# A quick tour to Regular Expression

## Preface

If you are new with Regular Expression, this document will try to instruct you to understand what it is, where you may meet it, how you can use it. If you had already been familiar with it, you can treat this document as a quick reference. Hope you and Regular Expression become good friends ever since.

Table of Contents

[A quick tour to Regular Expression 1](#_Toc318302097)

[Preface 1](#_Toc318302098)

[What is regular expression? 2](#_Toc318302099)

[You can back to these definitions when you have completed the next sections, then you will have better understanding about what the definitions are saying. 2](#_Toc318302100)

[A quick example: 2](#_Toc318302101)

[Basic concepts: 3](#_Toc318302102)

[Create your first pattern: 3](#_Toc318302103)

[Create more patterns: 3](#_Toc318302104)

[Regular expression characters 4](#_Toc318302105)

[Application Cases: 7](#_Toc318302106)

[BVT with MSJade 7](#_Toc318302107)

[Response validation in Web test 9](#_Toc318302108)

[Daily find/replace in Visual Studio 10](#_Toc318302109)

[Match rule in Fiddler Autoresponder 12](#_Toc318302110)

[Configurations in Web.config and WebContent.config 13](#_Toc318302111)

[Summary: 14](#_Toc318302112)

[References: 14](#_Toc318302113)

## What is regular expression?

**Regular expression:** A **pattern** used like a sieve to retrieve elements of strings. – ***Hackers & painters:*** *big ideas from the computer age, by Paul Graham*

Regular expressions are special characters that **match** or **capture** portions of a field, as well as the rules that govern all characters. – [Google Analytics](https://support.google.com/googleanalytics/bin/answer.py?hl=en&answer=55582)

## You can back to these definitions when you have completed the next sections, then you will have better understanding about what the definitions are saying.

## A quick example:

In case that you had built a web site, and there is form on the page to ask user to input their phone number. You may want to verify if the user typed valid numbers. How would you implement this verification functionality?

You can check each character of the user input, using a for-loop structure. If encountered a non-numeric character then fail, else success.

But there is a much better way to achieve this. See the definition for Regular Expression? You can predefine a pattern – this pattern indicates that the string should be consisted of digital numbers only - to test the user input. This way is faster, more maintainable and much more intuitive if you are getting to be used to using it.

Test it out!

The pattern for the all-digital numbers is “^\d\*$”. You can test it via [Online Regular Expression tester](http://www.pagecolumn.com/tool/regtest.htm) …

|  |  |
| --- | --- |
| **No match example** | **Match example** |
|  |  |

So “^\d\*$” is a pattern, which can be used to test if the user input is consisted of numbers only or not.

From this example, you should have gotten an intuitive impression what is pattern. Now you can back to the [First Section](#_What_is_regular) to read the definitions again, you should get a better understanding of them. The next sections will introduce you how to make your own patterns and list some frequently used patterns.

## Basic concepts:

The only concept you should understand is pattern, which you had learnt from the former sections.

## Create your first pattern:

Now you are building an awesome tool, which can send a web request to a remote server to publish an article, and the server responds your tool with a message. For instance, if your article is published successfully, it responds a string “Your article has been published successfully”; if your article contains some sensitive information and was detected by the server, or for other reasons, in a word if your publishing is failed unluckily, the server responds a string “Oh… I failed to publish your article”.

How can you let your tool know the publishing result?

Just set a regular expression pattern to test the response message! Your tool will use the created pattern to test if the response message contains the word “successfully” or not. What would this pattern be like?

One of the pattern candidates is “**successfully**”.

So the “**successfully**” is your first regular expression pattern.

|  |  |
| --- | --- |
| **No match example** | **Match example** |
|  |  |

## Create more patterns:

As you might have seen, the first pattern is very simple and can only match the exact word “successfully”. In most of cases, you want your patterns be vaguer to match different words at one time. You have met this situation in [the second section](#_A_quick_example:). You just want to match digital numbers, but you don’t care what numbers they exactly are. Another example can be that you want to test a user input is a valid email address or not.

To design these patterns you need to learn more about the **regular expression characters**.

Regular expression characters can be divided into 7 categories, they are:

* Wildcards and quantifier
* Anchors
* Grouping
* Predefined special characters
* Predefined classes
* Delimiters
* Other

#### Wildcards and quantifier:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| \* | [Matches zero or more of the previous item](http://www.google.com/support/analytics/bin/answer.py?answer=91154) | The default previous item is the previous character. **goo\*gle** matches **gooogle**, **goooogle** |
| + | Just like a star, except that a [plus sign must match at least one previous item](http://www.google.com/support/analytics/bin/answer.py?answer=91153) | **gooo+gle** matches **goooogle**, but never **google**. |
| ? | [Matches zero or one of the previous item](https://support.google.com/googleanalytics/bin/answer.py?answer=91149) | **labou?r** matches both **labor** and **labour** |
| {n} | Matches previous item with n occurrences | **Jef{2}** matches **Jeff**, but not **Jef**, nor **Jefff** |
| {n,m} | Matches previous item with at least n and at most m occurrences | **I{2,5}van** matches **IIvan**, and **IIIvan**, **IIIIIvan**, but not **Ivan**, nor **IIIIIIvan**, nor **IIIIIIIIIIvan** |
| {n,} | Matches at least n previous item | **Ka{5}te** matches **Kaaaaate**, and **Kaaaaaaaaaaate**, but not **Kate**, nor **Kaaaate** |
| | | [Lets you do an "or" match](https://support.google.com/googleanalytics/bin/answer.py?answer=91150) | **a|b** matches **a** or **b** |

#### Anchors:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| ^ | [Requires that your data be at the beginning of its field](https://support.google.com/googleanalytics/bin/answer.py?answer=91146) | **^site** matches **site** but not **mysite** |
| $ | [Requires that your data be at the end of its field](https://support.google.com/googleanalytics/bin/answer.py?answer=91148) | **site$** matches **site** but not **sitescan** |
| \b | Require that your data be at the border of a word | **\bsite\b** matches **my site** but not **mysite** |
| \B | Require that your data be at the border of non-word characters | **\Bsite\B** matches **\_site\_** but not **site** |

#### Grouping:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| () | [Use parenthesis to create an item](https://support.google.com/googleanalytics/bin/answer.py?answer=91151), instead of accepting the default | **Thank(s|you)** will match both **Thanks** and **Thankyou**, and will remember the matched characters **s** or **you**, you can use $1 to refer the remembered s or you in other places |
| [] | [Use brackets to create a list of items to match to](https://support.google.com/googleanalytics/bin/answer.py?answer=91152) | **[abc]** creates a list with **a**, **b** and **c** in it, it will match **a**, **b** and **c**. |
| - | [Use dashes with brackets to extend your list](https://support.google.com/googleanalytics/bin/answer.py?answer=91152) | **[A-Z]** creates a list for the uppercase English alphabet, it will match A, B, C, …, Z |

#### Predefined special characters:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| \t | Matches tab character | **I\tT** matches   |  |  | | --- | --- | | **I** | **T** |   But not **I T**, nor **IT** |
| \n | Matches new line character | **Hello\r\nWorld** matches **Hello**  **World**, but not **Hello World** |
| \r | Matches carriage return character | Same as above. In most cases, **\r\n** are used together |
| \f | Matches the form feed character | I can’t come up with one example now |
| \a | Matches the alert character | … |
| \e | Matches the escape character | … |
| \cX | Matches the control character corresponding to X | … |
| \b | Matches the back character | … |
| \v | Matches the vertical tab character | … |
| \0 | Matches the Null character | These special characters are rarely used in common cases |

#### Predefined classes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Equivalent expression** | **Remark** | **Example** |
| . | [^\n\r] (here ^ represents except) | [Matches any single character (letter, number or symbol)](https://support.google.com/googleanalytics/bin/answer.py?answer=91145) except new line and carriage return characters | **goo.gle** matches **gooogle**, **goodgle**, **goo8gle** |
| \d | [0-9] | Matches digital number character | You’ve already seen in the quick example.  **\d** matches **5**, but not **f**;  **\d+** matches **55**, but not **five** |
| \D | [^0-9]  (here ^ represents except) | Matches non-digital character | **\D** matches **f**, but not **5**;  **\D+** matches **five**, but not **55** |
| \s | [ \t\n\x0B\f\r] | Matches the whitespace character | **Hello\sWorld** matches **Hello World**, but not **HelloWorld** |
| \S | [^ \t\n\x0B\f\r] | Matches the non-whitespace character | **Hello\SWorld** matches **Hello\_World**, **Hello-World**, **HellooWorld**, but not **Hello World** |
| \w | [a-zA-Z\_0-9] | Matches the word character (letter, digital number and underscore) | **\w+** matches **Hello5\_**, but not **-#@!$%^&\*()-+** |
| \W | [^a-zA-Z\_0-9] | Matches the non-word character | **\W+** matches **-#@!$%^&\*()-+**, but not **hello** |

#### Delimiters:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| (?:x) | Matches x but not remembers the match | Please refer the Grouping section.  **Thank(?:s|you)** will match both **Thanks** and **Thankyou**, but will not remember the matched characters **s** or **you** |
| x(?=y) | Look ahead, matches the previous item x if it is followed by y | **Thank(?=you)** will match the word **Thank** in **Thankyou**, but not the one in **Thanks** |
| X(?!y) | Look ahead, matches the previous item x if it is not followed by y | **Thank(?!you)** will match the word **Thank** in **Thanks**, but not the one in **Thankyou** |

#### Other:

|  |  |  |
| --- | --- | --- |
| **Code** | **Remark** | **Example** |
| \ | [Turns a regular expression character into an everyday character](http://www.google.com/support/analytics/bin/answer.py?answer=91144) | **mysite\.com** keeps the dot from being a wildcard, it will match **mysite.com**;  **2\\*3** keeps the star from being a wildcard, it will match **2\*3**;  **2\^3** keeps the hat from being an anchor, it will match **2^3**;  **subway\(metro\)** keeps the parenthesis from being a grouping character, it will match **subway(metro)**. |

**Tips for Regular Expressions (From Google):**

1. Make the regular expression as simple as possible so that you and your colleagues can work with them easily in the future.
2. Be sure to use a backslash if you have characters like "?" or "." and you wish to match those literal characters -- otherwise, they will be interpreted as special regular expression characters.
3. Not all regular expressions include special characters. For example, you can specify that a Google Analytics goal be a regular expression, and even if you don't have any special characters, your goal will be interpreted according to the rules of regular expressions.
4. Regular expressions are greedy. For example, **site** matches **mysite** and **yoursite** and **sitescan**. If site is your regular expression, it is the equivalent of asking to match to all strings that contain site. Therefore, you should use anchors whenever necessary, to get a more accurate match. ^site$, which uses both a beginning ^ and ending $ anchor, will ensure that the expression has to start with site and end with site and include nothing else. Notice, too, that there were no special characters in the regular expression site -- it is interpreted as a regular expression only if it is in a regular expression-sensitive field.

## Application Cases:

Below sections list some real life cases that we use the regular expression in our test work. So if you still feel that regular expression is new to you after read this article, then you will find there are many chances for you to practice it. Practice makes perfect, hope regular expression be one of your good friends in testing before long.

### BVT with MSJade

In our team, we are executing the BVT daily with MSJade. The most common test cases are handler feed validation. When we want to validate a feed node value, we can ask for Regular Expression’s help. Take this feed (<http://sandblu.msn-int.com/channels/2742655/aspx/Campaign.aspx?cp-documentid=4734560>) for example, if we want to validate the Sizing node.

<?xml version="1.0" encoding="utf-8" ?>

<Campaign>

<Bracket>

<StartDate>2012-02-05</StartDate>

<EndDate>2012-02-05</EndDate>

<Label>Bracket!</Label>

<Groups>

<Group>

<MMCategoryID>17</MMCategoryID>

<Label>Group &lt;[D]&gt;</Label>

<Title>No Description</Title>

<Description></Description>

<Ads></Ads>

<VoteModel>pollVote</VoteModel>

<Sizing>7,5,3,1</Sizing>

<VotableNum>7,5,3,0</VotableNum>

<Round>1</Round>

</Group>

</Groups>

</Bracket>

</Campaign>

We can write below test case in MSJade:

<RDTestCase ID="0001" Product="Brackets">

<TestCaseInfo Title="Verify the node format of Campaign Info Handler - Group Sizing" Owner="Brackets Test Team" Priority="1" Frequency="BVT" />

<Command ID="1" ResultValue="PASS" Pre="" Post="" RepeatCount="0" Module="**GenericTestCases**" ApiName="**HandlerTC.CheckXmlNodeFormat**">

<Param name="handlerUrl" value="<http://sandblu.msn-int.com/channels/2742655/aspx/Campaign.aspx?cp-documentid=4734560>" />

<Param name="xPath" value="/Campaign/Bracket/Groups/Group/Sizing" />

<Param name="formatType" value="**Array[number]**" />

<Param name="**customRegex**" value=" **^(?:\d+\s\*,\s\*)\*\d+\s\*$**" />

<Param name="attribute" value="" />

<Param name="fullMatch" value="True" />

<Param name="allowEmpty" value="False" />

</Command>

</RDTestCase>

We want to make sure editor had written value in Sizing node in pattern of 7,5,3,1, so we use the pattern ^(?:\d+,\s\*)\*\d+\s\*$ to test it. The pattern ^(?:\d+,\s\*)\*\d+\s\*$ can be read as the value should begin with one or more digital numbers, then followed with a comma, before and after the comma, maybe some whitespaces there. The digital-comma will repeat sometimes or even not appear, but it must end with a digital number, and there may be some whitespaces around the digital numbers.

Below is the test result for pattern ^(?:\d+,\s\*)\*\d+\s\*$:

|  |  |
| --- | --- |
| **Entries** | **Result** |
| 7 | Match |
| 11,9,7,5 , 3,1 | Match |
| Hello | Not Match! |
| 9,,,5,1 | Not Match! |

In the feed nodes, there are many other kind of values need to be validated. I’ll summarize some and you can replace the ^(?:\d+,\s\*)\*\d+\s\*$ by them in the above BVT test case to make it yours.

#### Reusable patterns

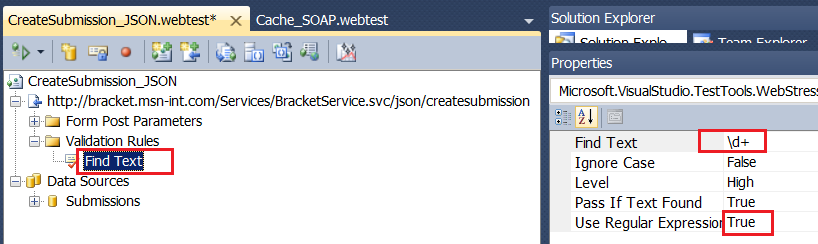
This table can be extended during our working experiences growing, if you identified a reusable pattern, please share to the team.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pattern** | **Code** | **Remark** | **Matches** | **Not Matches** |
| Array[Number] | ^(?:\d+,\s\*)\*\d+\s\*$ |  | 7  11,9,7,5, 3,1 | Hello  9,,,5,1 |
| Email | ^(?:\w+[\-\.]?)\*\w+@(?:\w+\.?)\*\w+$ |  | [Jeff.tian@facebook.com](mailto:Jeff.tian@facebook.com)  [v-jeff@somewhere.com](mailto:v-jeff@somewhere.com)  [admin@website.com](mailto:admin@website.com) | [Jeff^tian@somesite.com](mailto:Jeff%5etian@somesite.com)  -.\*@some |
| Date | ^(((0?[1-9]|1[012])/(0?[1-9]|1\d|2[0-8])|(0?[13456789]|1[012])/(29|30)|(0?[13578]|1[02])/31)/(19|[2-9]\d)\d{2}|0?2/29/((19|[2-9]\d)(0[48]|[2468][048]|[13579][26])|(([2468][048]|[3579][26])00)))$ |  | 12/31/2002  12/31/1998  2/29/2012 | 31/12/2002  12/31/98  2/29/2011 |
| Time | ^(([0-9])|([0-1][0-9])|([2][0-3])):(([0-9])|([0-5][0-9]))(?::(([0-9])|([0-5][0-9])))?$ |  | 23:59:59  11:05:23  16:05 | 24:00:00  11:05:60  16:05: |
| DateTime | ^(((0?[1-9]|1[012])/(0?[1-9]|1\d|2[0-8])|(0?[13456789]|1[012])/(29|30)|(0?[13578]|1[02])/31)/(19|[2-9]\d)\d{2}|0?2/29/((19|[2-9]\d)(0[48]|[2468][048]|[13579][26])|(([2468][048]|[3579][26])00)))(?: (([0-9])|([0-1][0-9])|([2][0-3])):(([0-9])|([0-5][0-9]))(?::(([0-9])|([0-5][0-9])))?)?$ |  | 12/31/2002 09:00:00  12/31/2002 | 09:00:00 |
| Url | (((ht|f)tp(s?):\/\/)|(www\.[^ \[\]\(\)\n\r\t]+)|(([012]?[0-9]{1,2}\.){3}[012]?[0-9]{1,2})\/)([^ \[\]\(\),;&quot;'&lt;&gt;\n\r\t]+)([^\. \[\]\(\),;&quot;'&lt;&gt;\n\r\t])|(([012]?[0-9]{1,2}\.){3}[012]?[0-9]{1,2}) | This is built-in in the MSJade Generic Test Cases, so you don’t really need it in BVT. But you can use it in other scenarios, such as web test. | [www.site.com](http://www.site.com)  <https://192.168.0.1:80/users/~fname.lname/file.txt>  <http://www.baidu.com> | Imap://.com |
| Xml tag | <(\w+)(\s(\w\*=".\*?")?)\*((/>)|((/\*?)>.\*?</\1>)) | Verify if the tags are closed correctly | <node>value</node>  <Campaign>  <Bracket>  <StartDate>2012-02-05</StartDate>  <EndDate>2012-02-05</EndDate>  <Label>Bracket!</Label>  <Groups>  <Group>  <Sizing>7,5,3,1</Sizing>  <VotableNum>7,5,3,0</VotableNum>  <Round>1</Round>  </Group>  </Groups>  </Bracket>  </Campaign> | <node>blablabla</anotherNode>  <Campaign>  <Bracket>  <Groups>  <Group>  </Groups>  </Bracket>  </Campaign> |
| Currency | ^(\$|)([1-9]\d{0,2}(\,\d{3})\*|([1-9]\d\*))(\.\d{2})?$ |  | $1,234,567.89  1234567.89  $9.99 | $1,2345,67.89  $1234,345,678.0  0 |

### Response validation in Web test

Almost every project will have web and load testing, during the web test, we’ll verify that if a handler respond correctly. In this place we may use the regular expression pattern to test if the response text is expected or not.

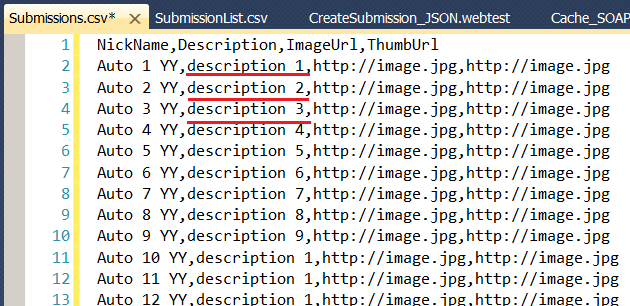
For example:



In the above screenshot, the web test is call a create submission handler, and the handler will respond the submission id (a number), so we can use pattern \d+ in the Validation Rules, once it fails, then it indicates the submission creating meet some errors. Also the Reusable Patterns can used here if applicable.

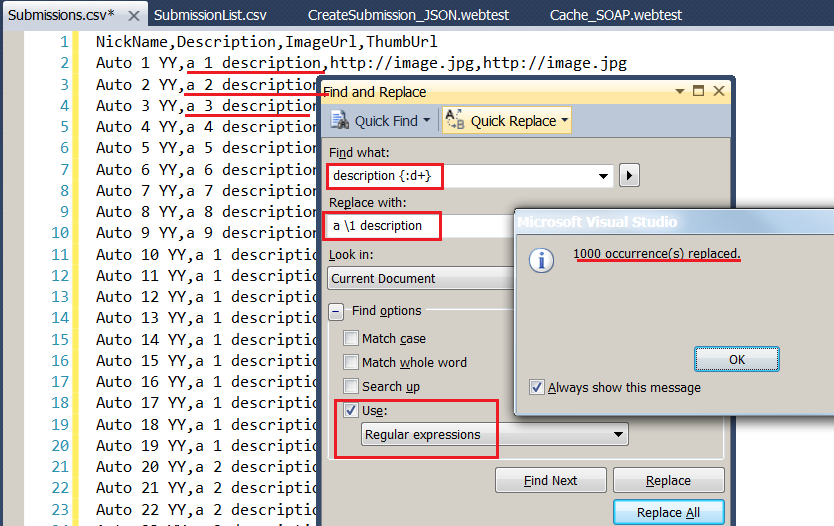
### Daily find/replace in Visual Studio

In case there is a csv file like below, and you want to replace the “description xx” to “a xx description”, how would you do (there are 1000 rows!)?



**Solution:**

Use this pattern: description {:d+}



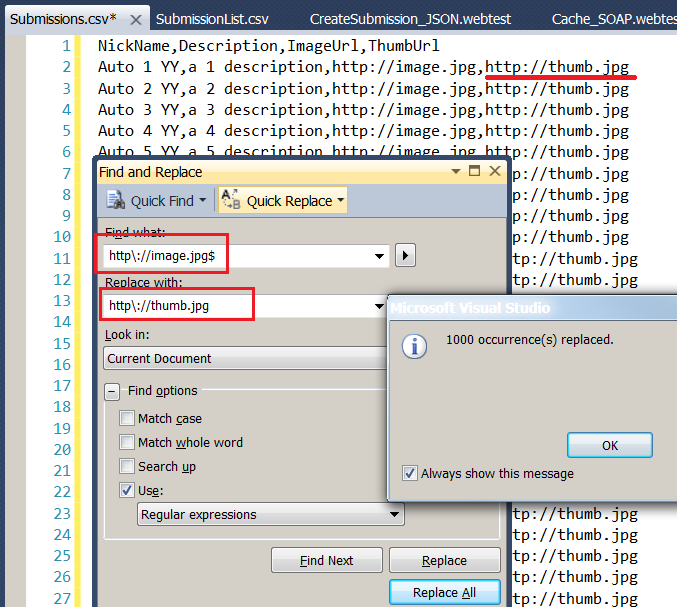
**Note:**

Visual Studio Find and Replace feature uses another Regular Expression grammar, which means that it uses some other characters to express the same meaning. For example, it uses :d to represent digital numbers, which is represented by \d in other systems, uses {} as () for grouping, uses \1 as $1 for back reference. For the other differences, you can refer to <http://msdn.microsoft.com/en-us/library/2k3te2cs.aspx>.

Take the above example again, if now you want to replace all the original ThumbUrl <http://image.jpg> with <http://thumb.jpg>, how would you do?

**Solution:**

Use pattern http\://image.jpg$

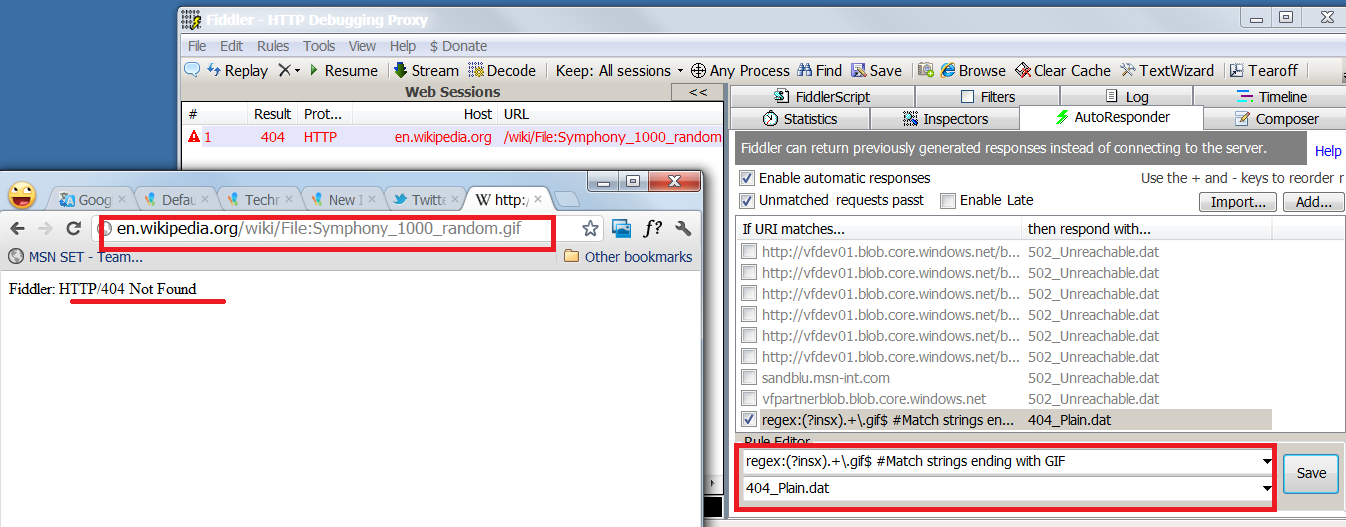


You should add a dollar sign $ behind the url to only replace the ThumbUrl which is positioned in the end of each line. If you miss the $, then the ImageUrl will also be replaced unexpectedly.

Regular Expression makes these daily work very efficient.

### Match rule in Fiddler Autoresponder

Sometimes you need to emulate some files are down with Fiddler. For example, by set the regular expression “.+\.gif$” you can block all the gif images:



.+\.gif$ includes all the uris that start with any character(s) and end with .gif.

### Configurations in Web.config and WebContent.config

If you download the project source code and read the Web.Config and WebContent.Config files, you will also find many regular expressions usage.

Below are some configurations for handlers from WebContent.Config:

<!-- Media Manager content source -->

<source name="MediaManager" baseUrl="http://api.mediamanager.msn-int.com/">

<cacheSettings duration="0" />

<storageSettings>

</storageSettings>

<contentItems>

<!--Comment-->

<contentItem match="**Comments/(?'postId'[0-9]+)/(?'start'[0-9]\*)/(?'end'[0-9]\*)**" **regex**="**true**">

<cache duration="0"/>

<processors>

<add name="Xslt" type="Microsoft.Msn.Set.Web.Content.Xml.XsltProcessor, Microsoft.Msn.Set.Web">

<xslt url="xslt/Comments.xslt">

<outputSettings indent="false" />

<xsltArgs>

<add name="startIndex" value="${start}"/>

</xsltArgs>

<xsltExtensions>

<add namespace="Website" type="Website.Core.XsltExtensions, Website" />

</xsltExtensions>

</xslt>

</add>

</processors>

<resources>

<add name="Comments" url="commentservice.svc/group/${MM\_PROJECTGROUPID}/project/${MM\_PROJECTID}/entitytype/Submission/entity/${postId}/comments/?commentstatus=${MM\_COMMENT\_STATUS}&amp;start\_index=${start}&amp;end\_index=${end}&amp;sortby=${MM\_COMMENT\_SORT}&amp;sortdirection=desc" />

</resources>

</contentItem>

Notice the bolded part, that means the handler url would be like [http://sp.msn-int.com/**MediaManager**/**Comments**/**136**/**1**/**9**](http://sp.msn-int.com/MediaManager/Comments/136/1/9). The 136 is the postId (digital numbers, required because the pattern uses + sign), the 1 is the start index (digital number, optional because pattern uses \*), the 9 is the end index (digital number, optional because pattern uses \*).

Below are part of the URL routing settings from Web.Config, it is also using Regular Expressions!

<!-- URL Routing -->

<routing>

<map>

<add url="**{campaignname}/admin/comments/{postID}/**" destination="~/admin/Comments.aspx">

<constraints>

<add name="campaignname" value="**.\***" />

<add name="postID" value="**[0-9]\***" />

</constraints>

</add>

<add url="**{campaignname}**/" destination="~/default.aspx">

<constraints>

<add name="campaignname" value="**((?!\.axd).)+**" />

</constraints>

</add>

</map>

</routing>

In the first <add /> node, the {campaignname} can be replaced with any string and the {postID} should be replaced with digital numbers, so the final url should be like <http://sp.msn-int.com/StopDiabetes/Comments/136/> .

In the second <add /> node, the pattern shows that the {campaignname} can be replaced with any string but not ends with “.axd”.

## Summary:

The Regular Expression is a powerful tool, can be used in many scenarios. If you use it well, it can help you yield twice the result with half the effort.

If you meet some situation that needs a regular expression pattern which is not mentioned in the document, you can try to find in the [Online Regular Expression Library](http://regexlib.com/). If even this doesn’t work, then it’s time to create on your own, and the [Online Regular Expression Tester](http://www.pagecolumn.com/tool/regtest.htm) will help you do this.

## References:

Regular Expression Library: <http://regexlib.com/>

Online Regular Expression Tester: <http://www.pagecolumn.com/tool/regtest.htm>