**C# Regex Class – Fundamental Functionalities**

The Regex Class allows users to define specific patterns and match input strings against them. This can be used to extract substrings following a pattern (e.g. phone numbers or e-mails) and/or execute replace and split operations too complex for the ones of the String class (e.g. splitting a string whose parts are separated by hexadecimal numbers of a length between two and three digits that might included a “-“).

The class can be used by integrating the ***System.Text.RegularExpressions*** into one’s project.

**Defining a pattern**

*string pattern = @”\d{1, 3}”;* //(match one to three digits)

*string specialChar\_pattern = “[a-zA-Z0-9]{1}\\.[a-zA-Z0-9]{1}@example.com}”;*   
//Matches emails of the type x.y@example.com, xxxx.yyy@example.com, etc.  
//Requires backslashes for “.”, as without them, it is considered a wildcard character.

**Finding Matches**

*String pattern = “…”;  
String stringToMatch = “someText”;  
Regex rgx = new Regex(pattern);*  
  
Only first match => *Regex.Match(stringToMatch); //Datatype: Match*  
All matches => *Regex.Matches(stringToMatch); //Datatype: MatchCollection*

**Splitting**

*String pattern = “…”;  
String stringToSplit = “someText”;  
Regex rgx = new Regex(pattern);*

*string[] splitResult = rgx.Split(stringToSplit); // Contains an array of strings resulting from the split*

**Replacing**

*string pattern = “…”;* // The pattern to define where replacement should take place  
*string input = “myinput”;* // The string containing substrings to replace  
*string replacement = “theCharacterToInsertAsSubstitute”*; // This string to insert as replacement  
*Regex.Replace(input, pattern, replacement);* //This is a static class

**Important pattern “parts” (extract from list in Source 1)**

|  |  |  |
| --- | --- | --- |
| **Item** | **Function** | **Example** |
| \w | Matches any word character | In “Hi, Max”,  matches “H”, “i”, “M”, “a”, and “x” |
| \s | Matches any non-white space | In “It “  Matches “I” and “t” |
| \d | Matches any decimal digit | In “10 K9 dogs”  Matches “1”, “0” and “9” |
| \* | Matches the previous element zero or more times | For when using “a\*b” this would, for example, match: “b”, “ab”, “aab”, “aaab”, … |
| + | Matches the previous element one or more times | For “a\*b”,  Matches “ab”, “aab”, “aaab”, … but NOT “b” |
| ? | Matches the previous element zero or one time | For “a\*b”,  Matches “b” and “ab” |
| { n }  { n, }  { n, m } | Matches the previous element n times  Matches the previous element at least n times  Matches the previous element at least n and up to m times | “c{3}d” matches “cccd”  “c{3, }d” matches “cccd”, “ccccd”, “cccccd”, …  “c{3, 4}d” matches only “cccd” and “ccccd” |
| | | Matches any one element separated by the vertical bar character | In “wr(i | o)te” matches “write” and “wrote” |
| [ ] | Defines a character group and matches any single character within that character group | “[abdcde12345]{2}” matches “a1” in “this is a1” |
| [^ ] | Matches any character NOT in the character group defined | “[^1234567890]” matches “a”, “b” and “c” in “a1b2c3” |
| [ x – y ] | Matches any single character in the range from x to y (based on their ASCII position) | “[a - z]” matches all lower case character, e.g.  “y”, “e” and “z” in “H12yLMe3z“ |
| . | Matches any single character but \n | In “farm over form”, “f.rm” matches “farm” and “form” |
| ( ) | Defines a subexpression |  |
| \ | Either matches an escape character or, if the character is not one, the character itself. |  |

**Sources:**   
[1] <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference> [Last accessed: 2021-04-16]  
[2] <https://www.c-sharpcorner.com/article/c-sharp-regex-examples/> [Last accessed: 2021-04-16]  
[3] <https://www.c-sharpcorner.com/UploadFile/955025/regular-expression-in-C-Sharp/> [Last accessed: 2021-04-16]