RegEx Tutorial - Blutige Anfänger und Fortgeschrittene

Hi comm und Leute von außerhalb des Forums,

-> Dieses Tutorial richtet sich an blutige Anfänger und alle die bereits schon Erfahrungen mit RegEx haben<-

RegEx braucht man einfach so gut wie immer wenn man professionell mit Strings arbeiten möchte.

Was durchgenommen wird

- Einführung (was ist RegEx, wofür ist es gut)
- Erklärung der Zeichen
- Escapen (was bewirkt es, was bedeutet escapen)
- Code Snippets (Code-Schnippsel)
- Ersetzung ((?<X>expr) <-> ${X})
- Beispielprojekt zum ausprobieren in Visual Basic

Bevor wir anfangen können, müssen einige wenige Vorbereitungen getroffen werden:
- Visual Basic muss installiert sein, davon ausgehend, dass Sie Ihr neu erworbenes Wissen direkt in die Tat umsetzen möchten ;-)
- Zeit und Lust :-)
- Lernbereitschaft, selbsterklärend
- und das wichtigste, die Klasse ***System.Text.RegularExpressions*** muss importiert werden.

So wird's gemacht:

```
1           Imports System.Text.RegularExpressions
```

Kurz ein Wort zur Ersetzung.
Moment mal, wieso brauchen wir dafür RegEx? VB stellt uns doch bereits ein eine Ersetzen-Funktion (Replace) zur Verfügung. Allgemein bekannt sieht das dann in etwa so aus:

```
1           Dim var As String = "Das Wetter ist heute schön"
2           var = var.Replace("heute", "heute nicht")
```

Da wir das bereits kennen, ist nun die Frage, wozu wir zum ersetzen noch RegEx brauchen. Das ist schnell erklärt, und jeder kennt die Situation: Manchmal möchte man nur bestimmte Muster eines Textes ersetzen, z.B. nur Wörter die mit "a" beginnen, oder nur Zahlen ersetzen, oder einfach nur alles ersetzen was zwischen zwei "" steht. Und das ist etwas, was mit der normalen Replace Funktion nur sehr schwer, und bis zu einem gewissen Grad teilweise gar nicht mehr zu bewerkstelligen ist. Zu Replace gleich noch Beispiele, zuerst wollen wir uns jedoch die alles

entscheidende Kernfrage stellen, die da lautet ...

Was ist denn nun RegEx?
RegEx steht für "*Regular Expressions*", zu Deutsch "*Reguläre Ausdrücke*", oft auch "RegExp". Hierbei können bestimmte Zeichen verwendet, um die Suche nach bestimmten Textstellen oder -mustern zu matchen und nichtgewolltes leichter auszuschließen. Zur Veranschaulichung: Mit folgendem Code ist es uns möglich, alle Wörter die mit "a" anfangen und mit "en" enden zu entfernen, dafür ersetzen wir einfach mit "", also mit nichts, das gefundene wird somit entfernt:

```
RichTextBox1.Text = RegEx.Replace(RichTextBox1.Text, "(^|\s|\n)a[a-z]+en(\n|\s|$)", "", RegexOptions.IgnoreCase)
```

Es ist noch nicht wichtig den Ausdruck zu verstehen, das kommt alles dran, also keine Panik wenn Ihr mit diesem kryptischen Stück Code noch nichts anfangen könnt. Nach diesem Tutorial wisst Ihr all das richtig einzuordnen. Zunächst einmal ist zu sehen, das RegEx.Replace() hier mit 4 Parametern aufgerufen wurde:
- dem EingabeString, das ist der Text aus dem etwas ersetzt werden soll, hier der Text in der RTB
- der Reguläre Ausdruck, der den EingabeString -die RTB in unserem Fall- nach einem bestimmten Muster durchsucht
- der Ersetzen-Zeichenkette, diesmal "", also eben nichts, und
- der Suchoption, in unserem Beispiel wurde Groß-/Kleinschreibung ignoriert. Dieser, also der 4. parameter ist optional.

Aber was ging da vor sich? Einige Zeichen im Suchmuster werden als Reguläre Ausdrücke behandelt und können so nicht im Text als Strings gefunden werden. Der Punkt(".") hat als Ausdruck eine eigene Bedeutung, er steht hierbei für ein beliebiges Zeichen. Das können Sonderzeichen sein, Buchstaben oder Zahlen, Tabstopps, Zeilenumbrüche, Textanfänge oder -enden. Der Ausdruck "a.s" würde also Beispielsweise die worte "als", "aus" oder "ass" finden, da das wort mit "a" beginnt, dem ein beliebiges Zeichen folgt und danach ein "s". Auch "a8s" oder "a!s" würde gefunden werden. Dem Fortgeschrittenen wird aufgefallen sein, dass das hier natürlich kein Wortanfang sein muss, auch "Haus" würde gefunden werden, dazu kommen wir aber gleich ;-)
-> Würde man nach Sonderzeichen als Strings suchen wollen geht das auch, wie, erfahrt ihr weiter unten, anzumerken hier auch Stichwort "escapen"

Hier eine Liste der Ausdrücke. Enthält [b]*Notation* + *Beschreibung* + *Beispiel*[/b]

Beliebiges Zeichen '.'

- Wert: Ein Beliebiges Zeichen
Notation[/b]: .
Beschreibung: Der "." entspricht einem beliebigen Zeichen, mit ausnahme des Zeilenumbruchs (dafür gibt's ein extra Zeichen)
Beispiel: "h.se" findet "hase", "hose" oder auch "h9se"
"h.s." findet "hass", "hast" oder "Schiss". Aber wieso "Schiss"? Ganz einfach, weil vor "h" oder nach dem letzten "s" kein Leerzeichen
steht, dieser Suchausdruck kann auch innerhalb eines Wortes sein.

0 oder mehr '*'

- Wert: 0(null) oder mehr
Notation: *

Beschreibung: Findet 0 (null) oder mehr Vorkommen des vorangehenden Ausdrucks und ergibt somit alle möglichen Übereinstimmungen.
Beispiel: "Han*es" findet "Hannes", "Hannnnnes" oder "Hannnnnnnnnnes" aber auch "Hanes" und sogar "Haes"(wie gesagt, möglich ist 0 oder mehrmals). "Ha.*es" findet "Hannes", "Hastiges" oder "Haltendes" aber auch "Ha325!%es", da ja nicht festgelegt ist, welche Zeichen innerhalt des Wortes erlaubt sind, aber das kommt nachher noch alles mit "[" und "]". Um ".*" mal in Worte zu fassen: Der Ausdruck sucht "Ha", wenn gefunden, wird alles was dann kommt ebenfalls positiv gematcht. In dem Fall ".*" was soviel heißt wie "ein beliebiges Zeichen beliebig oft oder gar nicht". Und am Ende dann irgendwann "es". Für den größeren Teil der Leser dieses Tuts wird es jetzt verwunderlich sein wenn Ich sage, dass damit auch "Ha123xyz!esesesesesses" gefunden würde, wir haben nicht festgelegt, dass die Übereinstimmungssuche gleich beim ersten vorkommen von "es" enden soll. Auch dafür gibt es eine Lösung, lest weiter, Ihr werdet wissen wann die Stelle kommt ;-)

1 oder mehr '+'

- Wert: Findet einen oder mehrere Vorkommen des vorangehenden Zeichens
Notation: +
Beschreibung: Findet mindestens ein oder mehrere Vorkommen des vorangehenden Ausdrucks
Beispiel: Im gegensatz zu "*" würde bei "Han+es" zwar auch "Hanes" "Hannes" oder "Hannnnnnnes" und so gefunden werden, jedoch NICHT "Haes", da der Ausdruck mindestens einmal vorkommen muss

Zeilenanfang '^'

- Wert: Zeilenanfang
Notation: ^
Beschreibung: Findet den Anfang des Strings
Beispiel: "^d" Findet ein "d" nur, wenn es am Anfang des Strings steht, der Ausdruck "^.*" findet Beispielsweise die erste Zeile des Strings bis zum Zeilenumbruch oder zum Textende, falls es nur eine Zeile ist.

Bis zum ersten Vorkommen '?'

- Wert: Bis zum ersten vorkommen
Notation: ?
Beschreibung: Sucht bis zum ersten vorkommen des vorangehenden suchmusters
Beispiel: "a.* " (<- am ende ein leerzeichen!!!) findet im String "a b c d" nur "a b c ", da logischerweise nach "d" kein leerzeichen kommt. Mittels ".*" wird bis dahin gesucht, was nach dem "*" steht, steht da nichts, dann wird höchstens bis zum nächsten Zeilenumbruch gesucht. ABER mit dem "?" Zeichen, wird nur bis zum >ERSTEN< vorkommen des nächsten Zeichens gesucht, also was nach dem "?" kommt. Schreiben wir nun ein Fragezeichen dazu, also "a.*? ", so wird nur "a " gefunden, da nach diesem ja bereits schon das nächststehende Leerzeichen kommt. Komischerweise scheint das mit dem "?" in der VB-Hilfe vergessen worden zu sein. Also "a.*? " findet alle wörter die mit "a" anfangen, und bis dahin, wenn ein Leerzeichen kommt. Im Text "Das ist mein RegEx-Tutorial fürs VB-Paradise Forum" würde mit dem eben genannten Suchausdruck "as ", "al " und "adise" gefunden werden. Würden wir bei dem Suchmuster vor dem "a" auch noch ein Leerzeichen platzieren, so würden wir im eben beschriebenen Text nichts finden da dort kein wort mit "a" anfängt.

Zeilenende '$'

- Wert: Zeilenende
Notation: $
Beschreibung: Findet das Ende des Strings
Beispiel: Naja, ist halt wie "^" nur dass es mit "$" das Ende eines Strings ist. Wäre der zu durchsuchende String nun:
"Ich schreibe gerade einen Teststring" und würden wir nach "...$" suchen, so würden wir "ing" finden.

Zeilenumbruch '\n'

- Wert: **Zeilenumbruch**
Notation: **\n**
Beschreibung: **Findet Zeilenumbrüche**
Beispiel: **Hier mal ein Test-text:**
"Heute bin ich so mo
tiviert, dass ich angefangen h
abe, ein Tutorial für das Foru
m zu schreiben."
So, und wenn wir nun nach ".\n." suchen würden, dann würden wir folgende Strings finden: "ot", "ha" und "um". Suchen wir nach "^.*?\n" so erhalten wir die komplette erste Zeile.

Hier der Zweite Teil

Leerzeichen '\s'

- Wert: Leerzeichen
Notation: \s
Beschreibung: Findet Leerzeichen
Beispiel: "\s...\s" findet alle wörter mit 3 Buchstaben. "\sa.*?\s" findet alle wörter die mit "a" anfangen bis zum nächsten Leerzeichen. ".\s." findet im String "Das ist ein kurzer Text" beispielsweise "s i", "t e", "n k" und "r T", also immer das erste Zeichen nach und vor dem Leerzeichen.

Beliebiges Zeichen in der Menge '[...]'

- Wert: Ein beliebiges Zeichen in der Menge
Notation(s): [und]
Beschreibung: Findet eines der Zeichen, die in [] enthalten sind. Geben Sie zum Festlegen eines Bereichs von Zeichen das Start- und das Endzeichen durch einen Bindestrich (-) getrennt ein, wie in [a-z]. *Beispiel:* "\sF[aeiou]s.*?\s" Findet worte die mit "F" anfangen und danach einen Selbstlaut(also einen der Buchstaben in den eckigen klammern) haben bis zum nächsten Leerzeichen, also Beispielsweise " Fasan ", " Fussball " oder " Fasten ", gefoch nicht "Frisch", "Flug" oder "Franz", da hierbei nach dem "F" keiner der Buchstaben a,e,i,o oder u kommt(also keiner DIREKT nach dem "F") "[a-z0-9]" findet jedes (einzelne) Zeichen wenn es ein Buchstabe oder eine Zahl ist. Mit dem "+" kann hierbei praktischerweise die Übereinstimmung fortgesetzt werden. Im Text "In diesem Text kommen auch Zahlen vor, wie 1, 23, 300 oder 19" finden wir mit dem Suchmuster "\s[a-z]+" Beispielsweise vom Wort "dem" bis zu der Stelle an der die Zahlen anfangen, da zahlen im Alphabet nicht vorkommen(PS: ab dem Wort "dem" deshalb, weil wir davor ein "\s" haben!! Und noch eine Falle: wir haben nach dem "z" in "[a-z]" ein leerzeichen,

hätten wir das nicht, würden wir NUR das wort "dem" finden, da Leerzeichen auch nicht im Alphabet enthalten sind! Immer gut aufpassen!). Im Text "8723 das waren zahlen, hier kommen noch mehr: 876578345" finden wir mit dem suchmuster "^[0-9]+" nur den ersten Zahlenblock, da ab dem Leerzeichen abgebrochen wird, welches ja ebenfalls bei den Zahlen 0-9 nicht enthalten ist.

Beliebiges Zeichen nicht in der Menge '[^ ...]'

- Wert: ein beliebiges, NICHT in der Menge enthaltenes Zeichen
Notation: [^...]
Beschreibung: Findet ein beliebiges Zeichen, das nicht im Satz von Zeichen nach ^ enthalten ist.
Beispiel: "\sa[^ub].*?\s" findet worte die mit a anfangen bis zum nächsten Leerzeichen NUR wenn nach dem "a" KEIN "u" oder "b" folgt. Gefunden werden würde " ast ", " altar ", " ampel " oder " access ", JEDOCH NICHT " auto ", " abfindung ", " auge " oder " aber ".

Gruppierung

- Wert: Gruppierung
Notation: (und)
Beschreibung: Ermöglicht es Ihnen, einen Satz von Ausdrücken zu gruppieren. Wenn Sie in einer Suche nach zwei verschiedenen Ausdrücken suchen möchten, können Sie diese mit dem Gruppierungsausdruck kombinieren.
Beispiel: ([0-9][a-z])+ findet alle Zeichenfolgen, wenn abwechselnd eine Zahl und dann ein Buchstabe folgt. Im String "3h4g9w2g6b5y0lzdh3j2q8f62441" würde gefunden werden: "3h4g9w2g6b5y0l" und "3j2q8f". Oder leichter Dargestellt: Im String "0F0F0F0F0FFFFFF0F0F0F0000" würde gefunden werden: "0F0F0F0F0F" und "0F0F0F".

ODER '|'

- Wert: Or -> Oder
Notation: |
Beschreibung: Findet entweder den Ausdruck vor oder nach dem Symbol OR (|). Wird meistens in einer Gruppe verwendet.
Beispiel: \s(ab|ein|aus|um|allein)gang\s findet die wörter "abgang", "eingang", "ausgang", "umgang" und "alleingang". Hierbei wird wird mit ODER gearbeitet, kommt also ein Ausdruck nicht vor, wird der nächste probiert. Der Ausdruck "ver(sau|ursach|schwende|damm)t" findet "versaut", "verursacht", "verschwendet" und "verdammt". Ist logischerweise nicht wie "ver[a-z]+t" zu behandeln, da die Strings die zwischen "ver" und "t" stehen vorgegeben sind, und nicht beliebig auftauchen.

Escape '\'

- Wert: Escape
Notation: \
Beschreibung: Findet das Zeichen, das dem umgekehrten Schrägstrich (\) als Literal folgt. Damit können Sie Zeichen wie { und ^ suchen, die in der Notation für reguläre Ausdrücke verwendet werden.
Beispiel: Mit \. kann in einem Text nach dem Punkt als String gesucht werden. "..\$" findet beispielsweise nicht das Zeilenende, sondern Beispielsweise "ab$" oder "var$". Der Ausdruck <.*?> findet alles was zwischen zwei "<>" steht, also Beispielsweise "<title>" oder "</body>". der

Ausdruck "\[.*?\]" findet alles was zwischen zwei [] steht, also "[Wort]", "[autorun]" und so weiter. Das "\" vor "[" und "]" ist wichtig, damit die [] Zeichen als String erkannt werden können. Das macht man am besten mit jedem Sonderzeichen, also statt "+" nach "\+" suchen, statt "/" ein "\/" oder statt "." ein "\.", naja ihr versteht schon.

Markierter Ausdruck '{ ... }'

- Wert: Markierter Ausdruck
Notation: { und }
Beschreibung: Findet den Text, der mit dem in Klammern stehenden Ausdruck übereinstimmt.
Beispiel: [0-9]{4} findet alle Strings die nur aus Zahlen bestehen und vierstellig sind. Findet beispielsweise "8932", "5897" und "2385" in "8932589723857". Der Ausdruck (ohne die "") "[a-z]*[a-z]{2}[a-z]*" findet "Mann", "Fitness", "Anerkennung" oder auch "Hallo", da nur Wörter gefunden werden, die zwei doppelte Buchstaben hintereinander enthalten. Der Ausdruck "\s[a-z]{3}\s" ist der selbe wie "\s...\s", es wird beide male nach Worten mit nur 3 Buchstaben gesucht. \s[a-z]{2,7}\s findet alle Wörter die zwischen 2 und 7 Buchstaben lang sind. \s[a-z]{4,}\s findet wörter die 4 ODER MEHR Buchstaben enthalten, der Ausdruck \s[a-z]{0,4}\s findet dagegen alle wörter die weniger ODER BIS ZU 4 Buchstaben enthalten.

Minimal, 0 oder mehr '@'

- Wert: Minimal, 0(null) oder mehr
Notation: @
Beschreibung: Findet 0 (null) oder mehr Vorkommen des vorhergehenden Ausdrucks, wobei die Übereinstimmung möglichst wenig Zeichen umfasst.
Beispiel: e.@e findet "ente" und "erprise" in "enterprise". a.@e Komischerweise funzt das nicht, das @ wird als reiner String behandelt...

==========

So, jetzt können wir eigentlich auch schon loslegen. Wir starten nun einfach einmal ein Beispielprojekt.
Dafür holen wir uns
1x TextBox
1x RichTextBox
1x ListBox
1x CheckBox

Wenn wir das alles schön positioniert haben, gehen wir in den Code-Editor und erstellen dort eine Sub. Nennen wir mal sie FilterExpr()

```
1           Imports System.Text.RegularExpressions
2
```

```
3           Public Class Form1
4
5                   Sub FilterExpr(ByVal FilterObject As String)
6
7                   End Sub
8
9           End Class
```

Nun füllen wir die Sub wie folgt:

```
1           Imports System.Text.RegularExpressions
2
3           Public Class Form1
4
5                   Sub FilterExpr(ByVal FilterObject As String)
6                   Dim Results As New List(Of String)  'Res als List
7                   Dim Curr As String = ""
8                   ListBox1.Items.Clear()  'ListBox items leeren
9                   Dim MC As MatchCollection   'Deklariere MC als MatchCollection
10                  'Je nachdem ob CheckBox1 gecheckt ist, wird Groß- und Kleinschreibung i
11                  If CheckBox1.Checked Then
12                          MC = Regex.Matches(RichTextBox1.Text, FilterObject, RegexOption
13                  Else
14                          MC = Regex.Matches(RichTextBox1.Text, FilterObject, RegexOption
15                  End If
16                  For i As Integer = 0 To MC.Count - 1    'Schleife durchlaufen
17                          If Results.Contains(MC(i).Value) = False Then
18                          Curr = MC(i).Value.ToString
19                          ListBox1.Items.Add(Curr)        'Übereinstimmungen in der ListB
20                          End If
21                  Next
22                  End Sub
23
24                  'Dann in TextBox1_TextChanged:
25                  Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e
26                  If Not TextBox1.Text = "." And Not TextBox1.Text = " " And Not TextBox1
27                          FilterExpr(TextBox1.Text)   'Unsere Sub FilterExpr wird aufgeru
28                  End If
29                  End Sub
30
31                  'Und noch in RichTextBox1_TextChanged:
32                  Private Sub RichTextBox1_TextChanged(ByVal sender As System.Object, ByV
33
34          RichTextBox1.TextChanged
35                  FilterExpr(TextBox1.Text)
36                  End Sub
37
38          End Class
```

Mit diesem Projekt wird es ab jetzt richtig spannend, wenn wir unser neu erworbenes Wissen gleich in die Tat umzusetzen. Wir debuggen erstmal und schauen, ob alles glatt geht.
Doch bevor es ans Werk geht: Kopiert euch wenn Ihr wollt einfach schnell irgendeinen Text in die RTB. Er sollte viele verschiedene Textmuster enthalten, um auch kompliziertere Code zu matchen

Diesen Text lass ich euch mal zum kopieren da, ist ganz praktisch zum üben eben.
Selbstverständlich könne Ihr auch eure eigenen Texte nehmen .

Jetzt könnt ihr nach Lust und Laune herumexperimentieren. Versucht zum Beispiel mal den Ausdruck <.*?> und schreibt Ihn in die TextBox.
Oder auch "\Dim .*?\s As .*\s|\n)"
Ist ganz interessant, das ganze mal auszuprobieren, damit kann man sich erst einmal ein wenig einarbeiten, in die ganzen RegEx-Codes.
Es muss nicht immer eine komplette Sub sein, Beispielsweise wenn man nur kurz etwas ersetzen will. Dafür reicht folgende kleine Codezeile:

```
1          RichTextBox1.Text = RegEx.Replace(RichTextBox1.Text, "\s...\s", "", RegExOption
```

damit entfernt man z.B. alle Worte mit 3 Zeichen.

Was kann RegEx noch alles?

Nun, da Ich es am leichtesten finde, immer gleich alles zu testen, schlage Ich vor, wir erstellen uns ein neues Projekt oder
lassen das schon erstellte offen und fügen dort einen Button hinzu. Jetzt wollen wir überprüfen, ob denn überhaupt in der RichTextBox ein
vordefiniertes Suchmuster vorhanden ist. Das machen wir mit RegEx.IsMatch(), ist eigentlich das ähnliche wie RichTextBox1.Text.Contains(), nur dass wir auch hierbei wieder Reguläre Ausdrücke angeben können, nach dessen Muster gesucht werden soll. Dafür erstellen wir zuerst eine Variable vom Typ Boolean, die am Ende angibt, ob ein solches Muster gefunden wurde.

```
1          Dim Exists As String = Regex.IsMatch(RichTextBox1.Text, "\sa[a-z]+en\s", RegExO
2          If Exists Then MsgBox("Es sind Übereinstimmungen Ihres Suchmusters vorhanden.")
```

In diesem Beispiel matcht RegEx ein Wort, welches mit "a" anfängt und mit "en" aufhört, also beispielsweise "arbeiten", "ahnen" oder "aufpassen". Wie immer pfeifen wir auf Groß-Kleinschreibung ;-) Somit werden zwar die gesuchten Strings nicht ausgegeben, aber es wird geprüft, ob ein solches Muster überhaupt auftaucht.

Wir können mit RegEx natürlich auch Strings splitten. Dafür brauchen wir lediglich einen InputString, d.h. der String in dem gesucht werden soll.
Ich habe zum testen einfach mal folgenden genommen:

```
1          Dim Input As String = "Ich gehe heute mit meinem Hund spazieren"
```

Wir legen dann eine weitere Variable an vom Typ String-Array, dort "lagern" wir dann die gesplitteten Strings:

```
1           Dim Input As String = "Ich gehe heute mit meinem Hund spazieren"
2           Dim Res() As String
3           ListBox1.Items.Clear()
4           Res = Regex.Split(Input, "\s[a-z]+\s", RegexOptions.IgnoreCase)
5           For Each el As String In Res
6                   ListBox1.Items.Add(el.ToString)
7           Next
```

Würden wir nun den String von "Input" in die RichTextBox1 schreiben und dann in unserer TextBox diesen Suchausdruck:
(\s[a-z]+\s)
eingeben, so fänden wir "gehe", "mit" und "Hund". Wenn wir aber auf unseren Button klicken wird gesplittet und übrig bleibt "Ich", "heute", "meinem" und "spazieren".

****
So, jetzt haben wir eine Sub FilterExpr. Um mit RegEx noch besser üben zu können, können wir eine weitere Sub einbauen, nennen wir sie mal ReplaceStr(). Dieser übergeben wir 2 Parameter.

```
1           Sub ReplaceStr(ByVal Pattern As String, Optional ByVal Replacement As String =
2                   RichTextBox1.Text = RegEx.Replace(RichTextBox1.Text, Pattern, Replaceme
3           End Sub
```

Und so brauchen wir fortan zum ersetzen oder entfernen von Suchmustern nur noch die Sub aufrufen[ReplaceStr("Ausdruck", "Irgendwas oder nichts")]und sparen uns jedesmal die Tipperei.

Das war's auch schon mit diesem Tutorial .. Ich hoffe es hat euch gefallen und Ihr habt was dazugelernt. Wenn das der Fall ist, hat es sich für mich gelohnt dieses Tutorial zu schreiben. Vergesst nicht, auf den "Bedanken"-Button zu klicken ;-) Bei Fragen, Anregungen oder Wünschen, wenn noch was fehlt oder was dazu soll, dann kontaktiert mich einfach über PM oder Skype/ICQ
Es folgen jetzt noch ein paar Beispiele, falls bisher etwas unklar war einfach mal anschauen

[Beispiel 1]
Der Ausdruck "\s[a-z]+i[a-z]+\s" (wie immer ohne "" außer die dazwischen vielleicht) matcht Wörter, in denen ein i vorkommt,
da wir jedoch [a-z]+ haben, kann darf das "i" schon mal nicht am Anfang vorkommen, auch nicht am Ende. Warum? Weil wir zuvor [a-z]+ geschrieben haben, was heißt, dass mindestens ein vorkommen des vorangehenden Ausdruck vorkommen muss. Genauso auch am Ende, da haben wir dasselbe stehen. Hierbei würden wir "biene" "irgendwie" oder "Mist" finden, jedoch nicht "ist", "Ascii", "immer" oder "Manni".
Mit folgendem Ausdruck würden wir alle Wörter finden, in denen ein "i" vorkommt, egal ob es am ende oder am Anfang oder dazwischen irgendwo steht: \s[a-z]*i[a-z]*\s Warum? Weil wir hier kein "+" sondern das "*" Zeichen haben, denn beim "*" Zeichen können es 0(null), 1 oder mehrere Vorkommen sein, beim "+" müssen es mindestens 1 oder mehr sein. Beliebter Fehler ^^; (PS: Selbstverständlich ist es bei beiden Ausdrücken egal, wieviele "i" drin vorkommen, logisch)

[Beispiel 2]
Wir haben einen String: "Wenn mir langwelig ist, gehe ich raus oder lese 1-2 Stunden." <-ohne "" !!! Wir gehen in den folgenden Beispielen davon aus, dass wir die Beispieltexte in unsere RTB

kopieren und in der TextBox die Ausdrücke eingeben !!! So, worin unterscheidet sich Ausdruck1 von Ausdruck2(ohne das erste leerzeichen nach dem doppelpunkt):
Ausdruck1: We.*\s
Ausdruck2: We.*?\s
Einfach mal beides versuchen, ist ganz interessant anzuschauen. Beim ersten Ausdruck finden wir den String "Wenn mir langwelig ist, gehe ich raus oder lese 1-2 " in der ListBox, beim zweiten Ausdruck finden wir "Wenn" und "weilig". warum? Weil wir beim ersten Suchausdruck von "We" angefangen suchen bis zum letzen leerzeichen vor dem Ende des Strings oder vor einem Zeilenumbruch. Da nach dem Letzen Wort(Stunden.) ja kein Leerzeichen mehr kommt, wird das nicht gefunden. Beim 2. Ausdruck haben wir noch ein "?" dazugeschrieben, was heißt, dass nur von der Zeichenfolge "we" bis zum nächststehenden Leerzeichen gesucht werden soll, das haben wir also einmal im wort "Wenn" und einmal in "langweilig". Anderes Beispiel: der String ist diemal folgender: "ente essen einzeln" Die Ausdrücke:
Ausdruck1: e[a-z]+e[a-z]*
Ausdruck2: e[a-z]+e[a-z]+
Auch hier wieder der Spaß mit "+" oder "*"
Ausdruck1 findet alle 3 Worte des Strings, Ausdruck2 findet nur "essen" und "einzeln". Warum? Weil bei "ente" der letzte Buchstabe das "e" ist und im Ausdruck aber steht, dass nach dem nächsten "e" noch mindestens ein Buchstabe kommen muss(-> "+" = 1 oder mehr vorkommen).

So, das wär's nun gewesen, wenn mir noch etwas einfällt, werd ich nicht zögern, es dazuschreiben. Anbei hier noch einige nützliche RegEx-Codes

****

```
1         ### HTML-tags finden:
2         <.*?>
3
4         ### Strings finden:
5         \".*?\"
6
7         ### Links aus HTML-Quelltext filtern:
8         <a href\=\".*?>
9
10        ### MAC-Adresse validieren:
11        ([a-f0-9]{2}\-){5}[a-f0-9]{2}
12
13        ### Dateipfade aufspüren:
14        [a-z]\:\\.*\.[a-z]{1-4}
15
16        ### Uhrzeit finden:
17        (0.|[1][0-9]|[2][0-3])\:[0-5][0-9]\:[0-5][0-9]
18        aber noch besser ist folgende Variante, da hierbei die Uhrzeit im format 23:15:
19        auch nur 23:15. Dasselbe funktioniert hierbei auch mit "-" oder "." als Trennze
20        \s(0.|[1][0-9]|[2][0-3])(\:|\.|\-)[0-5][0-9]((\:|\.|\-)[0-5][0-9]|)(\,[0-5][0-9
21
22        ### CLSID anzeigen:
23        \{[0-9a-f]{8}\-([0-9a-f]{4}\-){3}[0-9a-f]{12}\}
24
25        ### IP-Adresse finden:
26        (([0-9]|[1][0-9]|[1][0-9][0-9]|[2][0-5]{2})\.){3}([0-9]|[1][0-9]|[1][0-9][0-9]|
27
```

```
### Internetadressen finden:
(http\:\/\/|www.).*\.[a-z]{1-4}

### Email adressen rausfischen:
[a-zA-Z0-9|a-zA-Z0-9\.]+\@[a-zA-Z0-9|a-zA-Z0-9\-]+\.[com|de|net|mu]+
oder:
\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
oder:
[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[

gov|biz|info|name|aero|biz|info|jobs|museum)\b



####### Visual Basic #######

### Variablen aufspüren:
\s(dim|private|friend|static|public) [a-z0-9]+ As (new [a-z]+.*|[a-z]+)

### Importiertes
(^|\n)Imports .*?\n

### Kommentare finden
\'.*

### Subs finden
\s[a-z]+\ssub\s.*?\n
Dasselbe einfach auch mit Function oder Property

### Variablen eines bestimmten Datentyps finden(hier String)
\s(dim|private|friend|static|public) [a-z0-9]+ as (new [a-z]+.*string\)|string)

### True/False finden
.*(True|False).*

### Event handlers:
\s[a-z]+\ssub [a-z0-9_\(\)\,\. ]+ handles [a-z0-9]+\.[a-z]+
### NUR die Handler:
\shandles [a-z0-9]+\.[a-z]+
### Nur Subs/Functions ohne Parameter:
([a-z]+|) (sub|function) [a-z0-9_]+\(\)
### Nur Subs/Functions mit parametern:
([a-z]+|) (sub|function) [a-z0-9_\-]+\(((optional|byval|byref) .*?\)).*
```

Ich bedanke mich nochmal für's lesen meines Tutorials. Viel Spaß damit, Link