letters that you typed in as individual characters. When on, the lowercase and uppercase letter groups will also match both variants, even if you turned on only one of them.

Turn off to make the field respect the case of the individual characters you've typed in.

### Match All Characters Except The Selected Ones

Turn on to make the field match any character except those that you specified.

Turn off to make the field match only those characters that you specified.

### Unicode Categories

Turn on one or more Unicode categories to have the field include all characters that are in one of the selected categories. Turning on categories removes all characters with the selected categories from the list of individual characters above.

Turn off Unicode categories if you don't want the field to include any characters from certain categories, except for those characters that you have entered as individual characters. Turning off a Unicode category puts back any individual characters that were removed by turning that category on.

- lowercase letters: lowercase letters that have uppercase variants.
- uppercase letters: uppercase letters that have lowercase variants.
- title case letters: letters that appear at the start of a word when only the first letter of the word is capitalized.
- modifier letters: special characters that are used like a letter.
- other letter: letters or ideographs that do not have lowercase and uppercase variants.
- non-spacing marks: characters intended to be combined with other characters without taking up extra space (e.g. accents, umlauts, etc.).
- spacing combining marks: a character intended to be combined with another character that takes up extra space (vowel signs in many Eastern languages).
- enclosing marks: a character that encloses the character is is combined with (circle, square, keycap, etc.).
- decimal digits: digits 0 through 9 in all scripts except ideographic scripts.
- letter numbers: numbers that look like letters, such as a Roman numerals.
- other numbers: superscript or subscript digits, and numbers that are not digit 0..9 (excluding numbers from ideographic scripts).
- dash punctuation: all kinds of hyphens and dashes.
- open punctuation: all kinds of opening brackets.
- close punctuation: all kinds of closing brackets.
- initial punctuation: all kinds of opening quotes.
- final punctuation: all kinds of closing quotes.
- connector punctuation: punctuation characters such as underscores that connects words.
- other punctuation: all kinds of punctuation characters that are not dashes, brackets, quotes, or connectors.
- math symbols: all mathematical symbols.
- currency symbols: all currency signs.
- modifier symbols: combining characters (mark) as full characters on their own.
- other symbols: various symbols that are not math symbols, currency signs, or combining characters.
- space separators: a whitespace character that is invisible, but does take up space.
- line separators: line separator character U+2028.
- paragraph separators: paragraph separator character U+2029.
- control characters: ASCII 0x00..0x1F and Latin-1 0x80..0x9F control characters.
- formatting characters: invisible formatting indicators.
- private use characters: all code points reserved for private use.
- surrogates: one half of a surrogate pair in UTF-16 encoding.
- unassigned: all code points to which no characters have been assigned.

## 8. Pattern: Basic characters

“Basic characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of ASCII characters. The repetition settings for the field determine how many characters the field can or must match.

Individual characters:

Case insensitive

Lowercase letters  Uppercase letters  Match all characters except the selected ones

Digits  Punctuation and symbols

Whitespace  Line breaks

### Individual Characters

Type in a list of characters that you want to allow in this field.

You can only type in characters that don't fit any of the categories that you've turned on below.

### Case Insensitive

Turn on to allow the field to match both the lowercase and uppercase variants of all letters that you typed in as individual characters. When on, the lowercase and uppercase letter groups will also match both variants, even if you turned on only one of them.

Turn off to make the field respect the case of the individual characters you've typed in.

### Match All Characters Except The Selected Ones

Turn on to make the field match any character except those that you specified.

Turn off to make the field match only those characters that you specified.

### Lowercase Letters

Turn on to allow any lowercase letter a..z in this field. Doing so will remove all lowercase letters from the list of individual characters above.

Turn off to allow only those lowercase letters that you typed in as individual characters. Doing so will put back any individual lowercase letters that were removed by turning this option on.

### Uppercase Letters

Turn on to allow any uppercase letter A..Z in this field. Doing so will remove all uppercase letters from the list of individual characters above.

Turn off to allow only those uppercase letters that you typed in as individual characters. Doing so will put back any individual uppercase letters that were removed by turning this option on.

## **Digits**

Turn on to allow any digit 0..9 in this field. Doing so will remove all digits from the list of individual characters above.

Turn off to allow only those digits that you typed in as individual characters. Doing so will put back any individual digits that were removed by turning this option on.

## **Punctuation and symbols**

Turn on to allow any ASCII punctuation or symbol character in this field. Doing so will remove all punctuation and symbol characters from the list of individual characters above.

Turn off to allow only those punctuation and symbol characters that you typed in as individual characters. Doing so will put back any punctuation and symbol characters that were removed by turning this option on.

## **Whitespace**

Turn on to allow spaces and tabs in this field. Doing so will remove spaces and tabs from the list of individual characters above.

Turn off to allow only spaces and/or tabs if you typed them in as individual characters. Doing so will put back any individual spaces and tabs that were removed by turning this option on.

## **Line Breaks**

Turn on to allow this field to span across multiple lines.

Turn off to force the field to stay within one line.

## 9. Pattern: Character masks

“Character masks” is one of the patterns that you can select on the Match panel. With this pattern you can easily make a field match a certain combination of letters, digits, and punctuation. This pattern uses text masks similar to the masks used by masked edit controls in various development tools to force the user to enter a certain combination of letters, digits, and/or punctuation.



### Character Masks

Type in one or more character masks for the strings that the field is allowed to match. You can enter one mask per line. The characters listed below have a special meaning in the masks. All other characters simply match themselves. Masks are always applied case insensitively.

- **L**: Allows any letter.
- **A**: Allows any alphanumeric character (i.e. letter or digit).
- **C**: Allows any character.
- **9**: Allows any digit.
- **#**: Allows any digit or the + or - sign.
- **\**: Escapes the character that follows it, making it match itself literally. To match any of the 6 characters in this list (and the lowercase equivalents of the 3 letters), precede them with a backslash.

In the “none” field validation mode, the field will allow any of the characters allowed by any of the masks anywhere in the field. The shortest and longest mask determine the minimum and maximum length of the field.

In the “loose”, “average”, and “strict” field validation modes, the field will use the masks exactly as you specified them.

## 10. Pattern: List of literal text

“List of literal text” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a piece of text from a list that you provide. If you repeat the field, a different item from the list may be matched with each repetition.

Case insensitive     
  Delimit list with page breaks     
  Match any text except the sp     
  Can span across lines

Mary<sup>CL</sup><sub>RF</sub>  
 Sue<sup>CL</sup><sub>RF</sub>  
 Betty

### Case Insensitive

Turn on to make the field ignore the differences in uppercase and lowercase letters between the list of literal text below and the actual values the regex will match.

### Delimit List with Page Breaks

Turn on to use page breaks to delimit the items in the list of literal text below. This will allow you to specify values that span across multiple lines. Press Ctrl+Enter to insert a page break. A horizontal line will represent the page break. The Backspace and Delete keys will delete page breaks just like line breaks and other characters.

Turn off to use line breaks to delimit the list of literal text. This way you cannot specify text that spans across multiple lines, but you can easily paste in a list of text. Press Enter to insert a line break.

Whether you use line breaks or page breaks only affects the way you will specify the list in RegexMagic. It does not affect the final regular expression given the same list of literal text appropriately delimited with line breaks or page breaks. The automatic pattern detection only uses page breaks when it detects multi-line values.

### Match any text except the specified text

Turn on to make the field match any text except the text that you have specified. The minimum and maximum number of times you set to repeat this field then specifies the minimum and maximum number of characters this field can match.

Turn off to make the field match the text that you have specified. If you set the field to repeat more than once, the field will try to match the entire block of text as many times as you specified.

## **Can Span Across Lines**

Turn on to allow this field to match more than one line of text. Turn off to require this field's match to stay within one line.

This option is only used when “match any text except the specified text” is turned on.

## **List of Literal Text**

Enter the text this field should allow. Press Enter to insert a line break to delimit each piece of text. Press Ctrl+Enter to insert a page break when delimiting the list of values with page breaks.

This control is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual.

In the “none” field validation mode, the field will allow any combination of any of the characters present in the list of literal text. The shortest and longest pieces of text determine the minimum and maximum length of the field.

In the “loose”, “average”, and “strict” field validation modes, the field will use each piece of literal text exactly as you specified it.

## 11. Pattern: Literal text

“Literal text” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a specific piece of text exactly as you type it in.

Case insensitive
  Match any text except the specified text
  Can span across lines

```
*great*
```

### Case Insensitive

Turn on to make the field ignore the differences in uppercase and lowercase letters between the text you specify below and the actual text the regex will match.

### Match any text except the specified text

Turn on to make the field match any text except the text that you have specified. The minimum and maximum number of times you set to repeat this field then specifies the minimum and maximum number of characters this field can match.

Turn off to make the field match the text that you have specified. If you set the field to repeat more than once, the field will try to match the entire block of text as many times as you specified.

### Can Span Across Lines

Turn on to allow this field to match more than one line of text. Turn off to require this field’s match to stay within one line.

This option is only used when “match any text except the specified text” is turned on.

### Literal Text

Type in or paste in the text this field should match.

This control is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual.

The literal text can span across multiple lines. Press Enter to insert a line break.

## 12. Pattern: Literal bytes

“Literal bytes” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a specific piece of text exactly as you type it in.

Match any bytes except the specified by  Can span across lines

00000000	00	10	2A	3B	4C	5D	6E	7F	8A	FF	.*;L]n Šÿ
----------	----	----	----	----	----	----	----	----	----	----	-----------

### Match any bytes except the specified bytes

Turn on to make the field match any bytes except the sequence of bytes that you have specified. The minimum and maximum number of times you set to repeat this field then specifies the minimum and maximum number of bytes this field can match.

Turn off to make the field match the sequence of bytes that you have specified. If you set the field to repeat more than once, the field will try to match the entire sequence of bytes as many times as you specified.

### Can Span Across Lines

Turn on to allow this field to match line breaks. Turn off to disallow line breaks.

This option is only used when “match any bytes except the specified bytes” is turned on.

### Literal Bytes

Type in or paste in the sequence of bytes this field should match. Matching literal bytes only works correctly when the regular expression engine works on 8-bit data. RegexMagic’s regex engine works in 8-bit mode when your samples are in hexadecimal mode.

This control is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. If the clipboard contains the hexadecimal representation of the bytes you want, paste them into the left hand side of the editor. If the clipboard contains the actual bytes, paste them into the right hand side of the editor.

## 13. Pattern: Control characters

“Control characters” is one of the patterns that you can select on the Match panel. Use this pattern to restrict a field to a certain set of ASCII control characters. The repetition settings for the field determine how many characters the field can or must match.

Control characters:

<input type="checkbox"/> 00 Null	<input type="checkbox"/> 08 Backspace	<input type="checkbox"/> 10 Data link escape	<input type="checkbox"/> 18 Cancel
<input type="checkbox"/> 01 Start of heading	<input checked="" type="checkbox"/> 09 Character tabulation	<input type="checkbox"/> 11 Device control one	<input type="checkbox"/> 19 End of medium
<input type="checkbox"/> 02 Start of text	<input type="checkbox"/> 0A Line feed (LF)	<input type="checkbox"/> 12 Device control two	<input type="checkbox"/> 1A Substitute
<input type="checkbox"/> 03 End of text	<input checked="" type="checkbox"/> 0B Line tabulation	<input type="checkbox"/> 13 Device control three	<input type="checkbox"/> 1B Escape
<input type="checkbox"/> 04 End of transmission	<input type="checkbox"/> 0C Form feed (FF)	<input type="checkbox"/> 14 Device control four	<input type="checkbox"/> 1C Information separator four
<input type="checkbox"/> 05 Enquiry	<input type="checkbox"/> 0D Carriage return (CR)	<input type="checkbox"/> 15 Negative acknowledge	<input type="checkbox"/> 1D Information separator three
<input type="checkbox"/> 06 Acknowledge	<input type="checkbox"/> 0E Shift out	<input type="checkbox"/> 16 Synchronous idle	<input type="checkbox"/> 1E Information separator two
<input type="checkbox"/> 07 Bell	<input type="checkbox"/> 0F Shift in	<input type="checkbox"/> 17 End of transmission block	<input type="checkbox"/> 1F Information separator one

Match all characters except the selected ones

### Control Characters

Select the ASCII control characters that this field should match.

### Match All Characters Except The Selected Ones

Turn on to make the field match any character except those that you specified.

Turn off to make the field match only those characters that you specified.

## 14. Pattern: Number

“Number” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a decimal number that may or may not have decimals, thousand separators, currency symbols, signs, exponents, etc.

Minimum value of integer part: 1	Maximum value of integer part: 14	<input checked="" type="checkbox"/> Limit integer part <input type="checkbox"/> Allow plus sign <input type="checkbox"/> Allow minus sign <input type="checkbox"/> Allow parentheses <input type="checkbox"/> Sign is required <input type="checkbox"/> Whitespace allowed after sign <input type="checkbox"/> Thousand separators are required <input type="checkbox"/> Allow leading zeros <input checked="" type="checkbox"/> Require integer part <input type="checkbox"/> Allow exponent <input type="checkbox"/> Currency sign or code required
Decimal separator: Period	Min. and max. number of decimals 0 1	
Thousand separator: None	Currency sign/code position: Before only	
Currency sign: None	Currency codes:	

### Minimum Value of Integer Part

Type in the minimum number that the field is allowed to match when limiting values. The minimum only restricts the integer part. It does not restrict the fraction or exponent.

The minimum is ignored in the “none” and “loose” field validation modes, which always allow any integer. In “average” mode, the field will allow any integer part with at least as many digits as the minimum value. In “strict” mode, the regular expression will limit the integer part to the exact minimum value, which may require a complex regular expression.

### Maximum Value of Integer Part

Type in the maximum number that the field is allowed to match when limiting values. The maximum only restricts the integer part. It does not restrict the fraction or exponent.

The maximum is ignored in the “none” and “loose” field validation modes, which always allow any integer. In “average” mode, the field will allow any integer part with at most as many digits as the maximum value. In “strict” mode, the regular expression will limit the integer part to the exact maximum value, which may require a complex regular expression.

### Decimal Separator

Select the style of the decimal separator that the number should use. This option only has an effect when the maximum number of decimals is not zero.

- **Any:** Allow both the period and the comma as the decimal separator.
- **Period:** Allow the period as a decimal separator.
- **Comma:** Allow the comma as a decimal separator.

### Minimum Number of Decimals

Enter the minimum number of decimals that the number should have.

### Maximum Number of Decimals

Enter the maximum number of decimals that the number is allowed to have.

### Thousand Separator

Select the style of the thousand separators that the number is allowed to use.

- **None:** No thousand separators allowed.
- **Any:** Any of the 3 thousand separators below are allowed.
- **Comma:** Allow the comma as a thousand separator.
- **Quote:** Allow the single quote as a thousand separator.
- **Space:** Allow a space as a thousand separator.

Note that with the field validation mode set to “none” or “loose”, the regular expression will allow the thousands separator anywhere in the integer part of the number. In “average” and “strict” modes, the regular expression will require the thousands separator to properly split the number in groups of 3 digits.

### Currency Sign or Code Position

Choose whether the allowed currency sign(s) and/or code(s) should appear before the number, after the number, or either before or after the number.

Note that if you select “either”, the regular expression will allow the number to have two currency signs, one before and one after it, unless you set the field validation mode to “strict”.

### Currency Sign

Choose to allow one of five common currency signs, none of them, or all of them.

### Currency Codes

Type in a list of currency codes that the field should allow. Delimit the codes with semicolons.

### Limit Integer Part

Turn on to allow the field to match only numbers with the integer part between a certain minimum and maximum number.

Turn off to allow any number.

This option has no effect in the “none” and “loose” field validation modes, which always allow any number.

### **Allow Plus Sign**

Turn on to allow positive numbers to be preceded with an optional + sign.

Turn off to disallow the + sign.

### **Allow Minus Sign**

Turn on to allow negative numbers to be preceded with a - sign.

Turn off to disallow the - sign.

When limiting the integer part with a negative minimum, you must turn on either “allow minus sign” or “allow parentheses” or both.

### **Allow Parentheses**

Turn on to allow negative numbers to be enclosed between ( and ) parentheses.

Turn off to disallow the parentheses.

When limiting the integer part with a negative minimum, you must turn on either “allow minus sign” or “allow parentheses” or both.

### **Sign Is Required**

Turn on to require positive numbers to use the plus sign.

Turn off to allow positive numbers without any sign.

This option does not have any effect unless allow plus sign is turned on.

### **Whitespace Allowed After Sign**

Turn on to allow spaces between the sign and the number.

Turn off to require the sign to be immediately adjacent to the number.

This option does not have any effect unless one or more signs are allowed.

### **Thousand Separators Are Required**

Turn on to require the number to use the selected thousand separator.

Turn off to make the thousand separator optional.

This option does not have any effect when the thousand separator is set to “none”.

This option has no effect in the “none” and “loose” field validation modes, which do not check if the thousand separators are positioned exactly.

### **Allow Leading Zeros**

Turn on to allow the field to be preceded by one or more zeros beyond the normally required or allowed number of digits when limiting the values accepted by the field.

This option has no effect in the “none” and “loose” field validation modes, which always allow any integer with any number of digits.

### **Require Integer Part**

Turn on to require the number to always have an integer part, even when that is zero.

Turn off to make the integer part optional for fractional numbers between -1 and 1.

This option has no effect when the number cannot have decimals, or when the integer part is limited to exclude zero.

### **Allow Exponent**

Turn on to allow the number to be written in scientific notation with an exponent.

Turn off to disallow the exponent.

### **Currency Sign or Code Required**

Turn on to require the number to have a currency sign or code.

Turn off to make the currency sign or code optional.

This option only has an effect if one or more currencysign(s) and/or code(s) are allowed.

## 15. Pattern: Integer

“Integer” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an integer number in decimal, hexadecimal, octal, and/or binary notation.

Allowed integer formats:

<input checked="" type="checkbox"/> Decimal 987	<input type="checkbox"/> Hexadecimal 3DBh	<input type="checkbox"/> Octal 1733	<input type="checkbox"/> Binary 1111011011
<input type="checkbox"/> Decimal 987d	<input checked="" type="checkbox"/> Hexadecimal 0x3DB	<input type="checkbox"/> Octal 1733o	<input type="checkbox"/> Binary 1111011011b
<input type="checkbox"/> Hexadecimal 3DB	<input type="checkbox"/> Hexadecimal \$3DB	<input type="checkbox"/> Octal 01733	

Limit the integers to these ranges:  Allow leading zeros

128..255

### Allowed Integer Formats

Select the number bases (decimal, hexadecimal, octal and/or binary) that this field can use. For each base, you can choose one or more alternatives with or without various prefixes and/or suffixes.

When the field validation mode is set to “loose”, the field will allow any selected number base with any selected prefix and suffix. In “average” and “strict” modes, the regex will only accept prefix/suffix combinations with numbers in the correct base.

### Limit The Integers to These Ranges

Turn on to allow the field to match only integers in the range or ranges of integers that you specified.

Turn off to allow any integer.

This option has no effect when using the “none” and “loose” field validation modes, which always allow any integer. When using the “average” field validation mode, only the number of digits but not the actual integer values are restricted.

### Allow Leading Zeros

Turn on to allow the field to be preceded by one or more zeros beyond the normally required or allowed number of digits when limiting the values accepted by the field.

This option has no effect in the “none” and “loose” field validation modes, which always allow any integer with any number of digits.

### Integer Ranges

Type in the integers and/or integer ranges that this pattern is allowed to match when limiting the integers. Use semicolons to delimit multiple integers or ranges. Use two dots to specify a range between two integers.

You can specify the integers using any of the integer formats supported by RegexMagic, even those that you aren't allowing for this field. For example: `100b..7;42;0x100..999` matches an integer between 4 and 7, the number 42, or any integer between 256 and 999.

If the ranges you specify include integers like 123 that do not specify whether the notation is hexadecimal, decimal, octal, or binary, then RegexMagic looks at the allowed integer formats. If you allow only one format that doesn't use a prefix or suffix the integer to denote its base, then integers in the ranges that don't denote their base are taken to be in that one format you're allowing. Otherwise, they are taken to be decimal numbers.

Negative numbers are not permitted. If you want to match negative numbers, you can use the "number" field validation pattern instead of the "integer" pattern. Alternatively, you can use a "sequence" field that contains a field using the "literal text" pattern to match the sign followed by the "integer" pattern to match the number.

The ranges are ignored when using the "none" and "loose" field validation modes, which always allow any integer. In "average" mode, only the number of digits is restricted. The field will match any integer with at least as many digits as the minimum value and no more digits than the maximum value. In "strict" mode, the regular expression will limit the field to the exact ranges that you specify, which may result in a very long regular expression.

## 16. Pattern: Date and time

“Date and time” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a date, a time, or any part of or any combination of date and time indicators.

Date separators:	Time separators:	AM/PM indicators:
Slash, hyphen or dot	Colon or dot	AM;PM
Names of months:		
Jan, January; Feb, February; Mar, March; Apr, April; May, May; Jun, June; Jul, July; Aug, August; Sep, September; Oct, October; Nov, November		
Names of days:		
Sun, Sunday; Mon, Monday; Tue, Tuesday; Wed, Wednesday; Thu, Thursday; Fri, Friday; Sat, Saturday;		
<input type="checkbox"/> Limit date values	Beginning date:	Ending date:
	Monday , October 16, 2023	Monday , October 16, 2023
<input type="checkbox"/> Limit time values	Beginning time:	Ending time:
	12:00:00 AM	1:00:00 AM
		Leading zeros:
		Optional leading zeros
Allowed date and time formats:		

### Date Separators

Type in or select one or more characters that can be used to delimit the day, month and year in a numeric date format. In the list of formats, the forward slash / is a placeholder for the date separator characters you specify here. If you specify more than one character, the date separator can be any single one of those.

### Time Separators

Type in or select one or more characters that can be used to delimit the day, month and year in a numeric time format. In the list of formats, the forward slash / is a placeholder for the time separator characters you specify here. If you specify more than one character, the time separator can be any single one of those.

### AM and PM Indicators

Type the indicators for AM and PM, separated by a semicolon. You can provide more than two indicators if the data you are working with uses different variations (such as "AM,a;PM,p"). Delimit the variations of the same indicator with a comma.

In the list of formats, the lower case “a” is a placeholder for the AM and PM indicators you specify here. If you don’t use these indicators, you can leave this field blank.

## Names of Months

Type in the names of the 12 months of the year, delimited by semicolons. You can specify more than one name for each month, such as a full name and an abbreviated name. Delimit the alternatives for each month with commas.

In the list of formats, the upper case “M” is a placeholder for the month names you specify here. If you don’t use the month names, you can leave this field blank.

## Names of Days

Type in the names of the 7 days of the week, delimited by semicolons. You can specify more than one name for each day, such as a full name and an abbreviated name. Delimit the alternatives for each day with commas.

In the list of formats, the upper case “D” is a placeholder for the day names you specify here. If you don’t use the day names, you can leave this field blank.

## Limit Date Values

Turn on to allow the field to match only dates between a certain beginning and ending date.

Turn off to allow any date.

This option has no effect in the “none” and “loose” field validation modes, which always allow any date.

Limiting date values will make your regular expression significantly more complex. You should only use this feature if a single regular expression is all you can use (e.g. in a text editor). If you’ll be using this regular expression in source code, it’s far more efficient to let the regular expression match all dates, and put a bit of extra logic in your source code to process only those dates you’re interested in.

## Beginning Date

Type in the oldest date that the field is allowed to match when limiting date values.

The beginning date is ignored in the “none” and “loose” field validation modes, which always allow any date. In “average” mode, the field will restrict the date’s years only. If the years are the same, the months are restricted too. If the months are the same, the days are restricted. In “strict” mode, the regular expression will limit the field to the exact beginning date, which may require a complex regular expression.

## Ending Date

Type in the newest date that the field is allowed to match when limiting date values.

The ending date is ignored in the “none” and “loose” field validation modes, which always allow any date. In “average” mode, the field will restrict the date’s years only. If the years are the same, the months are restricted

too. If the months are the same, the days are restricted. In “strict” mode, the regular expression will limit the field to the exact ending date, which may require a complex regular expression.

## Limit Time Values

Turn on to allow the field to match only times between a certain beginning and ending time.

Turn off to allow any time.

This option has no effect in the “none” and “loose” field validation modes, which always allow any time.

Limiting time values will make your regular expression significantly more complex. You should only use this feature if a single regular expression is all you can use (e.g. in a text editor). If you’ll be using this regular expression in source code, it’s far more efficient to let the regular expression match all times, and put a bit of extra logic in your source code to process only those times you’re interested in.

## Beginning Time

Type in the oldest time that the field is allowed to match when limiting time values.

The beginning time is ignored in the “none” and “loose” field validation modes, which always allow any date. In “average” mode, the field will restrict the time’s hours only. If the years are the same, the minutes are restricted too. If the months are the same, the seconds are restricted. In “strict” mode, the regular expression will limit the field to the exact beginning time, which may require a complex regular expression.

## Ending Time

Type in the newest time that the field is allowed to match when limiting time values.

The ending time is ignored in the “none” and “loose” field validation modes, which always allow any time. In “average” mode, the field will restrict the time’s hours only. If the years are the same, the minutes are restricted too. If the months are the same, the seconds are restricted. In “strict” mode, the regular expression will limit the field to the exact ending time, which may require a complex regular expression.

## Leading zeros

Choose if day, month, hour, minute and second numbers should be padded to two digits using a leading zero. Years must always be padded to 2 or 4 digits. E.g. 1 January 2001 in m/d/y format can be written as:

- **No leading zeros allowed:** 1/1/01 only
- **Optional leading zeros:** 1/1/01 or 01/01/01
- **Leading zeros required:** 01/01/01 only

## Allowed Date and Time Formats

Specify the date and/or time format patterns that this field can use. If you specify multiple patterns, separate them with line breaks. The following characters have a special meaning in the date and time formats:

- **d**: Day of the month, a number between 1 and 31.
- **D**: Day of the week, one of the names in the “names of days” list.
- **m**: Month number, a number between 1 and 12.
- **M**: Month name, one of the names in the “names of months” list.
- **y**: 2-digit year, a number between 00 and 99.
- **Y**: 4-digit year, a number between 0000 and 9999.
- **/**: Date separator, one of the characters in the “date separators” list.
- **h**: Hour on a 12-hour clock, a number between 1 and 12.
- **H**: Hour on a 24-hour clock, a number between 0 and 23.
- **n**: Minutes, a number between 0 and 59.
- **s**: Seconds, a number between 0 and 59.
- **a**: AM or PM.
- **::**: Time separator, one of the characters in the “time separators” list.

## 17. Pattern: Email address

“Email address” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an email address. You can set the pattern to allow any email address, or only email addresses for specific users or specific domains.

User name:	Domain name:	Mails: prefix:
Allow any user name	Allow any domain name	No prefix

### User Names

The part of the email address before the @ sign is the user name. You can restrict it if you’re only interested in certain email addresses.

- **Allow any user name:** Allow any valid user name.
- **Basic characters only:** Allow only the characters [a-z0-9.\_-] in the user name. Use this if you’ll be passing the email address to software that has trouble with special characters.
- **Specific user names only:** The email address must use one of the user names in the list.

### List of User Names

Type in or paste in the user names this field should allow, one per line.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

### Domain Names

The part of the email address after the @ sign is the domain name. You can restrict it if you’re only interested in certain email addresses.

- **Allow any domain name:** Allow any syntactically valid domain name. No checks are done whether the domain or even top-level domain actually exists. E.g. user@adsf.qwer will be matched, even though there is no .qwer top-level domain.
- **Any domain on specific TLDs:** Allow any domain name on the top-level domains in the list. Top-level domains are generic TLDs like “com”, “net” and “org”, and country-code domains like “us”, “uk”, “au”, etc. When this option is selected, you can right-click on the list of domains to automatically add currently existing top-level domains.
- **Any subdomain on specific domains:** The email address must use one of the domain names in the list. Additional unspecified subdomains are allowed. If you specify "domain.com", then "user@domain.com", "user@mail.domain.com" and "user@pop.domain.com" are all valid.

- **Specific domains only:** The email address must use one of the domain names in the list. No unspecified subdomains are allowed. If you specify "domain.com", then only "user@domain.com" is valid.

### Mailto: Prefix

Choose if email addresses can or must be written as mailto:user@domain.com.

- **No prefix:** Never allow the mailto: prefix.
- **Optional prefix:** Allow email addresses to be specified with or without the prefix.
- **Require prefix:** Require the mailto: prefix to be specified.

### List of Domain Names

Type in or paste in the domain names this field should allow, one per line.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## 18. Pattern: URL

“URL” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match an internet address as you would enter into a web browser’s address bar. You can set the pattern to allow any URL, or to require specific schemes, ports, domains, file paths, parameters, etc.

Schemes: http;https	Port numbers: Optional port number	User names: No user names	Password: No password
Domain name: Allow any domain name	Folders: Safe URL characters only [a	File names: Safe characters only [a-z0-	Parameters: Safe characters only
	Minimum and maximum depth: 0 9	<input checked="" type="checkbox"/> Optional file name	

### List of Schemes

Enter the schemes or protocols the URL can use, delimited by semicolons. http;https;ftp allows the most common protocols. If you don’t specify any schemes, the field will require the URL to have a scheme but will allow any scheme, including a non-existent one.

If you want to allow URLs like www.regexmagic.com that don’t specify a URL scheme then specify the subdomain those URLs should begin with followed by a dot as the scheme. http;https;ftp;www.;ftp. allows URLs with the http, https and ftp protocols on any domain, as well as URLs without a protocol on www. and ftp. subdomains.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

### Port numbers

A URL can include a port number to connect to a different port than the one normally used by the protocol, e.g. http://www.domain.com:6969

- **No port number:** Only match URLs without port numbers.
- **Optional port number:** Allow any of the port numbers in the list, as well as URLs without port numbers.
- **Require port number:** Require the URL to include one of the specified port numbers.

### List of Port Numbers

A URL can include a port number to connect to a different port than the one normally used by the protocol, e.g. http://www.domain.com:6969

Type in or paste in the port numbers the URL can use, delimited by semicolons. You can use a hyphen to specify a range of ports. E.g.: 80;1024-65535 allows ports 80 and 1024 through 65535.

If you do not specify any port numbers, all ports from 1 to 65535 will be allowed.

The list of ports is ignored if the field validation mode is set to “none” or “loose”. To make the regex actually check the port numbers, set the field validation mode to “average” or “strict”.

## User Names

URLs can include a user name. It will appear before the domain, delimited by an @ sign. E.g.: `http://user@www.domain.com`. You can choose if this field should match URLs that include user names or not, or search for specific user names.

- **No user names:** Do not allow any user name.
- **Allow any user name:** Allow any valid user name.
- **Safe characters only:** Allow only the characters `[a-z0-9$_+!*(),-]` in the user name. These characters have no special meaning in URLs.
- **Basic characters only:** Allow only the characters `[a-z0-9._-]` in the user name.
- **Specific user names only:** The URL address must use one of the user names in the list.

## List of User Names

Type in or paste in the user names this field should allow, delimited by semicolons.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## Password

Choose if the URL can have an embedded password, as in `http://user:password@www.domain.com`. URLs can only contain a password if they also contain a user name.

- **No password:** Never allow embedded passwords.
- **Optional password:** Allow URL addresses to be specified with or without the password.
- **Require password:** Require the URL to include a password.

This syntax is not commonly used any more. The embedded passwords are plainly visible. Leaving passwords visible is usually not a good idea. RegexMagic still supports it. You may want to search for URLs with embedded passwords and remove the passwords from them.

## Domain Names

In a URL like `http://www.domain.com`, “`www.domain.com`” is the full domain name. You can restrict it if you’re only interested in certain URLs.

- **Allow any domain name:** Allow any syntactically valid domain name. No checks are done whether the domain or even top-level domain actually exists. E.g. “`adsf.qwer`” will be matched, even though there is no `.qwer` top-level domain.

- **Any domain on specific TLDs:** Allow any domain name on the top-level domains in the list. Top-level domains are generic TLDs like “com”, “net” and “org”, and country-code domains like “us”, “uk”, “au”, etc. When this option is selected, you can right-click on the list of domains to automatically add currently existing top-level domains.
- **Any subdomain on specific domains:** The URL must use one of the domain names in the list. Additional unspecified subdomains are allowed. If you specify "domain.com", then "www.domain.com", "ftp.domain.com“ and ”mail.domain.com" are all valid.
- **Specific domains only:** The URL must use one of the domain names in the list. No unspecified subdomains are allowed. If you specify only "domain.com", then "domain.com“ is accepted but ”www.domain.com" is not.

## List of Domain Names

Type in or paste in the domain names this field should allow, one per line.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## Folders

The file referenced by a URL can be in a folder or directory structure, e.g.:

<http://www.domain.com/folder/subfolder/file.html>. You can determine if the URL can include folders, and which ones.

- **No folders:** Do not allow any folders.
- **Allow any path:** Allow all valid folder names.
- **Safe URL characters only:** Allow only the characters [a-z0-9\$\_+!\*'(),-] in the path. These characters have no special meaning in URLs.
- **Safe folder characters only:** Allow only the characters [a-z0-9\$\_+'()(),-] in the path. These characters have no special meaning in URLs or folder names and exclude the dot which is typically used in file names.
- **Basic characters only:** Allow only the characters [a-z0-9.\_-] in the path.
- **Basic folder characters only:** Allow only the characters [a-z0-9.\_-] in the path. This excludes the dot which is typically used in file names.
- **Specific folders only:** Only folders in the list below can be used. They can be combined into paths in any way.
- **Specific paths only:** Only paths in the list below can be used.

## List of Folders or Paths

Type in or paste in the folders or paths allowed in the URL, one per line. If you set the folders choice to “specific paths only”, type in path names like “onefolder”, "folder/subfolder“ and ”folder/subfolder/3rdlevel". Only the exact paths you specify will be allowed.

Otherwise, type in individual folder names like “folder1”, “folder2” and “folder3”. The regular expression will allow any combination of these folders, like folder2/folder3/folder1/folder2.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## Minimum Folder Depth

The minimum number of folders that the URL should specify. Set to zero to allow URLs without a folder. Set to greater than zero to require a certain folder depth.

For example `http://www.domain.com/folder1/subfolder/level3/filename.html` has a folder depth of 3. However, if you set Folders to “allow any path”, “safe URL characters only”, or “basic characters only” (all of which allow dots in folder names) and you set File names to “no file name” then the generated regex will match this URL as if it had a folder depth of 4 with “filename.html” being the 4th folder. Likewise, if you set Folders to “allow any path” and Parameters to “no parameters” then the list of parameters will be included with the final folder.

## Maximum Folder Depth

The maximum number of folders that the URL should specify. This number must be equal to or greater than the minimum number of folders and must be greater than zero. Set Folders to “no folders” if you don’t want to allow any folders at all.

For example `http://www.domain.com/folder1/subfolder/level3/filename.html` has a folder depth of 3. However, if you set Folders to “allow any path”, “safe URL characters only”, or “basic characters only” (all of which allow dots in folder names) and you set File names to “no file name” then the generated regex will match this URL as if it had a folder depth of 4 with “filename.html” being the 4th folder. Likewise, if you set Folders to “allow any path” and Parameters to “no parameters” then the list of parameters will be included with the final folder.

## File names

Choose if the URL can or should include a file name, such as `http://www.domain.com/filename.ext`.

- **No file names:** Do not allow any file names.
- **Allow any file name:** Allow any valid file name. This choice generates a regex that matches URLs with parameters and treats the parameters as part of the file name if Parameters is set to “no parameters”.
- **Safe characters only:** Allow only the characters `[a-z0-9$_+!*'(),-]` in the file name. These characters have no special meaning in URLs.
- **Basic characters only:** Allow only the characters `[a-z0-9._-]` in the file name.
- **Specific extensions only:** Only allow file names with specific extensions from the list below.
- **Specific file names only:** Only file names in the list below can be used.

## List of File Names or Extensions

Type in or paste in the file names the URL can reference, one per line. If you selected “specific extensions only”, enter the extensions, e.g. “html” and “txt”.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## Optional file name

If you’ve specified that the URL should include a file name, you can choose whether the filename is optional or mandatory.

## Parameters

A URL can include a list of parameters at the end, e.g.:

`http://www.domain.com/action.php?param1=value1&param2=value2`. You can restrict this field to disallow parameters or to require certain parameters.

- **No parameters:** Do not allow any parameters.
- **Allow all parameters:** Allow anything to occur after the ? in the URL.
- **Safe characters only:** Allow only the characters `[a-z0-9$.+!*'(),-]` in the parameters and their values. These characters have no special meaning in URLs.
- **Basic characters only:** Allow only the characters `[a-z0-9._-]` in the parameters and their values.
- **Specific parameters only:** Only parameters in the list below can be used. Does not require all parameters to be present.

## List of Parameters

Type in or paste in the parameters that can occur in the URL, one per line.

This box is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Enter to insert a line break.

If the field validation mode is set to “none” or “loose”, the list only determines which characters are allowed. To make the regex actually check for the whole list, set the field validation mode to “average” or “strict”.

## 19. Pattern: Country

“Country” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a country code or country name as defined in the ISO 3166 standard. You can restrict the field to the codes or names of specific countries.

<input checked="" type="checkbox"/> Two-letter codes	<input type="checkbox"/> English names	<input type="checkbox"/> French names	<input type="checkbox"/> Make suffixes optional
--	--	---------------------------------------	---

<p>Country groups:</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> African Union</li> <li><input type="checkbox"/> Arab League</li> <li><input type="checkbox"/> ASEAN</li> <li><input type="checkbox"/> CIS</li> <li><input checked="" type="checkbox"/> European Union</li> <li><input checked="" type="checkbox"/> NATO</li> <li><input checked="" type="checkbox"/> OECD</li> <li><input type="checkbox"/> Org of American States</li> <li><input type="checkbox"/> SAARC</li> <li><input type="checkbox"/> Others</li> </ul>	<p>Countries:</p> <table border="0"> <tr> <td><input type="checkbox"/> AFGHANISTAN</td> <td><input type="checkbox"/> AZERBAIJAN</td> <td><input type="checkbox"/> BOUVET ISLAND</td> </tr> <tr> <td><input type="checkbox"/> ÅLAND ISLANDS</td> <td><input type="checkbox"/> BAHAMAS</td> <td><input type="checkbox"/> BRAZIL</td> </tr> <tr> <td><input type="checkbox"/> ALBANIA</td> <td><input type="checkbox"/> BAHRAIN</td> <td><input type="checkbox"/> BRITISH INDIAN OCEAN TERF</td> </tr> <tr> <td><input type="checkbox"/> ALGERIA</td> <td><input type="checkbox"/> BANGLADESH</td> <td><input type="checkbox"/> BRUNEI DARUSSALAM</td> </tr> <tr> <td><input type="checkbox"/> AMERICAN SAMOA</td> <td><input type="checkbox"/> BARBADOS</td> <td><input checked="" type="checkbox"/> BULGARIA</td> </tr> <tr> <td><input type="checkbox"/> ANDORRA</td> <td><input type="checkbox"/> BELARUS</td> <td><input type="checkbox"/> BURKINA FASO</td> </tr> <tr> <td><input type="checkbox"/> ANGOLA</td> <td><input checked="" type="checkbox"/> BELGIUM</td> <td><input type="checkbox"/> BURUNDI</td> </tr> <tr> <td><input type="checkbox"/> ANGUILLA</td> <td><input type="checkbox"/> BELIZE</td> <td><input type="checkbox"/> CAMBODIA</td> </tr> <tr> <td><input type="checkbox"/> ANTARCTICA</td> <td><input type="checkbox"/> BENIN</td> <td><input type="checkbox"/> CAMEROON</td> </tr> <tr> <td><input type="checkbox"/> ANTIGUA AND BARBUDA</td> <td><input type="checkbox"/> BERMUDA</td> <td><input type="checkbox"/> CANADA</td> </tr> <tr> <td><input type="checkbox"/> ARGENTINA</td> <td><input type="checkbox"/> BHUTAN</td> <td><input type="checkbox"/> CAPE VERDE</td> </tr> <tr> <td><input type="checkbox"/> ARMENIA</td> <td><input type="checkbox"/> BOLIVIA, PLURINATIONAL ST</td> <td><input type="checkbox"/> CAYMAN ISLANDS</td> </tr> <tr> <td><input type="checkbox"/> ARUBA</td> <td><input type="checkbox"/> BONAIRE, SINT EUSTATIUS A</td> <td><input type="checkbox"/> CENTRAL AFRICAN REPUBLIC</td> </tr> <tr> <td><input type="checkbox"/> AUSTRALIA</td> <td><input type="checkbox"/> BOSNIA AND HERZEGOVINA</td> <td><input type="checkbox"/> CHAD</td> </tr> <tr> <td><input checked="" type="checkbox"/> AUSTRIA</td> <td><input type="checkbox"/> BOTSWANA</td> <td><input type="checkbox"/> CHILE</td> </tr> </table>	<input type="checkbox"/> AFGHANISTAN	<input type="checkbox"/> AZERBAIJAN	<input type="checkbox"/> BOUVET ISLAND	<input type="checkbox"/> ÅLAND ISLANDS	<input type="checkbox"/> BAHAMAS	<input type="checkbox"/> BRAZIL	<input type="checkbox"/> ALBANIA	<input type="checkbox"/> BAHRAIN	<input type="checkbox"/> BRITISH INDIAN OCEAN TERF	<input type="checkbox"/> ALGERIA	<input type="checkbox"/> BANGLADESH	<input type="checkbox"/> BRUNEI DARUSSALAM	<input type="checkbox"/> AMERICAN SAMOA	<input type="checkbox"/> BARBADOS	<input checked="" type="checkbox"/> BULGARIA	<input type="checkbox"/> ANDORRA	<input type="checkbox"/> BELARUS	<input type="checkbox"/> BURKINA FASO	<input type="checkbox"/> ANGOLA	<input checked="" type="checkbox"/> BELGIUM	<input type="checkbox"/> BURUNDI	<input type="checkbox"/> ANGUILLA	<input type="checkbox"/> BELIZE	<input type="checkbox"/> CAMBODIA	<input type="checkbox"/> ANTARCTICA	<input type="checkbox"/> BENIN	<input type="checkbox"/> CAMEROON	<input type="checkbox"/> ANTIGUA AND BARBUDA	<input type="checkbox"/> BERMUDA	<input type="checkbox"/> CANADA	<input type="checkbox"/> ARGENTINA	<input type="checkbox"/> BHUTAN	<input type="checkbox"/> CAPE VERDE	<input type="checkbox"/> ARMENIA	<input type="checkbox"/> BOLIVIA, PLURINATIONAL ST	<input type="checkbox"/> CAYMAN ISLANDS	<input type="checkbox"/> ARUBA	<input type="checkbox"/> BONAIRE, SINT EUSTATIUS A	<input type="checkbox"/> CENTRAL AFRICAN REPUBLIC	<input type="checkbox"/> AUSTRALIA	<input type="checkbox"/> BOSNIA AND HERZEGOVINA	<input type="checkbox"/> CHAD	<input checked="" type="checkbox"/> AUSTRIA	<input type="checkbox"/> BOTSWANA	<input type="checkbox"/> CHILE
<input type="checkbox"/> AFGHANISTAN	<input type="checkbox"/> AZERBAIJAN	<input type="checkbox"/> BOUVET ISLAND																																												
<input type="checkbox"/> ÅLAND ISLANDS	<input type="checkbox"/> BAHAMAS	<input type="checkbox"/> BRAZIL																																												
<input type="checkbox"/> ALBANIA	<input type="checkbox"/> BAHRAIN	<input type="checkbox"/> BRITISH INDIAN OCEAN TERF																																												
<input type="checkbox"/> ALGERIA	<input type="checkbox"/> BANGLADESH	<input type="checkbox"/> BRUNEI DARUSSALAM																																												
<input type="checkbox"/> AMERICAN SAMOA	<input type="checkbox"/> BARBADOS	<input checked="" type="checkbox"/> BULGARIA																																												
<input type="checkbox"/> ANDORRA	<input type="checkbox"/> BELARUS	<input type="checkbox"/> BURKINA FASO																																												
<input type="checkbox"/> ANGOLA	<input checked="" type="checkbox"/> BELGIUM	<input type="checkbox"/> BURUNDI																																												
<input type="checkbox"/> ANGUILLA	<input type="checkbox"/> BELIZE	<input type="checkbox"/> CAMBODIA																																												
<input type="checkbox"/> ANTARCTICA	<input type="checkbox"/> BENIN	<input type="checkbox"/> CAMEROON																																												
<input type="checkbox"/> ANTIGUA AND BARBUDA	<input type="checkbox"/> BERMUDA	<input type="checkbox"/> CANADA																																												
<input type="checkbox"/> ARGENTINA	<input type="checkbox"/> BHUTAN	<input type="checkbox"/> CAPE VERDE																																												
<input type="checkbox"/> ARMENIA	<input type="checkbox"/> BOLIVIA, PLURINATIONAL ST	<input type="checkbox"/> CAYMAN ISLANDS																																												
<input type="checkbox"/> ARUBA	<input type="checkbox"/> BONAIRE, SINT EUSTATIUS A	<input type="checkbox"/> CENTRAL AFRICAN REPUBLIC																																												
<input type="checkbox"/> AUSTRALIA	<input type="checkbox"/> BOSNIA AND HERZEGOVINA	<input type="checkbox"/> CHAD																																												
<input checked="" type="checkbox"/> AUSTRIA	<input type="checkbox"/> BOTSWANA	<input type="checkbox"/> CHILE																																												

### Two-Letter Codes

Make the field match two-letter country codes as defined by ISO 3166.

### English Names

Make the field match English country names as defined by ISO 3166.

### French Names

Make the field match French country names as defined by ISO 3166.

### Make Suffixes Optional

Many of the country names in the ISO 3166 standard have suffixes, such as "IRAN, ISLAMIC REPUBLIC OF".

If you turn on this option, the field matches the country name with or without the suffix. E.g. both "IRAN" and "IRAN, ISLAMIC REPUBLIC OF" are accepted.

If you turn off this option, the field matches the country names only as listed in ISO 3166. E.g. only "IRAN, ISLAMIC REPUBLIC OF" is accepted.

## **Country Groups**

Use this list to quickly select or deselect certain groups of countries. The checkboxes in this list are automatically updated when you select or deselect individual countries.

## **Countries**

Select the countries of which this field should match the codes and/or names. Selecting none of the countries is the same as selecting all the countries.

## 20. Pattern: State or Province

“State or province” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match the name or code of a state or province from a list of specific states or all states from specific countries.

Two-letter codes       Full names

Countries:      States and Provinces:

<input type="checkbox"/> Canadian Provinces	<input checked="" type="checkbox"/> Alabama	<input checked="" type="checkbox"/> Iowa	<input checked="" type="checkbox"/> New Jersey
<input checked="" type="checkbox"/> US States	<input checked="" type="checkbox"/> Alaska	<input checked="" type="checkbox"/> Kansas	<input checked="" type="checkbox"/> New Mexico
	<input checked="" type="checkbox"/> Arizona	<input checked="" type="checkbox"/> Kentucky	<input checked="" type="checkbox"/> New York
	<input checked="" type="checkbox"/> Arkansas	<input checked="" type="checkbox"/> Louisiana	<input checked="" type="checkbox"/> North Carolina
	<input checked="" type="checkbox"/> California	<input checked="" type="checkbox"/> Maine	<input checked="" type="checkbox"/> North Dakota
	<input checked="" type="checkbox"/> Colorado	<input checked="" type="checkbox"/> Maryland	<input checked="" type="checkbox"/> Ohio
	<input checked="" type="checkbox"/> Connecticut	<input checked="" type="checkbox"/> Massachusetts	<input checked="" type="checkbox"/> Oklahoma
	<input checked="" type="checkbox"/> Delaware	<input checked="" type="checkbox"/> Michigan	<input checked="" type="checkbox"/> Oregon
	<input checked="" type="checkbox"/> District of Columbia	<input checked="" type="checkbox"/> Minnesota	<input checked="" type="checkbox"/> Pennsylvania
	<input checked="" type="checkbox"/> Florida	<input checked="" type="checkbox"/> Mississippi	<input checked="" type="checkbox"/> Rhode Island
	<input checked="" type="checkbox"/> Georgia	<input checked="" type="checkbox"/> Missouri	<input checked="" type="checkbox"/> South Carolina
	<input checked="" type="checkbox"/> Hawaii	<input checked="" type="checkbox"/> Montana	<input checked="" type="checkbox"/> South Dakota
	<input checked="" type="checkbox"/> Idaho	<input checked="" type="checkbox"/> Nebraska	<input checked="" type="checkbox"/> Tennessee
	<input checked="" type="checkbox"/> Illinois	<input checked="" type="checkbox"/> Nevada	<input checked="" type="checkbox"/> Texas
	<input checked="" type="checkbox"/> Indiana	<input checked="" type="checkbox"/> New Hampshire	<input checked="" type="checkbox"/> Utah

<  >

### Two-Letter Codes

Allow the field to match the two-letter codes of the selected states and provinces.

### Full Names

Allow the field to match the full names of the selected states and provinces.

### Countries

Select the countries of which you want to match the states or provinces. You have to select at least one country.

### States and Provinces

Select the states and provinces of which this field should match the codes and/or names. Selecting none of the states and provinces is the same as selecting all of them.

This list only shows the states and provinces of the countries you have selected in the list of countries. The field never matches states or provinces of countries that you didn't select.

## 21. Pattern: Currency

“Currency” is one of the patterns that you can select on the Match panel. Use this pattern to make a field match a currency code as defined in the ISO 4217 standard. You can restrict the field to specific currencies.

Show special currency codes       Show obsolete currencies

Currencies:

<input type="checkbox"/> AED United Arab Emirates dirh	<input type="checkbox"/> BIF Burundian franc	<input type="checkbox"/> COP Colombian peso	<input type="checkbox"/> FJD Fiji dollar
<input type="checkbox"/> AFN Afghan afghani	<input type="checkbox"/> BMD Bermudian dollar	<input type="checkbox"/> COU Unidad de Valor Real	<input type="checkbox"/> FKP Falkland Islands pound
<input type="checkbox"/> ALL Albanian lek	<input type="checkbox"/> BND Brunei dollar	<input type="checkbox"/> CRC Costa Rican colon	<input checked="" type="checkbox"/> GBP Pound sterling
<input type="checkbox"/> AMD Armenian dram	<input type="checkbox"/> BOB Boliviano	<input type="checkbox"/> CUC Cuban convertible peso	<input type="checkbox"/> GEL Georgian lari
<input type="checkbox"/> ANG Netherlands Antillean guilder	<input type="checkbox"/> BRL Brazilian real	<input type="checkbox"/> CUP Cuban peso	<input type="checkbox"/> GHS Ghanaian cedi
<input type="checkbox"/> AOA Angolan kwanza	<input type="checkbox"/> BSD Bahamian dollar	<input type="checkbox"/> CVE Cape Verde escudo	<input type="checkbox"/> GIP Gibraltar pound
<input type="checkbox"/> ARS Argentine peso	<input type="checkbox"/> BTN Bhutanese ngultrum	<input type="checkbox"/> CZK Czech koruna	<input type="checkbox"/> GMD Gambian dalasi
<input type="checkbox"/> AUD Australian dollar	<input type="checkbox"/> BWP Botswana pula	<input type="checkbox"/> DJF Djiboutian franc	<input type="checkbox"/> GNF Guinean franc
<input type="checkbox"/> AWG Aruban florin	<input type="checkbox"/> BYR Belarusian ruble	<input type="checkbox"/> DKK Danish krone	<input type="checkbox"/> GTQ Guatemalan quetzal
<input type="checkbox"/> AZN Azerbaijani manat	<input type="checkbox"/> BZD Belize dollar	<input type="checkbox"/> DOP Dominican peso	<input type="checkbox"/> GYD Guyanese dollar
<input type="checkbox"/> BAM Bosnia and Herzegovina convertible mark	<input type="checkbox"/> CAD Canadian dollar	<input type="checkbox"/> DZD Algerian dinar	<input type="checkbox"/> HKD Hong Kong dollar
<input type="checkbox"/> BBD Barbados dollar	<input type="checkbox"/> CDF Congolese franc	<input type="checkbox"/> EGP Egyptian pound	<input type="checkbox"/> HNL Honduran lempira
<input type="checkbox"/> BDT Bangladeshi taka	<input type="checkbox"/> CHF Swiss franc	<input type="checkbox"/> ERN Eritrean nakfa	<input type="checkbox"/> HRK Croatian kuna
<input type="checkbox"/> BGN Bulgarian lev	<input type="checkbox"/> CLP Chilean peso	<input type="checkbox"/> ETB Ethiopian birr	<input type="checkbox"/> HTG Haitian gourde
<input type="checkbox"/> BHD Bahraini dinar	<input type="checkbox"/> CNY Renminbi	<input checked="" type="checkbox"/> EUR Euro	<input type="checkbox"/> HUF Hungarian forint

### Show Special Currency Codes

Show currency codes that represent bond market units, special settlement units, and precious metals in the list of currencies.

### Show Obsolete Currency Codes

Show currency codes that are no longer used in active trade today. These codes are still valid for referencing obsolete currencies in historic data.

### Currencies

Select the currencies codes that this field should match. Selecting none of the currencies is the same as selecting all the currencies. The currency codes are defined by ISO 4217.

The currencies are listed in alphabetic order of the currency codes. When showing special and/or obsolete currencies, the normal currencies are listed first, then the special currencies, and finally the obsolete currencies.

Currency codes are only matched exactly when using the “average” or “strict” field validation modes. Invalid currencies may be matched in “loose” mode. All 3-letter combinations are accepted in “none” mode.

## 22. Pattern: Credit card number

“Credit card number” is one of the patterns that you can select on the Match panel. Use this pattern to match a series of digits that looks like a credit card number from one or more of the world’s major credit card issuers.

- |  |   |   |
|--|---|---|
| <input checked="" type="checkbox"/> Visa             | <input checked="" type="checkbox"/> Diners Club | <input checked="" type="radio"/> No spaces or dashes    |
| <input checked="" type="checkbox"/> MasterCard       | <input checked="" type="checkbox"/> Discover    | <input type="radio"/> Spaces and dashes to group digits |
| <input checked="" type="checkbox"/> American Express | <input checked="" type="checkbox"/> JCB         | <input type="radio"/> Spaces and dashes anywhere        |

### Visa

Turn on to accept Visa credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### MasterCard

Turn on to accept MasterCard credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### American Express

Turn on to accept American Express credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### Diners Club

Turn on to accept Diners Club credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### Discover

Turn on to accept Discover credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### JCB

Turn on to accept JCB credit card numbers.

Brand-specific number validation is only done when the field validation mode is set to “average” or “strict”.

### **No Spaces or Dashes**

Do not allow any spaces or dashes in the credit card number.

If you’re going to use your regular expression to validate user input on an order form, you should first use a regex like `\D+` to strip out all non-digits entered by the user. Then validate the remaining digits with the credit card number regex generated with the option “no spaces or dashes” selected.

### **Spaces and Dashes to Group Digits**

Allow spaces and dashes to group the digits in the credit card number, like it appears embossed on an actual credit card.

This option only works when the field validation mode is set to “average” or “strict”. If you set it to “none” or “loose”, selecting to allow spaces and dashes for grouping will allow them anywhere.

### **Spaces and Dashes Anywhere**

Allow any amount of spaces and dashes anywhere in the credit card number.

Use this option to scan for credit card numbers in documents, if you don’t want to miss any numbers that have been obfuscated with spaces or dashes. It may yield false positives in documents with lists of numbers.

## 23. Pattern: National ID

“National ID” is one of the patterns that you can select on the Match panel. Use it to make a field match a personal identification code used in a specific country such as social security numbers, identity numbers, citizen numbers, tax numbers, license plate numbers, etc.

Kind of ID:

US social security number 

### Kind of ID

Select which kind of national ID you want to match.

## 24. Pattern: VAT number

“VAT number” is one of the patterns that you can select on the Match panel. Use it to make a field match a value added tax number from one or more European countries.

Countries:

<input type="checkbox"/> AT Austria	<input type="checkbox"/> DK Denmark	<input type="checkbox"/> GB United Kingdom	<input checked="" type="checkbox"/> LU Luxembourg	<input type="checkbox"/> RO Romania
<input checked="" type="checkbox"/> BE Belgium	<input type="checkbox"/> EE Estonia	<input type="checkbox"/> HR Croatia	<input type="checkbox"/> LV Latvia	<input type="checkbox"/> SE Sweden
<input type="checkbox"/> BG Bulgaria	<input type="checkbox"/> EL Greece	<input type="checkbox"/> HU Hungary	<input type="checkbox"/> MT Malta	<input type="checkbox"/> SI Slovenia
<input type="checkbox"/> CY Cyprus	<input type="checkbox"/> ES Spain	<input type="checkbox"/> IE Ireland	<input checked="" type="checkbox"/> NL Netherlands	<input type="checkbox"/> SK Slovakia
<input type="checkbox"/> CZ Czech Republic	<input type="checkbox"/> FI Finland	<input type="checkbox"/> IT Italy	<input type="checkbox"/> PL Poland	<input type="checkbox"/> XI Northern Ireland
<input type="checkbox"/> DE Germany	<input type="checkbox"/> FR France	<input type="checkbox"/> LT Lithuania	<input type="checkbox"/> PT Portugal	

Country code:       Grouping characters:

### Allowed Countries

Each country uses a different format for its VAT numbers. Please select the countries you want to accept VAT numbers from.

### Country Code

VAT numbers normally begin with a two-letter country code. However, people sometimes omit the country code if the country is already specified elsewhere.

- **No country code:** Do not allow the VAT number to include the country code
- **Optional country code:** Match VAT numbers with and without country codes.
- **Require country code:** Require the VAT number to begin with a country code.

### Grouping Characters

People often split VAT numbers into groups of 3 or 4 digits to make them easier to read. E.g. BE 0123.456.789 uses a space to separate the country code from the number, and dots to group the digits.

Type all the extra characters like spaces and dots that you want to allow in the VAT number. If you don't type in any characters, or select “None”, only unformatted VAT numbers like BE0123456789 will be accepted.

## 25. Pattern: IPv4 Address

“IPv4 address” is one of the patterns that you can select on the Match panel. Use it to make a field match an internet address such as 127.0.0.1.

Allowed IPv4 notations:

- Dotted 192.168.0.1
- Decimal 3232235521
- Hexadecimal C0A80001
- Hexadecimal C0A80001h
- Hexadecimal 0xC0A80001

Subnet mask:

0 0

Limit the IPv4 addresses to these ranges  Validate 0..255 in dotted addresses

63.212.171.1..63.212.171.254

### Allowed IPv4 Notations

Select the IPv4 address notations that this field can match. You can choose the common dotted decimal notation, as well as decimal and hexadecimal integer notations.

When the field validation mode is set to “loose”, if you select the hexadecimal notation with the h suffix or the 0x prefix and also select the dotted and/or decimal notations, then the field will allow the prefix even when dotted or decimal notation is used. In “average” and “strict” modes, the regex will only allow the prefix or suffix when the address is matched in hexadecimal notation.

### Minimum Subnet Mask

If the matched IPv4 addresses must have a subnet mask (the 24 in 192.168.0.0/24), specify the smallest subnet mask you want to allow.

Set the minimum subnet mask to zero to make the subnet mask optional.

### Maximum Subnet Mask

If the matched IPv4 addresses must have a subnet mask (the 24 in 192.168.0.0/24), specify the largest subnet mask you want to allow.

Set the maximum subnet mask to zero if you don’t want the field to match subnet masks.

### Limit The IPv4 Addresses to These Ranges

Turn on to allow the field to match only IPv4 addresses in the range or ranges of addresses that you specified.

Turn off to allow any integer.

This option has no effect when using the “none” and “loose” field validation modes, which always allow any integer. When using the “average” field validation mode, only the number of digits but not the actual integer values are restricted.

### **Validate 0..255 in dotted addresses**

Turn on if the text may contain dotted numbers like 999.999.999.999 that aren't IPv4 addresses and you want to make sure your regular expression does not match these.

Turn on if all dotted numbers in the text you will be using this regex on are valid IPv4 addresses, meaning there is no need to make the regex exclude things like 999.999.999.999 because they don't occur in the text anyway.

This option only makes a difference when the field validation mode is set to “average” or “strict”. Only these modes generate regexes that restrict the 4 numbers in dotted IPv4 notation. If the regex needs to match the full range 0..255 for some or all of the 4 numbers in dotted notation, then turning on this option will generate a long regex that matches the range 0..255 exactly. Turning off this option will generate a short regex that allows any number between 0 and 999.

### **IPv4 Address Ranges**

Type in the IPv4 addresses and/or address ranges that this pattern is allowed to match when limiting the addresses. Use semicolons to delimit multiple integers or ranges. Use two dots to specify a range between two integers. You can specify the addresses using any of the IPv4 address notations supported by RegexMagic, even those that you aren't allowing for this field. You can also use the subnet mask notation to specify a range of addresses. For example: 192.168.0.0/24;134744072;10.0.0.3..10.0.0.7 matches an address between 192.168.0.0 and 192.168.0.255, the address 8.8.8.8, or an address between 10.0.0.3 and 10.0.0.7.

A decimal number like 134744072 without a prefix or suffix is interpreted as a hexadecimal number if you selected to allow the hexadecimal notation without a prefix or suffix and you disallowed the decimal notation. If you selected any other combination of notations, the decimal number is interpreted as a decimal number.

The ranges are ignored when using the “none” and “loose” field validation modes, which always allow any IPv4 address. In “average” mode, the dotted notation restricts each of the 4 numbers separately. This may allow the field to match more IP addresses than you specified. E.g. if you restricted the field to match 8.8.8.8 and 8.8.4.4, in “average” mode it will also match 8.8.8.4 and 8.8.4.8. For the decimal and hexadecimal notations, “average” mode only restricts the number of digits. The field will match any integer with at least as many digits as the minimum value and no more digits than the maximum value. In “strict” mode, the regular expression will limit the field to the exact ranges that you specify in all the notations that you allow, which may result in a very long regular expression.

## 26. Pattern: GUID

“GUID” is one of the patterns that you can select on the Match panel. Use it to make a field match a globally unique identifier such as {7E9081B5-9A6D-4CC1-A8C3-47F69FB4198D}.

This pattern can only be used to match arbitrary GUIDs. If you want to match specific GUIDs, use the “literal text” pattern to match a single GUID, or the “list of literal text” pattern to match one from a list of GUIDs.

Braces around the GUID: Required	Hyphens in the GUID: Required	Case: Case insensitive
-------------------------------------	----------------------------------	---------------------------

### Braces Around The GUID

A GUID is a 128-bit number usually written as {A4F90AD5-DC63-4A1D-A5EC-F44E5B15C5D9} with braces and hyphens. You can choose whether your regular expression requires or allows the braces.

- **Required:** Only match the GUID if surrounded by braces.
- **Optional:** Match the GUID with braces if they are present while allowing the GUID to be matched without braces if they are not present.
- **Not allowed:** Match the GUID without the braces. If braces are present, the pattern may still match a GUID between braces without matching the braces if the patterns before and after this pattern can match the braces.

### Hyphens in The GUID

A GUID is a 128-bit number usually written as {A4F90AD5-DC63-4A1D-A5EC-F44E5B15C5D9} with braces and hyphens. You can choose whether your regular expression requires or allows the hyphens. If the field validation mode is “strict” or “average”, the hyphens will only be matched at their correct positions. In “loose” or “none” mode, the pattern will allow any combination of hexadecimal digits with or without hyphens.

- **Required:** Only match the GUID if all 4 hyphens are present.
- **Optional:** Match the GUID with or without hyphens. If the field validation mode is set to “strict”, all 4 hyphens must be either present or omitted. In “average” mode, any of the 4 hyphens may be omitted separately.
- **Not allowed:** Do not allow any hyphens in the GUID.

### Case

Choose whether the hexadecimal digits A to F in the GUID should be uppercase or lowercase.

- **Case insensitive:** Match all GUIDs regardless of case.
- **Uppercase:** Only match GUIDs with uppercase hexadecimal digits.
- **Lowercase:** Only match GUIDs with lowercase hexadecimal digits.

## 27. Pattern: Regular expression

“Regular expression” is one of the patterns that you can select on the Match panel. With this pattern you can insert actual regular expression code into your regular expression.

Regular expression flavor:  Case insensitive  ^\$ match at line breaks  
 GNU BRE  Dot matches line breaks  Free-spacing RegexBuddy

```

\\(\\(I|you|we|they)\\s\\+love\\s\\+he\\s\\+loves)\\s*reg\\(ular\\
expressions|?\\exp\\(es|?\\)
  
```

### Regular Expression Flavor

Select the regular expression flavor this regular expression was created for.

If this flavor is different than the flavor for the final regular expression, RegexMagic will automatically convert the regular expression below before inserting it into the final regular expression.

### Dot Matches Line Breaks

Turn on to make the dot match absolutely any character, including line breaks.  
 Turn off to make the dot match any character except line break characters.

### Case Insensitive

Turn on to ignore differences between lowercase and uppercase letters in the regex and subject string. The regex “word” will match “Word”, “woRD”, etc.  
 Turn off to treat uppercase and lowercase letters as different letters. The regex “word” will match “word” but not “Word”.

### ^\$ Match at Line Breaks

Turn on to make the ^ and \$ match after and before line breaks embedded in the subject string, as well as at the start and end of the subject. In other words: make ^ and \$ match the start and end of a line. Turn off to allow the ^ and \$ to match at the start and end of the subject string only.

### Free-Spacing

Turn on to ignore whitespace and comments in the regular expression below.  
 Turn off to treat whitespace as significant, and the # character as an ordinary character.

## **RegexBuddy**

Invoke RegexBuddy to edit this regular expression.

## **List of Regex**

Type in or paste in the strings this field should allow.

This control is a full-featured text editor. You can cut, copy and paste with Ctrl+X, Ctrl+C and Ctrl+V as usual. Press Ctrl+Enter to insert a page break when delimiting the list of Regex with page breaks. Press Enter to insert a line break.

## 28. Action Panel Reference

The Action panel is the place where you tell RegexMagic which action you plan to take with your regular expression. You can collect parts of the text matched by the regular expression, or replace them with different parts or with new text. The settings on the Action panel determine the replacement text that is generated, if any. Together, the Samples, Match, and Action panels define a RegexMagic formula to generate a regular expression, and possibly a replacement text.

In the default layout, the Action panel shares the top left tab with the Samples and Match panels. Depending on which panel you last used, that tab will say “Samples, Match” or “Samples, Action”. Click that tab, and then on the Action tab inside it, to activate the Action panel in the default layout. In the side by side layout, the Match and Action panels are combined into one tab labeled “Match, Action”. Clicking that tab shows both panels. In the dual monitor tabbed layout, the Action panel has its own tab in RegexMagic’s main window. In the dual monitor side by side layout, the Action panel is always visible in the main window.

Press **Alt+4** on the keyboard to move keyboard focus to the Samples panel, and to make it visible if it was hidden. Or, select the Samples item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.

The screenshot shows the 'Action' panel with the following settings:

- Capture parts of the regular expression match**
  - Capturing groups:** TheMonth (2) [New] [Delete] [Delete All]
  - Group name:** TheMonth
  - Actual backreferences:** 2
  - What to capture:** Part of the text matched by a field
  - Field:** 1 Date and time
  - Detail:** Month number
- Action to take**
  - Action to take:** Replace one field
  - Field to replace:** 2 Date and time
  - How to replace:** Replace with another field
  - Field to replace with:** 3 Date and time

### Capture Parts of The Regular Expression Match

You can tell RegexMagic to add capturing groups to your regular expression. Capturing groups make it possible to extract different parts of the regex match. All programming languages that support regular expressions provide functions or variables that allow you to retrieve the text matched by each of the capturing groups. More advanced software tools also allow you to work with text matched by capturing groups. E.g. in PowerGREP, you can run a “collect data” action that collects the text matched by one or more capturing groups, instead of or in addition to the overall regex match.

The other major purpose of capturing groups is to reinsert parts of the regex match into the replacement text while doing a search-and-replace. RegexMagic automatically creates the needed capturing groups and uses backreferences to those groups to the replacement text when you tell RegexMagic to replace fields. If your regex matches three fields, and you tell RegexMagic to replace field **2** with field **3**, then RegexMagic adds capturing groups to capture the text matched by fields **1** and **3**. The replacement text will have one backreference to field **1** (essentially leaving the text matched by that field in place), and two backreferences to field **3** (the first one replacing field **2** with field **3**, and the second one leaving field **3** in place).

## Capturing Groups

This is the list of capturing groups that you have defined. The name of each group is shown. If you have generated the regular expression, the actual backreferences are shown between brackets.

Select a capturing group in this list to edit it.

RegexMagic may automatically create capturing groups if those are needed by other options that you have selected, such as an action to replace certain fields. Those capturing groups are marked as “auto-generated”. RegexMagic will automatically recreate or delete those as needed when you generate the regular expression.

## New Capturing Group

Add a new capturing group. After adding the group, type in its name and choose what to capture.

## Delete Capturing Group

Delete the selected capturing group.

## Delete All Capturing Groups

Delete all the capturing groups that you have added.

Auto-generated capturing groups that are needed for the “action to take” will not be deleted. Set “action to take” to “no action” and regenerate the regex if you want to remove the auto-generated groups too.

## Group Name

Type in a name for the capturing group. Only the letters A-Z and the digits 0-9 are permitted.

If your regular expression flavor supports named capture, RegexMagic will use this name in the regular expression. If not, you can still give the group a name for easy reference within RegexMagic, even though the actual backreference will be a number.

If you do not give the group a name, RegexMagic will generate a numbered capturing group.

## Actual Backreferences

After you generate the regular expression, this field will show the actual backreferences that you can use to get the text matched by this capturing group. Normally, this backreference will be the same as the name of the group, or a number if your regex flavor does not support named capture.

It is possible that RegexpMagic will generate the same capturing group more than once. This will happen when you want to capture a part of something that generates multiple alternatives in the regular expression. Usually, only one of those backreferences will have actually captured something when your regex finds a match.

If a capturing group is unused, RegexpMagic will display “none” as the actual backreference. This will happen if you chose to capture something that is not added to the regular expression, e.g. the decimal separator of a field that matches integer numbers only.

The actual backreferences will be indicated as “unknown” if you haven’t generated the regular expression yet.

## What to Capture

Choose which part of the regular expression the selected capturing group should capture.

- **Text matched by a field:** Capture the text matched by one field entirely. If the field is repeated, the group captures the text at the beginning of the first repetition until the end of the last repetition.
- **Part of the text matched by a field:** Capture a specific part of a particular field. The available parts depend on the pattern you’re using for the field.
- **Text before a field:** Capture the text from the start of the regex match, until the last character before the selected field.
- **Text after a field:** Capture the text from the first character after the selected field, until the end of the regular expression.
- **Text matched by a range of fields:** Capture the text matched by two fields, and all the fields between those two.

## Field

Select the field that the selected capturing group should capture. This is the list of fields you defined on the Match panel.

## Last field

Select the last field that the selected capturing group should capture. This is the list of fields you defined on the Match panel.

## Detail

Select which part of the field you want to capture. The available parts depend on the pattern you’re using for the field.

## Action to Take

Regular expressions are used for three common tasks: finding text, splitting text, and replacing text. If you just want to run a search, select “no action”. If you want to split text along regex matches, select “split subject along regex”. RegexMagic will not generate a replacement text for these two choices.

When doing a search-and-replace, the entire regex match is always replaced. To replace only part of the regex match, capturing groups have to be used to reinsert the parts that you don’t want to change back into the replacement text. RegexMagic automatically takes care of this when you select one of the actions that replace only one or only certain fields.

## Action to Take

Choose if you want to take a certain action on each regular expression match. The action you select here determines the functions available on the Use panel.

- **No action:** Just find regex matches. Shows “match” functions.
- **Replace whole regex match:** Replace the whole regular expression match with something new. Shows “replace” functions.
- **Replace one field:** Replace one of the fields you’ve defined with something new. Leave the remainder of the regex match unchanged. Shows “replace” functions.
- **Replace one capturing group:** Replace one of the capturing groups you’ve defined with something new. Leave the remainder of the regex match unchanged. Shows “replace” functions.
- **Replace all fields:** Replace all of the fields you’ve defined with something new, except for the fields that you choose to ignore. Shows “match” functions.
- **Replace all capturing groups:** Replace all of the capturing groups you’ve defined with something new, except for the capturing groups that you choose to ignore. Shows “replace” functions.
- **Split subject along regex:** Use the regular expression to split a string into a list of strings. Shows “split” functions.

## Capturing Group to Replace

Select the field that you want to replace with something new. You can choose from the list of capturing groups you defined on the Action panel.

## Field to Replace

Select the field that you want to replace with something new. This is the list of fields you defined on the Match panel.

## How to Replace

Choose if you want to take a certain action on each regular expression match. The action you select here determines the functions available on the Use tab.

- **Ignore this field or group:** The text matched by this field or capturing group will be left in place after the replacement, unless the field or capturing group is nested within another field or capturing group that is replaced. In that case, both are replaced together.
- **Leave the matched text in place:** The field or capturing group will be replaced with a backreference that reinserts the text matched by this field or capturing group.
- **Delete the matched text:** Omit this field or capturing group from the replacement.
- **Replace with another field:** Replace the field or capturing group with the text matched by another field.
- **Replace with another capturing group:** Replace the field or capturing group with the text matched by another capturing group.
- **Enter literal replacement text:** Replace the text matched by this field or capturing group with a fixed piece of literal text.
- **Enter a replacement with backreferences:** Enter your own replacement text for the field or capturing group. You can use backreferences to capturing groups, and any of the features supported by the selected replacement text flavor.

### Capturing Group to Replace With

A backreference that holds the text matched by this capturing group will be inserted into the replacement text. You can choose from the list of capturing groups you defined on the Action panel.

### Field to Replace With

A backreference that holds the text matched by this field will be inserted into the replacement text. This is the list of fields you defined on the Match panel. If you did not define a capturing group for this field, RegexpMagic will automatically create one.

### Replacement Text Flavor

Select the replacement text flavor this replacement text was created for.

If this flavor is different than the flavor for the final replacement text, RegexpMagic will automatically convert the replacement text below before inserting it into the final replacement text.

### Replacement Text

Type in the replacement text you want to use.

If you chose for RegexpMagic to interpret backreferences in this replacement text, they will be highlighted according to the regular expression flavor that you selected. Use the right-click menu to insert backreferences to the capturing groups that you defined.

## Split Limit

Turn on to limit the number of items in the resulting array when the subject string is split. Turn off to split the subject string as many times as possible along all regex matches.

## Split Limit

Enter the maximum the number of items in the resulting array when the subject string is split. The subject string will be split at most one time less than the number you enter, with the last element in the array being the unsplit remainder of the subject string. The minimum number you can enter is 2 to split the string once on the first regex match and return an array with 2 strings.

## Include Capturing Groups

Turn on the add text matched by capturing groups to the returned array between the split parts of the subject string. In most applications, capturing groups added to the array do *not* count towards the limit that you may have specified for the number of strings in the returned array.

Turn off to discard text matched by capturing groups.

## Include Empty Strings

Turn on to add empty strings to the array when adjacent regex matches are found. Depending on the application, the array may or may not have trailing empty strings when a regex match is found at the end of the subject string.

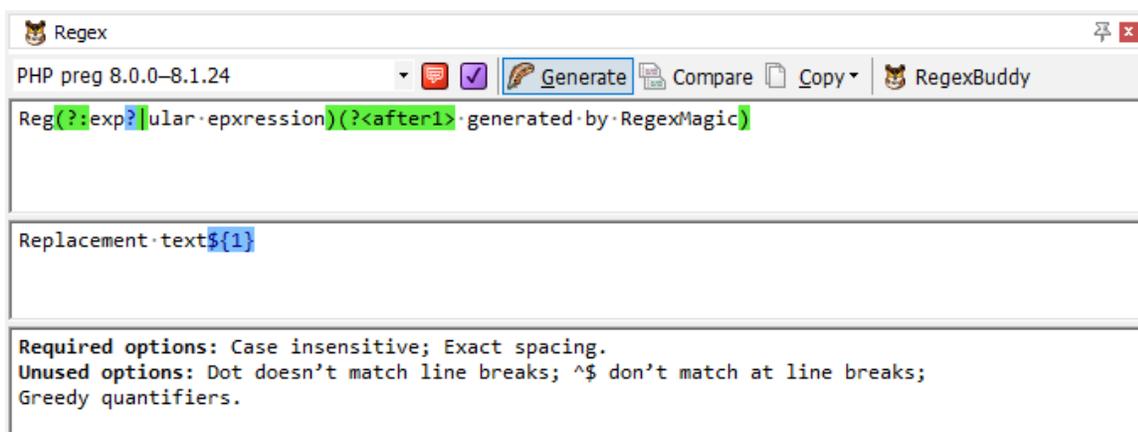
Turn off to never add any empty strings to the returned array.

## 29. Regex Panel Reference

The Regex panel is where you have RegexMagic generate your regular expression, and possibly a replacement text, according to your specifications on the Match and Action panels.

In the default layout and the dual monitor tabbed layout, the Regex panel is docked together under a tab with the Use panel. Click the “Regex, Use” tab near the top of the RegexMagic window to activate the Regex panel. In both side by side layouts, the Regex panel is permanently visible.

Press **Alt+5** on the keyboard to move keyboard focus to the Regex panel, and to make it visible if it was hidden. Or, select the Regex item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.



### Applications and Regular Expression Flavors

The good news about regular expressions is that you can use them with a wide variety of applications and programming languages. The bad news is that the developers of each regular expression engine (the software component that executes your regular expressions) has their own idea of what the regex syntax should be. That has resulted in a wide range of regular expression flavors. Though many claim to be “Perl compatible”, there are significant differences between most flavors.

Fortunately, you have RegexMagic to generate your regular expressions for you. All you need to do is to select the correct application in RegexMagic. RegexMagic supports 292 different applications and programming languages. The number is so large because RegexMagic supports many different versions of the same applications. This ensures that you get the same results when testing the generated regular expression in RegexMagic as when using it in your actual application.

If your application needs to work with multiple applications or multiple versions of the same application, select the primary application or version in the drop-down list and click the Compare button to select all your target applications for comparison.

## More Applications and Languages

Select “More applications and languages” at the top of the list or press Alt+F1 on the keyboard to show a dialog box with the complete list of predefined applications and languages. The ones you tick in the dialog box are the favorites shown directly in the drop-down list. The built-in applications cannot be edited or deleted. But you can edit their associated source code templates.

An “application” in RegexMagic is a group of five settings:

1. Regular expression flavor: The syntax supported by and the behavior of the regular expression engine used by this application.
2. Replacement text flavor: The replacement text syntax, if any, supported by this application.
3. Split flavor: The behavior of this application when splitting a string along regex matches, if supported.
4. String style: Rules for formatting a regex or replacement text as a literal string when copying and pasting.
5. Template for source code snippets: Template for generating code snippets on the Use panel. Click the Edit button to edit a template or to create a new one.

When adding custom applications, you are limited to selecting predefined regex, replacement, and split flavors. Because the flavor definitions are very complex, you cannot create your own. You can select them in combinations that were not previously used in any application. If you’re working with an application or programming language that uses a regex, replacement, or split flavor that is different from any of the flavors supported by RegexMagic, you can let us know as a feature request for future versions of RegexMagic.

## Regex Matching Options

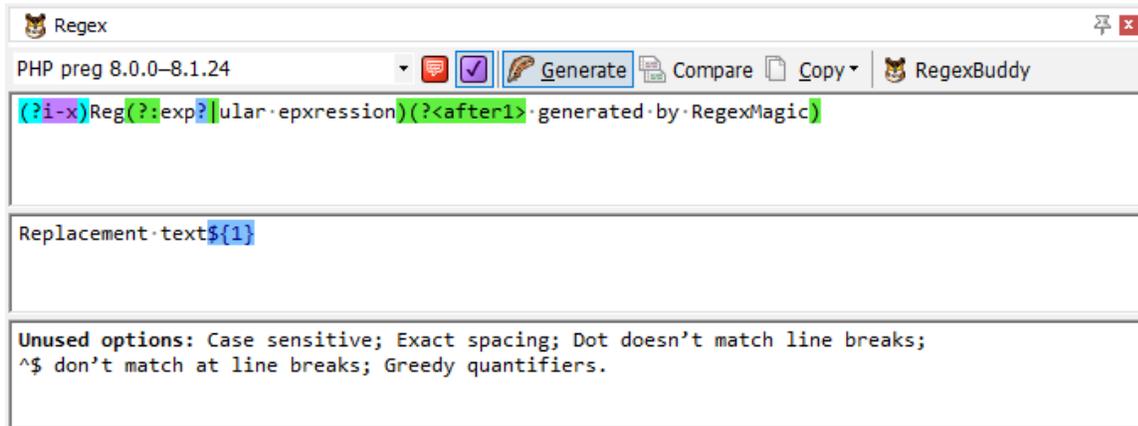
Most regular expression flavors support several regular expression matching options or modes that affect things like case sensitivity and line break handling. These options are traditionally set outside the regular expression. In an application, these can often be set using toggle buttons. In programming languages, these modes are set using flags at the end of a regex literal, or as an extra parameter to functions that take a regular expression as a parameter.

In RegexMagic, these options appear on the Regex panel. Free-spacing mode is available via the  button that you can press before generating the regular expression. You choose whether your regex is free-spacing or not. The other options appear as labels at the bottom of the panel. Required options must be set when using the generated regex to make sure you’ll get the same results as in RegexMagic. The options are always indicated with positive labels, such as “case sensitive” or “case insensitive”. In your actual application you may need to turn off the opposite option if it only has one option as an on/off toggle. For example, if your application has a “case insensitive” option, then you need to turn that off when RegexMagic indicates “case sensitive” is required. Turn it on when RegexMagic indicates “case insensitive” is required.

For completeness, RegexMagic also indicates unused options. These are options that your regular expression flavor allows to be toggled, but that do not affect the regular expression generated by RegexMagic. The labels are the default states for the unused options.

If you generate source code snippets on the Use panel, RegexMagic automatically sets the matching modes in the source code snippets. If you copy and paste your regex, you need to set the matching modes yourself. Or,

you can include the matching modes into the regular expression itself by turning on the  button, as the screen shot shows.



## Free-spacing

Free-spacing mode ignores whitespace and # comments in the regular expression. Most modern regex flavors support this.

Click the red button with the text balloon to turn on free-spacing mode. This is recommended. RegexMagic then adds comments to your regular expression, making it easier to remember what it does later.

If you turn on free-spacing mode in RegexMagic, but you don't turn on the Modifiers button, then you need to make sure to also turn on free-spacing mode in the application or programming language you're working with. If the Modifiers button is turned on, your regex will begin with (?x) to activate free-spacing mode.

## Mode Modifiers

Turn on the Modifiers button (purple button with a tick mark) to place mode modifiers at the start of the regular expression to set the dot matches newline, case insensitive and ^\$ match at line breaks matching modes. This results in a self-contained regular expression that doesn't depend on external options.

Only modern regex flavors support mode modifiers. When the Modifiers button is disabled, or you turned it off, you need to use whatever mechanism your application or programming library provides to set the matching modes your regular expression needs. RegexMagic indicates this in the bottommost box on the Regex panel. The list of "required options" are the options that your regular expression depends on. You have to turn them on to make your regex work as RegexMagic intended. For completeness, RegexMagic also indicates the remaining options that the application supports as "unused options". Turning unused options on or off will not affect your regular expression.

## Generate Your Regular Expression

After selecting the application and toggling the  and  buttons as needed, click Generate button to generate your regular expression following your specifications on the Match panel. If you specified a

replacement action on the Action panel, the Generate button generates a replacement text too. If the regular expression can't be generated for some reason, the RegexMagic Assistant tells you why.

Once clicked, the Generate button stays down until you click it again. While the button is down, RegexMagic will automatically regenerate your regular expression and replacement text each time you change something on the Match or Action panel. If the regular expression can't be generated for some reason, nothing will be displayed until you correct the error. To see the error message in the Assistant, click the Generate button again, even if it is already down.

## **Copy Your Regex to The Clipboard**

It's tempting to just hit Ctrl+A and Ctrl+C to select the regular expression you just generated, and copy it to the clipboard. That will work if you plan to paste it into an application that provides a box to paste a regular expression in, such as the Search boxes in EditPad Pro or PowerGREP. But you're making your life difficult if you plan to paste your regex "as is" into your source code.

Instead, click the Copy button on the Regex toolbar. RegexMagic presents you with a long list of programming language string styles to choose from. Select the string style you need in your source code, and RegexMagic will properly format your regex as a string, escaping characters that have a special meaning in strings in your programming language. The topic Copying and Pasting Regexes in this manual has all the details.

## 30. Copying and Pasting Regular Expressions

Copy and paste is the simple and easy way to transfer a regular expression between RegexMagic and your searching, editing and coding tools. Use the Copy button on the Regex panel to transfer the regular expression in different formats. If you often use RegexMagic in conjunction with a particular tool or application, you may want to see if it is possible to integrate RegexMagic with that tool.

The search boxes of text editors, grep utilities, etc. do not require the regular expression to be formatted in any way, beyond being a valid regex pattern. This is exactly the way the Regex panel in RegexMagic displays the generated regular expression. You can simply copy the regex “as is” if you want to paste it into the search box of an application.

If you want to use the regular expression in the source code of an application or script you are developing, the regex needs to be formatted according to the rules of your programming language. Some languages, such as Perl and JavaScript use a special syntax reserved for regular expressions. Other languages rely on external libraries for regular expression support, requiring you to pass regexes to their function calls as strings. This is where things get a bit messy.

### Matching Modes, Flavors, and String Styles

Matching modes like “case insensitive” and “dot matches line breaks” are generally not included as part of the regular expression. That means they aren’t included when copying and pasting regular expressions. The only exceptions are the JavaScript, Perl, PHP, and Ruby operators. For all the other string styles, you’ll need to make sure by yourself that you’re using the same matching modes in your actual tool or source code as those indicated by RegexMagic on the Regex panel. If you want RegexMagic to take care of this for you, use the source code snippets on the Use tab rather than direct copy and paste.

The string style for copying the regex is independent of the regular expression flavor. Suppose you select Python in the drop-down list with applications, generate a regular expression, and then select the Copy as JavaScript String command. That will give you a regular expression using Python’s regex flavor formatted as a JavaScript string. This could be useful if your Python application reads its regular expressions from a JSON (JavaScript Object Notation) file. But if you wanted to use it in a JavaScript application, you need to select JavaScript in the drop-down list with applications before selecting Copy as JavaScript String. The string style that best matches your chosen application is always listed directly under the Copy button. All the other string styles can be found in the submenu.

### Copying The Regex as a String or Operator

In regular expressions, metacharacters must be escaped with a backslash. In many programming languages, backslashes appearing in strings must be escaped with another backslash. This means that the regex `\\` which matches a single backslash, becomes `\\\\` in Java or C/C++. The regex `"\\\"` which matches a single backslash between double quotes, becomes `"\\\"\\\\\"`. How’s that for clarity?

When you generate code snippets on the Use tab, RegexMagic automatically inserts your regexes in the proper format into the code snippets, adapting them to the whims of each language. To use a regex you prepared in RegexMagic without creating a code snippet, click the Copy button on the toolbar on the Regex panel. You can copy the regular expression to the clipboard in one of several formats. Directly under the

Copy button, you can find the string style used by the programming language you've selected on the Use panel. If the current regular expression flavor usually uses a different string style, that one will also be listed directly under the Copy button. All other string styles will be listed under the "Copy regex as" submenu.

**As is:** Copies the regex unchanged. Appropriate for tools and applications, but not for source code.

**Basic-style string:** The style used by programming languages derived from Basic, including Visual Basic and Xojo (REALbasic). A double-quoted string. A double quote in the regex becomes two double quotes in the string.

**C string:** A string of char in C and C++. A double-quoted string. Backslashes and double quotes are escaped with a backslash. Supports escapes such as `\t`, `\n`, `\r`, and `\xFF` at the string level.

**C Wide string:** A string of `wchar_t` in C and C++. A double-quoted string prefixed with the letter L. Backslashes and double quotes are escaped with a backslash. Supports escapes such as `\t`, `\n`, `\r`, `\xFF`, and `\uFFFF` at the string level.

**C++11 Raw string:** A string of char in C++11 quoted as a Raw string. Raw strings can contain literal line breaks and unescaped quotes and backslashes. Does not support any character escapes. If the regex contains the characters `)"` then `RegexMagic` automatically uses a longer custom delimiter to make sure the delimiter does not occur in the regex. If the regex does not contain any line breaks, quotes, or backslashes a normal double-quoted string is copied as then there is no benefit to using a raw string.

**C++11 Wide Raw string:** A string of `wchar_t` in C++11 quoted as a Raw string.

**C# string:** If the regex contains backslashes, it will be copied as a verbatim string for C# which doesn't require backslashes to be escaped. Otherwise, it will be copied as a simple double-quoted string.

**Delphi string:** The style used by Delphi and other programming languages derived from Pascal. A single-quoted string. A single quote in the regex becomes two single quotes in the string.

**Delphi Prism string:** The style used by Delphi Prism, formerly known as Oxygene or Chrome. Either a single-quoted string on a single line, or a double-quoted string that can span multiple lines. A quote in the regex becomes two quotes in the string.

**Groovy string:** The Groovy programming language offers 5 string styles. Single-quoted and double-quoted strings require backslashes and quotes to be escaped. Using three single or three double quotes allows the string to span multiple lines. For literal regular expressions, the string can be delimited with two forward slashes, requiring only forward slashes to be escaped.

**Java string:** The style used by the Java programming language. A double-quoted string. Backslashes and double quotes are escaped with a backslash. Unicode escapes `\uFFFF` allowed.

**JavaScript operator:** A Perl-style `//` operator that creates a literal `RegExp` object in the ECMAScript programming language defined in the ECMA-262 standard, and its implementations like JavaScript, JScript and ActionScript. ECMA-262 uses mode modifiers that differ from Perl's.

**JavaScript string:** The string style defined in the ECMA-262 standard and used by its implementations like JavaScript, JScript and ActionScript. A single-quoted or double-quoted string. Backslashes and quotes are escaped with a backslash. Unicode escapes `\uFFFF` and Latin-1 escapes `\xFF` allowed.

**Perl-style string:** The style used by scripting languages such as Perl and Ruby, where a double-quoted string is interpolated, but a single-quoted string is not. Quotes used to delimit the string, and backslashes, are escaped with a backslash.

**Perl operator:** A Perl `m//` operator for match and split actions, and an `s///` operator for replace actions.

**PHP string:** A string for use with PHP's `ereg` functions. Backslashes are only escaped when strictly necessary.

**PHP `'/'` preg string:** A Perl-style `//` operator in a string for use with PHP's `preg` functions.

**PostgreSQL string:** A string for PostgreSQL, delimited by double dollar characters.

**PowerShell string:** A string for PowerShell. Uses “here strings” for multi-line strings. Quotes and non-printables are escaped with backticks.

**Python string:** Unless the regex contains both single and double quote characters, the regex is copied as a Python “raw string”. Raw strings do not require backslashes to be escaped, making regular expressions easier to read. If the regex contains both single and double quotes, the regex is copied as a regular Python string, with quotes and backslashes escaped.

**R string:** The string style used by the R programming language. A single-quoted or double-quoted string. Backslashes and quotes are escaped with a backslash. Unicode escapes `\U0010FFFF`, basic Unicode escapes `\uFFFF`, and Latin-1 escapes `\xFF` allowed.

**Ruby operator:** A Perl-style `//` operator for use with Ruby. Ruby uses mode modifiers that differ from Perl's.

**Scala string:** Copies the regex as a triple-quoted Scala string which avoids having to escape backslashes and allows line breaks which is handy for free-spacing regular expressions.

**SQL string:** The string style used by the SQL standard. A single-quoted string. A single quote in the regex becomes two single quotes in the string. The string can span multiple lines. Note that not all databases use this string style. E.g. MySQL uses C-style strings, and PostgreSQL uses either C-style strings or dollar-delimited strings.

**Tcl word:** Delimits the regular expression with curly braces for Tcl. In Tcl parlance, this is called a word.

**XML:** Replaces ampersands, angle brackets and quotes with XML entities like `&amp;`; suitable for pasting into XML files.

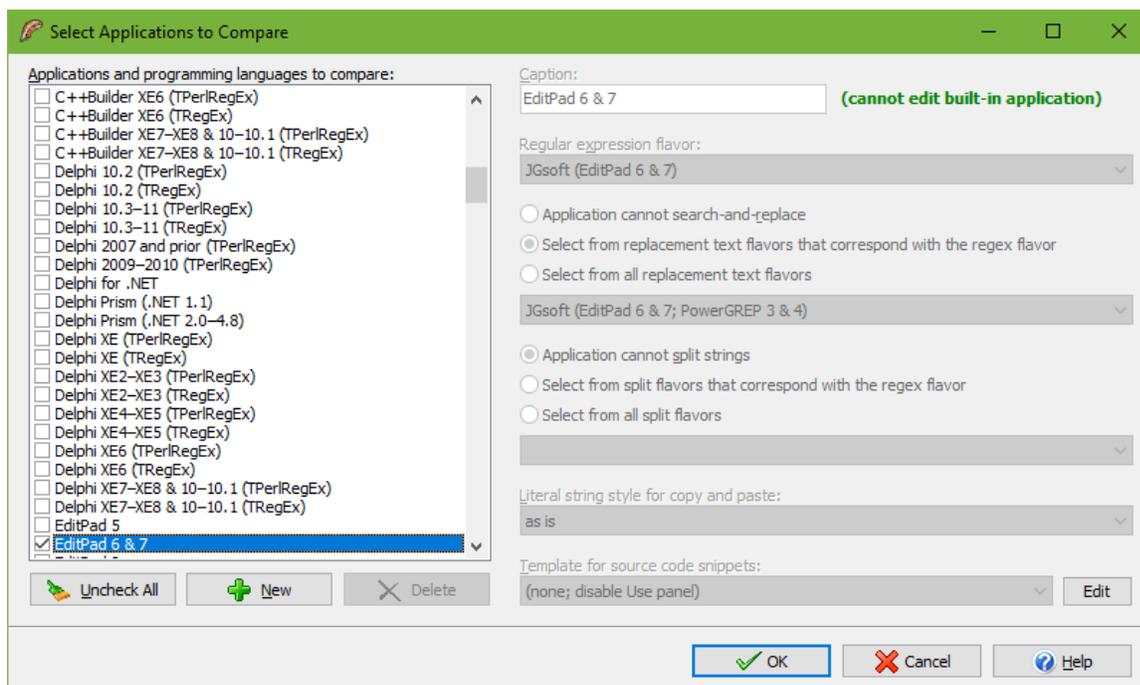
## 31. Compare Applications

Sometimes you may need a regular expression that works with multiple applications, or multiple versions of the same application. A regular expression that validates input on a web form, for example, may need to work on both the client side where JavaScript or HTML5 is used and on the server side where PHP or ASP.NET might be used. Then you'd be working with two incompatible regular expression flavors. On the client side you'll also want to support the JavaScript or HTML5 implementations in all modern browsers, which further add minor inconsistencies.

Fortunately, RegexMagic takes care of all issues with different regex flavors for you. All you need to do is click the Compare button to select all of the versions of all the applications that your regular expression needs to work with. RegexMagic then generates your regular expression for the regex flavors of all these applications and will tell you whether the same regex will work with all of them, or whether you need different regexes.

You should also provide plenty of samples on the Samples panel. RegexMagic tests the regular expressions it generated for all applications in the comparison on all your samples. This happens in the background. Providing lots of samples does not slow down RegexMagic. The tests are run by a special regular expression engine that emulates 574 different aspects (syntactic and behavioral differences) of 94 regular expression flavors used by the 133 different applications and programming languages that RegexMagic supports. You can be confident that RegexMagic's comparison corresponds with what the generated regexes will do in your actual applications. You only need to make sure that your samples are representative of the actual input the regular expressions will have to deal with.

### Selecting Applications for Comparison



The Select Applications to Compare dialog appears when you click the Compare button on the Regex panel. This dialog looks very similar to the More Applications and Languages dialog that appears when you select More Applications and Languages at the top of the drop-down list with applications on the Regex panel. The only difference is that the checkboxes in the list of flavors control which applications will be included in the comparison.

The application you selected in the drop-down list on the Regex panel is always included in the comparison. But you should tick that application in the Select Applications to Compare dialog too. That ensures that this application will continue to be included in the comparison when you select another application in the drop-down list.

To stop comparing applications, click the Uncheck All button and then click OK.

## Comparison Results

The results of the comparison appear at the bottom of the Regex panel. If you have lots of applications selected and/or have provided lots of samples on the Samples panel (as you should), then the comparison may take a moment. The Regex panel will indicate that comparison is in progress and automatically show the results when the comparison is done. You can continue using RegexMagic while the comparison is in progress. The comparison is automatically restarted if you do anything that might change the results.

This example shows all possible results you might get from a comparison:

**\d**

**Unused options:** Case insensitive; Dot doesn't match line breaks.

The regex flavor used by [AceText 2 & 3](#) is **identical** to the one used by EditPad 6 & 7. There is no need to compare these applications.

**Cannot generate the regular expression** for [XML Schema](#).

[Perl 5.30–5.32](#) uses the **same regex** and finds the **same matches** as EditPad 6 & 7.

[Java 19–21](#) uses a **different regex** yet finds the **same matches** as EditPad 6 & 7.

[JavaScript \(Chrome\)](#) uses the **same regex** but finds **different matches** than EditPad 6 & 7.

[MySQL](#) uses a **different regex** and finds **different matches** than EditPad 6 & 7.

“EditPad 6 & 7” was the application selected in the drop-down list on the Regex panel. The regex panel shows the regular expression and regex options generated for EditPad 6 & 7.

“The regex flavor used by AceText 2 & 3 is identical to the one used by EditPad 6 & 7” means that (as far as RegexMagic knows) these two applications use the exact same regular expression flavor. RegexMagic always generate the same regular expressions for them and always highlight the same matches for them on the Samples panel. When two or more such applications are included in the comparison, RegexMagic does the comparison only once for each of them. Note that if any of the other messages in the comparison results list multiple applications that does not mean that those applications use identical regex flavors. It only means that for the present comparison they happen to produce the same outcome.

“Cannot generate the regular expression for XML Schema” means that the settings on the Match panel require features that XML Schema's regex flavor does not support. When you see this message, click on the application's link. The error message for this flavor then appears in the Assistant panel. In this example, XML

Schema is not capable of finding partial matches like EditPad. If this message indicates multiple applications, they may have the same errors or different errors.

“Perl 5.18 uses the same regex and finds the same matches as EditPad 6 & 7” means that the same regex can be used in Perl and in EditPad. It will find the same matches in Perl as in EditPad, at least when used on the samples you provided on the Samples panel. The regular expression may need to be copied differently though. Select Copy Regex As Is under the Copy button to paste into EditPad’s Search box. Select Copy as Perl Operator to paste into Perl source code.

“Java 8 uses a different regex yet finds the same matches as EditPad 6 & 7” means that RegexMagic generates a different regular expression for Java than for EditPad. That does not mean that the regular expression for EditPad won’t work at all with Java. It means that RegexMagic decided that a different regular expression will give better results with Java than the regex for EditPad. When you see this message, click the application’s link to see the regular expression for that application on the Regex panel. In this example, RegexMagic generates `\p{Nd}` for Java to match Unicode digits like `\d` does in EditPad. You can use `\d` in Java, but it would match ASCII digits only. If this message indicates multiple applications, then all indicated applications need regexes that are different from the regex for the main application. The indicated applications may use need same regex or different regexes than each other. Click the link of one of the indicated applications to compare it with the others.

“JavaScript (Firefox) uses the same regex but finds different matches than EditPad 6 & 7” means that RegexMagic couldn’t generate a regular expression for JavaScript that finds the same matches as the regex for EditPad and that the regular expression that comes closest for JavaScript happens to be the same regex as the one for EditPad. When you see this, click the application’s link. The samples panel will then highlight the matches for the other application. If this message indicates multiple applications, then all indicated applications find matches that are different from the matches found by the main applications. The indicated applications may find the same matches or different matches than each other. Click the link of one of the indicated applications to compare it with the others.

“MySQL uses a different regex and finds different matches than EditPad 6 & 7” is the combination of the previous two results. It means that RegexMagic decided that a different regular expression will give better results with MySQL than the regex for EditPad, but that even the better regular expression does not give the same results. In this example, MySQL uses `[[[:digit:]]` instead of `\d` to match digits, but only matches ASCII digits.

If you want to experiment with this example, you can recreate this comparison with the following (contrived) RegexMagic Formula:

1. Click the New Formula button on the top toolbar to clear out all settings on the Samples, Match, and Action panels.
2. On the Samples panel, paste in one new sample:
  - 1.
3. Click the Add First Field button  to add the first (and only) field.
4. In the “pattern to match field” drop-down list, select “regular expression”.
5. In the “regular expression flavor” drop-down list, select “EditPad 6 & 7”.
6. Paste in this regular expression:

`\d`

7. On the Regex panel, select the JGsoft flavor, turn off free-spacing, and turn off mode modifiers.
8. Click the Compare button.
9. In the dialog, click Uncheck All.
10. Check AceText 2 & 3, EditPad 6 & 7, Java 8, JavaScript (Firefox), MySQL, and Perl 5.18.
11. Click the OK button.
12. Click the Generate button, and you'll get the comparison results from the example.

This example is a bit contrived because it uses the “regular expression” pattern. This makes RegexMagic try to convert the given regular expression to all the applications in the comparison. A more realistic scenario would be to use the “basic characters” or “Unicode characters” patterns to tell RegexMagic to match a digit. Then the comparison results would be far simpler. For “basic characters” you’d get the same matches for all applications because all regex engines can match ASCII digits. For “Unicode characters” you’d get the same matches for all applications that support Unicode, and an error for applications that don’t support Unicode.

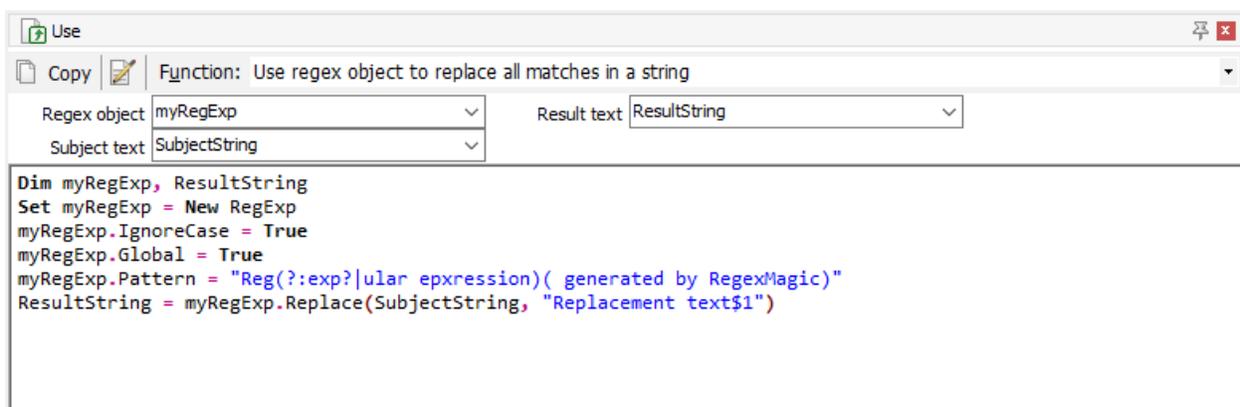
The example does show the importance of making sure your samples are representative of the actual data your regular expression will have to work with. If you change the sample to include only ASCII characters, then you will get the same matches for all applications for which a regex can be generated. But when the Thai digit ๖ is added, applications that don’t support Unicode produce different matches than those that do. If your actual application only ever works with ASCII data, then you should only provide ASCII samples to avoid needless false alarms about different match results. If your actual application needs to be able to text in many different scripts, then you’ll need to provide samples in all scripts that matter to make sure the comparison is sufficiently thorough.

## 32. Use Panel Reference

On the Use panel you can generate a source code snippet in your favorite programming language to perform a task of your choice with the regular expression generated by RegexMagic on the Regex panel.

In the default layout and the dual monitor tabbed layout, the Use panel is docked together under a tab with the Regex panel. Click the “Regex, Use” tab near the top of the RegexMagic window to activate the Use panel. In both side by side layouts, the Regex panel is permanently visible.

Press **Alt+6** on the keyboard to move keyboard focus to the Use panel, and to make it visible if it was hidden. Or, select the Use item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.



### Generate Source Code to Use a Regular Expression

Once you have generated a regular expression, you are ready to use it. One way is to simply tell RegexMagic to copy the regular expression to the clipboard, so you can paste it into tool or editor where you want to use the regex. If you do this often, you may want to see if it is possible to integrate RegexMagic and the software you use regular expressions with.

If you want to use the regular expression in the source code for an application you are developing, switch to the Use panel in RegexMagic. RegexMagic can generate code snippets in a variety of programming languages that have particularly strong support for regular expressions, or have a popular regular expression library available for them.

### Benefits of RegexMagic’s Code Snippets

Using RegexMagic’s code snippets gives you several major benefits. You don’t have to remember all the details how to use a particular language’s or library’s regex support. Though most languages and libraries have similar capabilities, the actual implementation is quite different. With RegexMagic, you simply select the task you want to accomplish from the list, and RegexMagic generates source code you can readily copy and paste into your code editor or IDE.

Another key benefit is that when using a regular expression in a programming language, certain characters need special treatment, up and above the special treatment they may need in the regular expression itself. E.g. the regex `\` to match a single backslash is automatically inserted as `\"` in a Java or C code snippet, and as `\\` in a JavaScript or Perl snippet. Never again mess around with pesky backslashes!

For replacement actions, RegexMagic also knows if a particular regex library uses backslashes or dollar signs to insert backreferences into the replacement text. RegexMagic supports both, but most libraries accept only one style.

For languages that support exception handling, RegexMagic's code snippets also contain `try..catch` or equivalent code blocks to handle any exception that could be thrown by any of the methods the code snippet calls.

## How to Generate a Code Snippet

To generate a code snippet, first generate the regular expression for your programming language on the Regex panel. Then switch to the Use panel and select the type of function you want to implement. Some functions are only available for certain languages. E.g. the functions that create an object for repeatedly applying a regular expression are only available in languages with object-oriented regex libraries.

The available functions also depend on the kind of action you defined. Search and extraction functions are listed if you don't specify an action, search-and-replace functions are listed for replace actions, and splitting functions are listed for split actions.

Most functions allow you to specify certain parameters, such as the variable or string constant you want to use as the subject for the regex operation, the variable to store the results in, the name of the regex or matcher object, etc. RegexMagic does not verify if you entered a valid constant or variable. Whatever you enter is inserted into the code snippet unchanged. The parameters you entered are automatically remembered. For each parameter, the same values are carried over across all the functions, for each language. So you need to enter the names you typically use only once.

RegexMagic keeps the code snippet in sync with your regular expression at all times, even when the regex is syntactically incorrect. The snippet is ready for copying and pasting at all times. If you want, you can edit the snippet in RegexMagic. But remember that your changes will be lost as soon as you regenerate the regular expression.

You can transfer the snippet into your code editor or IDE by clicking the Copy button on the Use tab's toolbar. This copies the entire snippet to the Windows clipboard, ready for pasting. If you only want to use part of the snippet, either select it and press `Ctrl+C` on the keyboard to copy it, or select it and then drag-and-drop it with the mouse into your code editor.

## Available Languages

RegexMagic ships with source code templates for all of the world's most popular programming languages. Each template implements a comprehensive set of functions, or things that you can do with your regular expression in the selected programming language.

If the provided functions don't match your coding style, you can easily edit any of the templates by clicking the Edit button on the toolbar on the Use panel. You can edit or delete the provided functions, and add as many new functions as you like.

You can also create new templates. These can be alternative sets of functions for a programming language already supported by RegexMagic, such as a set of functions specific to a particular development project. You can also create templates for languages not supported by RegexMagic. The only limitation is that you'll be limited to languages that use one of the regex flavors and string styles already supported by RegexMagic.

## 33. Available Functions in Source Code Snippets

To make it easy for you to apply the same regex techniques in a variety of programming languages, RegexMagic uses the same function descriptions for all languages it supports. Some functions are not available for all languages, depending on the functionality provided by the language or its regular expression library.

**Import regex library:** If the regular expression support is not a core part of the language, you need to reference the regex library in your code, before you can use any of the other code snippets.

**Test if the regex matches (part of) a string:** Tests if the regular expression can be matched anywhere in a subject string, and stores the result in a boolean variable.

**Test if the regex matches a string entirely:** Tests if the regular expression matches the subject string entirely, and stores the result in a boolean variable. This function is appropriate when validating user input or external data, since you typically want to validate everything you received. If the language does not have a separate function for testing if a string can be matched entirely, RegexMagic places anchors around your regular expression to obtain the same effect.

**If/else branch whether the regex matches (part of) a string:** Tests if the regular expression can be matched anywhere in a subject string. Add code to the *if* section to take action if the match is successful. Add code to the *else* section to take action if the match fails.

**If/else branch whether the regex matches a string entirely:** Tests if the regex matches a subject string in its entirety, adding anchors to the regex if necessary.

**Create an object with details about how the regex matches (part of) a string:** Applies the regular expression to a subject string, and returns an object with detailed information about the part of the subject string that could be matched. The details usually include the text that was matched, the offset position in the string where the match started, and the number of characters in the match. If the regex contains capturing groups, the details also contain the same information for each of the capturing groups. The groups are numbered starting at one, with group number zero being the overall regex match.

Note that for regular expressions that consist only of anchors or lookaround, or where everything is optional, it is possible for a successful match to have a length of zero. Such regexes can also match at the point right after the end of the string, in which case the offset points to a non-existent character.

**Get the part of a string matched by the regex:** Applies the regular expression to a subject string, and returns the part of the subject string that could be matched. This function is appropriate if you are only interested in the first (or only) possible match in the subject string.

**Get the part of a string matched by a numbered group:** Applies the regular expression to a subject string, and returns the part of the subject string that was matched by a capturing group in the regex. This function is appropriate if you are only interested in a particular part of the first (or only) possible match in the subject string. One of the parameters for this function is the number of the capturing group. You can easily obtain this number from the “actual backreferences” field on the Action panel.

**Get the part of a string matched by a named group:** Applies the regular expression to a subject string, and returns the part of the subject string that was matched by a named capturing group in the regex. This function is appropriate if you are only interested in a particular part of the first (or only) possible match in the

subject string. One of the parameters for this function is the name of the capturing group. You can easily obtain this name from the “actual backreferences” field on the Action panel.

**Iterate over all matches in a string:** Applies and re-applies a regular expression to a subject string, iterating over all matches in the subject, until no further matches can be found. Insert your own code inside the loop to process each match. Match details such as offset and length of the match, the matched text, the results from capturing groups, etc. are available inside the loop.

**Iterate over all matches and capturing groups in a string:** Iterates over all matches in the string like the previous function, and also iterates over all capturing groups of each match. Note that the number of available capturing groups may be different for each match. If certain capturing groups did not participate at all in the overall match, no details will be available for those groups. In practice, you will usually use the previous function, and only work with the capturing groups that you are specifically interested in. This function illustrates how you can access each group.

**Validate input with a regular expression:** Applies the regular expression to the text entered in a field on a form to validate the field before processing the form.

## Replace Functions

These functions are available for languages that can use a regular expression to search-and-replace through a string. These functions only appear in the Function drop-down list when you’re defining a replace action.

**Replace all matches in a string:** Executes a replace action. All regex matches in the subject string are substituted with the replacement text. References to capturing groups in the replacement text are expanded. Unless otherwise indicated in the function’s name, the “replace all” function does not modify the original subject string. It returns a new string with the replacements applied.

**Search and replace through a string:** Executes a replace action using a callback function that returns the replacement text. You can edit this callback function to use procedural code to modify or build the replacement for each regex match.

## Split Functions

These functions are available for languages that can use a regular expression to split a string. These functions only appear in the Function drop-down list when you’re defining a split action.

**Split a string:** Executes a split action. Returns an array or list of strings from a given subject string. The first string in the array is the part of the subject before the first regex match. The second string is the part between the first and second matches, the third string the part between the second and third matches, etc. The final string is the remainder of the subject after the last regex match. The text actually matched by the regex is not added to the array.

## Object-Oriented Functions

These functions are available for programming languages that use an object-oriented regular expression library.

**Create an object to use the same regex for many operations:** Before a regular expression can be applied to a subject string, the regex engine needs to convert the textual regex that you (or the user of your software) entered into a binary format that can be processed by the engine's pattern matcher. If you use any of the functions listed above, the regex is (re-)compiled each time you call one of the functions. That is inefficient if you use the same regex over and over. Therefore, you should create an object that stores the regular expression and its associated internal data for the regex engine. Your source code becomes easier to read and maintain if you create regex objects with descriptive names.

**Create an object to apply a regex repeatedly to a given string:** Some regex libraries, such as Java's `java.util.regex` package, use a separate pattern matcher object to apply a regex to a particular subject string, instead of using the regex object created by the function above. Explicitly creating and using this object, instead of using one of the convenience functions that create and discard it behind the scenes, makes repeatedly applying the same regex to the same string more efficient. This way you can find all regex matches in the string, rather than just the first one.

**Apply the same regex to more than one string:** Illustrates how to use the regex object to apply a given regular expression to more than one subject string.

**Use regex object to ...:** This series of functions creates and uses a regex object to perform its task. The results of these functions are identical to the results of the similarly named functions already described above. The advantage of these functions over the ones above, is that you can reuse the same regex object to use a given regular expression more than once. For the second and following invocations, you obviously only copy and paste the part of the code snippet that uses the regex object into your source code.

## Database Functions

These functions are available for database languages (various flavors of SQL).

**Select rows in which a column is/cannot (partially) matched by the regex:** SQL code for selecting rows from a database checking for a regex match on a certain column. Only those rows where the column does/doesn't match the regex are returned.

**Select rows in which a column is/cannot entirely matched by the regex:** SQL code for selecting rows from a database checking for a regex match on a certain column. Anchors are added to the regex to require it to match the column's value entirely. Only those rows where the column does/doesn't match the regex are returned.

**Select the part of a column matched by a regex:** SQL code for selecting rows from a database checking for a regex match on a certain column. Returns only the part of the column that was matched by the regular expression, and only for those rows where the column matched at all.

## 34. Source Code Template Editor

RegexMagic makes it easy to generate source code snippets for all of the world's most popular programming languages. Each language template implements a comprehensive set of functions, or things that you can do with your regular expression in the selected programming language.

If the provided functions don't match your coding style, you can easily edit any of the templates by clicking the Edit button on the toolbar on the Use tab. You can edit or delete the provided functions, and add as many new functions as you like.

You can also create new templates. These can be alternative sets of functions for a programming language already supported by RegexMagic, such as a set of functions specific to a particular development project. You can also create templates for languages not supported by RegexMagic. The only limitation is that you'll be limited to languages that use one of the regex flavors and string styles already supported by RegexMagic.

### RegexMagic Source Code Template Editor

The RegexMagic Source Code Template Editor appears when you click the Edit button on the toolbar on the Use tab. To start with a fresh template, click the New button. To open an existing template, click the Open button and select the template from the drop-down list. Click the Save button to save your modified template, overwriting the existing template. Click the Save As button to save the template under a new name, so both the original and the modified template will be available.

Templates that ship with RegexMagic are stored in the same folder as RegexMagic itself. The default installation folder is "C:\Program Files\JGsoft\RegexMagic". Templates that you save are always saved under the "Application Data\RegexMagic 3" folder under your Windows user profile. If two templates with the same file name exist in both folders, only the template in your user folder will be used. The effect is that if you open a template included with RegexMagic and save it with the same name, only the modified template will be available in RegexMagic, but the original template will still be available on disk.

### General

On the General tab in the template editor, you'll be asked to set the general properties of the language you're creating a template for. You should also enter your contact details, in case you want to share your template on RegexMagic's user forums.

**Language:** Enter the name of the programming language the template is for. You can also specify a more descriptive purpose if you're creating an alternative template for a language that already has a template. This name will appear in the Language drop-down list on the Use tab.

**Regex and replacement flavor:** Select the regular expression flavor and the replacement text flavor used by the language. The available replacement text flavors depend on the regular expression flavor that you selected. It is not possible to define your own flavors. The flavor you select is the target flavor that RegexMagic will use when converting your regular expression.

**String style:** Select the kind of string or operator that the language uses to specify literal regular expressions in source code. The %REGEX% and %REPLACEMENT% placeholders will insert the regular expression

and replacement text into the generated source code snippet after formatting it using the string style you select here. This way, RegexMagic automatically takes care of placing quotes around the string and escaping quotes and backslashes in the regex, or whatever the selected string style requires. The available string styles are the same ones available via the Copy and Paste buttons on RegexMagic's main toolbar.

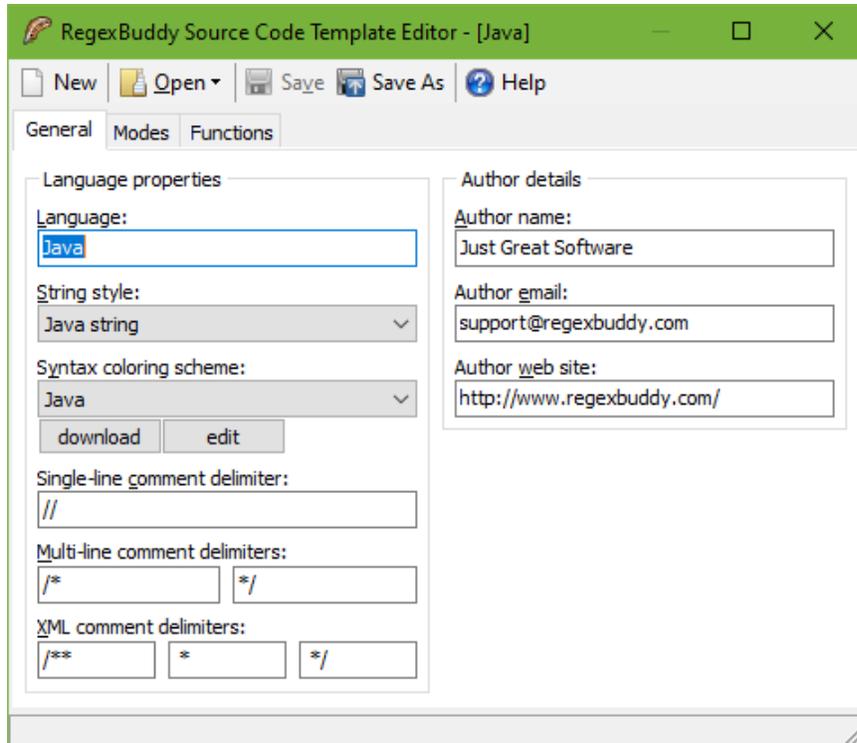
**Syntax coloring scheme:** Select the syntax coloring scheme that RegexMagic should use to highlight keywords, comments, strings, etc. when displaying your source code snippet on the Use tab. This only affects how your code snippets are displayed in RegexMagic. It does not alter the code snippet. The highlighting is not copied over when you copy your code snippet to the clipboard. Click the Download button to download coloring schemes shared by other RegexMagic users. Click the Edit button to edit the selected syntax coloring scheme with the syntax coloring scheme editor. The editor is an external application that licensed RegexMagic users can download for free from the EditPad Pro web site. (RegexMagic uses the same syntax coloring system as EditPad Pro.)

**Single-line comment delimiter:** Enter the character or characters that start a comment that runs until the end of the line in the language you're creating the template for. Leave this box blank if the language doesn't support single-line comments.

**Multi-line comment delimiters:** In the box at the left, enter the character or characters that begin a multi-line comment. In the box at the right, enter the characters that end a multi-line comment. Leave both boxes blank if the language doesn't support multi-line comments.

You have to specify either the single-line comment delimiter or the multi-line comment delimiter pair for RegexMagic to be able to add conversion warnings to the generated source code snippet, and for the "comment with RegexMagic's regex tree" function to be available. If your language supports both single-line and multi-line comments, please specify both styles. RegexMagic will then choose the comment style best suited for each situation.

**Region delimiters:** If your programming language supports the concept of regions, enter the compiler directive that starts the region in the box at the left, and the directive that ends the region in the box at the right. If your language allows regions to be named, you should include a name such as "RegexMagic" or "regular expression tree" in the region delimiters. When you specify region delimiters, the function "comment inside a region with RegexMagic's regex tree" will automatically become available for your language.



## Modes

On the Modes tab in the template editor, you'll need to specify the code that RegexMagic needs to use to make sure your code snippet uses the same matching modes as RegexMagic itself. The labels on the Modes tab correspond with the labels on the mode buttons in RegexMagic's toolbar.

What you specify on the Modes tab is *not* automatically inserted into the source code snippets. Rather, the code you specify for each mode is used to build up the placeholders `%MODELIST%`, `%DOTALL%`, `%CASELESS%`, `%MULTILINE%` and `%FREESPACING%`. The `%MODELIST%` placeholder combines the code from “code to turn on modes” or from “code to turn off modes” (depending on whether the mode is on or off) for each mode as specified in the “code to make a list of modes” section. The other four placeholders simply insert the code from “code to turn on modes” or from “code to turn off modes”, depending on whether the mode is on or off.

**Code to turn on modes:** For modes that are off by default in your language, you should specify the code that turns them on. Leave the code blank for modes that are on by default, or that your programming language does not provide an option for. If the code to set a mode needs to reference a variable or object that you want to be dynamic, you can use one of the `%PARAM1%` through `%PARAM4%` placeholders in the code to turn on the mode.

**Code to turn off modes:** Use this instead of “code to turn on modes” for modes that are on by default in your language. If your language requires the mode to be set whether it's on or off (i.e. there's no default), then you can specify both the “on” and “off” code for the mode.

**Code to make a list of modes:** If you want to use the `%MODELIST%` placeholder in your code snippets to insert all modes together in one spot, you can specify how RegexMagic should build that list. If you leave all the boxes blank and the checkbox off, RegexMagic will simply concatenate all the modes without any spacing between them. In the “begin list” and “end list” boxes you can specify which code, if any, RegexMagic should place at the start or the end of the list. The begin and end code is only used if the list contains at least one mode. In the “delimit modes” box, you can type the characters that RegexMagic should place between two modes. If you want the delimiter to include a line break, turn on the line break checkbox. The delimiter is only used if the list contains two or more modes. In the “empty list substitute” box you can type the code that `%MODELIST%` should insert when all modes are set to their defaults (i.e. all the relevant code to turn modes on and off is blank).

If your language’s regex flavor supports a particular mode via a mode modifier inside the regular expression, but the language or library doesn’t support it via a flag or property that you can set in your application’s source code (other than using the mode modifier), then you should leave both the “code to turn on modes” and “code to turn off modes” blank for that mode. When both these are blank, RegexMagic will automatically insert the mode modifier for that mode at the start of the regular expression to turn the mode on or off, as necessary.

E.g. in REALbasic, “dot matches newlines” is off by default, and “case insensitive” and “^ and \$ match at line breaks” are on by default. It takes a whole line of code to toggle them. “Free-spacing” syntax is supported by the PCRE flavor used by REALbasic, but REALbasic doesn’t provide an option for it. If you generate a REALbasic snippet with the free-spacing option turned on, RegexMagic will automatically prepend the `(?x)` mode modifier to your regular expression to make it work with REALbasic.

## Functions

The Functions tab is where the real action is. Here, you'll define all the functions that will appear in the Functions drop-down list on the Use tab when your template is selected.

To add a function, click the New button. The first function you add will be blank. The next function you add will automatically take over everything from the selected function, except its description. This way you can easily create different variants of a particular function.

The order in which the functions are listed in the template editor is also the order they'll have in RegexMagic. RegexMagic doesn't sort them alphabetically. You should put related functions next to each other. You can rearrange the functions with the Move Up and Move Down buttons. You can delete functions with the Delete button.

The function's description is the label it will have on the Use tab in RegexMagic. The Function Description drop-down list in the template editor lists all the function descriptions in all other RegexMagic source code templates that you haven't used yet in the template you're editing now. This way you can easily make your function descriptions consistent between various templates. It's also a handy reminder of the functions your template doesn't support yet. Of course, you can always enter a new function description. If you then edit another template, the new description will also appear in the drop-down list.

The Match, Replace and Split checkboxes determine for which kinds of actions the function will be available on the Use panel. Since a comprehensive template will have a lot of functions, you can reduce clutter by restricting functions to the kinds of actions that they perform.

Each function can have up to four parameters that the user of your template can provide. You should use these parameters for your code snippet's input and output variables. Such variable names tend to change whenever a code snippet is used. Allowing the variable name to be set in RegexMagic before the snippet is generated is easier than changing it after pasting the snippet into your source code editor.

For each parameter, you can specify a label in the drop-down list at the left, and a default value at the right. The drop-down list will show all function labels that you've already used in this template. If you select a parameter from the drop-down list, RegexMagic will also set the default value you previously set for that parameter. If you change the default value, it will change throughout the current template for all parameters with the same label. There is no limit to the number of different parameters you can use in a language template. However, each function can use only four.

Using three checkboxes you can specify if RegexMagic should modify the regular expression before inserting it into the function's source code. Turn on "prepend mode modifiers to regex" if you want RegexMagic to put mode modifiers for all the matching modes at the start of the regex. As explained in the above section about modes, RegexMagic normally only prepends the mode modifiers for those modes for which you did neither specify "turn on" nor "turn off" code. If your programming language has certain regular expression functions that allow modes to be set in source code, while others don't (leaving mode modifiers as the only way), you should turn on the "prepend mode modifiers to regex" for those functions that don't. For the functions that do, you'll use placeholders like %MODELIST%. Note that some regex flavors, like JavaScript, POSIX and XML don't support mode modifiers. For these flavors, the "prepend mode modifiers" checkbox will have no effect.

There's a difference between checking if a regular expression matches part of a string, versus checking if it matches the whole string. Some programming languages have separate routines for these two tasks. Other

languages don't have a special method to check if a regex matches a string entirely. For those languages, you can create a function that checks if a regex matches a string entirely by turning on the options to prepend and append an anchor. RegexMagic automatically uses the anchors appropriate for your language's regular expression flavor.

Finally, you'll need to enter the actual code snippet. The box in the lower right corner is where you enter your snippet using placeholders. RegexMagic will show a preview of the source code it will generate in the lower left corner. The preview uses the regular expression and options you used in RegexMagic before opening the source code template editor.

You can insert placeholders by typing them in, or by clicking the Placeholders button and selecting one from the list. The following placeholders are available:

- Regular expression: `%REGEX%` inserts the regular expression that RegexMagic generated formatted using the string style you specified on the General tab.
- Replacement text: `%REPLACEMENT%` inserts the replacement text that RegexMagic generated formatted using the string style you specified on the General tab.
- Parameter 1..4: `%PARAM1%` through `%PARAM4%` insert the value for each of the parameters. The preview shows the default values you specify in the template editor. The Use panel will use the user-specified values.
- List of all modes: `%MODELIST%` inserts the list of modes built according to the code and options you specified on the Modes tab to set all of the matching modes.
- List of specific modes: `%BEGINMODELIST%..%ENDMODELIST%` inserts a list of specific modes. Between these tags you can use any of the placeholders from `%CASELESS%` until `%LINEBREAK%`. You should not place anything else between the two tags. The placeholders between the list will automatically be formatted as a list of modes using the delimiter and begin/end list code you specified on the Modes tab. If you click the Placeholder button and select "list of specific modes" you'll get a dialog box where you can select the modes that you want.
- Case insensitive mode: `%CASELESS%` inserts the code you specified on the Modes tab to turn on the "case insensitive" or the "case sensitive" mode, as generated on the Regex panel.
- Free-spacing mode: `%FREESPACING%` inserts the code you specified on the Modes tab to turn on the "free-spacing syntax" mode or the "exact spacing syntax" mode., as generated on the Regex panel
- Dot matches line breaks mode: `%DOTALL%` inserts the code you specified on the Modes tab to turn on the "dot matches line breaks" or the "dot doesn't match line breaks" mode, as generated on the Regex panel.
- `^` and `$` match at line breaks mode: `%MULTILINE%` inserts the code you specified on the Modes tab to turn on the "`^` and `$` match at line breaks" or the "`^` and `$` don't match at line breaks" mode, as generated on the Regex panel.
- Explicit capture mode: `%EXPLICITCAPTURE%` inserts the code you specified on the Modes tab to turn on the "named capture only" or the "numbered capture" mode, as generated on the Regex panel.
- Duplicate capturing group names mode: `%NAMEDDUPLICATE%` inserts the code you specified on the Modes tab to turn on the "allow duplicate names" or the "names must be unique" mode, as generated on the Regex panel.
- Ungreedy mode: `%UNGREEDY%` inserts the code you specified on the Modes tab to turn on the "lazy quantifiers" or the "greedy quantifiers" mode, as generated on the Regex panel.
- Skip zero-length matches mode: `%SKIPZEROLENGTH%` inserts the code you specified on the Modes tab to turn on the "skip zero-length matches" or the "allow zero-length matches" mode, as generated on the Regex panel.
- Split limit: `%SPLITLIMIT%` is the Split Limit entered on the Action panel.

- Splitting adds capturing groups mode: `%SPLITCAPTURE%` inserts the code you specified on the Modes tab to turn on the “splitting adds capturing groups” or the “splitting doesn’t add capturing groups” mode, as chosen via the Add Capturing Groups option on the Action panel.
- Splitting adds empty strings mode: `%SPLITEMPTY%` inserts the code you specified on the Modes tab to turn on the “splitting adds empty strings” or the “splitting doesn’t add empty strings” mode, as chosen via the Add Empty Strings option on the Action panel.
- Line break handling mode: `%LINEBREAK%` inserts the code you specified on the Modes tab in the “code to set the line break handling” group for the line break handling mode, as generated on the Regex panel.
- Number of capturing groups: `%GROUPCOUNT%` inserts a number indicating the total number of capturing groups in the regular expression, which can be zero. This is also the number assigned to the highest-numbered capturing group, if any.
- Number of capturing groups: `%GROUPCOUNTPLUS1%` inserts a number indicating the total number of capturing groups in the regular expression plus one. This is typically the length of a returned array that includes the overall regex match and all capturing groups.

You can also use conditionals to include certain code in your snippet only under certain circumstances. This is particularly useful to create snippets that call functions with optional parameters. For example, typing `“%IFSPLITLIMIT%, %SPLITLIMIT%”` will insert a comma, space and the split limit only when the split limit is not blank. The following conditionals are available:

- List of all modes: `%IFMODELIST%..%ENDIFMODELIST%` inserts its contents only when `%MODELIST%` is not blank. Note that this is not the same as all modes being turned off. The list of modes is blank whenever the code you specified (or left blank) on the Modes tab results in an empty string given the modes used by the regular expression.
- Case insensitive mode: `%IFCASELESS%..%ENDIFCASELESS%` inserts its contents if `%CASELESS%` is not blank. It will be blank if the regex is case insensitive and you did not specify any code for “case insensitive” on the Modes tab, or when the regex is case sensitive and you did not specify any code for “case sensitive” on the Modes tab. You can but don’t need to use `%CASELESS%` inside the conditional.
- Free-spacing mode: `%IFFREESPACING%..%ENDIFFREESPACING%` inserts its contents if `%FREESPACING%` is not blank.
- Dot matches newlines mode: `%IFDOTALL%..%ENDIFDOTALL%` inserts its contents if `%DOTALL%` is not blank.
- `^` and `$` match at line breaks mode: `%IFMULTILINE%..%ENDIFMULTILINE%` inserts its contents if `%MULTILINE%` not blank.
- Explicit capture mode: `%IFEXPLICITCAPTURE%..%ENDIFEXPLICITCAPTURE%` inserts its contents if `%EXPLICITCAPTURE%` is not blank.
- Duplicate capturing group names mode: `%IFNAMEDDUPLICATE%..%ENDIFNAMEDDUPLICATE%` inserts its contents if `%NAMEDDUPLICATE%` is not blank.
- Ungreedy mode: `%IFUNGREEDY%..%ENDIFUNGREEDY%` inserts its contents if `%UNGREEDY%` is not blank.
- Skip zero-length matches mode: `%IFSKIPZEROLENGTH%..%ENDIFSKIPZEROLENGTH%` inserts its contents if `%SKIPZEROLENGTH%` is not blank.
- Split limit: `%IFSPLITLIMIT%..%ENDIFSPLITLIMIT%` inserts its contents if the user specified a limit for splitting the string (and thus `%SPLITLIMIT%` is not blank).
- Split without limit: `%IFSPLITNOLIMIT%..%ENDIFSPLITNOLIMIT%` inserts its contents if the user did not specify a limit for splitting the string (and thus `%SPLITLIMIT%` is blank).
- Splitting adds capturing groups mode: `%IFSPLITCAPTURE%..%ENDIFSPLITCAPTURE%` inserts its contents if `%SPLITCAPTURE%` is not blank.

- Splitting adds empty strings mode: %IFSPLITEMPTY%..%ENDIFSPLITEMPTY% inserts its contents if %SPLITEMPTY% is not blank.
- Line break handling mode: %IFLINEBREAK%..%ENDIFLINEBREAK% inserts its contents if %LINEBREAK% is not blank.

The screenshot shows the 'RegexBuddy Source Code Template Editor - [Csharp]' window. The 'Functions' list on the left includes 'Iterate over all matches in a string', which is selected. The 'Selected function' panel on the right shows the function description 'Iterate over all matches in a string' and configuration options: Match (checked), Replace (unchecked), and Split (unchecked). Parameters are set as follows: Parameter 1: Regex object (regexObj); Parameter 2: Subject text (subjectString); Parameter 3: Match object (matchResults); Parameter 4: (empty). The 'Code snippet' area displays the following C# code:

```
try {
    Regex %PARAM1% = new Regex(%REGEX%MODELIST%);
    Match %PARAM3% = %PARAM1%.Match(%PARAM2%);
    while (%PARAM3%.Success) {
        // matched text: %PARAM3%.Value
        // match start: %PARAM3%.Index
        // match length: %PARAM3%.Length
        %PARAM3% = %PARAM3%.NextMatch();
    }
} catch (ArgumentException ex) {
    // Syntax error in the regular expression
}
```

The 'Preview' section shows the rendered C# code for the selected function:

```
try {
    Regex regexObj = new Regex(
        @"# 1. Literal text
        <b>
        # 2. Match anything
        [^\n\r<]+
        # 3. Literal text
        </b>",
        RegexOptions.IgnoreCase | RegexOptions.IgnorePatternl
    );
    Match matchResults = regexObj.Match(subjectString);
    while (matchResults.Success) {
        // matched text: matchResults.Value
        // match start: matchResults.Index
        // match length: matchResults.Length
        matchResults = matchResults.NextMatch();
    }
}
```

## 35. Library Panel Reference

On the Library panel you can store your RegexMagic formulas for later reuse. A RegexMagic formula consists of everything you specify on the Samples, Match, and Action panels. That is everything RegexMagic needs to generate your regular expression and replacement text.

In the layouts, you can access the Library panel by clicking on the Library tab. In the default layout, you can find that tab near the top of the RegexMagic window. In the dual monitor layout, the Library tab is on the floating panel on the second monitor.

Press **Alt+7** on the keyboard to move keyboard focus to the Library panel, and to make it visible if it was hidden. Or, select the Library item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.

### Storing RegexMagic Formulas in Libraries

When you have gone through the effort of providing samples and specifying what you want to match and what action you want to take with it to generate a regular expression, it would be a waste to use it just once. It makes a lot of sense to store the RegexMagic formulas you create for later reuse, by yourself or by friends and colleagues.

A RegexMagic formula consists of the samples that you provide on the Samples panel, the fields and patterns that you define on the Match panel, and the capturing groups and replacements that you specify on the Action panel. Everything shown on these three panels is always saved together as one formula.

To save a RegexMagic formula, switch to the Library panel in RegexMagic. If you want to store it into an existing library, click the Open button to select the RegexMagic Library file. Or click the small downward pointing arrow on the Open button to reopen a library that you recently worked with.

Any changes you make to a RegexMagic Library are saved automatically. RegexMagic makes sure that you never lose any patterns you worked so hard on. If you open the same library in more than one instance of RegexMagic, RegexMagic makes sure all the instances are kept in sync, by automatically saving and reloading the library.

In other words: you don't have to worry at all about making sure your work is saved. As a precaution against inadvertent modifications, all libraries are opened as "read only" by default. You can toggle the read only state with the check box just below the New and Open buttons on the Library tab's toolbar.

One exception is the sample RegexMagic library that is included with RegexMagic. When you open it, RegexMagic will turn on the "read only" state permanently. The reason is that when you upgrade RegexMagic, the sample library will be upgraded too. If you want to edit the sample library, click the Save As button to make a copy.

Since RegexMagic automatically and regularly saves libraries, there is no button or command for you to do so manually. If you want to edit a library, and preserve the original for safekeeping, click the Save As button before turning off the read only option. Save As tells RegexMagic to create a copy of the library under a new name. Any subsequent changes are automatically saved into the new file. The original file stays the way it is.

## Adding a Formula to The Library

First, you need to use the Samples, Match, and Action panels to create your RegexMagic formula. Then, click on the Library panel's Add button. If the button is disabled, you forgot to turn off the "read only" checkbox on the Library panel.

After you add the formula to the library, it appears at the right hand side of the Library panel. The middle area displays a summary of the Match and Action settings. The bottom area shows a drop-down list with all the samples in the formula, and the full text of the selected sample.

Don't forget to type a clear description in the edit box just below the Library tab's toolbar. You can enter as much text as you want. A good description will greatly help to look up the regex later. The first line of the description appears in the list at the left. The list at the left shows all the formulas in the library, alphabetically sorted by their descriptions. If a regex does not have a description, it is listed as (untitled).

## Updating Regex Actions in The Library

You cannot directly edit regular expressions and test data stored in the library. To change a regular expression, click the Update button and select to update the regular expression. This will replace the library item's regular expression with the one you're currently editing in RegexMagic. Click the Update button and select to update the sample to replace the library item's sample with the text currently on the Test tab. Both updates will leave the description unchanged.

## Copying and Pasting Formulas Between Libraries

Each instance of RegexMagic can open only a single library. If you want to move or duplicate regex actions between two libraries, start a second instance of RegexMagic. Then use the Cut, Copy and Paste buttons on the Library toolbar to move or copy a formula from one instance of RegexMagic to the other.

If you open the same library in more than once RegexMagic instance, both instances are automatically kept in sync. If you make changes in one instance, they automatically appear in the other instance when you switch to it.

## Using a Formula from a Library

On the Library panel, click the the Open button to open the RegexMagic library from which you want to reuse a RegexMagic formula. Or click the small downward pointing arrow on the Open button to reopen a library that you recently worked with. If this is the first time you are using RegexMagic's Library tab, the list of libraries that you recently worked with will list the standard RegexMagic library that is included with RegexMagic itself. This library contains a wide range of samples that you can use to learn how RegexMagic works, and adapt for your own purposes.

To use a formula from the library, select it from the list at the left, and click the Use button. This replaces everything on the Samples, Match, and Action panels with what's in the formula you selected in the library. You can then immediately generate the formula's regular expression, or you can adapt the formula first and then generate your regular expression.

## 36. GREP Panel Reference

On the GREP panel you can search (and replace) through large numbers of files and folders using the regular expression (and replacement text) that RegexMagic generated.

In all layouts, you can access the GREP panel by clicking on the GREP tab. In the default layout, you can find that tab near the top of the RegexMagic window. In the dual monitor layout, the GREP tab is on the floating panel on the second monitor.

Press **Alt+8** on the keyboard to move keyboard focus to the GREP panel, and to make it visible if it was hidden. Or, select the GREP item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.

### GREP: Search and Replace through Files and Folders

The GREP panel enables you to search through any number of files and folders using the regular expression you generated. If you defined a replace action, you can use it to search-and-replace through files and folders. This way you can test your regular expression on a larger number of files (rather than just a few samples on the Samples panel), and even perform actual search or file maintenance tasks.

Type the folder you want to search through in the Folders field, or click the ellipsis button next to it to select the folder from a folder tree. If you want to search through multiple folders, simply enter them all delimited by semicolons. Turn on “recurse subfolders” to make RegexMagic search through all subfolders of the folder(s) you specified.

If you don’t want to search through all files in the folder, you can restrict the search using file masks. In a file mask, the asterisk (\*) represents any number (including none) of any character, similar to `.*` in a regular expression. The question mark (?) represents one single character, similar to `.` in a regular expression. E.g. the file mask `*.txt` tells RegexMagic to search through any file with a `.txt` extension. You can delimit multiple masks with semicolons. To search through all C source and header files, use `*.c;*.h`.

File masks also support a simple character class notation, which matches one character from a list or a range of characters. E.g. to search through all web logs from September 2008, use a file mask such as `www.200809[0123][0-9].log` or `www.200809??.log`.

Sometimes it is easier to specify what you do not want to search through. To do so, mark the “invert mask” option. With this option, a file mask of `*.c;*.h` tells RegexMagic to search through all files except C source and header files.

By default, RegexMagic searches through a whole file at once, and allows regex matches to span multiple lines. Traditional grep tools, however, search through files line by line. You can make RegexMagic do the same by turning on the “line-based” option. When grepping line by line, you can turn on “invert results” to make RegexMagic list all lines in each file which the regular expression does *not* match.

## Target and Backup Files

When searching, RegexMagic does not modify any files by default. The search matches are simply displayed on screen. To save all the search matches into a single file, select the “save results into a single file” option. Click the ellipsis button to the far right of the target setting to select the file you want to save the results into. To create one file for each file searched through, select “save one file for each searched file”. Click the ellipsis button to select the folder to save the files into. Each saved file will have the same name as the file searched through.

When executing a search-and-replace, RegexMagic can either modify the files it searched through, or create copies of those files and modify the copies. If you select to “copy only modified files” or “copy all searched files”, click the ellipsis button to select the folder to save the target files into. Again, each target file will have the same name as the original.

RegexMagic will not prompt you when the GREP action overwrites one or more files. Therefore, you should make sure the backup option is set to anything except “no backups”. The “single .bak” and “single .~??” options create a backup file for each target file that is overwritten. If the backup file already existed, the previous backup is lost. The “multi .bak” and “multi backup N of” options create numbered backup files without overwriting existing backup files. The “same file name” option keeps one backup for each target file, and gives the backup copy the same name. The “hidden history” option creates a hidden folder named “\_\_history” in each folder where files are overwritten which keeps backups with numbered extensions.

The “same file name” backup option requires you to specify a folder into which backup files should be stored. The other backup options except “hidden history” allow you to choose. If you specify a backup folder, all backups will be stored in that folder. If not, backups are saved in the same folders as the target files that were overwritten. Keeping backups in a separate folder makes it easy to get rid of the backups later, but makes it harder to find a backup file if you want to restore a file manually.

## Preview, Execute and Quick Execute

RegexMagic can GREP in three modes: preview, execute and quick execute. You can access these modes through the GREP button. A preview displays detailed results in RegexMagic, without modifying any files. Execute an action to create or modify target files as you specified, and to see detailed results afterwards to verify the results.

Use “quick execute” when you know the GREP action will do what you want, to quickly update files without displaying detailed results in RegexMagic. The “quick execute” option can be much faster than “execute” when performing a search-and-replace through large numbers of files, since it doesn’t require RegexMagic to keep track of each individual search match. On a typical computer, “preview” and “execute” can handle about 100,000 search matches, while “quick execute” has no practical limits.

## Backup Files and Undo

If you followed my advice to have RegexMagic create backup files, you can instantly undo the effects of a GREP action. Click on the GREP button, and then select Undo from the menu. This will replace all files that were overwritten with their backup copies. The backups are removed in the process. Only the very last action

can be undone by RegexMagic, and only if you do not close RegexMagic. If you have multiple instances of RegexMagic open, each instance can undo its own very last GREP action.

If you've verified the results, and all target files are in order, click the GREP button and select Delete Backup Files to delete the backup files created by the very last GREP action.

## Opening and Saving Actions and Results

The Clear button clears the GREP action and results. It is not necessary to click the Clear button before starting a new action, but it may make things easier for you by reducing clutter.

Click the Open button to open a GREP action previously saved with the Save button. Note that only the action (regular expression with folder, mask, target and backup settings) is saved into rmg files.

To save the results of an action, click the Export button. When prompted, enter a file name with a .txt or .html extension. If you enter a .html extension, the results are saved into an HTML file. When you view the HTML file in a web browser, it will look just like the results are displayed in RegexMagic. If you enter any other extension, the results are saved as plain text. If you want to save search results without additional information such as the number of matches, set the Target option to save the results to file *before* executing the action. Target files only contain search matches, one on each line.

## PowerGREP: The Ultimate GREP for Windows

Since RegexMagic's focus is on creating and testing regular expressions, it only offers basic GREP functionality. While certainly useful, you may want to invest in a powerful GREP tool such as PowerGREP. PowerGREP can search using any number of regexes at once, post-process replacement texts during search-and-replace, arbitrarily section files using an extra set of regular expressions, search through proprietary file formats such as PDF, MS Word and Excel, etc. PowerGREP also keeps a permanent undo history, a feature that can save your day all by itself.

## 37. Share Experiences and Get Help on The User Forums

If you are a licensed RegexMagic user, you have instant access to the RegexMagic online forums. Obviously, your computer will need to be connected to the Internet.

In the layouts, you can access the Forum panel by clicking on the Forum tab. In the default layout, you can find that tab near the top of the RegexMagic window. In the dual monitor layout, the Forum tab is on the floating panel on the second monitor.

Press **Alt+9** on the keyboard to move keyboard focus to the Forum panel, and to make it visible if it was hidden. Or, select the Forum item in the View menu. You can access that menu by the View button on the Assistant panel, and by right-clicking on the caption or tab of any panel in RegexMagic.

### Logging In

When you click the Login button you will be asked for a name and email address. The name you enter is what others will see when you post a message to the forum. It is polite to enter your real, full name. The forums are private, friendly and spam-free, so there's no need to hide behind a pseudonym. While you can use an anonymous handle, you'll find that people (other RegexMagic users) are more willing to help you if you let them know who you are. Support staff from Just Great Software will answer technical support questions anyhow.

The email address you enter is used to email you whenever others participate in one of your discussions. The email address is never displayed to anyone, and will never be used for anything other than the automatic notifications. RegexMagic's forum system does not have a function to respond privately to a message. If you don't want to receive automatic email notifications, there's no need to enter an email address.

If you select "never email replies", you'll never get any email. If you select "email replies to conversations you start", you'll get an email whenever somebody replies to a conversation that you started. If you select "email replies to conversations that you participate in", you'll get an email whenever somebody replies to a conversation that you started or replied to. The From address on the email notifications will be [forums@jgsoft.com](mailto:forums@jgsoft.com). You can filter the messages based on this address in your email software.

RegexMagic's forum system uses the standard HTTP protocol which is also used for regular web browsing. If your computer is behind an HTTP proxy, click the Proxy button to configure the proxy connection.

If you prefer to be notified of new messages via an RSS feed instead of email, log in first. After RegexMagic has connected to the forums, you can click the Feeds button to select RSS feeds that you can add to your favorite feed reader.

### Various Forums

Below the Login button, there's a drop-down list where you can select which particular forum you want to participate in. The "Just Great Software news" forum is read-only. Announcements about new RegexMagic releases and other Just Great Software products will appear there.

The “RegexMagic discussion” forum is for discussing anything related to the RegexMagic software itself. This is the place for technical support questions, feature requests and other feedback regarding the functionality and use of RegexMagic.

The “regular expressions” forum is for discussing regular expressions in general. Here you can talk about creating regular expressions for particular tasks, and exchange ideas on how to implement regular expressions with whatever application or programming language you work with.

## Searching The Forums

Before starting a new conversation, please check first if there’s already a conversation going on about your topic. In the top right corner of the Forum panel, there is a box on the toolbar that you can use to search for messages. When you type something into that box, only conversations that include at least one message containing the word or phrase you typed in will be shown. The filtering happens in real time as you enter your word or phrase.

Note that you can enter only one search term, which will be searched for literally. If you type “find me”, only conversations containing the two words “find me” next to each other and in that order will be shown. You cannot use boolean operators like “or” or “and”. Since the filtering is instant, you can quickly try various keywords.

If you find a conversation about your subject, start with reading all the messages in that conversation. If you have any further comments or questions on that conversation, reply to the existing conversation instead of starting a new one. That way, the thread of the conversation will stay together, and others can instantly see what you’re talking about. It doesn’t matter if the conversation is a year old. If you reply to it, it will move to the top automatically.

## Conversations and Messages

The left hand half of the Forum pane shows two lists. The one at the top shows conversations. The bottom one shows the messages in the selected conversation. You can change the order of the conversations and messages by clicking on the column headers in the lists. A conversation talks about one specific topic. In other forums, a conversation is sometimes called a thread.

If you want to talk about a topic that doesn’t have a conversation yet, click the New button to start a new conversation. A new entry will appear in the list of conversations with an edit box. Enter a brief subject for your conversation (up to 100 characters) and press Enter. Please write a clear subject such as “scraping an HTML table in Perl” rather than “need help with HTML” or just “help”. A clear subject significantly increases the odds that somebody who knows the answer will actually click on your conversation, read your question and reply. A generic scream for help only gives the impression you’re too lazy to enter a clear subject, and most forum users don’t like helping lazy people.

After typing in your subject and pressing Enter, the keyboard focus will move to the box empty box where you can enter the body text of your message. Please try to be as clear and descriptive as you can. The more information you provide, the more likely you’ll get a timely and accurate answer. If your question is about a particular regular expression, don’t forget to attach your regular expression or test data. Use the forum’s attachment system rather than copying and pasting stuff into your message text.

If you want to reply to an existing conversation, select the conversation and click the Reply button. It doesn't matter which message in the conversation you selected. Replies are always to the whole conversation rather than to a particular message in a conversation. RegexMagic doesn't thread messages like newsgroup software tends to do. This prevents conversations from veering off-topic. If you want to respond to somebody and bring up a different subject, you can start a new conversation, and mention the new conversation in a short reply to the old one.

When starting a reply, a new entry will appear in the list of messages. Enter a summary of your reply (up to 100 characters) and press Enter. Then you can enter the full text of your reply, just like when you start a new conversation. However, doing so is optional. If your reply is very brief, simply leave the message body blank. When you send a reply without any body text, the forum system will use the summary as the body text, and automatically prepends [nt] to your summary. The [nt] is an abbreviation for "no text", meaning the summary is all there is. If you see [nt] on a reply, you don't need to click on it to see the rest of the message. This way you can quickly respond with "Thank you" or "You're welcome" and other brief courtesy messages that are often sadly absent from online communication.

When you're done with your message, click the Send button to publish it. There's no need to hurry clicking the Send button. RegexMagic will forever keep all your messages in progress, even if you close and restart RegexMagic, or refresh the forums. Sometimes it's a good idea to sleep on a reply if the discussion gets a little heated. You can have as many draft conversations and replies as you want. You can read other messages while composing your reply. If you're replying to a long question, you can switch between the message with the question and your reply while you're writing.

## **Directly Attach Regexes, Test Data, etc.**

One of the greatest benefits of RegexMagic's built-in forums is that you can directly attach regular expressions, test data, regex libraries, source code templates and GREP actions. Simply click the Attach button and select the item you want to attach. You can add the same item more than once. E.g. to add two regular expressions, click on the first one in the History, click the Attach button and choose Regular Expression. Then click on the second regex and click Attach, Regular Expression again.

To attach a screen shot, press the Print Screen button on the keyboard to capture your whole desktop. Or, press Alt+Print Screen to just capture the active window (e.g. RegexMagic's window). Then switch to the Forum tab, click the Attach button, and select Clipboard. You can also attach text you copied to the clipboard this way.

It's best to add your attachments while you're still composing your message. The attachments will appear with the message, but won't be uploaded until you click the Send button to post your message. If you add an attachment to a message you've written previously, it will be uploaded immediately. You cannot attach anything to messages written by others. Write your own reply, and attach your data to that.

To check out an attachment uploaded by somebody else, click the Use or Save button. The Use button will load the attachment directly into RegexMagic. This may replace your own data. E.g. the Test tab can hold only one sample, so using an attachment with test data will replace whatever you typed or loaded into the Test tab. If you click the Save button, RegexMagic will prompt for a location to save the attachment. RegexMagic will not automatically open attachments you save.

RegexMagic automatically compresses attachments in memory before uploading them. So if you want to attach an external file, there's no need to compress it using a zip program first. If you compress the file

manually, everybody who wants to open it will have to decompress it manually. If you let RegexMagic compress it automatically, decompression will also be automatic.

## Taking Back Your Words

If you regret anything you wrote, simply delete it. There are three Delete buttons. The one above the list of conversations deletes the whole conversation. You can only delete a conversation if nobody else participated in it. The Delete button above the edit box for the message body deletes that message, if you wrote it. The Delete button above the list of attachments deletes the selected attachment, if it belongs to a message that you wrote.

If somebody already downloaded your message before you got around to deleting it, it won't vanish magically. The message will disappear from their view of the forums the next time they log onto the forums or click Refresh. If you see messages disappear when you refresh your own view of the forums, that means the author of the message deleted it. If you replied to a conversation and the original question disappears, leaving your reply as the only message, you should delete your reply too. Otherwise, your reply will look silly all by itself. When you delete the last reply to a conversation, the conversation itself is also deleted, whether you started it or not.

## Changing Your Opinion

If you think you could better phrase something you wrote earlier, select the message and then click the Edit button above the message text. You can then edit the subject and/or body text of the message. Click the Send button to publish the edited message. It will replace the original. If you change your mind about editing the message, click the Delete button. Make sure to click it only once! When editing a message, clicking Delete will revert the message to what it was before you started editing it. If you click Delete a second time (i.e. while the message is no longer being edited), you'll delete the message from the forum.

If other people have already downloaded your message, their view of the message will magically change when they click Refresh or log in again. Since things may get confusing if people respond to your original message before they see the edited message, it's best to restrict your edits to minor errors like spelling mistakes. If you change your opinion, click the Reply button to add a new message to the same conversation.

## Updating Your View

When you click the Login button, RegexMagic automatically downloads all new conversations and message summaries. Message bodies are downloaded one conversation at a time as you click on the conversations. Attachments are downloaded individually when you click the Use or Save button.

RegexMagic keeps a cache of conversations and messages that persists when you close RegexMagic. Attachments are cached while RegexMagic is running, and discarded when you close RegexMagic. By caching conversations and messages, RegexMagic improves the responsiveness of the forum while reducing the stress on the forum server.

If you keep RegexMagic running for a long time, RegexMagic will not automatically check for new conversations and messages. To do so, click the Refresh button.

Whenever you click Login or Refresh, all conversations and messages are marked as “read”. They won’t have any special indicator in the list of conversations or messages. If the refresh downloads new conversation and message summaries, those will be marked “unread”. This is indicated with the same “people” icon as shown next to the Login button.

## 38. Forum RSS Feeds

When you're connected to the user forum, you can click the Feeds button to select RSS feeds that you can add to your favorite feed reader. This way, you can follow RegexMagic's discussion forums as part of your regular reading, without having to start RegexMagic. To participate in the discussions, simply click on a link in the RSS feed. All links in RegexMagic's RSS feeds will start RegexMagic and present the forum login screen. After you log in, wait a few moments for RegexMagic to download the latest conversations. RegexMagic will automatically select the conversation or message that the link points to. If RegexMagic was already running and you were already logged onto the forums, the conversation or message that the link points to is selected immediately.

You can choose which conversations should be included in the RSS feed:

- All conversations in all groups: show all conversations in all the discussion groups that you can access in RegexMagic.
- All conversations in the selected group: show the list of conversations that RegexMagic is presently showing on the Forum tab.
- All conversations you participated in: show all conversations that you started or replied to in all the discussion groups that you can access in RegexMagic.
- All conversations you started: show all conversations that you started in all the discussion groups that you can access in RegexMagic.
- Only the selected conversation: show only the conversation that you are presently reading on the Forum tab in RegexMagic.

In addition, you can choose how the conversations that you want in your RSS feed should be arranged into items or entries in the feed:

- One item per group, with a list of conversations: Entries link to groups as a whole. The entry titles show the names of the groups. The text of each entry shows a list of conversation subjects and dates. You can click the subjects to participate in the conversation in RegexMagic. Choose this option if you prefer to read discussions in RegexMagic itself (with instant access to attachments), and you only want your RSS reader to tell you if there's anything new to be read.
- One item per conversation, without messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the date the conversation was started and last replied to. If your feed has conversations from multiple groups, those will be mixed among each other.
- One item per conversation, with a list of messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the list of replies with their summaries, author names, and dates. You can click a message summary to read the message in RegexMagic. If your feed has conversations from multiple groups, those will be mixed among each other.
- One item per conversation, with a list of messages: Entries link to conversations. The entry titles show the subjects of the conversations. The entry text shows the list of replies, each with their full text. If your feed has conversations from multiple groups, those will be mixed among each other. This is the best option if you want to read full discussions in your RSS reader.
- One item per message with its full text: Entries link to messages (responses to conversations). The entry titles show the message summary. The entry text shows the full text of the reply, and the conversation subject that you can click on to open the conversation in RegexMagic. If your feed lists multiple conversations, replies to different conversations are mixed among each other. Choose this option if you want to read full discussions in your RSS reader, but your RSS reader does not mark old entries as unread when they are updated in the RSS feed.

## 39. Change RegexMagic's Appearance

RegexMagic's interface is completely modular. The application consists of nine panels. You can freely arrange all nine panels to best suit the way you like to work with RegexMagic. You can even hide the ones you don't use.

You can activate each of the panels through the View menu, whether you closed the panel or not. You can access the View menu through the View toolbar button on the toolbar at the top. If you closed the Assistant panel, you can access its toolbar buttons, including the View menu, by right-clicking on the tab or caption of any of the panels in RegexMagic.

If you like to keep your hands on the keyboard, you can press Alt+1 through Alt+9 to move keyboard focus to each of the nine panels. If the panel wasn't visible, it will be shown. The View menu reminds you of the number of each panel.

The default layout is optimized for a computer with a single monitor that has an average screen resolution. If your computer has a single large resolution monitor, you can make use of the additional space by docking side-by-side some of the panels that are tabbed by default. If your computer has more than one monitor, take advantage of both monitors by making one or more panels float. Then drag the floating panels off to the second monitor.

### How to Rearrange RegexMagic's Panels

To move a panel, use the mouse to drag and drop its caption bar (for a panel docked to the side, or a floating panel) or its tab (for a tabbed panel). While you drag the panel, a thick gray rectangle moves around while you move the mouse. This rectangle indicates the position and size the panel will occupy if you were to release the mouse button.

To dock a panel to the side, move the mouse pointer near the edge while dragging the panel. You can dock panels to the side of the entire RegexMagic window. You can also dock panels to the side of other panels, and even to the side of a panel that is inside a tab. While dragging a panel, moving the mouse pointer a few pixels closer to or away from the edge of RegexMagic's window makes the difference between docking the panel to the side of RegexMagic itself, or to the side of the panel that touches that edge of RegexMagic's window.

To arrange two panels inside a tabbed container, drag one panel and move the mouse pointer to the center of the other panel. The gray rectangle's shape will change slightly, showing an extension at the top that looks like an invisible tab.

To make a panel float freely, drag it away from RegexMagic or simply double-click its caption or tab. Floating a panel is very useful if your computer has more than one monitor. Move the floating panel to your second monitor to take full advantage of your multi-monitor system. If you drag a second panel onto the floating panel, you can dock both panels together in a single floating container. This way you can conveniently display several panels on the second monitor.

## Assistant

Show the RegexMagic Assistant. In the default view, the RegexMagic Assistant is visible along the right hand side of the RegexMagic window. The assistant displays helpful hints as well as error messages while you work with RegexMagic.

## Samples

Show the Samples panel, where you provide sample text to base your regular expression on, and to set the regular expression against. In the default view, the Samples panel shares the top left tab with the Match and Action panels.

## Match

Show the Match panel, where you tell RegexMagic which parts of the text you want your regular expression to match, and what that text should look like or which patterns that text should fit. In the default view, the Match panel shares the top left tab with the Samples and Action panels. Depending on which panel you last used, that tab will say “Samples, Match” or “Samples, Action”. Click that tab, and then on the Match tab inside it, to activate the Match panel in the default layout.

## Action

Show the Action panel, where you tell RegexMagic which action you plan to take with your regular expression. You can collect parts of the text matched by the regular expression, or replace them with different parts or with new text. The settings on the Action panel determine the replacement text that is generated, if any. In the default view, the Action panel shares the top left tab with the Samples and Match panels. Depending on which panel you last used, that tab will say “Samples, Match” or “Samples, Action”. Click that tab, and then on the Action tab inside it, to activate the Action panel in the default layout.

## Regex

Show the Regex panel, where you have RegexMagic generate your regular expression, according to your specifications on the Match and Action panels. In the default view, the Regex panel is docked together under a tab with the Use panel. Click the “Regex, Use” tab near the top of the RegexMagic window to activate the Regex panel.

## Use

Show the Use panel, where you can generate a source code snippet in your favorite programming language to perform a task of your choice with the regular expression generated by RegexMagic. In the default view, the Use panel is docked together under a tab with the Regex panel. Click the “Regex, Use” tab near the top of the RegexMagic window to activate the Use panel.

## Library

Show the Library panel, where you can store your RegexMagic formulas for later reuse. A RegexMagic formula consists of everything you specify on the Samples, Match, and Action panels. That is everything RegexMagic needs to generate your regular expression and replacement text. In the default view, you can access the Library panel by clicking on the Library tab near the top of the RegexMagic window.

## GREP

Show the GREP panel, where you can search (and replace) through large numbers of files and folders using the regular expression that RegexMagic generated. In the default view, you can access the GREP panel by clicking on the GREP tab near the top of the RegexMagic window.

## Forum

Show the Forum panel, where you can share your experiences with other RegexMagic users, and get help. In the default view, you can access the Forum panel by clicking on the Forum tab near the top of the RegexMagic window.

## Large Toolbar Icons

If you have a high resolution monitor, the icons on RegexMagic's various toolbars may be too small to discern properly. Select Large Toolbar Icons in the View menu to make them 50% larger. Select the same command to restore the toolbar icons to their default size.

## Office 2003 Display Style

If you find RegexMagic's looks a bit bland, select Office 2003 Display Style from the View menu to make RegexMagic mimic the looks of Microsoft Office 2003. This will make RegexMagic rather colorful. Select the item again to restore the default looks. On Windows XP, the default looks will use the Windows XP theme you selected in your computer's display settings.

## Restore Default Layout

Use the Restore Default Layout item in the View menu to quickly reset the RegexMagic window to its default layout, with the Assistant docked to the right, and one row of tabs at the top. The first tab docks the Samples, Match, and Action panels. The second tab docks the Regex and Use panels. The other panels each have a tab of their own.

This layout keeps each panel sufficiently large to be workable on a computer with a single medium resolution monitor.

## Side by Side Layout

The side by side layout docks the Samples panel at the top, and the Regex and Assistant panels at the right hand side. These three panels are permanently visible. The other panels are arranged in a row of tabs below the Samples panel. The Match and Action panels are docked side by side under the first tab, while the other panels all have tabs of their own.

If you have a large resolution wide screen monitor, use this layout to work more comfortably by keeping the most commonly used panels permanently visible.

## Dual Monitor Tabbed Layout

This option is only available if your computer has more than one monitor. Use this layout if your computer has two average resolution monitors to take advantage of the second monitor. RegexMagic's main window has the Samples panel at the top, and the Regex and Assistant panels at the right hand side. These three panels are permanently visible. The Match and Action panels are tabbed together in the remaining corner.

The floating window on the second monitor shows one row of tabs. The first tab docks the Regex and Use panels together. The other panels each have a tab of their own. It arranges the Action, Library and Undo History panel in tabs, with the File Selector and Assistant docked to the left and the bottom.

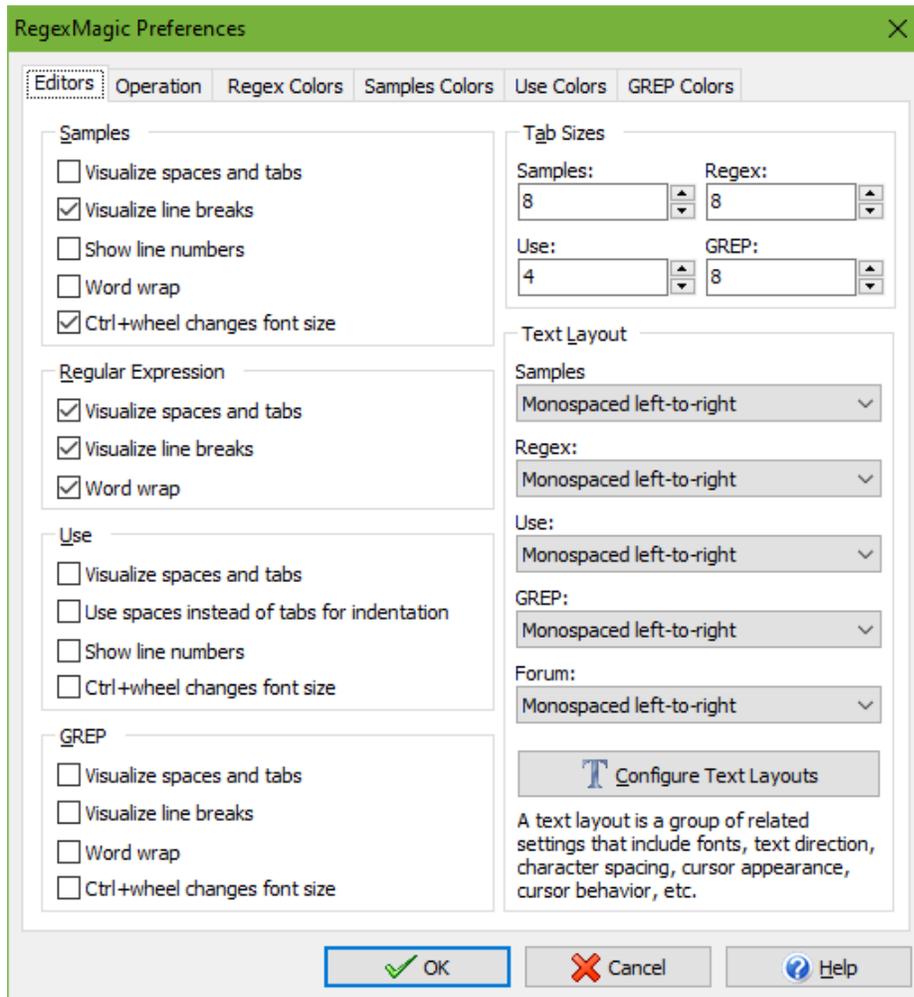
## Dual Monitor Side by Side Layout

This option is only available if your computer has more than one monitor. Use this layout if your computer has two high resolution wide screen monitors to put your computer's screen size to maximum use. It arranges the Samples, Assistant, Match, and Action panels side by side in RegexMagic's main window. Those four panels are permanently visible. The floating window on the second monitor has the Regex panel permanently visible in the top left corner. The bottom left corner has two tabs for the Use and GREP panels, and the right hand side has two tabs for the Library and Forum panels.

## 40. Adjust RegexMagic to Your Preferences

Click the Preferences button on the toolbar at the top to adjust RegexMagic to your tastes and habits.

### Editors



**Samples:** Configures the edit box on the Samples panel. You can choose whether spaces and tabs should be indicated by small dots and >> signs, and whether line breaks should be indicated by paragraph symbols. You can also specify if line numbers should be shown in the left margin, and whether long lines should wrap at the edge of the edit box instead of requiring horizontal scrolling.

**Regular Expression:** Configures the edit boxes for entering the regular expression and the replacement text on the Regex panel. You can choose whether spaces and tabs should be indicated by small dots and >> signs, and whether line breaks should be indicated by paragraph symbols.

**Use:** Configures the edit boxes on the Use panel. You can choose whether spaces and tabs should be indicated by small dots and >> signs, whether the tab key should insert spaces or tabs, and whether line numbers should be shown in the left margin.

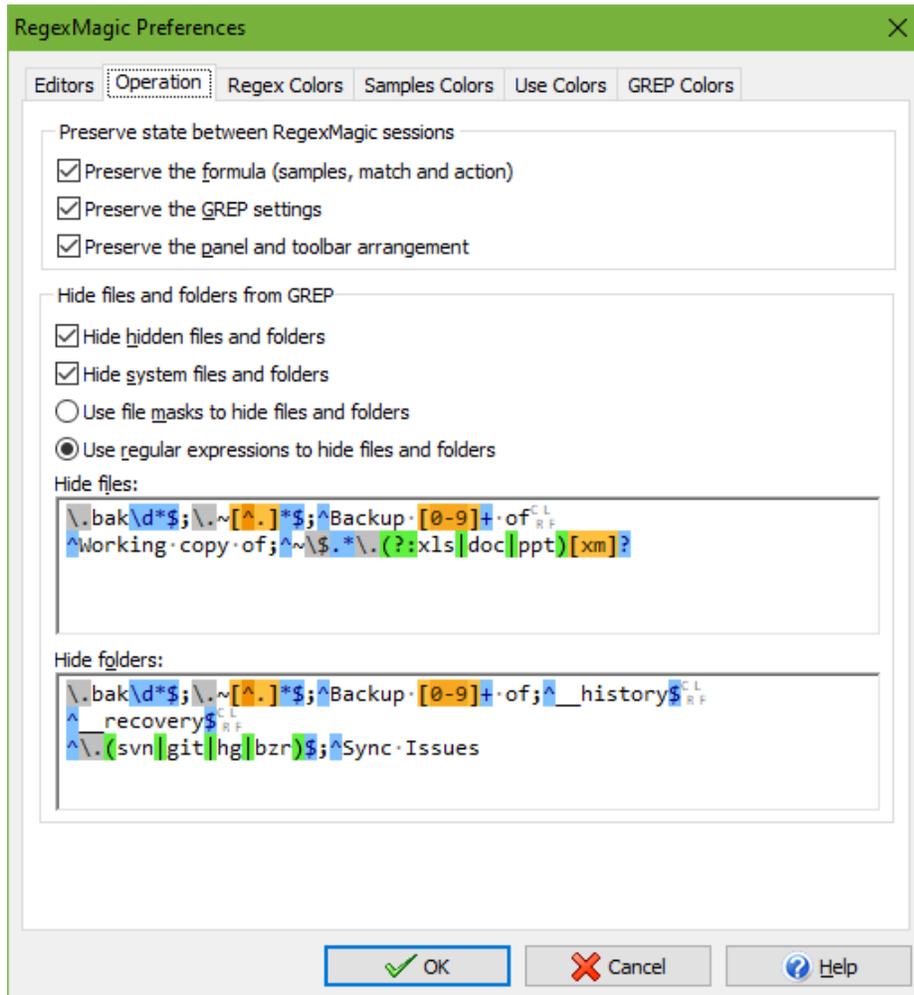
**GREP:** Configures the results display on the GREP panel. You can choose whether spaces and tabs should be indicated by small dots and >> signs, and whether line breaks should be indicated by paragraph symbols. You can also specify whether long lines should wrap at the edge of the edit box instead of requiring horizontal scrolling.

**Tab Sizes:** Specify the width in spaces of tabs in the edit boxes on the Samples, Regex, Use, and GREP panels.

**Text Layout:** In RegexMagic, a “text layout” is a combination of settings that control how text is displayed and how the text cursor navigates through that text. The settings include the font, text direction, text cursor behavior, text cursor appearance, which characters are word characters, and how the text should be spaced. You can select up to four different text layouts that are used for various parts of RegexMagic. The Regex text layout is used for edit boxes for regular expressions and replacement texts. The Convert text layout is used for the warning and error messages on the Convert panel. The font from the Convert text layout is also used for the regex tree on the Create panel. The editors on the Test, Debug, Use, and GREP panels all use the same text layout. You’ll likely want to select a monospaced text layout for this one, so that columns line up correctly. The Forum text layout is used to edit messages on the built-in user forum.

You can select predefined text layouts from the drop-down lists. Or, you can click the Configure Text Layouts button to show the Text Layout Configuration window to customize the text layouts.

## Operation



**Preserve state between RegexMagic sessions:** With these checkboxes you can choose what RegexMagic should remember when you close it and restart it. Turning these on allows you to continue working where you left off. Turning these off allows you to instantly start with a clean slate.

**Hide files and folders from GREP:** The options “hide hidden files and folders” and “hide system files and folders” are turned on by default. These make all files and folders that have the hidden or system attribute set invisible to the GREP panel. Turn these options off if you want to grep through hidden and system files and folders.

Choose “use file masks to hide files and folders” to use traditional wildcard file masks to hide files and folders from the GREP panel. Choose “use regular expressions to hide files and folders” to use regular expressions to hide files and folders. Wildcards have to match the entire name of the file or folder to hide it. Regular expressions hide any file or folder with a name matched by the regex, even if the name is only matched partially. The regular expression “joe”, for example, hides all files that contain “joe” anywhere in the file name. The equivalent file mask using wildcards is “\*joe\*”. When using wildcards, “joe” hides files exactly named “joe” only.

In the “hide files” box you can enter a semicolon-delimited list of file masks or regular expressions. All files of which the names match one of these file masks are automatically skipped by the GREP panel. By default, the permanent exclusion masks are set to a regular expression that matches all backup files created by RegexBuddy, as well as temporary working copies of files created by EditPad Pro and MS Office. If you’d ever need to search through backup files or working copies, you will need to adjust the permanent exclusion file masks first.

The “hide folders” box works in the same way, except that the file masks you specify are tested against individual folder names. If the name of a folder matches one of these file masks then the GREP panel automatically skips that folder and any files or folders inside that folder. The default settings exclude `__history`, which is the folder name that RegexBuddy uses for backup files when you select the “hidden history” backup option. Also excluded by default is `.svn` which is the folder name that Subversion uses to cache data in each folder that you have added to version control.

## Regex Colors

Configure the colors used to highlight different parts of the regular expression and replacement text on the Regex panel. You can select one of the preset configurations at the top of the screen. To customize the colors, click on an item in the list of individual colors. Then click the Background Color and Text Color buttons to change the item’s colors. You can also change the font style with the bold, italic and underline checkboxes. At the bottom, a sample edit box will show what the color configuration looks like. You can edit the text in the example box to further test the colors.

## Sample Colors

Configure the colors used by the Samples panel to highlight marked and matched fields. You can select one of the preset configurations at the top of the screen. To customize the colors, click on an item in the list of individual colors. Then click the Background Color and Text Color buttons to change the item’s colors. You can also change the font style with the bold, italic and underline checkboxes.

At the bottom, a sample edit box will show what the color configuration looks like. RegexMagic supports up to 99 fields. Fields 11, 21, 31, etc. are highlighted with the color “field 1”, fields 12, 22, 32, etc. with “field 2”, and so on. The color will be used as is, dimmed, or underlined depending on whether the field was matched by the generated regular expression, marked manually, or both.

## Use Colors

Configure the colors used by syntax coloring of the code snippets on the Use panel. You can select one of the preset configurations at the top of the screen. To customize the colors, click on an item in the list of individual colors. Then click the Background Color and Text Color buttons to change the item’s colors. You can also change the font style with the bold, italic and underline checkboxes. At the bottom, a sample edit box will show what the color configuration looks like. You can see the effect for several different programming languages. You can edit the text in the example box to further test the colors.

## **GREP Colors**

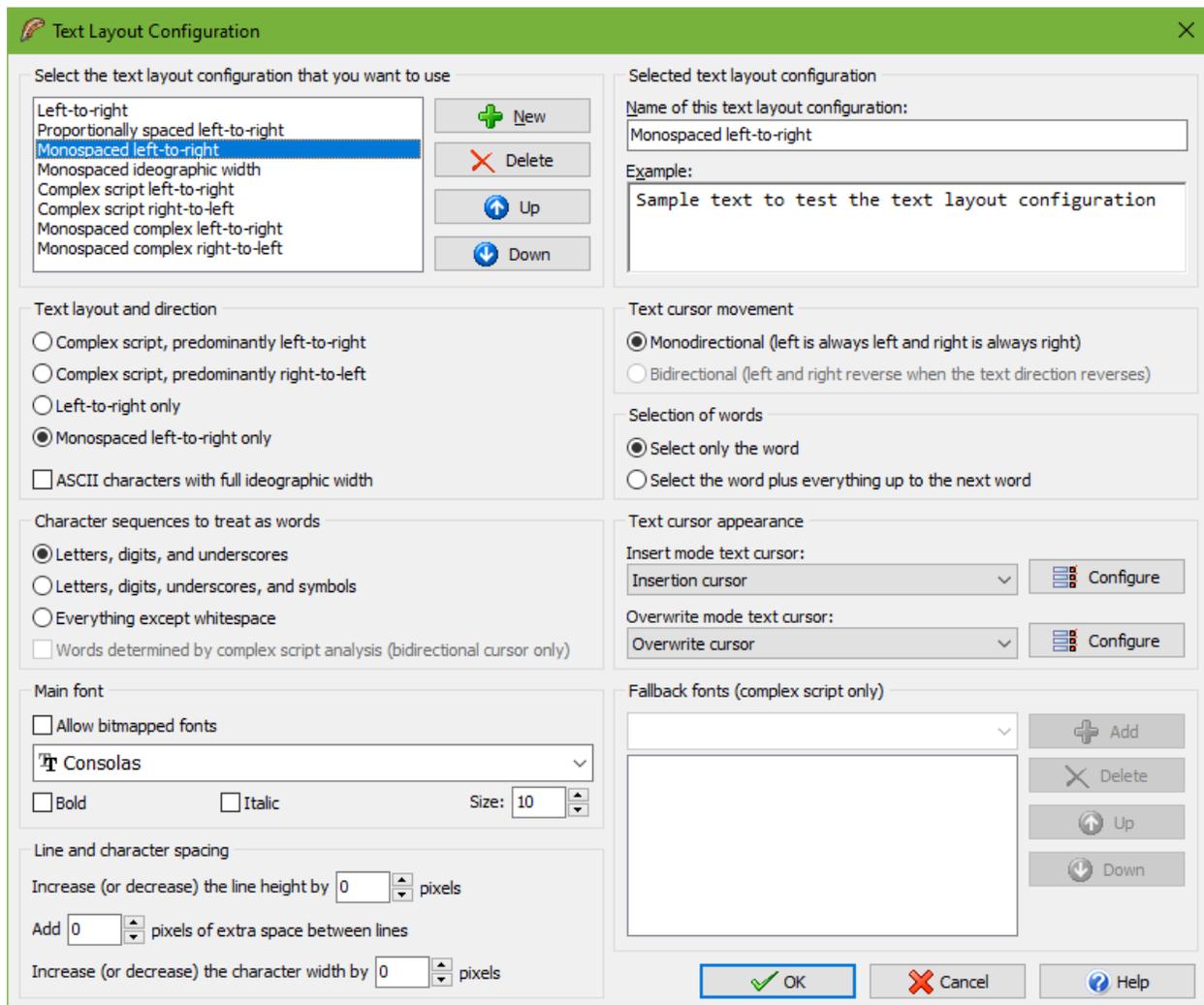
Configure the colors used by the results display on the GREP panel. You can select one of the preset configurations at the top of the screen. To customize the colors, click on an item in the list of individual colors. Then click the Background Color and Text Color buttons to change the item's colors. You can also change the font style with the bold, italic and underline checkboxes. At the bottom, a sample results display will show what the color configuration looks like.

## 41. Text Layout Configuration

In RegexMagic, a “text layout” is a combination of settings that control how text is displayed and how the text cursor navigates through that text. The settings include the font, text direction, text cursor behavior, text cursor appearance, which characters are word characters, and how the text should be spaced.

On the Editors tab in the Preferences, you can select up to four different text layouts that are used for various parts of RegexMagic. The Regex text layout is used for edit boxes for regular expressions and replacement texts. The Convert text layout is used for the warning and error messages on the Convert panel. The font from the Convert text layout is also used for the regex tree on the Create panel. The editors on the Test, Debug, Use, and GREP panels all use the same text layout. You’ll likely want to select a monospaced text layout for this one, so that columns line up correctly. The Forum text layout is used to edit messages on the built-in user forum.

You can select predefined text layouts from the drop-down lists. Or, you can click the Configure Text Layouts button to show the Text Layout Configuration window to customize the text layouts.



## Select The Text Layout Configuration That You Want to Use

The Text Layout Configuration screen shows the details of the text layout configuration that you select in the list in the top left corner. Any changes you make on the screen are automatically applied to the selected layout and persist as you choose different layouts in the list. The changes become permanent when you click OK. The layout that is selected in the list when you click OK becomes the new default layout.

Click the New and Delete buttons to add or remove layouts. You must have at least one text layout configuration. If you have more than one, you can use the Up and Down buttons to change their order. The order does not affect anything other than the order in which the text layouts configurations appear in selection lists.

RegexMagic comes with a number of preconfigured text layouts. If you find the options on this screen bewildering, simply choose the preconfigured layout that matches your needs, and ignore all the other settings. You can fully edit and delete all the preconfigured text layouts if you don't like them.

- Left-to-right: Normal settings with best performance for editing text in European languages and ideographic languages (Chinese, Japanese, Korean). The default font is monospaced. The layout does respect individual character width if the font is not purely monospaced or if you select another font.
- Proportionally spaced left-to-right: Like left-to-right, but the default font is proportionally spaced.
- Monospaced left-to-right: Like left-to-right, but the text is forced to be monospaced. Columns are guaranteed to line up perfectly even if the font is not purely monospaced. This is the best choice for working with source code and text files with tabular data.
- Monospaced ideographic width: Like monospaced left-to-right, but ASCII characters are given the same width as ideographs. This is the best choice if you want columns of mixed ASCII and ideographic text to line up perfectly.
- Complex script left-to-right: Supports text in any language, including complex scripts (e.g. Indic scripts) and right-to-left scripts (Hebrew, Arabic). Choose this for editing text that is written from left-to-right, perhaps mixed with an occasional word or phrase written from right-to-left.
- Complex script right-to-left: For writing text in scripts such as Hebrew or Arabic that are written from right-to-left, perhaps mixed with an occasional word or phrase written from left-to-right.
- Monospaced complex left-to-right: Like “complex script left-to-right”, but using monospaced fonts for as many scripts as possible. Text is not forced to be monospaced, so columns may not line up perfectly.
- Monospaced complex right-to-left: Like “complex script right-to-left”, but using monospaced fonts for as many scripts as possible. Text is not forced to be monospaced, so columns may not line up perfectly.

## Selected Text Layout Configuration

The section in the upper right corner provides a box to enter the name of the text layout configuration. This name is only used to help you identify it in selection lists when you have prepared more than one text layout configuration.

In the Example box you can enter some text to see how the selected text layout configuration causes the editor to behave.

## Text Layout and Direction

- Complex script, predominantly left-to-right: Text is written from left to right and can be mixed with text written from right to left. Choose this for complex scripts such as the Indic scripts, or for text in any language that mixes in the occasional word or phrase in a right-to-left or complex script.
- Complex script, predominantly right-to-left: Text is written from right to left and can be mixed with text written from left to right. Choose this for writing text in scripts written from right to left such as Hebrew or Arabic.
- Left-to-right only: Text is always written from left to right. Complex scripts and right-to-left scripts are not supported. Choose this for best performance for editing text in European languages and ideographic languages (Chinese, Japanese, Korean) that is written from left to right without exception.
- Monospaced left-to-right only: Text is always written from left to right and is forced to be monospaced. Complex scripts and right-to-left scripts are not supported. Each character is given the same horizontal width even if the font specifies different widths for different characters. This guarantees columns to be lined up perfectly. To keep the text readable, you should choose a monospaced font.
- ASCII characters with full ideographic width: You can choose this option in combination with any of the four preceding options. In most fonts, ASCII characters (English letters, digits, and punctuation) are about half the width of ideographs. This option substitutes full-width characters for the ASCII characters so they are the same width as ideographs. If you turn this on in combination with “monospaced left-to-right only” then columns that mix English letters and digits with ideographs will line up perfectly.

## Text Cursor Movement

- Monodirectional: The left arrow key on the keyboard always moves the cursor to the left on the screen and the right arrow key always moves the cursor to the right on the screen, regardless of the predominant or actual text direction.
- Bidirectional: This option is only available if you have chosen one of the complex script options in the “text layout and direction” list. The direction that the left and right arrow keys move the cursor into depends on the predominant text direction selected in the “text layout and direction” list and on the actual text direction of the word that the cursor is pointing to when you press the left or right arrow key on the keyboard.
  - Predominantly left-to-right: The left key moves to the preceding character in logical order, and the right key moves to the following character in logical order.
    - Actual left-to-right: The left key moves left, and the right key moves right.
    - Actual right-to-left: The actual direction is reversed from the predominant direction. The left key moves right, and the right key moves left.
  - Predominantly right-to-left: The left key moves to the following character in logical order, and the right key moves to the preceding character in logical order.
    - Actual left-to-right: The actual direction is reversed from the predominant direction. The left key moves right, and the right key moves left.
    - Actual right-to-left: The left key moves left, and the right key moves right.

## Selection of Words

- Select only the word: Pressing Ctrl+Shift+Right moves the cursor to the end of the word that the cursor is on. The selection stops at the end of the word. This is the default behavior for all Just Great Software applications. It makes it easy to select a specific word or string of words without any extraneous spaces or characters. To include the space after the last word, press Ctrl+Shift+Right once more, and then Ctrl+Shift+Left.
- Select the word plus everything to the next word: Pressing Ctrl+Shift+Right moves the cursor to the start of the word after the one that the cursor is on. The selection includes the word that the cursor was on and the non-word characters between that word and the next word that the cursor is moved to. This is how text editors usually behave on the Windows platform.

## Character Sequences to Treat as words

- Letters, digits, and underscores: Characters that are considered to be letters, digits, or underscores by the Unicode standard are selected when you double-click them. Ctrl+Left and Ctrl+Right move the cursor to the start of the preceding or following sequence of letters, digits, or underscores. If symbols or punctuation appear adjacent to the start of a word, the cursor is positioned between the symbol and the first letter of the word. Ideographs are considered to be letters.
- Letters, digits, and symbols: As above, but symbols other than punctuation are included in the selection when double-clicking. Ctrl+Left and Ctrl+Right never put the cursor between a symbol and another word character.
- Everything except whitespace: All characters except whitespace are selected when you double-click them. Ctrl+Left and Ctrl+Right move the cursor to the preceding or following position that has a whitespace character to the left of the cursor and a non-whitespace character to the right of the cursor.
- Words determined by complex script analysis: If you selected the “bidirectional” text cursor movement option, you can turn on this option to allow Ctrl+Left and Ctrl+Right to place the cursor between two letters for languages such as Thai that don’t write spaces between words.

## Text Cursor Appearance

Select a predefined cursor or click the Configure button to show the text cursor configuration screen. There you can configure the looks of the blinking text cursor (and even make it stop blinking).

A text layout uses two cursors. One cursor is used for insert mode, where typing in text pushes ahead the text after the cursor. The other cursor is used for overwrite mode, where typing in text replaces the characters after the cursor. Pressing the Insert key on the keyboard toggles between insert and overwrite mode.

## Main Font

Select the font that you want to use from the drop-down list. Turn on “allow bitmapped fonts” to include bitmapped fonts in the list. Otherwise, only TrueType and OpenType fonts are included. Using a TrueType or OpenType font is recommended. Bitmapped fonts may not be displayed perfectly (e.g. italics may be clipped) and only support a few specific sizes.

If you access the text layout configuration screen from the print preview, then turning on “allow bitmapped fonts” will include printer fonts rather than screen fonts in the list, in addition to the TrueType and OpenType fonts that work everywhere. A “printer font” is a font built into your printer’s hardware. If you select a printer font, set “text layout and direction” to “left to right only” for best results.

## Fallback Fonts

Not all fonts are capable of displaying text in all scripts or languages. If you have selected one of the complex script options in the “text layout and direction” list, you can specify one or more “fallback” fonts. If the main font does not support a particular script, RegexMagic will try to use one of the fallback fonts. It starts with the topmost font at the list and continues to attempt fonts lower in the list until it finds a font that supports the script you are typing with. If none of the fonts supports the script, then the text will appear as squares.

To figure out which scripts a particular font supports, first type or paste some text using those scripts into the Example box. Make sure one of the complex script options is selected. Then remove all fallback fonts. Now you can change the main font and see which characters that font can display. When you’ve come up with a list of fonts that, if used together, can display all of your characters, select your preferred font as the main font. Then add all the others as fallback fonts.

## Line and Character Spacing

By default all the spacing options are set to zero. This tells RegexMagic to use the default spacing for the font you have selected.

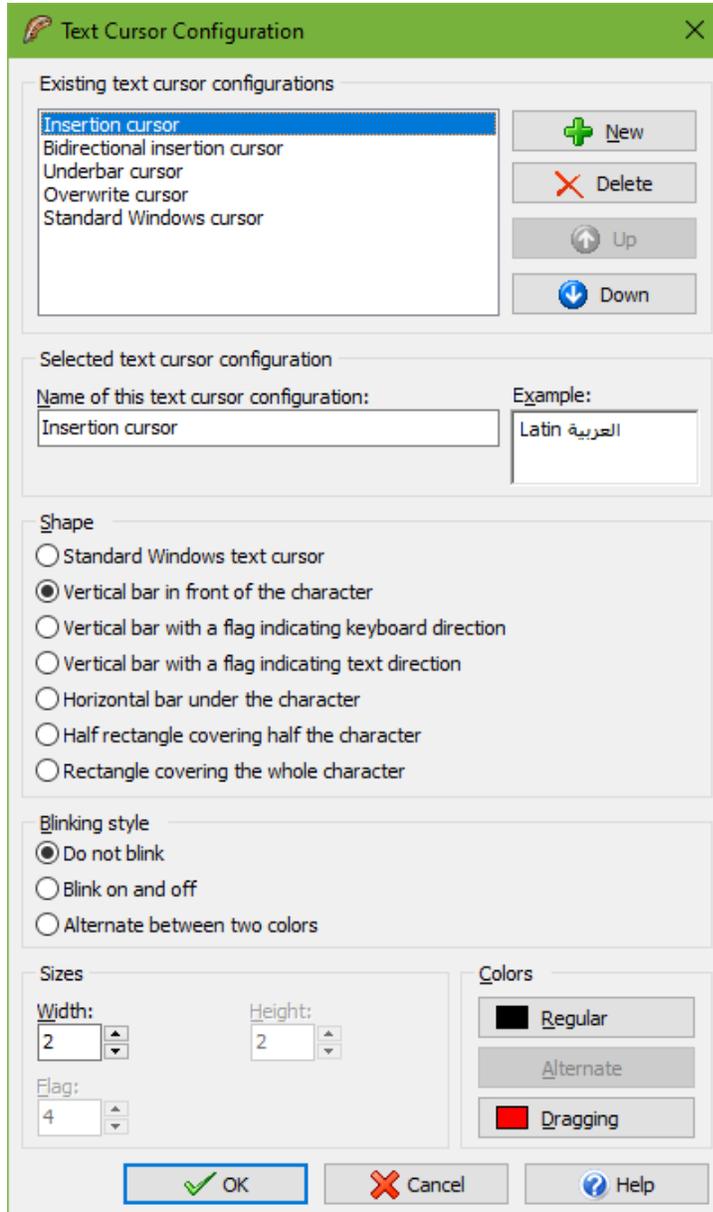
If you find that lines are spaced apart too widely, specify a negative value for “increase (or decrease) the line height”. Set to “add 0 pixels of extra space between lines”.

If you find that lines are spaced too closely together, specify a positive value for “increase (or decrease) the line height” and/or “add ... pixels of extra space between lines”. The difference between the two is that when you select a line of text, increasing the line height increases the height of the selection highlighting, while adding extra space between lines does not. If you select multiple lines of text, extra space between lines shows up as gaps between the selected lines. Adding extra space between lines may make it easier to distinguish between lines.

The “increase (or decrease) the character width by ... pixels” setting is only used when you select “monospaced left-to-right” only in the “text layout and direction” list. You can specify a positive value to increase the character or column width, or a negative value to decrease it. This can be useful if your chosen font is not perfectly monospaced and because of that characters appear spaced too widely or too closely.

## 42. Text Cursor Configuration

You can access the text cursor configuration screen from the text layout configuration screen by clicking one of the Configure buttons in the “text cursor appearance” section.



### Existing Text Cursor Configurations

The Text Cursor Configuration screen shows the details of the text cursor configuration that you select in the list at the top. Any changes you make on the screen are automatically applied to the selected cursor and persist as you choose different cursors in the list. The changes become permanent when you click OK. The cursor that is selected in the list when you click OK becomes the new default cursor.

Click the New and Delete buttons to add or remove cursors. You must have at least one text cursor configuration. If you have more than one, you can use the Up and Down buttons to change their order. The order does not affect anything other than the order in which the text cursor configurations appear in selection lists.

RegexMagic comes with a number of preconfigured text cursors. You can fully edit or delete all the preconfigured text cursors if you don't like them.

- Insertion cursor: Blinking vertical bar similar to the standard Windows cursor, except that it is thicker and fully black, even on a gray background.
- Bidirectional insertion cursor: Like the insertion cursor, but with a little flag that indicates whether the keyboard layout is left-to-right (e.g. you're typing in English) or right-to-left (e.g. you're typing in Hebrew). The flag is larger than what you get with the standard Windows cursor and is shown even if you don't have any right-to-left layouts installed.
- Underbar cursor: Blinking horizontal bar that lies under the character. This mimics the text cursor that was common in DOS applications.
- Overwrite cursor: Blinking rectangle that covers the bottom half of the character. In RegexMagic this is the default cursor for overwrite mode. In this mode, which is toggled with the Insert key on the keyboard, typing text overwrites the following characters instead of pushing them ahead.
- Standard Windows cursor: The standard Windows cursor is a very thin blinking vertical bar that is XOR-ed on the screen, making it very hard to see on anything except a pure black or pure white background. If you have a right-to-left keyboard layout installed, the cursor gets a tiny flag indicating keyboard direction. You should only use this cursor if you rely on accessibility software such as a screen reader or magnification tool that fails to track any of RegexMagic's other cursor shapes.

## Selected Text Cursor Configuration

Enter the name of the text cursor configuration. This name is only used to help you identify it in selection lists when you have prepared more than one text cursor configuration.

In the Example box you can enter some text to see what the cursor looks like. The box has a word in Latin and Arabic so you can see the difference in cursor appearance, if any, based on the text direction of the word that the cursor is on.

## Shape

- Standard Windows Text cursor: The standard Windows cursor is a very thin blinking vertical bar that is XOR-ed on the screen, making it very hard to see on anything except a pure black or pure white background. If you have a right-to-left keyboard layout installed, the cursor gets a tiny flag indicating keyboard direction. You should only use this cursor if you rely on accessibility software such as a screen reader or magnification tool that fails to track any of RegexMagic's other cursor shapes. The standard Windows cursor provides no configuration options.
- Vertical bar in front of the character: On the Windows platform, the normal cursor shape is a vertical bar that is positioned in front of the character that it points to. That is to the left of the character for left-to-right text, and to the right of the character for right-to-left text.
- Vertical bar with a flag indicating keyboard direction: A vertical bar positioned in front of the character that it points to, with a little flag (triangle) at the top that indicates the direction of the active keyboard layout. When the cursor points to a character in left-to-right text, it is placed to the

left of that character. When the cursor point to a character in right-to-left text, it is placed to the right of that character. The direction of the cursor's flag is independent of the text under the cursor. The cursor's flag points to the right when the active keyboard layout is for a left-to-right language. The cursor's flag points to the left when the active keyboard layout is for a right-to-left language.

- Vertical bar with a flag indicating text direction: A vertical bar positioned in front of the character that it points to, with a little flag (triangle) at the top that points to that character. When the cursor points to a character in left-to-right text, it is placed to the left of that character with its flag pointing to the right towards that character. When the cursor point to a character in right-to-left text, it is placed to the right of that character with its flag pointing to the left towards that character.
- Horizontal bar under the character: In DOS applications, the cursor was a horizontal line under the character that the cursor points to.
- Half rectangle covering half the character: The cursor covers the bottom half of the character that it points to. This is a traditional cursor shape to indicate typing will overwrite the character rather than push it ahead.
- Rectangle covering the whole character: The cursor makes the character invisible. This can also be used to indicate overwrite mode.

## Blinking Style

- Do not blink: The cursor is permanently visible in a single color. Choose this option if the blinking distracts you or if it confuses accessibility software such as screen readers or magnification tools.
- Blink on and off: The usual blinking style for text cursors on the Windows platform. The cursor is permanently visible while you type (quickly). When you stop typing for about half a second, the cursor blinks by becoming temporarily invisible. Blinking makes it easier to locate the cursor with your eyes in a large block of text.
- Alternate between two colors: Makes the cursor blink when you stop typing like “on and off”. But instead of making the cursor invisible, it is displayed with an alternate color. This option gives the cursor maximum visibility: the blinking animation attracts the eye while keeping the cursor permanently visible.

## Sizes

- Width: Width in pixels for the vertical bar shape.
- Height: Height in pixels for the horizontal bar shape.
- Flag: Length in pixels of the edges of the flag that indicates text direction.

## Colors

- Regular: Used for all shapes and blinking styles except the standard Windows cursor.
- Alternate: Alternate color used by the “alternate between two colors” blinking style.
- Dragging: Color of a second “ghost” cursor that appears while dragging and dropping text with the mouse. It indicates the position the text is moved or copied to when you release the mouse button.

## 43. Integrate RegexMagic with Searching, Editing and Coding Tools

RegexMagic is designed to be used as a companion to whatever software you use regular expressions with. Such applications include search tools, text editing and processing tools, programming and development tools, etc. Whenever you need to create a regular expression while working with those applications, RegexMagic pops up to build the regex, and disappears when you finish creating it.

Without integrating RegexMagic with your tools, this workflow takes many steps:

1. Start RegexMagic via a desktop or start menu shortcut.
2. Create the regular expression.
3. Copy the regular expression from RegexMagic to the clipboard, manually selecting the correct format.
4. Close RegexMagic.
5. Paste the regular expression into your tool.

### Basic Integration

Integrating RegexMagic with your favorite tools can significantly speed up this workflow by automating most of the steps. The simplest form of integration is by starting RegexMagic with certain command line parameters. This method is appropriate when you want to integrate RegexMagic with a 3rd party tool. If the tool has the ability to run external applications, you can easily launch RegexMagic from within the tool.

E.g. in Microsoft Visual Studio 2003, you could select Tools|External Tools from the menu, and specify the following arguments: `-putonclipboard -string basic -flavor dotnet -appname "Visual Studio"`. Use `-string c#` instead of `-string basic` if you develop in C# rather than Visual Basic.

When adding RegexMagic to Delphi's Tools menu, you can specify `-putonclipboard -string pascal -flavor pcre -flavorreplace jgsoft -appname Delphi` for the parameters. This selects the regex flavor used by TPerlRegEx.

This style of integration, which only takes a few minutes to implement, reduces the workflow to:

1. Start RegexMagic from a menu item, button or keyboard shortcut in your tool.
2. Create the regular expression.
3. Click the Send To button in RegexMagic, which closes RegexMagic and automatically puts the regex on the clipboard, in the right format.
4. Paste the regular expression into your tool.

Though you still need to copy and paste the regular expression from and into your tool, the reduced workflow is a lot more convenient. You can launch RegexMagic from within your tool, and RegexMagic automatically uses the correct regex flavor and string style for your tool.

## Advanced Integration

Tighter integration is possible with tools that have a programming interface (API), and with software that you develop yourself. RegexMagic provides a COM Automation interface, which you can easily import and call from any development tool or language that supports COM. The interface enables you to launch RegexMagic, and send and receive regex actions in response to the user's actions.

Except for actually creating the regular expression, the workflow is automated entirely:

1. Start RegexMagic from a menu item, button or keyboard shortcut in your tool.
2. Create the regular expression.
3. Click the Send To button in RegexMagic, which automatically updates the regex in your tool, and closes RegexMagic.

If you are a software developer, and some of your products support regular expressions, make your customers happy and provide tight integration with RegexMagic. Take a look at PowerGREP and EditPad Pro to see how convenient they make it to create a regular expression with RegexMagic.

## 44. Basic Integration with RegexMagic

Integrating RegexMagic with your favorite tools can significantly speed up this workflow by automating most of the steps. The simplest form of integration is by starting RegexMagic with certain command line parameters. This method is appropriate when you want to integrate RegexMagic with a 3rd party tool. If the tool has the ability to run external applications, you can easily launch RegexMagic from within the tool.

RegexMagic's command line parameters are case insensitive. The order in which they appear is irrelevant, except for parameters that must be followed by a second "data" parameter. The second parameter must immediately follow the first parameter, separated with a space. Values with spaces in them must be delimited by a pair of double quotes. Parameters can start with a hyphen or a forward slash.

### -putonclipboard

RegexMagic will show the Send To button. When clicked, the regex is copied to the clipboard. The regex is copied "as is", unless you specify one of the format options.

### -app *app*

Specify the application that the generated regular expression should work with. The *app* value must be an application identifier. RegexMagic will use this application's string style, regex flavor, replacement flavor, split flavor, and source code template. If there is no application identifier for the combination of string style and flavors that you want, then you cannot use the -app parameter. You can use the separate -string and -flavor parameters to specify custom combinations.

### -string *stringstyle*

Tell RegexMagic to use a certain string format for -putonclipboard. The *stringstyle* value must be a string style identifier. If you use both -app and -string, then -string takes precedence.

### -flavor *flavor*

Specify the regular expression flavor that the generated regex should work with. The *flavor* value must be a valid regex flavor identifier. If you use both -app and -flavor, then -flavor takes precedence.

### -flavorreplace *flavor*

Specify the replacement text flavor you're working with. The *flavor* value must be a valid replacement flavor identifier. If you use both -app and -flavorreplace, then -flavorreplace takes precedence. -flavorreplace is ignored unless you also use -app or -flavor to set the regex flavor.

### -flavorsplit *flavor*

Specify the split flavor you're working with. The *flavor* value must be a valid split flavor identifier. If you use both -app and -flavorsplit, then -flavorsplit takes precedence. -flavorsplit is ignored unless you also use -app or -flavor to set the regex flavor.

### -template *filename.rbsct*

Select the RegxBuddy source code template for the Use panel. You can specify the file name of the template without the path and without the extension. If you use both `-app` and `-template`, then `-template` takes precedence.

#### `-delimitclipboard` *delimiter*

Specify a string to use as a delimiter so that more text than just the regular expression can be passed via the clipboard. Your chosen delimiter must not be something that will appear in the regex or replacement text generated by RegxBuddy. Something like `***?***?` that isn't valid regex syntax is a good choice.

If you do not use the `-delimitclipboard` parameter, then only the regular expression is copied to the clipboard when you click the Send button.

If you use `-delimitclipboard` then RegxBuddy copies the regular expression followed by the delimiter followed by the list of options. No line breaks or whitespace are added around the delimiter. The regular expression is still formatted using the string style specified by the `-app` or `-string` parameter. The list of options is a comma-delimited list of options that need to be turned on. Options that need to be turned off are omitted.

<code>caseless</code>	Case insensitive
<code>freespacing</code>	Free-spacing regex
<code>dotall</code>	Dot matches line breaks
<code>multiline</code>	<code>^</code> and <code>\$</code> match at line breaks
<code>lbfonly</code>	LF only line break mode
<code>lbcronly</code>	CR only line break mode
<code>lbcrlfonly</code>	CRLF pairs only line break mode
<code>lbcrlf</code>	CR, LF, or CRLF line break mode
<code>lbunicode</code>	Unicode line break mode
<code>explicitcapture</code>	Parentheses do not create numbered groups
<code>namedduplicate</code>	Named capturing groups can have the same name
<code>ungreedy</code>	Quantifiers are lazy and appending <code>?</code> makes them greedy
<code>skipzerolength</code>	Skip zero-length matches
<code>stringsyntax</code>	Support literal string escapes on top of the regex or replacement text syntax
<code>splitlimit1234</code>	Split action with 1234 as the limit; 1234 can be any positive or negative integer; omit 1234 to split without a limit
<code>splitcapture</code>	Add capturing groups to the resulting array when splitting a string
<code>splitempty</code>	Add empty strings to the resulting array when splitting a string

If a replacement text is generated, the options are followed by another delimiter and the replacement text. The replacement text is also formatted using the string style specified by the `-app` or `-string` parameter.

#### `-sampleclipboard`

Start with a blank RegxBuddy Formula and paste the text on the clipboard as a sample on the Samples panel. The text on the clipboard is always used "as is".

`-appname "Application Name"`

The `-appname` parameter must be immediately followed with a parameter that indicates the name of the application that is invoking RegexMagic. RegexMagic will show this name in its caption bar, to make it clear which application the Regex will be sent to. You should always specify `-appname` when using `-putonclipboard`.

`-formula filename.rmf`

Loads the specified RegexMagic Formula file. Use this parameter if you want to associate RegexMagic with `rmf` files. This file format is used to share individual RegexMagic Formulas on the RegexMagic user forum.

`-sample document.txt`

Loads the file as a sample on the Samples tab.

`-library filename.rml`

Opens the specified library on the Library tab. Use this parameter if you want to associate RegexMagic with `rml` files.

`-grep filename.rmg`

Opens the specified GREP action on the GREP tab. Use this parameter if you want to associate RegexMagic with `rmg` files.

`-activate 1234`

Use this parameter in combination with `-putonclipboard`. `-activate` must be followed by a decimal unsigned integer that indicates the window handle of the active top-level form in your application. When the Send To button is clicked, RegexMagic will use the `SetForeground()` Win32 API call to bring your application to the front. That way, the Sent To button enables you to continue working with that application right away.

## 45. Integrating RegexMagic Using COM Automation

If you are a software developer, and some of your products support regular expressions, make your customers happy and provide tight integration with RegexMagic. That way, they can use RegexMagic to generate regular expressions to be used with your software, without having to manually copy and paste regexes from RegexMagic to your software.

RegexMagic provides a COM Automation interface, which you can easily import and call from any development tool or language that supports COM. The interface enables you to launch RegexMagic, receive the final regular expression, and even store the RegexMagic formula allowing the regular expression to be edited later with RegexMagic. It is a single instance interface, which means that each application has its own private instance of RegexMagic.

Take a look at PowerGREP and EditPad Pro to see how convenient they make it to create a regular expression with RegexMagic.

### Demo Applications Implementing RegexMagic's COM Automation

At <http://download.jgsoft.com/magic/RegexMagic2Clients.zip> you can download two sample applications that communicate with RegexMagic through its COM Automation interface. One is written in Delphi 2009, and the other is written in C# (Visual Studio 2010).

### Importing RegexMagic's Type Library

RegexMagic's installer automatically registers RegexMagic's automation interface with Windows. To automate RegexMagic via COM, you need to import its type library. It is stored in `RegexMagic2.exe`, which is installed under `C:\Program Files\Just Great Software\RegexMagic2` by default.

In Delphi, select `Component | Import Component` from the menu. Select "Import a Type Library" and click Next. Select "RegexMagic API" version "2.0" from `RegexMagic2.exe` and click Next. Choose a palette page and a directory, make sure `Generate Component Wrappers` is ticked, and click Next. Install into a new or existing package as you prefer. Two new component called `TRegexMagicIntf` and `TRegexMagicIntf2` appear on the component palette. These components implement the methods and events you can use to communicate with RegexMagic. Drop `TRegexMagicIntf2` on a form or data module. Set `ConnectKind` to `ckNewInstance`, and make sure `AutoConnect` is `False`. This ensures your application has its own instance of RegexMagic, and that RegexMagic only appears when the user actually wants to edit a regular expression. Call the component's `Connect()` method the first time the user wants to edit a regular expression with RegexMagic. For efficiency, do not call the `Disconnect()` method until your application terminates (at which point it is called automatically). Assign an event handler to `OnFinishRegex`. Calling `TRegexMagicIntf2.Connect()` raises an exception if the user has an older version of RegexMagic. If you want to support older versions, drop a `TRegexMagicIntf` on your form as well, and call `TRegexMagicIntf.Connect()` when `TRegexMagicIntf2.Connect()` fails. Once a call to `Connect()` succeeds, your application should not make any other calls to `Connect()` so that only one instance of the `TRegexMagicIntf` component is live. Otherwise, it will launch multiple instances of RegexMagic. Doing so would not cause any errors, but does waste resources.

In Visual Studio, right-click on “References” in the Solution Explorer, and pick “Add Reference”. Switch to the COM tab, and choose “RegexMagic API” version “2.0” from `RegexMagic2.exe`. After adding the reference, import the `RegexMagic` namespace with `using RegexMagic`. Then you can easily access the `RegexMagicIntf2` class. Create a new object from this class the first time the user wants to edit a regular expression with `RegexMagic`. Use the same object for all subsequent times the user wants to edit the same or another regex. Do not delete the object until your application terminates. Each time you create an object of `RegexMagicIntf2`, a new `RegexMagic` instance is launched. You also need to create an instance of `IRegexMagicIntfEvents_FinishRegexEventHandler` and assign it to the `FinishRegex` event of your `RegexMagicIntf2` object. Do not assign both. Only assign the one you will actually use.

Note that to successfully communicate with `RegexMagic`, two-way communication is required. Your application not only needs to call the methods of the `RegexMagicIntf2` interface to send the regular expression to `RegexMagic`. It also needs to implement an event sink for the `FinishRegex` method defined in the `IRegexMagicIntfEvents` interface. Not all development and scripting tools that can call COM automation objects can also implement event sinks. The tool must support “early binding”.

## RegexMagicIntf Interface

The `RegexMagicIntf` COM automation interface provides the following methods:

```
void IndicateApp(BSTR AppName, uint Wnd)
```

Call `IndicateApp` right after connecting to the COM interface. `AppName` will be displayed in `RegexMagic`’s caption bar, to make it clear that this instance is connected to your application. `Wnd` is the handle of your application’s top-level form where the regex is used. `RegexMagic` will call `SetForegroundWindow(Wnd)` to activate your application when the user is done with `RegexMagic` (i.e. after a call to `FinishRegex`). You can call `IndicateApp` as often as you want, as long as you also call it right after connecting to the COM interface.

```
uint GetWindowHandle()
```

Returns the window handle of `RegexMagic`’s top window, which is either the main window, or the topmost modal dialog box in `RegexMagic`. Pass this to `SetForegroundWindow()` after calling `InitRegex` or `InitAction`. `SetForegroundWindow()` only works when called by the thread that has input focus, so `RegexMagic` cannot bring itself to front. That is your application’s duty. (Likewise, `RegexMagic` will bring your application to front after calling `FinishRegex`, using the window handle you passed on in `IndicateApp()`.)

```
void SetOptions(BSTR StringStyle, BSTR RegexFlavor, BSTR ReplaceFlavor, VARIANT Options)
```

Do not call `SetOptions` when using the `RegexMagicIntf2` interface as that will make `RegexMagic 2` fall back to `RegexMagic 1` support. Call `SetOptions2` instead.

`RegexMagic 1` requires you to call `SetOptions` before calling `InitFormula` the first time. After that, you only need to call `SetOptions` if your application wants to use different options.

`StringStyle` expects a string style identifier. It tells `RegexMagic` which string format it should use when returning the regular expression via `FinishRegex`. By using `StringStyle`, you can let

RegexMagic take care of adding and removing quote characters, escaping characters, etc. If you pass an unsupported string type, “as is” will be used instead.

`RegexFlavor` and `ReplaceFlavor` tell RegexMagic which regular expression flavor and which replacement text flavor RegexMagic should use to generate the regular expression. The user will not be able to select different flavors.

`Options` expects an array of 4 integer values that indicate which options your application can set outside of the regular expression. If you pass `NULL` as the `Options` parameter, RegexMagic assumes your application supports all options, as if you had specified 1 for each value.

### Index Type Option

<b>0</b>	int	Dot matches line breaks
<b>1</b>	int	Case insensitive
<b>2</b>	int	^ and \$ match at line breaks
<b>3</b>	int	Free-spacing syntax

You can specify 3 values for each option:

### Value Meaning

<b>0</b>	The option is always turned off in the application.
<b>1</b>	The option can be turned on or off in the application. RegexMagic will tell your application whether it wants to turn the option on or off via the <code>Options</code> parameter of <code>FinishRegex</code> .
<b>2</b>	The option is always turned on in the application.

If you set an option to 0 or 2, and the string style does not include regex flags (e.g. `/ismx` in Perl), RegexMagic uses mode modifiers instead, if the chosen regex flavor supports them. If the flavor does not support mode modifiers, and the string style does not support regex flavors, then the options your application cannot set will be unavailable. That prevents the user from generating regular expressions that rely on those options being on or off.

```
void InitFormula(BSTR Formula)
```

Call `InitFormula` to make RegexMagic show up with the given RegexMagic formula. RegexMagic will follow up with a call to `FinishRegex` when the user closes RegexMagic by clicking the Send To button.

`Formula` is the RegexMagic formula (Samples, Match, and Action panel settings) to be used as the basis for creating the regular expression. Pass `NULL` or an empty string when you call `InitFormula` the first time to start with a blank formula. Pass the formula returned by `FinishRegex` if the user wants to continue editing the same regular expression. If you pass `NULL` upon the second call, RegexMagic shows up with the same formula the user edited last time, provided RegexMagic has not been shut down between calls. You should treat `Formula` as an opaque string that your application can use to restore RegexMagic’s state. You can persist this string with your application’s settings. If your application uses regular expressions in multiple areas, you can persist multiple formulas and make RegexMagic switch between them depending on the area the user is working in.

Since a `RegexMagic` formula includes samples, passing a formula to `InitFormula` replaces any samples you may have added with `AddSampleString` and `AddSampleFile`. If you want `RegexMagic` to edit an existing formula using new samples, call `ClearSamples`, `AddSampleString`, and/or `AddSampleFile` immediately after calling `InitFormula`, instead of before.

```
void ClearSamples()
```

Call `ClearSamples` to clear the `Samples` panel in `RegexMagic`. This clears out all samples set by previous calls to `AddSampleString` and `AddSampleFile`.

```
void AddSampleString(BSTR Caption, BSTR Sample)
```

Call `AddSampleString` to add (a sample of) the data your application will use the regular expression on. `RegexMagic` adds it to the `Samples` panel. That way the user can instantly test the regex on the data it'll actually be used on. You should only use `AddSampleString` if the data is not stored as a file on disk. Otherwise, `AddSampleFile` is more efficient. You can call `AddSampleString` multiple times to provide multiple samples. Use the `Caption` parameter to give each sample a different caption in the list on the `Samples` panel.

```
void AddSampleFile(BSTR Caption, BSTR Filename)
```

Call `AddSampleFile` to make `RegexMagic` load one of the files your application will use the regular expression on. `RegexMagic` adds it to the `Samples` panel. That way the user can instantly test the regex on the data it'll actually be used on. You can call `AddSampleFile` multiple times to provide multiple samples. Use the `Caption` parameter to give each sample a different caption in the list on the `Samples` panel. If you don't specify a caption, the file's name is used.

## RegexMagicIntf2 Interface

`RegexMagic 2.0.0` and later provide an additional interface called `RegexMagicIntf2`. This interface descends from `RegexMagicIntf`, so all the above methods are also supported by `RegexMagicIntf2`. You should use this interface to make use of the new functionality in `RegexMagic 2`. If your application is unable to instantiate this interface, then it can fall back on `RegexMagicIntf` if you want to support `RegexMagic 1` also.

```
BOOL CheckVersion(uint Version)
```

Call `CheckVersion()` before calling `SetOptions2` with an application identifier that is not supported by `RegexMagic 2.0.0`. `RegexMagic 2.0.0` returns `TRUE` if you pass `200` for `Version` and `FALSE` for all other versions. Since `RegexMagic 2.0.0` is the first version to support this call, you cannot query for `1.x.x` support. `RegexMagic 2.0.0` supports all regex flavors supported by `1.x.x`.

Future versions of `RegexMagic` will continue to return `TRUE` if you pass `200` for `Version`. They will return `TRUE` for additional version numbers if (and only if) new applications or regular expression flavors are added.

```
void SetOptions2(BSTR StringStyle, VARIANT Application, VARIANT Options)
```

Call `SetOptions2()` before calling `InitFormula` the first time. After that, you only need to call `SetOptions2` if your application wants to use different options.

`StringStyle` expects a string style identifier. It tells `RegexMagic` which string format it should use when returning the regular expression via `FinishRegex`. By using `StringStyle`, you can let `RegexMagic` take care of adding and removing quote characters, escaping characters, etc. If you pass an unsupported string type, “as is” will be used instead.

`Application` can be set to a string with an application identifier. If the identifier’s age is younger than `RegexMagic 2.0.0` then you can only use it if `CheckVersion()` returned `TRUE` for the identifier’s age. E.g. if the identifier’s age is `2.1.0` then you can only use it if `CheckVersion(210)` returns `TRUE`. There is no need to call `CheckVersion()` for identifiers introduced in `RegexMagic 2.0.0` or `1.x.x`.

Alternatively, `Application` can be set to a variant array that defines a custom application.

Index	Type	Description
0	BSTR	Name the flavor is indicated with in flavor selection lists. This is only used if the custom application is unknown to <code>RegexMagic</code> . If the user previously added the same application under a different name, then the user’s chosen name will be preserved.
1	BSTR	Regular expression flavor identifier
2	BSTR or NULL	Replacement flavor identifier. You can set this to the empty string or <code>NULL</code> to disable Replace mode if the application cannot search-and-replace using a regex.
3	BSTR or NULL	Split flavor identifier. You can set this to the empty string or <code>NULL</code> to disable Split mode if the application cannot split strings using a regex.
4	BSTR or NULL	String style identifier. Used only for the Copy and Paste menus in <code>RegexMagic</code> . Not used to pass regexes via the COM interface. Defaults to “asis” if omitted.
5	BSTR or NULL	File name without extension of the <code>RegexMagic</code> source code template for generating source code snippets on the Use panel. This can be a built-in or a custom template. Use panel is disabled if you don’t specify a template file name or if the file cannot be found.

`Options` expects an array of 4 integer values that indicate which options your application can set outside of the regular expression. If you pass `NULL` as the `Options` parameter, `RegexMagic` assumes your application supports the same options as the regular expression flavor specified by the `Application` parameter (whether explicitly as a custom application or implicitly with an application identifier).

#### Index Type Option

0	int	Dot matches line breaks
1	int	Case insensitive
2	int	^ and \$ match at line breaks
3	int	Free-spacing syntax

You can specify 3 values for each option:

#### Value Meaning

0	The option is always turned off in the application.
---	---

- 1 The option can be turned on or off in the application. `RegexMagic` will tell your application whether it wants to turn the option on or off via the `Options` parameter of `FinishRegex`.
- 2 The option is always turned on in the application.

If you set an option to 0 or 2, and the string style does not include regex flags (e.g. `/ismx` in Perl), `RegexMagic` uses mode modifiers instead, if the chosen regex flavor supports them. If the flavor does not support mode modifiers, and the string style does not support regex flavors, then the options your application cannot set will be unavailable. That prevents the user from generating regular expressions that rely on those options being on or off.

```
void AddSampleString2(BSTR Caption, BSTR Sample, VARIANT Options)
```

`AddSampleString2` does the same as `AddSampleString` but takes a third parameter that allows you to control how `RegexMagic` interprets the sample string. You should pass an array with 4 elements. If you don't need these options, then you can use `AddSampleString` instead.

### Index Type Description

- |   |      |   |
|---|------|---|
| 0 | BSTR | String style identifier that indicates how <code>RegexMagic</code> should interpret the Subject string. Defaults to <code>asis</code> if you do not specify a valid string style identifier.<br>If your application converted the string from a particular encoding to the Unicode BSTR needed for the Subject argument, then you can specify one of the text file encoding identifiers |
| 1 | BSTR | to have <code>RegexMagic</code> convert the string back to that encoding so the user can work with the same encoding on the Samples panel in <code>RegexMagic</code> as in your application. The default is <code>utf16le</code> which is the native encoding of the COM interface.   |
| 2 | BSTR | Specify <code>file</code> , <code>page</code> , or <code>line</code> to set the scope to “whole file”, “page by page”, or “line by line”. Pass <code>NULL</code> or an empty string to leave the scope unchanged.   |
| 3 | BSTR | You should set this to <code>mixed</code> to load the string without changing its line breaks. Other supported line break options are <code>auto</code> , <code>crlf</code> , <code>lf</code> , and <code>cr</code> .   |

```
void AddSampleFile2(BSTR Caption, BSTR Filename, VARIANT Options)
```

`AddSampleFile2` does the same as `AddSampleFile` but takes a third parameter that allows you to control how `RegexMagic` reads the sample file. You should pass an array with 4 elements. If you don't need these options, then you can use `AddSampleFile` instead.

### Index Type Description

- |   |      |   |
|---|------|---|
| 0 | BSTR | Specify one of the text file encoding identifiers to interpret the file as a text file with the given encoding. Specify <code>bytes</code> to load the file as a binary file in hexadecimal mode. Pass <code>NULL</code> or an empty string to have <code>RegexMagic</code> auto-detect the encoding.   |
| 1 | BSTR | Specify <code>file</code> , <code>page</code> , or <code>line</code> to set the scope to “whole file”, “page by page”, or “line by line”. Pass <code>NULL</code> or an empty string to leave the scope unchanged.<br>Specify <code>auto</code> convert the line breaks in the file to the style expected by the regex flavor. Specify <code>mixed</code> to load the file without converting its line breaks. Specify <code>crlf</code> , <code>lf</code> , or <code>cr</code> to convert the |
| 2 | BSTR | line breaks to the specified style. The conversion is only applied to the copy of the file in memory. The file on disk is not changed. Pass <code>NULL</code> or an empty string to leave the line break mode unchanged.  |

## RegexMagicIntfEvents Interface

`FinishRegex(BSTR Regex, BSTR Replacement, VARIANT Options, BSTR Formula)`

You must provide an event handler for this event to receive the regular expression generated by `RegexMagic`. After you call `InitFormula`, `RegexMagic` will call `FinishRegex` when the user closes `RegexMagic` by clicking the Send To button. The call to `InitFormula` returns immediately. The regular expression and replacement text returned by `FinishRegex` use the flavors and options you passed to the most recent call to `SetOptions`.

`RegexMagic` passes an array of `BOOL` values to `Options` to indicate which options your application should turn on (`TRUE`) or turn off (`FALSE`). If you called `SetOptions2()` then the array will have 12 elements. If you called `SetOptions()`, then the array will have only 4 elements (indexes 0 to 3 in the table). The number of elements does not depend on the the parameters you passed to `SetOptions2()` or `SetOptions()`. It only depends on which of these two methods you called. Options not supported by your application are still included in the array.

Index	Type	Option
0	BOOL	Dot matches line breaks ( <code>TRUE</code> ) or dot doesn't match line breaks ( <code>FALSE</code> ).
1	BOOL	Case insensitive ( <code>TRUE</code> ) or case sensitive ( <code>FALSE</code> ).
2	BOOL	<code>^</code> and <code>\$</code> match at line breaks ( <code>TRUE</code> ) or <code>^</code> and <code>\$</code> don't match at line breaks ( <code>FALSE</code> ).
3	BOOL	Free-spacing ( <code>TRUE</code> ) or exact spacing ( <code>FALSE</code> ).
4	BSTR	Line break style. One of: <code>default</code> , <code>lf</code> , <code>cr</code> , <code>crlfonly</code> , <code>crlf</code> , <code>unicode</code>
5	BOOL	Named capture only ( <code>TRUE</code> ) or numbered capture ( <code>FALSE</code> ).
6	BOOL	Allow duplicate names ( <code>TRUE</code> ) or require names to be unique ( <code>FALSE</code> ).
7	BOOL	Lazy quantifiers ( <code>TRUE</code> ) or greedy quantifiers ( <code>FALSE</code> ).
8	BOOL	Skip zero-length matches ( <code>TRUE</code> ) or allow zero-length matches ( <code>FALSE</code> ).
9	int or NULL	Limit for "split" actions. Set to a <code>NULL</code> variant to indicate that no limit is set at all (which is not the same as setting the limit to zero).
10	BOOL	Split: add groups ( <code>TRUE</code> ) or don't add groups ( <code>FALSE</code> ).
11	BOOL	Split: add empty strings ( <code>TRUE</code> ) or don't add empty strings ( <code>FALSE</code> ).

`RegexMagic` also returns the formula that was used to generate the regular expression. Your application can save this formula and pass it to the next call to `InitFormula` if the user wants to edit the same regular expression.

If the user closes `RegexMagic` without clicking the Send To button, your application will not receive a call to `FinishRegex`.

## Which Methods to Call and Events to Handle

In summary, you must call `IndicateApp` immediately after connecting to `RegexMagic`. Then call `SetOptions2` for `RegexMagic 2` or `SetOptions` for `RegexMagic 1` to indicate your application's capabilities. Then, call `InitFormula` each time the user wants to create a regular expression with `RegexMagic`. `RegexMagic` follows up with a corresponding call to `FinishRegex`. If you call `InitFormula` again before receiving a call to `FinishRegex`, the effects of the previous call to `InitFormula` are canceled.

## 46. Application Identifiers

Identifiers for the `-app` command line parameter and for the `RegexMagic` action variant structure in the COM interface. Each application defines a regular expression flavor and a string style. Most applications also define a replacement text flavor, a split flavor, and a source code template.

Identifier	Age	Application
act2	2.0.0	AceText 2 & 3
ace4	<b>2.13.0</b>	AceText 4
aspnet11	2.0.0	ASP.NET 1.1
aspnet20	2.0.0	ASP.NET 2.0–4.8
boostreansi1038	2.6.0	boost::regex 1.38–1.39
boostreansi1042	2.6.0	boost::regex 1.42–1.43
boostreansi1044	2.6.0	boost::regex 1.44–1.46
boostreansi1047	2.6.0	boost::regex 1.47
boostreansi1048	2.12.0	boost::regex 1.48–1.49
boostreansi1050	2.12.0	boost::regex 1.50–1.53
boostreansi1054	2.6.0	boost::regex 1.54–1.57
boostreansi1058	2.6.0	boost::regex 1.58–1.59
boostreansi1060	2.6.0	boost::regex 1.60–1.61
boostreansi1062	2.6.0	boost::regex 1.62–1.63
boostreansi1064	2.7.0	boost::regex 1.64–1.65
boostreansi1066	2.8.0	boost::regex 1.66–1.77
boostreansi1078	<b>2.13.0</b>	boost::regex 1.78–1.83
boostrewrite1038	2.6.0	boost::wregex 1.38–1.39
boostrewrite1042	2.6.0	boost::wregex 1.42–1.43
boostrewrite1044	2.6.0	boost::wregex 1.44–1.46
boostrewrite1047	2.6.0	boost::wregex 1.47–1.53
boostrewrite1054	2.6.0	boost::wregex 1.54–1.57
boostrewrite1058	2.6.0	boost::wregex 1.58–1.59
boostrewrite1060	2.6.0	boost::wregex 1.60–1.63
boostrewrite1064	2.7.0	boost::wregex 1.64–1.65
boostrewrite1066	2.8.0	boost::wregex 1.66–1.77
boostrewrite1078	<b>2.13.0</b>	boost::wregex 1.78–1.83
csharp11	2.0.0	C# (.NET 1.1)
csharp20	2.0.0	C# (.NET 2.0–7.0)
cppb102core	2.7.0	C++Builder 10.2 (TPerlRegEx)
cppb102	2.7.0	C++Builder 10.2 (TRegEx)
cppb103core	2.9.0	C++Builder 10.3–11 (TPerlRegEx)
cppb103	2.9.0	C++Builder 10.3–11 (TRegEx)
cppbxecore	2.0.0	C++Builder XE (TPerlRegEx)
cppbxecore	2.0.0	C++Builder XE (TRegEx)

cppbxe2core	2.0.0	C++Builder XE2–XE3 (TPerlRegEx)
cppbxe2	2.0.0	C++Builder XE2–XE3 (TRegEx)
cppbxe4core	2.0.0	C++Builder XE4–XE5 (TPerlRegEx)
cppbxe4	2.0.0	C++Builder XE4–XE5 (TRegEx)
cppbxe6core	2.0.0	C++Builder XE6 (TPerlRegEx)
cppbxe6	2.0.0	C++Builder XE6 (TRegEx)
cppbxe7core	2.1.0	C++Builder XE7–XE8 & 10–10.1 (TPerlRegEx)
cppbxe7	2.1.0	C++Builder XE7–XE8 & 10–10.1 (TRegEx)
delphi102core	2.7.0	Delphi 10.2 (TPerlRegEx)
delphi102	2.7.0	Delphi 10.2 (TRegEx)
delphi103core	2.9.0	Delphi 10.3–11 (TPerlRegEx)
delphi103	2.9.0	Delphi 10.3–11 (TRegEx)
delphi2007	2.0.0	Delphi 2007 and prior (TPerlRegEx)
delphi2009	2.0.0	Delphi 2009–2010 (TPerlRegEx)
delphinet	2.0.0	Delphi for .NET
prism11	2.0.0	Delphi Prism (.NET 1.1)
prism20	2.0.0	Delphi Prism (.NET 2.0–4.8)
delphixecore	2.0.0	Delphi XE (TPerlRegEx)
delphixe	2.0.0	Delphi XE (TRegEx)
delphixe2core	2.0.0	Delphi XE2–XE3 (TPerlRegEx)
delphixe2	2.0.0	Delphi XE2–XE3 (TRegEx)
delphixe4core	2.0.0	Delphi XE4–XE5 (TPerlRegEx)
delphixe4	2.0.0	Delphi XE4–XE5 (TRegEx)
delphixe6core	2.0.0	Delphi XE6 (TPerlRegEx)
delphixe6	2.0.0	Delphi XE6 (TRegEx)
delphixe7core	2.1.0	Delphi XE7–XE8 & 10–10.1 (TPerlRegEx)
delphixe7	2.1.0	Delphi XE7–XE8 & 10–10.1 (TRegEx)
epp5	2.0.0	EditPad 5
epp6	2.0.0	EditPad 6 & 7
epp8	2.9.0	EditPad 8
gnubre	2.0.0	GNU BRE
gnuere	2.0.0	GNU ERE
groovy4	2.0.0	Groovy (JDK 1.4)
groovy5	2.0.0	Groovy (JDK 1.5)
groovy6	2.0.0	Groovy (JDK 1.6)
groovy7	2.0.0	Groovy (JDK 1.7)
groovy8	2.0.0	Groovy (JDK 1.8)
groovy9	2.8.0	Groovy (JDK 9–12)
groovy13	2.9.0	Groovy (JDK 13–14)
groovy15	2.11.0	Groovy (JDK 15)
groovy16	2.11.0	Groovy (JDK 16)

groovy17	2.12.0	Groovy (JDK 17–18)
groovy19	<b>2.13.0</b>	Groovy (JDK 19–21)
html5chrome	2.0.0	HTML5 Pattern (Chrome)
html5edge	2.4.0	HTML5 Pattern (Edge)
html5firefox	2.0.0	HTML5 Pattern (Firefox)
html5msie	2.0.0	HTML5 Pattern (MSIE standard)
html5opera	2.0.0	HTML5 Pattern (Opera)
java4	2.0.0	Java 4
java5	2.0.0	Java 5
java6	2.0.0	Java 6
java7	2.0.0	Java 7
java8	2.0.0	Java 8
java9	2.8.0	Java 9–12
java13	2.9.0	Java 13–14
java15	2.11.0	Java 15
java16	2.11.0	Java 16
java17	2.12.0	Java 17–18
java19	<b>2.13.0</b>	Java 19–21
chrome	2.0.0	JavaScript (Chrome)
edge	2.4.0	JavaScript (Edge)
firefox	2.0.0	JavaScript (Firefox)
quirks	2.0.0	JavaScript (MSIE quirks)
msie	2.0.0	JavaScript (MSIE standard)
opera	2.0.0	JavaScript (Opera)
mysql	2.0.0	MySQL
oracle10gR1	2.0.0	Oracle 10gR1
oracle10gR2	2.0.0	Oracle 10gR2
oracle11g	2.0.0	Oracle 11gR1, 11gR2 & 12c
pcre40	2.0.0	PCRE 4.0–4.4
pcre45	2.7.0	PCRE 4.5
pcre50	2.0.0	PCRE 5.0–6.4
pcre65	2.0.0	PCRE 6.5–6.6
pcre67	2.0.0	PCRE 6.7
pcre70	2.0.0	PCRE 7.0
pcre71	2.0.0	PCRE 7.1
pcre72	2.0.0	PCRE 7.2–7.3
pcre74	2.0.0	PCRE 7.4
pcre75	2.0.0	PCRE 7.5–7.6
pcre77	2.0.0	PCRE 7.7–7.9
pcre800	2.0.0	PCRE 8.00
pcre801	2.0.0	PCRE 8.01–8.02

pcr810	2.0.0	PCRE 8.10
pcr811	2.0.0	PCRE 8.11–8.12
pcr813	2.0.0	PCRE 8.13
pcr820	2.0.0	PCRE 8.20
pcr821	2.0.0	PCRE 8.21
pcr830	2.0.0	PCRE 8.30–8.33 UTF-8
pcr830_16	2.0.0	PCRE 8.30–8.33 UTF-16
pcr832_32	2.0.0	PCRE 8.32–8.33 UTF-32
pcr834	2.0.0	PCRE 8.34–8.35 UTF-8
pcr834_16	2.0.0	PCRE 8.34–8.35 UTF-16
pcr834_32	2.0.0	PCRE 8.34–8.35 UTF-32
pcr836	2.2.0	PCRE 8.36–8.38 UTF-8
pcr836_16	2.2.0	PCRE 8.36–8.38 UTF-16
pcr836_32	2.2.0	PCRE 8.36–8.38 UTF-32
pcr839	2.7.0	PCRE 8.39 UTF-8
pcr839_16	2.7.0	PCRE 8.39 UTF-16
pcr839_32	2.7.0	PCRE 8.39 UTF-32
pcr840	2.7.0	PCRE 8.40–8.45 UTF-8
pcr840_16	2.7.0	PCRE 8.40–8.45 UTF-16
pcr840_32	2.7.0	PCRE 8.40–8.45 UTF-32
pcr1010	2.2.0	PCRE2 10.10
pcr1020	2.4.0	PCRE2 10.20
pcr1021	2.5.0	PCRE2 10.21–10.22
pcr1023	2.7.0	PCRE2 10.23
pcr1030	2.8.0	PCRE2 10.30–10.31
pcr1032	2.9.0	PCRE2 10.32–10.34
pcr1035	2.11.0	PCRE2 10.35–10.39
perl508	2.0.0	Perl 5.8
perl510	2.0.0	Perl 5.10
perl512	2.0.0	Perl 5.12
perl514	2.0.0	Perl 5.14–5.16
perl518	2.0.0	Perl 5.18
perl520	2.1.0	Perl 5.20
perl522	2.3.0	Perl 5.22
perl524	2.7.0	Perl 5.24
perl526	2.7.0	Perl 5.26–5.28
perl530	2.9.0	Perl 5.30–5.32
phpereg	2.0.0	PHP ereg 4.3.3–5.2.17
phppreg4303	2.0.0	PHP preg 4.3.3–4.3.4
phppreg4305	2.7.0	PHP preg 4.3.5–4.3.11
phppreg4402	2.0.0	PHP preg 4.4.0–4.4.2

phppreg4403	2.0.0	PHP preg 4.4.3–4.4.4
phppreg4405	2.0.0	PHP preg 4.4.5
phppreg4406	2.0.0	PHP preg 4.4.6–4.4.8
phppreg5000	2.0.0	PHP preg 5.0.0–5.0.4
phppreg5005	2.0.0	PHP preg 5.0.5–5.1.2
phppreg5103	2.0.0	PHP preg 5.1.3–5.1.6
phppreg5200	2.0.0	PHP preg 5.2.0–5.2.1
phppreg5202	2.0.0	PHP preg 5.2.2–5.2.3
phppreg5204	2.0.0	PHP preg 5.2.4
phppreg5205	2.0.0	PHP preg 5.2.5
phppreg5206	2.0.0	PHP preg 5.2.6
phppreg5207	2.0.0	PHP preg 5.2.7–5.2.13
phppreg5214	2.0.0	PHP preg 5.2.14–5.2.17
phppreg5300	2.0.0	PHP preg 5.3.0–5.3.1
phppreg5302	2.0.0	PHP preg 5.3.2
phppreg5303	2.0.0	PHP preg 5.3.3
phppreg5304	2.0.0	PHP preg 5.3.4–5.3.5
phppreg5306	2.0.0	PHP preg 5.3.6–5.3.18
phppreg5319	2.0.0	PHP preg 5.3.19–5.3.29
phppreg5400	2.0.0	PHP preg 5.4.0–5.4.8
phppreg5409	2.0.0	PHP preg 5.4.9–5.4.40
phppreg5441	2.3.0	PHP preg 5.4.41–5.4.45
phppreg5500	2.0.0	PHP preg 5.5.0–5.5.9
phppreg5510	2.0.0	PHP preg 5.5.10–5.5.24
phppreg5525	2.3.0	PHP preg 5.5.25–5.5.26
phppreg5527	2.4.0	PHP preg 5.5.27–5.5.31
phppreg5532	2.5.0	PHP preg 5.5.32–5.5.38
phppreg5600	2.1.0	PHP preg 5.6.0–5.6.8
phppreg5609	2.3.0	PHP preg 5.6.9–5.6.10
phppreg5611	2.4.0	PHP preg 5.6.11–5.6.17
phppreg5618	2.5.0	PHP preg 5.6.18–5.6.40
phppreg7000	2.5.0	PHP preg 7.0.0–7.0.2
phppreg7003	2.5.0	PHP preg 7.0.3
phppreg7004	2.5.0	PHP preg 7.0.4–7.1.33
phppreg7200	2.8.0	PHP preg 7.2.0–7.2.34
phppreg7300	2.9.0	PHP preg 7.3.0–7.3.33
phppreg7400	2.10.0	PHP preg 7.4.0–7.4.11
phppreg7412	2.11.0	PHP preg 7.4.12–7.4.33
phppreg8000	2.11.0	PHP preg 8.0.0–8.1.24
bre	2.0.0	POSIX BRE
ere	2.0.0	POSIX ERE

postgresql	2.0.0	PostgreSQL
pgr2	2.0.0	PowerGREP 2
pgr4	2.0.0	PowerGREP 3 & 4
pgr5	2.5.0	PowerGREP 5
powershell	2.0.0	PowerShell
powershellops	2.4.0	PowerShell operators
python24	2.0.0	Python 2.4–2.6
python27	2.0.0	Python 2.7
python30	2.0.0	Python 3.0
python31	2.0.0	Python 3.1
python32	2.0.0	Python 3.2
python33	2.0.0	Python 3.3
python34	2.0.0	Python 3.4
python35	2.4.0	Python 3.5
python36	2.7.0	Python 3.6
python37	2.8.0	Python 3.7–3.10
python311	<b>2.13.0</b>	Python 3.11–3.12
r2140	2.0.0	R 2.14.0–2.14.1
r2142	2.0.0	R 2.14.2
r2150	2.0.0	R 2.15.0–3.0.2
r3003	2.0.0	R 3.0.3–3.1.2
r3013	2.2.0	R 3.1.3–3.4.4
r3050	2.8.0	R 3.5.0–3.6.3
r4000	2.10.0	R 4.0.0–4.1.3
r4020	<b>2.13.0</b>	R 4.2.0–4.2.1
ruby18	2.0.0	Ruby 1.8
ruby19	2.0.0	Ruby 1.9
ruby20	2.0.0	Ruby 2.0–2.1
ruby22	2.2.0	Ruby 2.2–2.3
ruby24	2.7.0	Ruby 2.4–3.2
scala4	2.0.0	Scala (JDK 1.4)
scala5	2.0.0	Scala (JDK 1.5)
scala6	2.0.0	Scala (JDK 1.6)
scala7	2.0.0	Scala (JDK 1.7)
scala8	2.0.0	Scala (JDK 1.8)
scala9	2.8.0	Scala (JDK 9–12)
scala13	2.9.0	Scala (JDK 13–14)
scala15	2.11.0	Scala (JDK 15)
scala16	2.11.0	Scala (JDK 16)
scala17	2.12.0	Scala (JDK 17–18)
scala19	<b>2.13.0</b>	Scala (JDK 19–21)

stdreansicx10	2.4.0	std::regex (C++Builder 10–10.4)
stdreansicb11	2.12.0	std::regex (C++Builder 11)
stdreansicbx3	2.1.0	std::regex (C++Builder Win64 XE3–XE6)
stdreansicbx7	2.1.0	std::regex (C++Builder Win64 XE7–XE8)
stdreansivc2008	2.1.0	std::regex (Visual C++ 2008)
stdreansivc2010	2.1.0	std::regex (Visual C++ 2010)
stdreansivc2012	2.1.0	std::regex (Visual C++ 2012)
stdreansivc2013	2.1.0	std::regex (Visual C++ 2013)
stdreansivc2015	2.4.0	std::regex (Visual C++ 2015)
stdreansivc2017	2.7.0	std::regex (Visual C++ 2017–2022)
stdrewidecx10	2.4.0	std::wregex (C++Builder 10–10.4)
stdrewidecb11	2.12.0	std::wregex (C++Builder 11)
stdrewidecbx3	2.1.0	std::wregex (C++Builder Win64 XE3–XE6)
stdrewidecbx7	2.1.0	std::wregex (C++Builder Win64 XE7–XE8)
stdrewidevc2008	2.1.0	std::wregex (Visual C++ 2008)
stdrewidevc2010	2.1.0	std::wregex (Visual C++ 2010)
stdrewidevc2012	2.1.0	std::wregex (Visual C++ 2012)
stdrewidevc2013	2.4.0	std::wregex (Visual C++ 2013)
stdrewidevc2015	2.4.0	std::wregex (Visual C++ 2015)
stdrewidevc2017	2.7.0	std::wregex (Visual C++ 2017–2022)
tcl84	2.0.0	Tcl 8.4
tcl85	2.2.0	Tcl 8.5
tcl86	2.0.0	Tcl 8.6
vbscript	2.0.0	VBScript
vbnet11	2.0.0	Visual Basic (.NET 1.1)
vbnet20	2.0.0	Visual Basic (.NET 2.0–7.0)
vb6	2.0.0	Visual Basic 6
vs2012	2.0.0	Visual Studio 2012–2022 IDE
wxWidgets	2.0.0	wxWidgets
xml	2.0.0	XML Schema
xpath	2.0.0	XPath
xregexp2chrome	2.0.0	XRegExp 2 (Chrome)
xregexp2edge	2.4.0	XRegExp 2 (Edge)
xregexp2firefox	2.0.0	XRegExp 2 (Firefox)
xregexp2quirks	2.0.0	XRegExp 2 (MSIE quirks)
xregexp2msie	2.0.0	XRegExp 2 (MSIE standard)
xregexp2opera	2.0.0	XRegExp 2 (Opera)
xregexp3chrome	2.4.0	XRegExp 3 (Chrome)
xregexp3edge	2.4.0	XRegExp 3 (Edge)
xregexp3firefox	2.4.0	XRegExp 3 (Firefox)
xregexp3quirks	2.4.0	XRegExp 3 (MSIE quirks)

xregexp3msie	2.4.0	XRegExp 3 (MSIE standard)
xregexp3opera	2.4.0	XRegExp 3 (Opera)
xregexp4chrome	2.8.0	XRegExp 4 (Chrome)
xregexp4edge	2.8.0	XRegExp 4 (Edge)
xregexp4firefox	2.8.0	XRegExp 4 (Firefox)
xregexp4msie	2.8.0	XRegExp 4 (MSIE standard)
xregexp4opera	2.8.0	XRegExp 4 (Opera)
xregexp5chrome	2.11.0	XRegExp 5 (Chrome)
xregexp5edge	2.11.0	XRegExp 5 (Edge)
xregexp5firefox	2.11.0	XRegExp 5 (Firefox)
xregexp5msie	2.11.0	XRegExp 5 (MSIE standard)

## Regex Flavor Identifiers

Identifiers for the `-flavor` command line parameter and for the custom application variant structure in the COM interface.

Identifier	Age	Regex Flavor
jgsoft5	2.0.0	JGsoft V2 (EditPad 8; PowerGREP 5)
jgsoft	1.0.0	JGsoft (PowerGREP 3 & 4)
jgsoft3	2.0.0	JGsoft (EditPad 6 & 7)
jgsoftpcr	2.0.0	JGsoft PCRE (EditPad 5; PowerGREP 2)
dotnet11	2.0.0	.NET 1.1
dotnet	1.0.0	.NET 2.0–7.0
vs2012	2.0.0	Visual Studio 2012–2022 IDE
dotnetecma11	2.4.0	.NET 1.1 (ECMAScript)
dotnetecma20	2.4.0	.NET 2.0–7.0 (ECMAScript)
java4	2.0.0	Java 4
java	1.0.0	Java 5
java6	2.0.0	Java 6
java7	2.0.0	Java 7
java8	2.0.0	Java 8
java9	2.8.0	Java 9–12
java13	2.9.0	Java 13–14
java15	2.11.0	Java 15
java16	2.11.0	Java 16
java17	2.12.0	Java 17–18
java19	<b>2.13.0</b>	Java 19–21
perl	1.0.0	Perl 5.8
perl510	2.0.0	Perl 5.10
perl512	2.0.0	Perl 5.12

perl514	2.0.0	Perl 5.14–5.16
perl518	2.0.0	Perl 5.18
perl520	2.1.0	Perl 5.20
perl522	2.3.0	Perl 5.22
perl524	2.7.0	Perl 5.24
perl526	2.7.0	Perl 5.26–5.28
perl530	2.9.0	Perl 5.30–5.32
pcre40	2.0.0	PCRE 4.0–4.4
pcre45	2.7.0	PCRE 4.5
pcre50	2.0.0	PCRE 5.0–6.4
pcre65	2.0.0	PCRE 6.5–6.6
pcre	1.0.0	PCRE 6.7
pcre70	2.0.0	PCRE 7.0
pcre71	2.0.0	PCRE 7.1
pcre72	2.0.0	PCRE 7.2–7.3
pcre74	2.0.0	PCRE 7.4
pcre75	2.0.0	PCRE 7.5–7.6
pcre77	2.0.0	PCRE 7.7–7.9
pcre800	2.0.0	PCRE 8.00
pcre801	2.0.0	PCRE 8.01–8.02
pcre810	2.0.0	PCRE 8.10
pcre811	2.0.0	PCRE 8.11–8.12
pcre813	2.0.0	PCRE 8.13
pcre820	2.0.0	PCRE 8.20
pcre821	2.0.0	PCRE 8.21
pcre830	2.0.0	PCRE 8.30–8.33
pcre834	2.0.0	PCRE 8.34–8.35
pcre836	2.2.0	PCRE 8.36–8.38
pcre839	2.7.0	PCRE 8.39
pcre840	2.7.0	PCRE 8.40–8.45
PCRE1010	2.2.0	PCRE2 10.10
pcre1020	2.4.0	PCRE2 10.20
pcre1021	2.5.0	PCRE2 10.21–10.22
pcre1023	2.7.0	PCRE2 10.23
pcre1030	2.8.0	PCRE2 10.30–10.31
pcre1032	2.9.0	PCRE2 10.32–10.34
pcre1035	2.11.0	PCRE2 10.35–10.39
delphixe	2.0.0	Delphi XE (TRegEx)
delphixe2	2.0.0	Delphi XE2–XE5 (TRegEx)
delphixe6	2.0.0	Delphi XE6 (TRegEx)
delphixe7	2.1.0	Delphi XE7–XE8 & 10–10.1 (TRegEx)

delphi102	2.7.0	Delphi 10.2 (TRegEx)
delphi103	2.9.0	Delphi 10.3–11 (TRegEx)
delphi2007	2.0.0	Delphi 2007 (TPerlRegEx)
delphixecore	2.0.0	Delphi XE (TPerlRegEx)
delphixe2core	2.0.0	Delphi XE2–XE3, 2009 (TPerlRegEx)
delphixe4core	2.0.0	Delphi XE4–XE5 (TPerlRegEx)
delphixe6core	2.0.0	Delphi XE6 (TPerlRegEx)
delphixe7core	2.1.0	Delphi XE7–XE8 & 10–10.1 (TPerlRegEx)
delphi102core	2.7.0	Delphi 10.2 (TPerlRegEx)
delphi103core	2.9.0	Delphi 10.3–11 (TPerlRegEx)
phppreg4303	2.0.0	PHP preg 4.3.3–4.3.4
phppreg4305	2.7.0	PHP preg 4.3.5–4.3.11, 5.0.0–5.0.4
phppreg4402	2.0.0	PHP preg 4.4.0–4.4.2, 5.0.5–5.1.2
phppreg4403	2.0.0	PHP preg 4.4.3–4.4.4
phppreg4405	2.0.0	PHP preg 4.4.5
phppreg4406	2.0.0	PHP preg 4.4.6–4.4.8
phppreg5103	2.0.0	PHP preg 5.1.3–5.1.6
phppreg5200	2.0.0	PHP preg 5.2.0–5.2.1
phppreg5202	2.0.0	PHP preg 5.2.2–5.2.3
phppreg5204	2.0.0	PHP preg 5.2.4
phppreg5205	2.0.0	PHP preg 5.2.5
phppreg5206	2.0.0	PHP preg 5.2.6
phppreg5207	2.0.0	PHP preg 5.2.7–5.2.13, 5.3.0–5.3.1
phppreg5214	2.0.0	PHP preg 5.2.14–5.2.17, 5.3.3
phppreg5303	2.0.0	PHP preg 5.3.2
phppreg5304	2.0.0	PHP preg 5.3.4–5.3.5
phppreg5306	2.0.0	PHP preg 5.3.6–5.3.18, 5.4.0–5.4.8
phppreg5319	2.0.0	PHP preg 5.3.19–5.3.29, 5.4.9–5.4.40, 5.5.0–5.5.9
phppreg5510	2.0.0	PHP preg 5.5.10–5.5.24, 5.6.0–5.6.8
phppreg5609	2.3.0	PHP preg 5.4.41–5.4.45, 5.5.25–5.5.26, 5.6.9–5.6.10
phppreg5611	2.4.0	PHP preg 5.5.27–5.5.31, 5.6.11–5.6.17
phppreg5618	2.5.0	PHP preg 5.5.32–5.5.38, 5.6.18–5.6.40
phppreg7000	2.5.0	PHP preg 7.0.0–7.0.2
phppreg7003	2.5.0	PHP preg 7.0.3–7.1.33
phppreg7200	2.8.0	PHP preg 7.2.0–7.2.34
phppreg7300	2.9.0	PHP preg 7.3.0–7.3.33
phppreg7400	2.10.0	PHP preg 7.4.0–7.4.11
phppreg7412	2.11.0	PHP preg 7.4.12–7.4.33
phppreg8000	2.11.0	PHP preg 8.0.0–8.1.24
r2140	2.0.0	R 2.14.0–2.14.1
r2142	2.0.0	R 2.14.2

r2150	2.0.0	R 2.15.0–3.0.2
r3003	2.0.0	R 3.0.3–3.1.2
r3013	2.2.0	R 3.1.3–3.4.4
r3050	2.8.0	R 3.5.0–3.6.3
r4000	2.10.0	R 4.0.0–4.1.3
r4020	<b>2.13.0</b>	R 4.2.0–4.2.1
chrome	2.0.0	JavaScript (Chrome & Firefox)
javascript	1.0.0	JavaScript (MSIE standard)
quirks	2.0.0	JavaScript (MSIE quirks)
html5chrome	2.0.0	HTML5 Pattern (Chrome & Firefox)
html5	2.0.0	HTML5 Pattern (MSIE standard)
xregexp2firefox	2.0.0	XRegExp 2 (Chrome & Firefox)
xregexp2	2.0.0	XRegExp 2 (MSIE standard)
xregexp2quirks	2.0.0	XRegExp 2 (MSIE quirks)
xregexp3firefox	2.4.0	XRegExp 3 (Chrome & Firefox)
xregexp3	2.4.0	XRegExp 3 (MSIE standard)
xregexp3quirks	2.4.0	XRegExp 3 (MSIE quirks)
xregexp4firefox	2.8.0	XRegExp 4 (Chrome & Firefox)
xregexp4msie	2.8.0	XRegExp 4 (MSIE standard)
xregexp5firefox	2.11.0	XRegExp 5 (Chrome & Firefox)
xregexp5msie	2.11.0	XRegExp 5 (MSIE standard)
python	1.0.0	Python 2.4–2.6
python27	2.0.0	Python 2.7
python30	2.0.0	Python 3.0–3.1
python32	2.0.0	Python 3.2
python33	2.0.0	Python 3.3–3.4
python35	2.4.0	Python 3.5
python36	2.7.0	Python 3.6
python37	2.8.0	Python 3.7–3.10
python311	<b>2.13.0</b>	Python 3.11–3.12
ruby	1.0.0	Ruby 1.8
ruby19	2.0.0	Ruby 1.9
ruby20	2.0.0	Ruby 2.0–2.1
ruby22	2.2.0	Ruby 2.2–2.3
ruby24	2.7.0	Ruby 2.4–3.2
tcl	1.0.0	Tcl ARE 8.4
tcl85	2.2.0	Tcl ARE 8.5
tcl86	2.0.0	Tcl ARE 8.6
postgres	2.0.0	PostgreSQL ARE
tclbre84	2.0.0	Tcl BRE 8.4
tclbre85	2.2.0	Tcl BRE 8.5

tclbre86	2.0.0	Tcl BRE 8.6
postgresbre	2.0.0	PostgreSQL BRE
tclere84	2.0.0	Tcl ERE 8.4
tclere85	2.2.0	Tcl ERE 8.5
tclere86	2.0.0	Tcl ERE 8.6
postgresere	2.0.0	PostgreSQL ERE
phpereg	2.0.0	PHP ereg 4.3.3–5.2.17
bre	1.0.0	POSIX BRE
ere	1.0.0	POSIX ERE
gnubre	1.0.0	GNU BRE
gnuer	1.0.0	GNU ERE
oracle10gR1	2.0.0	Oracle 10gR1
oracle	1.0.0	Oracle 10gR2–12cR1
xml	1.0.0	XML Schema
xpath	1.0.0	XPath
stdreecmaansivc2008	2.1.0	std::regex ECMAScript (Visual C++ 2008)
stdreecmaansivc2010	2.1.0	std::regex ECMAScript (Visual C++ 2010)
stdreecmaansivc2012	2.1.0	std::regex ECMAScript (Visual C++ 2012)
stdreecmaansivc2013	2.1.0	std::regex ECMAScript (Visual C++ 2013)
stdreecmaansivc2015	2.4.0	std::regex ECMAScript (Visual C++ 2015 & C++Builder 10–10.4)
stdreecmaansivc2017	2.7.0	std::regex ECMAScript (Visual C++ 2017–2022)
stdreecmaansicbx3	2.1.0	std::regex ECMAScript (C++Builder Win64 XE3–XE6)
stdreecmaansicbx7	2.1.0	std::regex ECMAScript (C++Builder Win64 XE7–XE8)
stdreecmaansicb11	2.12.0	std::regex ECMAScript (C++Builder 11)
stdreecmawidevc2008	2.1.0	std::wregex ECMAScript (Visual C++ 2008)
stdreecmawidevc2010	2.1.0	std::wregex ECMAScript (Visual C++ 2010)
stdreecmawidevc2012	2.1.0	std::wregex ECMAScript (Visual C++ 2012–2013)
stdreecmawidevc2015	2.4.0	std::wregex ECMAScript (Visual C++ 2015)
stdreecmawidevc2017	2.7.0	std::wregex ECMAScript (Visual C++ 2017–2022)
stdreecmawidecbx3	2.1.0	std::wregex ECMAScript (C++Builder Win64 XE3–XE6)
stdreecmawidecbx7	2.1.0	std::wregex ECMAScript (C++Builder Win64 XE7–XE8)
stdreecmawidecx10	2.4.0	std::wregex ECMAScript (C++Builder 10–10.4)
stdreecmawidecb11	2.12.0	std::wregex ECMAScript (C++Builder 11)
stdrebreansivc2008	2.1.0	std::regex basic (Visual C++ 2008)
stdrebreansivc2010	2.1.0	std::regex basic (Visual C++ 2010)
stdrebreansivc2012	2.6.0	std::regex basic (Visual C++ 2012–2013)
stdrebreansivc2015	2.4.0	std::regex basic (Visual C++ 2015 & C++Builder 10–10.4)
stdrebreansicbx3	2.1.0	std::regex basic (C++Builder Win64 XE3–XE6)
stdrebreansicbx7	2.6.0	std::regex basic (C++Builder Win64 XE7–XE8)
stdrebreansicb11	2.12.0	std::regex basic (C++Builder 11)
stdrebrewidevc2008	2.1.0	std::wregex basic (Visual C++ 2008)

<code>stdrebrewidevc2010</code>	2.1.0	<code>std::wregex basic</code> (Visual C++ 2010)
<code>stdrebrewidevc2012</code>	2.6.0	<code>std::wregex basic</code> (Visual C++ 2012–2015)
<code>stdrebrewidecbxe3</code>	2.1.0	<code>std::wregex basic</code> (C++Builder Win64 XE3–XE6)
<code>stdrebrewidecbxe7</code>	2.6.0	<code>std::wregex basic</code> (C++Builder Win64 XE7–XE8 & C++Builder 10–10.4)
<code>stdrebrewidecb11</code>	2.12.0	<code>std::wregex basic</code> (C++Builder 11)
<code>stdregrepansivc2008</code>	2.1.0	<code>std::regex grep</code> (Visual C++ 2008)
<code>stdregrepansivc2010</code>	2.1.0	<code>std::regex grep</code> (Visual C++ 2010)
<code>stdregrepansivc2012</code>	2.6.0	<code>std::regex grep</code> (Visual C++ 2012–2013)
<code>stdregrepansivc2015</code>	2.4.0	<code>std::regex grep</code> (Visual C++ 2015 & C++Builder 10–10.4)
<code>stdregrepansicbxe3</code>	2.1.0	<code>std::regex grep</code> (C++Builder Win64 XE3–XE6)
<code>stdregrepansicbxe7</code>	2.6.0	<code>std::regex grep</code> (C++Builder Win64 XE7–XE8)
<code>stdregrepansicb11</code>	2.12.0	<code>std::regex grep</code> (C++Builder 11)
<code>stdregrepwidevc2008</code>	2.1.0	<code>std::wregex grep</code> (Visual C++ 2008)
<code>stdregrepwidevc2010</code>	2.1.0	<code>std::wregex grep</code> (Visual C++ 2010)
<code>stdregrepwidevc2012</code>	2.6.0	<code>std::wregex grep</code> (Visual C++ 2012–2015)
<code>stdregrepwidecbxe3</code>	2.1.0	<code>std::wregex grep</code> (C++Builder Win64 XE3–XE6)
<code>stdregrepwidecbxe7</code>	2.6.0	<code>std::wregex grep</code> (C++Builder Win64 XE7–XE8 & C++Builder 10–10.4)
<code>stdregrepwidecb11</code>	2.12.0	<code>std::wregex grep</code> (C++Builder 11)
<code>stdreereansivc2008</code>	2.1.0	<code>std::regex extended</code> (Visual C++ 2008)
<code>stdreereansivc2010</code>	2.1.0	<code>std::regex extended</code> (Visual C++ 2010–2013)
<code>stdreereansivc2015</code>	2.4.0	<code>std::regex extended</code> (Visual C++ 2015 & C++Builder 10–10.4)
<code>stdreereansicbxe3</code>	2.1.0	<code>std::regex extended</code> (C++Builder Win64 XE3–XE8)
<code>stdreereansicb11</code>	2.12.0	<code>std::regex extended</code> (C++Builder 11)
<code>stdreerewidevc2008</code>	2.1.0	<code>std::wregex extended</code> (Visual C++ 2008)
<code>stdreerewidevc2010</code>	2.1.0	<code>std::wregex extended</code> (Visual C++ 2010–2015)
<code>stdreerewidecbxe3</code>	2.1.0	<code>std::wregex extended</code> (C++Builder Win64 XE3–XE8 & C++Builder 10–10.4)
<code>stdreerewidecb11</code>	2.12.0	<code>std::wregex extended</code> (C++Builder 11)
<code>stdreegrepansivc2008</code>	2.1.0	<code>std::regex egrep</code> (Visual C++ 2008)
<code>stdreegrepansivc2010</code>	2.1.0	<code>std::regex egrep</code> (Visual C++ 2010–2013)
<code>stdreegrepansivc2015</code>	2.4.0	<code>std::regex egrep</code> (Visual C++ 2015 & C++Builder 10–10.4)
<code>stdreegrepansicbxe3</code>	2.1.0	<code>std::regex egrep</code> (C++Builder Win64 XE3–XE8)
<code>stdreegrepansicb11</code>	2.12.0	<code>std::regex egrep</code> (C++Builder 11)
<code>stdreegrepwidevc2008</code>	2.1.0	<code>std::wregex egrep</code> (Visual C++ 2008)
<code>stdreegrepwidevc2010</code>	2.1.0	<code>std::wregex egrep</code> (Visual C++ 2010–2015)
<code>stdreegrepwidecbxe3</code>	2.1.0	<code>std::wregex egrep</code> (C++Builder Win64 XE3–XE8 & C++Builder 10–10.4)
<code>stdreegrepwidecb11</code>	2.12.0	<code>std::wregex egrep</code> (C++Builder 11)
<code>stdreawkansivc2008</code>	2.1.0	<code>std::regex awk</code> (Visual C++ 2008)
<code>stdreawkansivc2010</code>	2.1.0	<code>std::regex awk</code> (Visual C++ 2010–2012)
<code>stdreawkansivc2013</code>	2.1.0	<code>std::regex awk</code> (Visual C++ 2013)
<code>stdreawkansivc2015</code>	2.4.0	<code>std::regex awk</code> (Visual C++ 2015 & C++Builder 10–10.4)
<code>stdreawkansicbxe3</code>	2.1.0	<code>std::regex awk</code> (C++Builder Win64 XE3–XE6)

stdreawkansi1038	2.1.0	std::regex awk (C++Builder Win64 XE7–XE8)
stdreawkansi1042	2.12.0	std::regex awk (C++Builder 11)
stdreawkwidevc2008	2.1.0	std::wregex awk (Visual C++ 2008)
stdreawkwidevc2010	2.1.0	std::wregex awk (Visual C++ 2010–2015)
stdreawkwidecbxe3	2.1.0	std::wregex awk (C++Builder Win64 XE3–XE8 & C++Builder 10–10.4)
stdreawkwidecb11	2.12.0	std::wregex awk (C++Builder 11)
boostecmaansi1038	2.6.0	boost::regex ECMAScript 1.38–1.39
boostecmaansi1042	2.6.0	boost::regex ECMAScript 1.42–1.43
boostecmaansi1044	2.6.0	boost::regex ECMAScript 1.44–1.46
boostecmaansi1047	2.6.0	boost::regex ECMAScript 1.47
boostecmaansi1048	2.7.0	boost::regex ECMAScript 1.48–1.49
boostecmaansi1050	2.12.0	boost::regex ECMAScript 1.50–1.53
boostecmaansi1054	2.6.0	boost::regex ECMAScript 1.54–1.57
boostecmaansi1058	2.6.0	boost::regex ECMAScript 1.58–1.59
boostecmaansi1060	2.6.0	boost::regex ECMAScript 1.60–1.61
boostecmaansi1062	2.6.0	boost::regex ECMAScript 1.62–1.63
boostecmaansi1064	2.7.0	boost::regex ECMAScript 1.64–1.65
boostecmaansi1066	2.8.0	boost::regex ECMAScript 1.66–1.77
boostecmaansi1078	<b>2.13.0</b>	boost::regex ECMAScript 1.78–1.83
boostecmawide1038	2.6.0	boost::wregex ECMAScript 1.38–1.39
boostecmawide1042	2.6.0	boost::wregex ECMAScript 1.42–1.43
boostecmawide1044	2.6.0	boost::wregex ECMAScript 1.44–1.46
boostecmawide1047	2.6.0	boost::wregex ECMAScript 1.47
boostecmawide1048	2.7.0	boost::wregex ECMAScript 1.48–1.49
boostecmawide1050	2.12.0	boost::wregex ECMAScript 1.50–1.53
boostecmawide1054	2.6.0	boost::wregex ECMAScript 1.54–1.57
boostecmawide1058	2.6.0	boost::wregex ECMAScript 1.58–1.59
boostecmawide1060	2.6.0	boost::wregex ECMAScript 1.60–1.63
boostecmawide1064	2.7.0	boost::wregex ECMAScript 1.64–1.65
boostecmawide1066	2.8.0	boost::wregex ECMAScript 1.66–1.77
boostecmawide1078	<b>2.13.0</b>	boost::wregex ECMAScript 1.78–1.83
boostbreansi1038	2.6.0	boost::regex basic 1.38–1.39
boostbreansi1042	2.6.0	boost::regex basic 1.42–1.61
boostbreansi1062	2.6.0	boost::regex basic 1.62–1.83
boostbrewide1038	2.6.0	boost::wregex basic 1.38–1.39
boostbrewide1042	2.6.0	boost::wregex basic 1.42–1.59
boostbrewide1060	2.6.0	boost::wregex basic 1.60–1.83
boostgrepansi1038	2.6.0	boost::regex grep 1.38–1.39
boostgrepansi1042	2.6.0	boost::regex grep 1.42–1.61
boostgrepansi1062	2.6.0	boost::regex grep 1.62–1.83
boostgrepwide1038	2.6.0	boost::wregex grep 1.38–1.39

boostgrepwide1042	2.6.0	boost::wregex grep 1.42–1.59
boostgrepwide1060	2.6.0	boost::wregex grep 1.60–1.83
boostereansi1038	2.6.0	boost::regex extended 1.38–1.39
boostereansi1042	2.6.0	boost::regex extended 1.42–1.61
boostereansi1062	2.6.0	boost::regex extended 1.62–1.77
boostereansi1078	<b>2.13.0</b>	boost::regex extended 1.78–1.83
boosterewide1038	2.6.0	boost::wregex extended 1.38–1.39
boosterewide1042	2.6.0	boost::wregex extended 1.42–1.59
boosterewide1060	2.6.0	boost::wregex extended 1.60–1.77
boosterewide1078	<b>2.13.0</b>	boost::wregex extended 1.78–1.83
boostegrepansi1038	2.6.0	boost::regex egrep 1.38–1.39
boostegrepansi1042	2.6.0	boost::regex egrep 1.42–1.61
boostegrepansi1062	2.6.0	boost::regex egrep 1.62–1.77
boostegrepansi1078	<b>2.13.0</b>	boost::regex egrep 1.78–1.83
boostegrepwide1038	2.6.0	boost::wregex egrep 1.38–1.39
boostegrepwide1042	2.6.0	boost::wregex egrep 1.42–1.59
boostegrepwide1060	2.6.0	boost::wregex egrep 1.60–1.77
boostegrepwide1078	<b>2.13.0</b>	boost::wregex egrep 1.78–1.83
boostawkansi1038	2.6.0	boost::regex awk 1.38–1.39
boostawkansi1042	2.6.0	boost::regex awk 1.42–1.61
boostawkansi1062	2.6.0	boost::regex awk 1.62–1.77
boostawkansi1078	<b>2.13.0</b>	boost::regex awk 1.78–1.83
boostawkwide1038	2.6.0	boost::wregex awk 1.38–1.39
boostawkwide1042	2.6.0	boost::wregex awk 1.42–1.59
boostawkwide1060	2.6.0	boost::wregex awk 1.60–1.77
boostawkwide1078	<b>2.13.0</b>	boost::wregex awk 1.78–1.83

## Replacement Flavor Identifiers

Identifiers for the `-flavorreplace` command line parameter and for the custom application variant structure in the COM interface.

Identifier	Age	Replace Flavor
jpgsoft5	2.0.0	JGsoft V2 (EditPad 8; PowerGREP 5)
jpgsoft	1.0.0	JGsoft (EditPad 6 & 7; PowerGREP 3 & 4)
jpgsoftpcr	2.0.0	JGsoft (EditPad 5; PowerGREP 2)
delphixe	2.1.0	Delphi XE–XE3 (TRegEx)
delphixe4	2.0.0	Delphi XE4–XE5 (TRegEx)
delphi	1.0.0	Delphi 10.2–11 (TRegEx & TPerlRegEx) & Delphi 2007–2009 (TPerlRegEx)
delphixecore	2.1.0	Delphi XE–XE3 (TPerlRegEx)

delphixe4core	2.1.0	Delphi XE6–10.1 (TRegEx & TPerlRegEx) & Delphi XE4–XE5 (TPerlRegEx)
dotnet	1.0.0	.NET 1.1–7.0
dotnetecma	2.4.0	.NET 1.1–7.0 (ECMAScript)
java	1.0.0	Java 4–6
java7	2.0.0	Java 7–21
perl	1.0.0	Perl 5.8
perl510	2.0.0	Perl 5.10–5.12
perl514	2.0.0	Perl 5.14–5.16
perl518	2.0.0	Perl 5.18–5.20
perl522	2.3.0	Perl 5.22–5.32
pcre1010	2.2.0	PCRE2 10.10–10.20
pcre1021	2.5.0	PCRE2 10.21–10.39
pcrex1021	2.5.0	PCRE2 extended 10.21–10.39
javascript	1.0.0	JavaScript (Chrome)
firefox	2.0.0	JavaScript (Firefox)
msie	2.0.0	JavaScript (MSIE standard)
quirks	2.0.0	JavaScript (MSIE quirks)
xregexp2firefox	2.4.0	XRegExp 2–3 (Chrome & Firefox)
xregexp2	2.0.0	XRegExp 2-3 (MSIE standard & quirks)
xregexp4firefox	2.12.0	XRegExp 4 (Chrome & Firefox)
xregexp4msie	2.12.0	XRegExp 4 (MSIE standard)
xregexp5firefox	2.11.0	XRegExp 5 (Chrome & Firefox)
xregexp5msie	2.11.0	XRegExp 5 (MSIE standard)
vbscript	1.0.0	VBScript
python	1.0.0	Python 2.4–3.4
python35	2.4.0	Python 3.5–3.6
python37	2.8.0	Python 3.7–3.12
ruby	1.0.0	Ruby 1.8
ruby19	2.0.0	Ruby 1.9–3.2
tcl	1.0.0	Tcl 8.4–8.5
tcl86	2.0.0	Tcl 8.6
phpereg	1.0.0	PHP ereg 4.3.3–5.2.17
phppreg	1.0.0	PHP preg 4.3.3–5.6.40
phppreg7	2.5.0	PHP preg 7.0.0–8.1.24
r	1.0.0	R 2.14.0–3.6.3
r4000	2.10.0	R 4.0.0–4.2.1
realbasic	1.0.0	REALbasic/Xojo
oracle	1.0.0	Oracle
postgres	1.0.0	PostgreSQL
xpath	1.0.0	XPath
stdreansivc	2.1.0	std::regex ECMAScript (Visual C++ 2008–2013)

stdreansivc2015	2.4.0	std::regex ECMAScript (Visual C++ 2015–2022 & C++Builder 10–10.4)
stdreansicb	2.1.0	std::regex ECMAScript (C++Builder Win64 XE3–XE8)
stdrewide	2.1.0	std::wregex ECMAScript (Visual C++ 2008–2013 & C++Builder Win64 XE3–XE8)
stdrewidevc2015	2.4.0	std::wregex ECMAScript (Visual C++ 2015–2022 & C++Builder 10–11)
stdresedansivc2008	2.5.0	std::regex sed (Visual C++ 2008–2013)
stdresedansivc2015	2.5.0	std::regex sed (Visual C++ 2015–2022 & C++Builder 10–10.4)
stdresedansicbx3	2.5.0	std::regex sed (C++Builder Win64 XE3–XE8)
stdresedwidevc2008	2.5.0	std::wregex sed (Visual C++ 2008–2013 & C++Builder Win64 XE3–XE8)
stdresedwidevc2015	2.5.0	std::wregex sed (Visual C++ 2015–2022 & C++Builder 10–11)
boostallansi1038	2.6.0	boost::regex all 1.38–1.39
boostallansi1042	2.6.0	boost::regex all 1.42–1.83
boostallwide1038	2.6.0	boost::wregex all 1.38–1.39
boostallwide1042	2.6.0	boost::wregex all 1.42–1.83
boostecmaansi1038	2.6.0	boost::regex ECMAScript 1.38–1.39
boostecmaansi1042	2.6.0	boost::regex ECMAScript 1.42–1.83
boostecmawide1038	2.6.0	boost::wregex ECMAScript 1.38–1.39
boostecmawide1042	2.6.0	boost::wregex ECMAScript 1.42–1.83
boostsedansi1038	2.6.0	boost::regex sed 1.38–1.39 & 1.42–1.83
boostsedwide1038	2.6.0	boost::wregex sed 1.38–1.39 & 1.42–1.83

## Split Flavor Identifiers

Identifiers for the `-flavorsplit` command line parameter and for the custom application variant structure in the COM interface.

Identifier	Age	Split Flavor
delphi	2.0.0	Delphi XE–XE5 (TRegEx)
delphixe6	2.0.0	Delphi XE6–XE8 & 10–11 (TRegEx)
tperlregex	2.0.0	Delphi 2007–XE8 & 10–11 (TPerlRegEx)
dotnet11	2.0.0	.NET 1.1
dotnet	2.0.0	.NET 2.0–7.0
java	2.0.0	Java 4–7
java8	2.0.0	Java 8–21
perl	2.0.0	Perl 5.8–5.32
javascript	2.0.0	JavaScript (Chrome & Firefox)
msie	2.0.0	JavaScript (MSIE standard)
quirks	2.0.0	JavaScript (MSIE quirks)
python	2.0.0	Python 2.4–3.4
python35	2.4.0	Python 3.5–3.6
python37	2.8.0	Python 3.7–3.12

ruby	2.0.0	Ruby 1.8–3.2
phpereg	2.0.0	PHP ereg 4.3.3–5.2.17
phppreg	2.0.0	PHP preg 4.3.3–5.6.40, 7.0.5–8.1.24
phppreg7000	2.5.0	PHP preg 7.0.0–7.0.4
r	2.0.0	R 2.14.0–4.2.1
postgres	2.0.0	PostgreSQL
xpath	2.0.0	XPath

## String Style Identifiers

Identifiers for the `-string` command line parameter and for the `StringType` parameter in the COM interface.

Identifier	Age	String Style
asis	1.0.0	as is
basic	1.0.0	Basic string
c	1.0.0	C string
cL	2.1.0	C wide string
c11	2.4.0	C++11 raw string
c11L	2.4.0	C++11 wide raw string
c#	1.0.0	C# string
pascal	1.0.0	Delphi string
prism	1.0.0	Delphi Prism string
groovy	1.0.0	Groovy string
java	1.0.0	Java string
javascriptstring	1.0.0	JavaScript string
javascript	1.0.0	JavaScript // operator
perl	1.0.0	Perl string
perlop	1.0.0	Perl m// or s/// operator
phpereg	1.0.0	PHP string
phppreg	1.0.0	PHP "/" string
postgres	1.0.0	PostgreSQL string
powershell	1.0.0	PowerShell string
python	1.0.0	Python string
r	2.0.0	R string
ruby	1.0.0	Ruby // operator
scala	1.0.0	Scala string
sql	1.0.0	SQL string
tcl	1.0.0	Tcl word
xml	1.0.0	XML

## Text File Encoding Identifiers

<b>Identifier</b>	<b>Encoding</b>
bytes	Binary file to be opened in hexadecimal mode
utf8	Unicode, UTF-8
utf32le	Unicode, UTF-32 little endian
utf32be	Unicode, UTF-32 big endian
utf16le	Unicode, UTF-16 little endian
utf16be	Unicode, UTF-16 big endian
win1250	Windows 1250: Central European
win1251	Windows 1251: Cyrillic
win1252	Windows 1252: Western European
win1253	Windows 1253: Greek
win1254	Windows 1254: Turkish
win1255	Windows 1255: Hebrew
win1256	Windows 1256: Arabic
win1257	Windows 1257: Baltic
win1258	Windows 1258: Vietnam
win874	Windows 874: Thai
8859-1	ISO-8859-1 Latin-1 Western European
8859-2	ISO-8859-2 Latin-2 Central European
8859-3	ISO-8859-3 Latin-3 South European
8859-4	ISO-8859-4 Latin-4 North European
8859-5	ISO-8859-5 Cyrillic
8859-6	ISO-8859-6 Arabic
8859-7	ISO-8859-7 Greek
8859-8	ISO-8859-8 Hebrew
8859-9	ISO-8859-9 Latin-5 Turkish
8859-10	ISO-8859-10 Latin-6 Nordic
8859-11	ISO-8859-11 Thai (TIS-620)
8859-13	ISO-8859-13 Latin-7 Baltic Rim
8859-14	ISO-8859-14 Latin-8 Celtic
8859-15	ISO-8859-15 Latin-9
8859-16	ISO-8859-16 Latin-10 South-Eastern European
dos437	DOS 437: United States
dos737	DOS 737: Greek
dos775	DOS 775: Baltic Rim
dos850	DOS 850: Western European
dos852	DOS 852: Central European
dos855	DOS 855: Cyrillic
dos857	DOS 857: Turkish

dos860	DOS 860: Portuguese
dos861	DOS 861: Icelandic
dos862	DOS 862: Hebrew
dos863	DOS 863: Canadian French
dos864	DOS 864: Arabic
dos865	DOS 865: Nordic
dos866	DOS 866: Cyrillic Russian
dos869	DOS 869: Greek 2
koi8r	KOI8-R: Russian
koi8u	KOI8-U: Ukrainian
ebcdic037	EBCDIC 037: US & Canada
ebcdic424	EBCDIC 424: Hebrew
ebcdic500	EBCDIC 500: International
ebcdic875	EBCDIC 875: Greek
ebcdic1026	EBCDIC 1026: Turkish
10585	ISO-10585: Armenian
armscii7	ArmSCII-7: Armenian
armscii8	ArmSCII-8: Armenian
armscii8a	ArmSCII-8A: Armenian
geostd8	GEOSTD8: Georgian
isiri-3342	ISIRI 3342: Farsi
kamenicky	Kamenický: Czech & Slovak
kz-1048	KZ-1048: Kazach
mazovia	Mazovia: Polish
mik	MIK: Bulgarian
ptcp154	PTCP154: Cyrillic Asian
tcvn	TCVN: Vietnamese
viscii	VISCII: Vietnamese
vps	VPS: Vietnamese
yuscii-cyr	YUSCII Cyrillic: Yugoslavia
yuscii-lat	YUSCII Latin: Yugoslavia
ascii	US-ASCII (7-bit)
win932	Windows 932: Japanese (Shift-JIS)
win949	Windows 949: Korean
win936	Windows 936: Simplified Chinese (GBK)
win950	Windows 950: Traditional Chinese (Big5)
euc-jp	EUC-JP: Japanese (JIS 201+208)
euc-jp-0212	EUC-JP-212: Japanese (JIS 201+208+212)
euc-kr	EUC-KR: Korean (KS 1001)
euc-cn	EUC-CN: Simplified Chinese (GB 2312)
euc-tw	EUC-TW: Traditional Chinese (CNS 11643)

uffff	ASCII + \uFFFF Unicode Escapes
ncrhex	ASCII + &#xFFFF; NCR
ncrdec	ASCII + &#65535; NCR
htmlentities	ASCII + &htmlchar;
iscii-dev	ISCII Devanagari
iscii-bng	ISCII Bengali & Assamese
iscii-pnj	ISCII Punjabi (Gurmukhi)
iscii-gjr	ISCII Gujarati
iscii-ori	ISCII Oriya
iscii-tml	ISCII Tamil
iscii-tlg	ISCII Telugu
iscii-knd	ISCII Kannada
iscii-mlm	ISCII Malayalam
vni	VNI: Vietnamese
viqr	VIQR: Vietnamese
mac-arabic	Mac Arabic
mac-celtic	Mac Celtic
mac-ce	Mac Central European
mac-croatian	Mac Croatian
mac-cyrillic	Mac Cyrillic
mac-dingbats	Mac Dingbats
mac-farsi	Mac Farsi
mac-gaelic	Mac Gaelic
mac-greek	Mac Greek
mac-hebrew	Mac Hebrew
mac-icelandic	Mac Icelandic
mac-inuit	Mac Inuit
mac-roman	Mac Roman (Western European)
mac-romanian	Mac Romanian
mac-symbol	Mac Symbol
mac-thai	Mac Thai
mac-turkish	Mac Turkish
mac-chinesesimp	Mac Chinese Simplified
mac-chinesetrad	Mac Chinese Traditional
mac-devanagari	Mac Devanagari
mac-gujarati	Mac Gujarati
mac-gurmukhi	Mac Gurmukhi
mac-japanese	Mac Japanese
mac-korean	Mac Korean
utf7	Unicode, UTF-7
2022-jp	ISO-2022-JP: Japanese (JIS 201+208)

2022-jp-1	ISO-2022-JP-1: Japanese (JIS 201+208+212)
2022-jp-2	ISO-2022-JP-2: Japanese multilingual (JIS 201+208+212)
2022-kr	ISO-2022-KR: Korean (KS 1001)
2022-cn	ISO-2022-CN: Chinese (GB 2312 + CNS 11643)
hz	HZ: Simplified Chinese
tscii	TSCII: Tamil



## Index

- .rmf files, 214
- .rmg files, 186
- 3rd party integration, 210
- .rml files, 214
- .rmg files, 214
- .bak files, 200
- Action panel, 151
- ActionScript, 162
- Add button, 183
- alternation, 64, 66
- any text, 57, 110
- anything, 57, 110
- API, 211
- application integration, 210
- Arabic, 203
- arguments
  - command line, 210
- arrange panels, 195
- array, 172
- ASCII, 16, 21, 113, 120
- aspects, 164
- Assistant, 95
- automatic save, 182
- backreference, 55, 109
- backslash, 161
- backup, 185
- backup files, 200
- bak files, 200
- Basic, 162
- basic characters, 16, 113
- bidirectional, 204
- binary, 25, 125
- blinking, 209
- C, 162
- C#, 162, 210
- C++, 162
- C++11, 162
- callback function, 172
- capturing groups, 69
- char, 162
- character masks, 17, 115
- characters, 15, 16, 21, 111, 113, 120
- Chrome, 162
- citizen numbers, 43, 144
- Clear button, 186
- clipboard, 161
- close panels, 193
- code editor, 168
- code snippets, 168
- colors, 200
- COM Automation interface, 215
- command line examples, 210
- command line parameters, 212
- complex script, 203
- configure RegexMagic, 197
- control characters, 21, 120
- copy, 161
- copy all searched files, 185
- copy only modified files, 185
- Country, 34, 138
- country code, 34, 138
- country name, 34, 138
- CR, 100
- credit card number, 40, 142
- CRLF, 100
- Currency, 38, 141
- currency code, 38, 141
- currency symbols, 22, 121
- cursor, 204
- cursor configuration, 207
- date, 28, 127
- date and time, 28, 127
- decimal, 22, 25, 121, 125
- default layout, 195
- delete backup files, 186
- Delphi, 88, 162, 210
- details of a match, 171
- discussion group, 187
- domains, 30, 131
- double quote, 161
- Dual Monitor Layout, 196
- ECMA-262, 162
- ECMAScript, 162
- email address, 30, 131
- entirely matching, 171
- ereg, 163
- escape, 161
- examples
  - command line, 210
- exception handling, 169, *see* exception handling,
  - see* exception handling
- exclude files, 200
- exclude folders, 200
- execute, 185
- exponents, 22, 121
- Export button, 186

- feeds, 192
- fields, 59
  - alternation, 64, 66
  - mark, 97, 102
  - replace, 71
  - sequence, 61, 66
- file
  - exclude, 200
  - hide, 200
- file mask, 184
- files
  - hidden, 199
  - system, 199
- flag, 208
- flavors, 164
- floating panels, 195
- folder
  - exclude, 200
  - hide, 200
- folders, 184
- font, 205
- forum, 187
- Generate button, 159
- globally unique identifier, 50, 148
- GREP button, 185
- GREP panel, 184
- Groovy, 162
- GUID, 50, 148
- handle exceptions, 169
- Hebrew, 203
- hexadecimal, 25, 125
- hidden files, 199
- hidden history, 185
- hide files, 200
- hide folders, 200
- hide panels, 193
- icons, 195
- IDE, 168
- identifiers, 222
- identity numbers, 43, 144
- ImageIndex, 88
- implementation, 168
- Indic, 203
- integer, 25, 125
- integration with other tools, 210
- internet address, 32, 47, 133, 146
- invert mask, 184
- invert results, 184
- IPv4 address, 47, 146
- ISO 3166, 34, 138
- ISO 4217, 38, 141
- Java, 162
- JavaScript, 162
- JScript, 162
- language
  - template, 174
- languages, 161, 168
- Large Toolbar Icons, 195
- launch RegexMagic, 210
- layout, 195
- left-to-right, 203, 208
- LF, 100
- libraries, 168
- library, 182
- license plate numbers, 43, 144
- line breaks, 100, 197
- line feed, 100
- line-based, 184
- list of literal text, 18, 116
- literal bytes, 20, 119
- literal text, 18, 19, 116, 118
- Mark button, 97
- masks, 17, 115
- match anything, 57, 110
- match details, 171
- Match panel, 101
- matching entirely, 171
- modify libraries, 182
- monospaced, 203
- More applications and languages, 158
- multi .bak, 185
- multi backup N of, 185
- multi-monitor, 193
- multiple applications, 164
- multiple versions, 164
- MySQL, 163
- national id, 43, 144
- newline, 100
- news feeds, 192
- no backups, 185
- number, 22, 121
- object with regex, 173
- octal, 25, 125
- Office 2003 Display Style, 195
- online forum, 187
- Open button, 182, 183, 186
- open libraries, 182, 183
- open panels, 193
- operator, 163
- options, 197
- Oxygene, 162
- Pascal, 162
- paste, 161
- pattern

- basic characters, 16, 113
- character masks, 17, 115
- control characters, 21, 120
- credit card number, 40, 142
- date and time, 28, 127
- email address, 30, 131
- GUID, 50, 148
- integer, 25, 125
- IPv4 address, 47, 146
- list of literal text, 18, 116
- literal bytes, 20, 119
- literal text, 19, 118
- match anything, 57, 110
- national id, 43, 144
- number, 22, 121
- pattern used by another field, 53, 108
- regula expression, 149
- regular expression, 52
- state or province, 36, 140
- text matched by another field, 55, 109
- Unicode characters, 15, 111
- URL, 32, 34, 38, 133, 138, 141
- VAT number, 44, 145
- pattern used by another field, 53, 108
- permanent exclusion, 200
- PHP, 163
- PostgreSQL, 163
- PowerGREP, 88
- PowerShell, 163
- preferences, 197
- preg, 163
- preserve state, 199
- preview, 185
- Prism, 162
- programming languages, 161, 168
- province, 36, 140
- Python, 163
- quick execute, 185
- quote, 161
- raise. *see* exception handling
- raw string, 163
- read only, 182
- REALbasic, 162
- regex object, 173
- Regex panel, 157
- RegexMagic Library file, 182
- RegexMagic.rml, 183
- regular expression, 52, 149
- remember state, 199
- replace, 71, 169, 172
- replacement text, 73
- Restore Default Layout, 195
- reuse, 182
- reuse pattern, 53, 108
- right-to-left, 203, 208
- rmf files, 214
- rmg files, 186, 214
- rml files, 214
- RSS feeds, 192
- Ruby, 163
- same file name, 185
- same pattern, 53, 108
- Samples panel, 96
- Save As button, 182
- Save button, 186
- save libraries, 182
- save one file for each searched file, 185
- save regexes, 182
- save results into a single file, 185
- save state, 199
- Scala, 163
- search-and-replace, 71
- second monitor, 193
- send, 161
- sequence, 61, 66
- show panels, 193
- show spaces, 197
- Side by Side Layout, 196
- signs, 22, 121
- single .~??, 185
- single .bak, 185
- single quote, 161
- SKU code, 17, 59
- social security numbers, 43, 144
- software integration, 210
- source code, 168
- spaces, 197
- split, 169, 172
- SQL, 163
- state, 199
- State or province, 36, 140
- store regexes, 182
- syntax coloring, 200
- system files, 199
- tabs, 197
- target, 185
- task, 168
- tax numbers, 43, 144
- Tcl, 163
- template, 174
- text cursor configuration, 207
- text layout, 202
- text masks, 17, 115
- text matched by another field, 55, 109

text mode, 100  
thousand separators, 22, 121  
throw. *see* exception handling  
TImageList, 88  
time, 28, 127  
tool integration, 210  
toolbar icons, 195  
transfer, 161  
type library, 215  
undo GREP, 185  
Unicode characters, 15, 111  
URL, 32, 133  
use, 168  
Use button, 183  
users, 30, 131  
value added tax number, 44, 145  
variable names, 78  
VAT number, 44, 145  
verbatim string, 162  
View menu, 193  
Visual Basic, 162, 210  
Visual Studio, 210  
visualize spaces, 197  
wchar\_t, 162  
wide screen, 196  
word, 205  
workflow, 210  
working copy, 200  
wrap, 197  
XML, 163  
Xojo, 162