By: Christopher Singleton

# VALIDATING USING REGULAR EXPRESSIONS

## Understanding Regular Expression Characters:

**Simple:** `"^\S+@\S+$"`

**Advanced:** `"^([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([\w-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$"`

| Character | Description |
|---|---|
| / .. / | All regular expressions start and end with forward slashes. |
| ^ | Matches the beginning of the string or line. |
| \w+ | Matches one or more word characters including the underscore. Equivalent to [A-Za-z0-9_]. |
| [\.-] | \ Indicates that the next character is special and not to be interpreted literally.<br>.- matches character . or -. |
| ? | Matches the previous character 0 or 1 time. Here previous character is [.-]. |
| \w+ | Matches 1 or more word characters including the underscore. Equivalent to [A-Za-z0-9_]. |
| * | Matches the previous character 0 or more times. |
| ([.-]?\w+)* | Matches 0 or more occurrences of [.-]?\w+. |
| \w+([.-]?\w+)* | The sub-expression \w+([.-]?\w+)* is used to match the username in the email. It begins with at least one or more word characters including the underscore, equivalent to [A-Za-z0-9_]. , followed by . or - and . or - must follow by a word character (A-Za-z0-9_). |
| @ | It matches only @ character. |
| \w+([.-]?\w+)* | It matches the domain name with the same pattern of user name described above.. |
| \.\w{2,3} | It matches a . followed by two or three word characters, e.g., .edu, .org, .com, .uk, .us, .co etc. |
| + | The + sign specifies that the above sub-expression shall occur one or more times, e.g., .com, .co.us, .edu.uk etc. |
| $ | Matches the end of the string or line. |

# USING HTML (EMAIL VALIDATION)

**HTML Email Validation:**

```
<div id="Email">
    <asp:TextBox ID="txb_Email" Csstype="email" Cssclass="BoxStyleMULI"
         placeholder="Email" runat ="server" required ="required" >
    </asp:TextBox>
    <asp:RegularExpressionValidator runat="server"
            ControlToValidate="txb_Email"
            ErrorMessage="*Not a valid email!"
            ValidationExpression="^([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]”+
                                 ”{1,3}\.)|(([\w-]+\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})”+
                                 ”(\]?)$">
    </asp:RegularExpressionValidator>
    <br />
</div>
```

# USING JAVASCRIPT (EMAIL VALIDATION)

**HTML:**

```
<div class="email">
<h2>Input an email and Submit</h2>
<ul>
<li><input type='text' name='text1'/></li>
<li> </li>
<li class="submit"><input type="submit" name="submit" value="Submit"
                                onclick="ValidateEmail(document.formMULI.txb_Email)"/>
</li>
<li> </li>
</ul>
</div>
<script src="email-validation.js"></script>
```

**JavaScript:**

```
<%@ Page Language = "C#" %>
<script runat="server">
    function ValidateEmail(inputText)
    {
        var mailformat = /^\w + ([\.-] ?\w +)*@\w + ([\.-] ?\w +)*(\.\w{ 2,3})+$/;
        if (inputText.value.match(mailformat))
        {
            document.formMULI.txb_Email.focus();
            return true;
        }
        else
        {
            alert("You have entered an invalid email address!");
            document.formMULI.txb_Email.focus();
            return false;
        }
    }
    </script>
```

## USING C# (EMAIL VALIDATION/REGEX)

**Note: Writes emails into two separate text files (Good emails and bad emails) from the original flat file.**

```
static void TestWritingToDataToSeperateTextFiles()
{
  string strSourceDataFileName =
                  @"C:\_BISolutions\Modlule07\CSharpEmailValidatorSol\DataToProcess\EmailData.txt";
  string strValidDataFileName =
              @"C:\_BISolutions\Modlule07\CSharpEmailValidatorSol\DataToProcess\ValidEmailData.txt";
  string strInValidDataFileName =
            @"C:\_BISolutions\Modlule07\CSharpEmailValidatorSol\DataToProcess\InValidEmailData.txt";
  try
  {
  System.IO.StreamReader objReadFile = new System.IO.StreamReader(strSourceDataFileName);
  System.IO.StreamWriter objWriteValidFile = new System.IO.StreamWriter(strValidDataFileName, false);
  System.IO.StreamWriter objWriteInValidFile =
                                          new System.IO.StreamWriter(strInValidDataFileName, false);
        string strLineData = "";
        while ((strLineData = objReadFile.ReadLine()) != null)
        {
            if (IsValid(strLineData))
            { objWriteValidFile.WriteLine(strLineData); }
            else
            { objWriteInValidFile.WriteLine(strLineData); }
        }

        objReadFile.Close();
        objWriteValidFile.Close();
        objWriteInValidFile.Close();
    }
    catch (System.Exception objException)
    {
        System.Console.WriteLine(objException.ToString());
    }
}
```

# USING C# (DATE VALIDATION/ENV-CULTURE-INFO)

**Note: Date validation has an added bonus, because you can use environment culture info or Regex.**

```
Console.Write("\n\n  Date (mm/dd/yyyy): ");
myDate = Console.ReadLine();

if (myDate == "")
{
    DateTime result;
    dateString = "10/13/2015";
    format = "d";
    CultureInfo provider = CultureInfo.InvariantCulture;
    result = DateTime.ParseExact(dateString, format, provider);
    System.Globalization.CultureInfo MyCultureInfo =
                                          new System.Globalization.CultureInfo("en-US");
    DateTime MyDateTime = DateTime.Parse(myDate, MyCultureInfo);

}
else
{
    DateTime result;
    dateString = "11/30/2015";
    format = "d";
    CultureInfo provider = CultureInfo.InvariantCulture;
    result = DateTime.ParseExact(dateString, format, provider);
    System.Globalization.CultureInfo MyCultureInfo =
                                          new System.Globalization.CultureInfo("en-US");
    DateTime MyDateTime = DateTime.Parse(myDate, MyCultureInfo);
}
```

# USING C# (DATE VALIDATION/REGEX)

```
Console.Write("\n\n  Date (mm/dd/yyyy): ");
string myDate = Console.ReadLine();

Regex reg = new Regex("^(0[1-9]|1[012])[/](0[1-9]|[12][0-9]|3[01])[/](19|20)\\d\\d+$");

while (!reg.IsMatch(myDate))
{
    Console.ForegroundColor = ConsoleColor.DarkBlue;
    Console.Clear();
    Console.Write("\n  Vulcan Values Bookstore --- The Logical Choice LLAP" +
              "\n\n  ****** Purchase Book Information Screen ******\n\n");
    Console.ForegroundColor = ConsoleColor.Red;

    Console.Write("\n  Some of the date did not come out right." +
                  " Please put the date in again. ");
    Console.ForegroundColor = ConsoleColor.DarkBlue;
    Console.Write("\n  Date (mm/dd/yyyy): ");
    myDate = Console.ReadLine();
}
```

## USING POWERSHELL (EMAIL VALIDATION/REGEX)

```powershell
## PowerShell, Regex
## Test email addresses to see which ones are valid.
## This function creates invalid/valid csv files and returns valid/invalid to those files.

Function IsValidEmail    {
Param ([string] $In)
# Returns true if In is in valid e-mail format.
[system.Text.RegularExpressions.Regex]::IsMatch($In,
                "^([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([\w-]+\.)+))”+
                ”([a-zA-Z]{2,4}|[0-9]{1,3})(\]?)$");
} # End of IsValidEmail

## Now we need to check the original file for invalid and valid emails.
$list = Get-Content C:\Emails\OriginalEmails\emailAddresses.csv

##============== Test to see if the file exists ==========================
if (!(Test-Path "C:\Emails\ValidEmails\ValidEmails.csv"))
{
   New-Item -path C:\Emails\ValidEmails -name ValidEmails.csv -type "file"
   Write-Host "Created new file and text content added"
}
else
{
  Write-Host "File already exists and new text content added"
}
##=======================================================================

if (!(Test-Path "C:\Emails\InValidEmails\InValidEmails.csv"))
{
   New-Item -path C:\Emails\InValidEmails -name InValidEmails.csv -type "file"
   Write-Host "Created new file and text content added"
}
else
{
  # Add-Content -path C:\Emails\ValidEmails -value "new text content"
  Write-Host "File already exists and new text content added"
}

$EmailAddresses = Import-Csv "C:\Emails\OriginalEmails\emailAddresses.csv" -Header FirstName,
LastName, Email, Address, City, State, ZipCode  | Select -Skip 1
```

```powershell
                                                    # (Continued)

##================ Our Testing Emails using an Array ======================
# [string[]] $emailAddresses = "david.jones@proseware.com", "d.j@server1.proseware.com",
                               # "jones@ms1.proseware.com", "j.@server1.proseware.com",
                               # "j@proseware.com9", "js#internal@proseware.com",
                               # "j_9@[129.126.118.1]", "j..s@proseware.com",
                               # "js*@proseware.com", "js@proseware..com",
                               # "js@proseware.com9", "j.s@server1.proseware.com",
                               # "tfl@psp.co.uk", "cuddly.penguin@cookham.net"
##=========================================================================

# Array buffers:
$ValidEmails = @()
$InvalidEmails = @()

##==================== Use A ForEach Loop ================================

ForEach ($emailAddress in $list)
{
$ValidEmails   = $list |Where-Object {IsValidEmail $_.Email}
$InvalidEmails = $list |Where-Object {-not(IsValidEmail $_.Email)}

    if ($ValidEmails.Count -gt $50)

    {

       $EmailAddresses |Where-Object {IsValidEmail $_.Email} | Export-Csv
                                      "C:\Emails\ValidEmails\ValidEmails.csv" -NoTypeInformation

    }

    if ($InvalidEmails.Count -gt $50)

    {

      $EmailAddresses |Where-Object {-not(IsValidEmail $_.Email)} | Export-Csv
                                    "C:\Emails\InValidEmails\InValidEmails.csv" -NoTypeInformation

    }
}
```