André Obelink (MCSD) is als Technical Manager werkzaam bij AcouSoft Informatisering B.V.

Ontdek de krachtige techniek van Regular Expressions

.NET FRAMEWORK MAAKT TOEPASSING REGULAR EXPRESSIONS EENVOUDIG

Met de komst van het .NET Framework heeft elke Visual Studio .NET-ontwikkelaar een solide engine tot zijn beschikking gekregen waarmee het werken met regular expressions erg eenvoudig wordt. Toch laten vele ontwikkelaars deze mogelijkheden nog steeds links liggen. Dat is jammer, omdat het juist zo'n krachtig gereedschap is dat je veel tijd en energie kan besparen. Het is dus de hoogste tijd om kennis te maken met deze mooie techniek.

Wat zijn regular expressions eigenlijk? Regular expressions zijn patronen of patterns die een bepaalde tekst of een deel ervan beschrijven. Met regular expressions kun je zoeken naar de aanwezigheid van dergelijke tekstblokken en optioneel kun je ze ook vervangen of verwijderen. Het beschrijven van patronen is een programmeertaal op zich; hier zijn dan ook al veel boeken over verschenen. In de eenvoudigste vorm kan je ze vergelijken met 'wildcard'-karakters uit MS-DOS om een groep van bestanden aan te geven, zoals '*.exe'. Regular expressions kunnen er heel ingewikkeld uitzien, maar het hoeft niet. Een regular expression waarmee men een Nederlandse postcode kan valideren ziet er als volgt uit:

```
^[1-9][0-9]{3}\s?[a-zA-Z]{2}$
```

Met deze regular expression kan men dus snel vaststellen dat '7101CK', '7101 CK', '7101ck' en '7101 ck' geldige postcodes zijn. Hij is zelfs zo slim dat hij kan bepalen dat de postcode '0710CK' niet geldig is. Als regular expressions nieuw voor je zijn, roept de bovenstaande 'regex' waarschijnlijk meteen de vraag op: Wat betekenen al die karakters? Een regex kan beginnen met een '^' en eindigen met een '$'. Wanneer je deze regex gebruikt om te kijken of er zich een postcode in de tekst bevindt, zal het niet werken. De tekst die wordt aangeboden mag dus alleen de postcode zijn. De beginmarkering wordt gevolgd door '[1-9]' wat inhoudt dat het eerste getal een getal tussen de 1 en 9 moet zijn. Door deze restrictie wordt de postcode '0710CK' direct afgekeurd. Het volgende blok wordt gevormd door '[0-9]' direct gevolgd door '{3}'. Analoog aan het eerste blok betekent '[0-9]' een getal tussen de 0 en 9, maar doordat er een '{3}' achter staat moet deze voorwaarde exact 3 keer voorkomen. Als gevolg van deze beperking wordt een postcode zoals '710CK' niet goedgekeurd.

```
' Maak object met het patroon (pattern); indit geval
' bevat txtRegExe dus: ^[1-9][0-9]{3}\s?[a-zA-Z]{2}$
Dim objRegEx As New Regex(txtRegEx.Text)

' Test ingevoerde postcode --> txtPostcode bevat '7101CK'
txtResult.Text = objRegEx.Match(txtPostcode.Text).Success
```

Codevoorbeeld 1. Controleer een postcode

De volgende combinatie is ook erg interessant. We kijken eerst naar '\s' wat een spatie inhoudt. Doordat deze direct gevolgd wordt door een '?' wordt de '\s' uitgebreid met de regel dat de spatie of 0 of 1 keer voor moet komen. Hierdoor worden postcodes met zowel een spatie als zonder spatie goedgekeurd. Omdat ik het niet belangrijk vind dat mijn gebruikers hoofdletters voor de lettercombinatie moeten intypen, wordt de lettercombinatie als [a-zA-Z] gedefinieerd. Deze letter moet wel 2 keer achter elkaar voorkomen. Dit verklaart de laatste '{2}'.

Vanzelfsprekend is het belangrijk dat je deze, maar ook andere taalkenmerken van de regular expressions in de vingers krijgt. Daarnaast moet je natuurlijk ook weten hoe je een hierboven beschreven controle met bijvoorbeeld Visual Basic .NET implementeert. Ik kan je geruststellen, dat gaat erg eenvoudig. Je moet weten dat de regular expression-classes zich in de namespace **System.Text.RegularExpressions** bevinden. In alle codevoorbeelden ga ik er van uit dat deze namespace geïmporteerd is.

In codevoorbeeld 1 wordt de *Regex*-class gebruikt. Dit is de belangrijkste class in de namespace System.Text.RegularExpressions. Bijna alle codevoorbeelden gebruiken shared methoden van deze class of een instantie er van.

DE REGULAR EXPRESSION-TAAL

In het voorbeeld met de postcodecontrole heb je al kennisgemaakt met een aantal elementen van de regular expression-taal. Het ontbreekt in dit artikel aan de ruimte om alle elementen aan bod te laten komen, laat staan uitgebreid te belichten. Ik probeer in een aantal categorieën de belangrijkste elementen te bespreken.

Afbeelding 1. Controleer een postcode

Afbeelding 2. Vervang dubbele woorden die direct achter elkaar staan

De groepen zijn gebaseerd op de indeling zoals die veel wordt gebruikt op diverse websites, waaronder ook de MSDN-site.

Character escapes

De elementairste vorm van een regex bestaat uit één letter, bijvoorbeeld 'a'. Het zal matchen met het eerste karakter dat hiermee overeenkomt. In de zin 'Ga maar naar huis' zal het in eerste instantie dus matchen met de 'a' achter de 'G'. Zo zou je ook kunnen zoeken op een combinatie van letters zoals 'aar'. Omdat we in veel gevallen natuurlijk meer willen doen dan zoeken alleen, is er een aantal karakters gereserveerd. Deze 12 karakters, .$^{[(|)*+?\, hebben een speciale betekenis. Soms wil je de functie van deze metakarakters uitschakelen, waardoor ze tijdelijk uit hun functie worden ontheven. Je kunt ze daarna als gewone karakter gebruiken. Bekijk het volgende voorbeeld maar eens.

```
^2+3=5$
^2\+3=5$
```

Beide voorbeelden zijn geldige regular expressions, maar de eerste regular expression zal '223=5' valide vinden en dat is in deze context dus niet juist.

Character classes

Een 'character class', ook wel 'character set' genoemd, zal matchen met een groep van karakters die binnen blokhaken is gedefinieerd. Een voorbeeld is '[1-9]' of '[^3-5]', waarbij de laatste overeenkomt met alle karakters die juist *niet* in de groep zitten. In dit geval dus niet de getallen 3, 4 en 5. Je hoeft metakarakters niet uit te schakelen wanneer ze binnen blokhaken voorkomen, behalve het liggend streepje '-' en de rechter blokhaak ']' die binnen de blokhaken een speciale betekenis hebben.

Atomic zero-width assertions

Er zijn ook metakarakters die niet zozeer een bepaald karakter of groep controleren, maar die de match baseren op de huidige positie binnen de tekst. Neem bijvoorbeeld het '^' karakter dat aangeeft dat het huidige karakter aan het begin van de tekst zit. De regular expression '^http' retourneert dus alleen die matches die aan het begin van de tekst staan. Zo zijn er ook metakarakters die aangeven dat de expression juist aan het einde van de tekst moet staan en dat rekening gehouden moet worden met meer regels of juist niet, of dat het binnen een woord op een bepaalde positie moet staan. Met dit soort karakters kan men bijvoorbeeld razendsnel alle URL's uit een tekst filteren.

Quantifiers

De zogenaamde 'quantifiers' geven een aantal aan binnen de regular expression. De quantifier heeft altijd betrekking op de karakter, groep of karakter-class die direct *ervoor* staat. In onze postcodecontrole hebben we reeds kennisgemaakt met de '{3}', '{2}' maar ook met de '?'. Naast andere karakters zoals de '*' of '+' zijn er ook variaties te bedenken op {*n*}, zoals {*n*,} of {*n*,*m*}. De regular expression '[0-9]{1,}' zegt dat er ten minste één getal moet staan, terwijl '[0-9]{1,3}' aangeeft dat er ten minste één getal moet staan, maar niet meer dan drie.

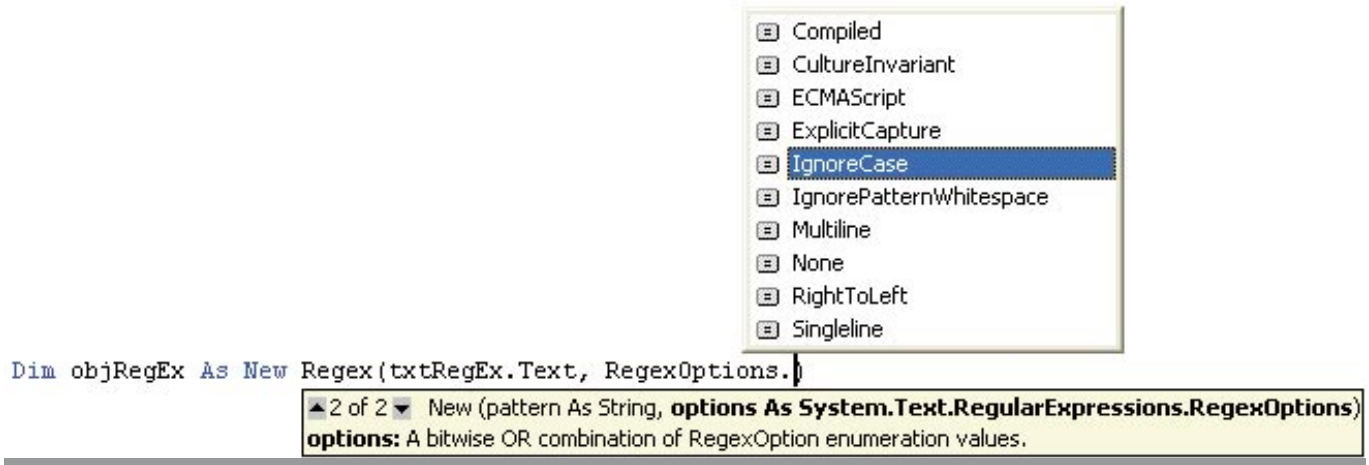
Grouping constructs

Met de ronde haken '()' kan men groepen definiëren van zogenaamde subexpressions. Dit heeft onder meer als doel de efficiëntie van de regular expression te verhogen. Dit kan omdat je bepaalde regex-operatoren kan laten toepassen op de hele groep. Naast het groeperen genereert het plaatsen van een expression tussen ronde haken ook een 'backreference', waardoor het gedeelte van de expression dat overeenkomt bewaard wordt. Hierdoor kun je dat deel weer hergebruiken. Denk hierbij aan het zoeken naar dubbele woorden in een tekst die direct achter elkaar staan om deze later te laten vervangen door het enkele woord. Met een regular expression is dit namelijk vrij gemakkelijk te realiseren. Dit zou je kunnen doen met de .Replace()-methode van de Regex-class, maar ook met de Replace()-methode van een string-object. In afbeelding 2 zie je hoe een en ander er in de praktijk uitziet.

De ronde haken worden in dit geval gebruikt om een woord te selecteren dat direct achter elkaar twee keer voorkomt. De gevonden woorden worden in een collectie geplaatst (MatchCollection) en binnen de items van deze collectie in de groep 'DubbelWoord' gezet. Doordat er in dit geval een groepsnaam wordt aangegeven, kan men hier later ook 'named' naar refereren. Zoals je kunt zien in codevoorbeeld 2 hoef je niet al te veel te coderen om al die functionaliteit te implementeren. Het commentaar in code zal waarschijnlijk voldoende houvast geven om te begrijpen hoe een en ander werkt.

```
' Maak object aan op basis van het pattern en geef
' tegelijkertijd aan dat het niet contextgevoelig is
Dim objRegEx As New Regex(txtRegEx.Text, RegexOptions.IgnoreCase)

' Vul collectie met alle woorden die dubbel voorkomen
Dim mclMatchCollection As MatchCollection = objRegEx.Matches(txtTest.Text)

' Toon een lijst met alle matches
For intA As Int32 = 0 To mclMatchCollection.Count - 1
    txtResult.Text += mclMatchCollection.Item(intA).Value & _
    Environment.NewLine
Next

' Scheidingslijn
txtResult.Text += "----------" & Environment.NewLine

Dim strModidiedText As String = txtTest.Text

' Je kunt binnen de groepen van het item zoeken naar het woord dat
' dan dubbel voorkomt dus bijvoorbeeld 'en' i.p.v. 'en en'
For intA As Int32 = 0 To mclMatchCollection.Count - 1 txtResult.Text +=
  mclMatchCollection.Item(intA).Groups("DubbelWoord").ToString & _
  Environment.NewLine()

    ' Vervang de dubbele woorden door de enkele
    With mclMatchCollection.Item(intA)
       strModidiedText = strModidiedText.Replace(.Value.ToString, _
       .Groups("DubbelWoord").ToString)
    End With
Next

' Toon aangepast tekst
txtModifiedText.Text = strModidiedText
```

Codevoorbeeld 2. Vervang dubbele woorden die direct achter elkaar staan



Afbeelding 3. Alternation constructs

Alternation constructs

Wanneer een (sub)expression moet overeenkomen met een specifiek aantal mogelijkheden, dan kan men er voor kiezen de mogelijkheden te scheiden door het pipe-teken '|' vast te leggen. Stel dat je in een tekst zoekt naar of 'xxx' of 'yyy' of 'zzz', dan kan men deze scheiden door het '|' teken en vervolgens deze mogelijkheden omsluiten door ronde haken. In het volgende voorbeeld klopt de regular expression wanneer zich in de zin of het woord 'audi', 'volvo' of 'bmw' bevindt. Doordat ik in de constructor van de class heb aangegeven om niet contextgevoelig te vergelijken, ziet men dat 'Volvo' ook overeenkomt met het pattern. In de volgende onderdeel ga ik hier dieper op in.

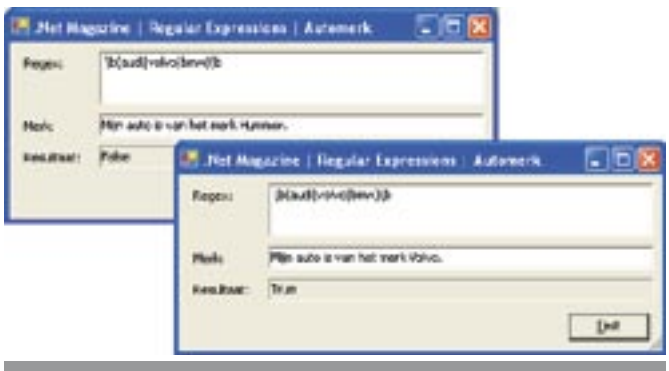
In de meeste codevoorbeelden toon ik in het tekstvak 'Resultaat' de waarde van de .Success-property van de Match-class. Ik zou echter ook de .Value-property kunnen tonen. Deze bevat dan altijd het meest linkse expression-deel dat overeenkomt. In de zin: "*Mijn auto's zijn van het merk Volvo en BMW.*" retourneert deze eigenschap, op basis van de gebruikte regular expression, de waarde 'Volvo'. In het codevoorbeeld dat je van de website van Microsoft (www.microsoft.nl/netmagazine9) kunt downloaden, zit ook nog een voorbeeld waarmee men met behulp van 'Alternating constructs' een tijd kan controleren.

Matching modes

Onder een 'matching mode' wordt eigenlijk niets ander dan een manier van vergelijken verstaan. De meeste regular expression-engines ondersteunen ten minste deze drie matching modes.

- '/i' maakt een regex context*on*gevoelig.
- '/s' zet de zogenaamde 'singleline mode'aan. In dit geval geeft de punt '.' een nieuwe regel aan.
- '/m' zet de zogenaamde 'multiline mode'aan. In dit geval geeft de '^' en '$' het begin en einde van de string aan.

Bepaalde regex-dialecten kennen nog meer manieren of opties die door een enkele letter worden aangegeven. Met het .NET Framework en de Regex-class kan men deze ook met bijvoorbeeld VB.NET-code instellen. Zo wordt in codevoorbeeld 2 bij het instantiëren van obj-Regex de optie *RegexOptions.IgnoreCase* meegegeven, waarmee het object feitelijk de '/i'-modifier toevoegt. Binnen het .NET Framework kan men echter kiezen uit nog een aantal andere opties.



Afbeelding 4. System.Text.RegularExpressions.RegexOptions

Substitutions

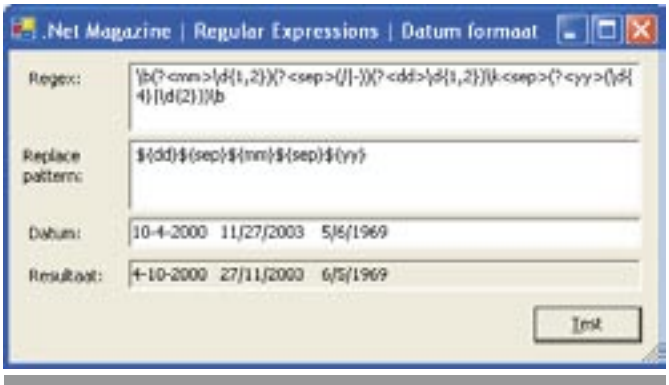
Tot nu toe hebben we het alleen gehad over 'matching patterns', dus het testen of de pattern in een tekst voorkomt. Naast deze matching patterns kennen we ook nog 'replacement patterns'. Substitutions zijn toegestaan binnen deze replacement patterns. Hiermee kun je tekst ook wijzigen door de gevonden expression(s) te vervangen door een bepaalde waarde. Deze patterns kunnen worden gebruikt binnen de Regex.Replace-methode. Door gebruik te maken van een pattern kun je dus heel veel logica in de Replace-methode implementeren en is deze Replace-methode superieur aan de Replace-methode van een string-object. Het volgende voorbeeld illustreert de kracht van deze methodiek. Het voert op dit moment te ver om uitgebreid op regular expressions in te gaan, maar het komt er in grote lijnen op neer dat de datum in groepjes wordt geknipt, namelijk in <mm>, <dd> en <yy>. De separator wordt bewaard in <sep> en met het replace pattern wordt vervolgens de volgorde van deze groepjes gewijzigd. Hierdoor blijft ook de oorspronkelijke separator bewaard.
De code die nodig is om de feitelijke converteeractie uit te voeren is minimaal.

.NET FRAMEWORK

We hebben al kennis gemaakt met de Regex-class en de methoden *Match*, *Matches* en *Replace*. Er zijn echter meer methoden die we kunnen aanspreken. Daarnaast heb je in afbeelding 4 ook kunnen zien dat er een aantal opties aangegeven kan worden. In dit onderdeel wil ik een aantal ervan kort toelichten.

RegexOptions.Multiline

Deze optie maakt het mogelijk om in multiline mode te werken. Dit is vooral handig wanneer je grote hoeveelheden tekst



Afbeelding 5. Een voorbeeld van substitutie

```
' Maak object met het patroon (pattern)
' \b(?<mm>\d{1,2})(?<sep>(/|-))(?<dd>\d{1,2})\k<sep>(?<yy>(\d{4}|\d{2}))\b
Dim objRegEx As New Regex(txtRegEx.Text)

' Vervang de 'blokken' en converteer dus het datum
' formaat van mm-dd-yyyy naar dd-mm-yyyy
txtResult.Text = objRegEx.Replace(txtDate.Text, txtReplacePattern.Text)
```

Codevoorbeeld 3. Converteer datums naar Nederlands datumformaat

```
' Maak object met alle informatie die nodig is
Dim rciPostcode As New
RegexCompilationInfo("^[1-9][0-9]{3}\s?[a-zA-Z]{2}$", _
RegexOptions.Compiled, "RegexPostcode", "DotNetMagazineRegex", True)

' Maak een array en vul deze, in dit geval met één object
Dim objRegexCompInfoArray() As RegexCompilationInfo = {rciPostcode}

' Definieer assembly --> stel naam DLL in
Dim objAssemblyName As New System.Reflection.AssemblyName
objAssemblyName.Name = "MijnRegularExpressions"

' Maak de assembly
Regex.CompileToAssembly(objRegexCompInfoArray, objAssemblyName)
```

Codevoorbeeld 4. Maak MijnRegularExpressions.dll

```
' Maak PostcodeRegex aan --> uit DLL
Dim rgxPostcode As New DotNetMagazineRegex.RegexPostcode

' Gebruik direct de functionaliteit
MessageBox.Show(rgxPostcode.Match("7101 CK").Success)
```

Codevoorbeeld 5. Test MijnRegularExpressions.dll

```
' Maak een Regexobject aan
Dim objRegex As New Regex("\s*,\s*")
Dim strCSV As String = "Audi,   BMW,,Volvo   ,  Peugeot  "

' Split de CSV string naar een array
For Each strElement As String In objRegex.Split(strCSV)
    ' Toon elk element, zonder (voorloop)spaties
    MessageBox.Show("-" & strElement & "-")
Next
```

Codevoorbeeld 6. De Split()-methode

moet verwerken. Deze optie wijzigt eigenlijk de werking van de eerder besproken karakters ^ en $, waardoor deze het begin en einde van een regel markeren in plaats van de hele expression.

RegexOptions.Compiled

Standaard transformeert de Regex-class de regular expression naar zogenaamde 'opcodes' die worden geïnterpreteerd wanneer het desbetreffende pattern wordt uitgevoerd tegen de brontekst. Je kunt ook de optie RegexOptions.Compiled aangeven. Hierdoor wordt de regular expression expliciet geconverteerd naar Microsoft Intermediate Language (MSIL). Hierdoor kan de JIT-compiler de expression converteren naar native CPU-instructies, waardoor de performance van de regular expression toeneemt. Omdat het extra compileren overhead met zich meebrengt, moet je deze optie alleen gebruiken wanneer de regular expression vaker gebruikt wordt. In theorie zal een gecompileerde regular expression ook meer geheugenruimte in beslag nemen omdat de MSIL-code, na het releasen van het Regex-object, niet wordt opgeruimd door de Garbage Collector.

CompileToAssembly-methode

Hoewel de vertraging van het compileren van de regular expression in veel gevallen te verwaarlozen is, zou je ook kunnen besluiten een of meer regular expressions voor te compileren. Dit kun je doen met de shared methode CompileToAssembly. De assembly die gegenereerd wordt bevat voor elke regular expression een type dat is afgeleid van de Regex-class. Vervolgens kun je binnen de Visual Studio .NET-omgeving een referentie zetten naar deze dll en de functionaliteit aanspreken. In het volgende codevoorbeeld gaan we onze regular expression, waarmee een Nederlandse postcode wordt gecontroleerd, beschikbaar maken als assembly. Je zou echter ook verscheidene regular expressions kunnen toevoegen. Het definiëren komt natuurlijk overeen met de definitie van in dit geval rciPostcode en is van het type RegexCompilationInfo. Deze objecten voeg je vervolgens toe aan de array, bijvoorbeeld door ze tussen de accolades te plaatsen en door kommascheidingen toe te passen.

Het gebruiken van onze assembly is zeer eenvoudig. Het enige dat we moeten doen is een referentie maken naar onze 'MijnRegularExpressions.dll' en twee regels code toevoegen. Deze dll bevindt zich overigens in de \bin map. Als de referentie is toegevoegd, kunnen we volstaan met de code uit codevoorbeeld 5.

Split-methode

De Split-methode op de Regex-class komt sterk overeen met de String.Split-methode. Het enige verschil zit hem in de definitie van het scheidingsteken; deze wordt bij de Regex-class natuurlijk gedefinieerd door een regular expression. Codevoorbeeld 6 toont alle woorden in een kommagescheiden tekst, waarbij spaties die voor of achter het woord staan worden genegeerd. Een schoolvoorbeeld van hoe je met een minimale hoeveelheid code een maximaal rendement kan behalen.

Shared methoden

De methoden die we tot nu toe hebben besproken zijn ook allemaal beschikbaar als Shared methoden. Hierdoor hoef je niet eerst een Regex-object te declareren en instantiëren. Alle informatie die de methode nodig heeft om goed te functioneren dient als parameters te worden meegegeven. Ook is er een aantal shared methoden die geen overeenkomstige instance-methode heeft. Voorbeelden hiervan zijn de Escape- en Unescape-methode, die een metakarakter converteren naar de escaped versie van dat karakter en eventueel weer terug.

```
'Share method voor unescaping van karakters \(i\) --> (i)
MessageBox.Show(Regex.Unescape("\(i\)"))

' Je kunt unescaping ook gebruiken om Carriage Returns
' of Line feeds toe te voegen. Dit resulteert in:
' Naam
' Adres
' Postcode Plaats
MessageBox.Show(Regex.Unescape("Naam\r\nAdres\r\nPostcode Plaats"))
```

Codevoorbeeld 7. De shared methode Unescape()

```
Dim strResult As String

' Maak object met het patroon (pattern)
Dim objRegex As New Regex(txtRegex.Text)

' Doorloop alle gevonden datums en plaats deze
For Each objMatch As Match In objRegex.Matches(txtSource.Text)
    strResult += objMatch.Value & Environment.NewLine
Next

' Toon resultaat
txtResult.Text = strResult
```

Codevoorbeeld 8. De MatchCollection kun je ook doorlopen met een For... Each-lus

Afbeelding 6. Toon elke match in de MatchCollection

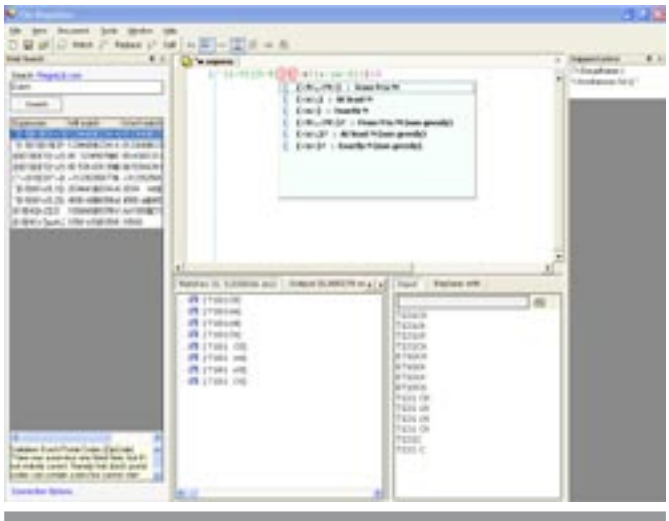
Match, MatchCollection en Group-class

Naast de Regex-class kent de System.Text.RegularExpressions-namespace ook nog een drietal andere classes. In onze codevoorbeelden zijn we ze alle drie al tegengekomen.

De MatchCollection-class is een collectie van matches, dus objecten van het type Match. Een object van het type MatchCollection is het resultaat van de Matches-methode op de Regex-class. In codevoorbeeld 2 kun je hier een voorbeeld van vinden. Een Match-object heeft een aantal properties waaronder Value, IsSuccess en Length (de lengte van de Value). Op zich spreken deze properties voor zich. De combinatie van deze twee classes kunnen tot heel krachtige toepassingen leiden. In het navolgende voorbeeld wordt de MatchCollection gevuld met alle data die zich in de tekst bevinden. De datum mag een voorloopnul hebben. Eventueel mag het jaartal uit slechts twee cijfers bestaan.

HULPMIDDELEN

Het bouwen van een regular expression kan een ingewikkelde en tijdrovende bezigheid zijn. Gelukkig staat een aantal tools tot je beschikking waarmee het bouwen van of experimenteren met regular expressions een stuk eenvoudiger wordt. Een van de tools die niet op je pc mag ontbreken is 'The Regulator'. Met dit freeware programma kun je een regular expression schrijven en testen tegen een aantal expressions. Daarnaast heeft het programma de mogelijkheid een koppeling tot stand te brengen met de webservice van RegexLib.com. Op deze website staat een groot aantal regular expressions dat is bijgedragen en getest door collega-ontwikkelaars. Vanuit de Regulator kan je de webservice raadplegen en een overzicht krijgen van alle regular expressions die voldoen aan bijvoorbeeld het zoekcriterium 'Dutch'. Je kunt deze regular expressions direct testen en/of aanpassen en de configuratie opslaan voor later gebruik. De editor van de Regulator is meer dan alleen een editor zoals Notepad; het markeert bijvoorbeeld het begin en einde van de huidige groep en kent ook een soort Intellisense. Een must have.



Afbeelding 7. De Regulator

Een andere tool die iets minder uitgebreid maar wel overzichtelijker is, is RegexDesigner .NET. Het aardige van deze tool is dat het ook de broncode in VB.NET en C# kan genereren waarmee je de regex kan aanspreken. Beide tools zijn vrij te gebruiken en te downloaden van SourceForge.com en GotDotNet.com.

André Obelink (MCSD) is als Technical Manager werkzaam bij AcouSoft Informatisering B.V. Sinds 1995 is hij als eindredacteur betrokken bij de Nederlandse Visual Basic Groep (http://www.vbgroup.nl). Voor vragen en opmerkingen kun je hem bereiken op andre@obelink.com.

Nuttige internetadressen

http://www.regexlib.com/
http://www.regular-expressions.info
http://regexadvice.com
http://gmckinney.info/resources/regex.pdf
De besproken tools kun je downloaden van http://sourceforge.net/projects/regulator en http://www.sellsbrothers.com/tools/#regexd
De broncode behorende bij dit artikel is te downloaden van de site van Microsoft Nederland (http://www.microsoft.nl/netmagazine9) of van zijn eigen site (http://www.obelink.com).