Anchors in Regular Expressions

Article • 09/15/2021 • 21 minutes to read • 17 contributors

In this article

Anchors, or atomic zero-width assertions, specify a position in the string where a match must occur. When you use an anchor in your search expression, the regular expression engine does not advance through the string or consume characters; it looks for a match in the specified position only. For example, `^` specifies that the match must start at the beginning of a line or string. Therefore, the regular expression `^http:` matches "http:" only when it occurs at the beginning of a line. The following table lists the anchors supported by the regular expressions in .NET.

Anchor	Description
`^`	By default, the match must occur at the beginning of the string; in multiline mode, it must occur at the beginning of the line. For more information, see Start of String or Line.
`$`	By default, the match must occur at the end of the string or before `\n` at the end of the string; in multiline mode, it must occur at the end of the line or before `\n` at the end of the line. For more information, see End of String or Line.
`\A`	The match must occur at the beginning of the string only (no multiline support). For more information, see Start of String Only.
`\Z`	The match must occur at the end of the string, or before `\n` at the end of the string. For more information, see End of String or Before Ending Newline.

Anchor	Description
`\z`	The match must occur at the end of the string only. For more information, see End of String Only.
`\G`	The match must start at the position where the previous match ended. For more information, see Contiguous Matches.
`\b`	The match must occur on a word boundary. For more information, see Word Boundary.
`\B`	The match must not occur on a word boundary. For more information, see Non-Word Boundary.

Start of String or Line: ^

By default, the `^` anchor specifies that the following pattern must begin at the first character position of the string. If you use `^` with the RegexOptions.Multiline option (see Regular Expression Options), the match must occur at the beginning of each line.

The following example uses the `^` anchor in a regular expression that extracts information about the years during which some professional baseball teams existed. The example calls two overloads of the Regex.Matches method:

- The call to the Matches(String, String) overload finds only the first substring in the input string that matches the regular expression pattern.
- The call to the Matches(String, String, RegexOptions) overload with the `options` parameter set to RegexOptions.Multiline finds all five substrings.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-
1957\n" +
                       "Chicago Cubs, National League, 1903-present\n" +
                       "Detroit Tigers, American League, 1901-present\n" +
                       "New York Giants, National League, 1885-1957\n" +
                       "Washington Senators, American League, 1901-1960\n";
```

```
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-
(\d{4}|present))?,?)+";
        Match match;

        match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();

        match = Regex.Match(input, pattern, RegexOptions.Multiline);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//    The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-
1957.
//
//    The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-
1957.
//    The Chicago Cubs played in the National League in 1903-present.
//    The Detroit Tigers played in the American League in 1901-present.
//    The New York Giants played in the National League in 1885-1957.
//    The Washington Senators played in the American League in 1901-1960.
```

The regular expression pattern `^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+` is defined as shown in the following table.

Pattern	Description
`^`	Begin the match at the beginning of the input string (or the beginning of the line if the method is called with the RegexOptions.Multiline option).

Pattern	Description
`((\w+(\s?)){2,}`	Match one or more word characters followed either by zero or by one space at least two times. This is the first capturing group. This expression also defines a second and third capturing group: The second consists of the captured word, and the third consists of the captured white space.
`,\s`	Match a comma followed by a white-space character.
`(\w+\s\w+)`	Match one or more word characters followed by a space, followed by one or more word characters. This is the fourth capturing group.
`,`	Match a comma.
`\s\d{4}`	Match a space followed by four decimal digits.
`(-(\d{4}|present))?`	Match zero or one occurrence of a hyphen followed by four decimal digits or the string "present". This is the sixth capturing group. It also includes a seventh capturing group.
`,?`	Match zero or one occurrence of a comma.
`(\s\d{4}(-(\d{4}|present))?,?)+`	Match one or more occurrences of the following: a space, four decimal digits, zero or one occurrence of a hyphen followed by four decimal digits or the string "present", and zero or one comma. This is the fifth capturing group.

End of String or Line: $

The `$` anchor specifies that the preceding pattern must occur at the end of the input string, or before `\n` at the end of the input string.

If you use `$` with the RegexOptions.Multiline option, the match can also occur at the end of a line. Note that `$` matches `\n` but does not match `\r\n` (the combination of carriage return and newline characters, or CR/LF). To match the CR/LF character combination, include `\r?$` in the regular expression pattern.

The following example adds the `$` anchor to the regular expression pattern used in the example in the Start of String or Line section. When used with the original input string, which includes five lines of text, the Regex.Matches(String, String) method is unable to find a match, because the end of the first line does not match the `$` pattern. When the original input string is split into a string array, the Regex.Matches(String, String) method succeeds in matching each of the five lines. When the Regex.Matches(String, String, RegexOptions)

method is called with the `options` parameter set to RegexOptions.Multiline, no matches are found because the regular expression pattern does not account for the carriage return element (\u+000D). However, when the regular expression pattern is modified by replacing `$` with `\r?$`, calling the Regex.Matches(String, String, RegexOptions) method with the `options` parameter set to RegexOptions.Multiline again finds five matches.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string cr = Environment.NewLine;
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-
1957" + cr +
                       "Chicago Cubs, National League, 1903-present" + cr +
                       "Detroit Tigers, American League, 1901-present" + cr +
                       "New York Giants, National League, 1885-1957" + cr +
                       "Washington Senators, American League, 1901-1960" + cr;
        Match match;

        string basePattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-
(\d{4}|present))?,?)+";
        string pattern = basePattern + "$";
        Console.WriteLine("Attempting to match the entire input string:");
        match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();

        string[] teams = input.Split(new String[] { cr },
StringSplitOptions.RemoveEmptyEntries);
        Console.WriteLine("Attempting to match each element in a string
array:");
        foreach (string team in teams)
        {
            match = Regex.Match(team, pattern);
            if (match.Success)
```

```
            {
                Console.Write("The {0} played in the {1} in",
                              match.Groups[1].Value, match.Groups[4].Value);
                foreach (Capture capture in match.Groups[5].Captures)
                    Console.Write(capture.Value);
                Console.WriteLine(".");
            }
        }
        Console.WriteLine();

        Console.WriteLine("Attempting to match each line of an input string
with '$':");
        match = Regex.Match(input, pattern, RegexOptions.Multiline);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();

        pattern = basePattern + "\r?$";
        Console.WriteLine(@"Attempting to match each line of an input string
with '\r?$':");
        match = Regex.Match(input, pattern, RegexOptions.Multiline);
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//    Attempting to match the entire input string:
//
//    Attempting to match each element in a string array:
//    The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-
1957.
//    The Chicago Cubs played in the National League in 1903-present.
//    The Detroit Tigers played in the American League in 1901-present.
//    The New York Giants played in the National League in 1885-1957.
```

```
//    The Washington Senators played in the American League in 1901-1960.
//
//    Attempting to match each line of an input string with '$':
//
//    Attempting to match each line of an input string with '\r?$':
//    The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-
1957.
//    The Chicago Cubs played in the National League in 1903-present.
//    The Detroit Tigers played in the American League in 1901-present.
//    The New York Giants played in the National League in 1885-1957.
//    The Washington Senators played in the American League in 1901-1960.
```

Start of String Only: \A

The `\A` anchor specifies that a match must occur at the beginning of the input string. It is identical to the `^` anchor, except that `\A` ignores the RegexOptions.Multiline option. Therefore, it can only match the start of the first line in a multiline input string.

The following example is similar to the examples for the `^` and `$` anchors. It uses the `\A` anchor in a regular expression that extracts information about the years during which some professional baseball teams existed. The input string includes five lines. The call to the Regex.Matches(String, String, RegexOptions) method finds only the first substring in the input string that matches the regular expression pattern. As the example shows, the Multiline option has no effect.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "Brooklyn Dodgers, National League, 1911, 1912, 1932-
1957\n" +
                       "Chicago Cubs, National League, 1903-present\n" +
                       "Detroit Tigers, American League, 1901-present\n" +
                       "New York Giants, National League, 1885-1957\n" +
                       "Washington Senators, American League, 1901-1960\n";

        string pattern = @"\A((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-
(\d{4}|present))?,?)+";

        Match match = Regex.Match(input, pattern, RegexOptions.Multiline);
```

```
        while (match.Success)
        {
            Console.Write("The {0} played in the {1} in",
                          match.Groups[1].Value, match.Groups[4].Value);
            foreach (Capture capture in match.Groups[5].Captures)
                Console.Write(capture.Value);

            Console.WriteLine(".");
            match = match.NextMatch();
        }
        Console.WriteLine();
    }
}
// The example displays the following output:
//    The Brooklyn Dodgers played in the National League in 1911, 1912, 1932-
1957.
```

End of String or Before Ending Newline: \Z

The `\Z` anchor specifies that a match must occur at the end of the input string, or before `\n` at the end of the input string. It is identical to the `$` anchor, except that `\Z` ignores the RegexOptions.Multiline option. Therefore, in a multiline string, it can only match the end of the last line, or the last line before `\n`.

Note that `\Z` matches `\n` but does not match `\r\n` (the CR/LF character combination). To match CR/LF, include `\r?\Z` in the regular expression pattern.

The following example uses the `\Z` anchor in a regular expression that is similar to the example in the Start of String or Line section, which extracts information about the years during which some professional baseball teams existed. The subexpression `\r?\Z` in the regular expression `^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+\r?\Z` matches the end of a string, and also matches a string that ends with `\n` or `\r\n`. As a result, each element in the array matches the regular expression pattern.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "Brooklyn Dodgers, National League, 1911, 1912,
```

```
1932-1957",
                           "Chicago Cubs, National League, 1903-present" +
Environment.NewLine,
                           "Detroit Tigers, American League, 1901-present" +
Regex.Unescape(@"\n"),
                           "New York Giants, National League, 1885-1957",
                           "Washington Senators, American League, 1901-1960" +
Environment.NewLine};
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-
(\d{4}|present))?,?)+\r?\Z";

        foreach (string input in inputs)
        {
            Console.WriteLine(Regex.Escape(input));
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine("   Match succeeded.");
            else
                Console.WriteLine("   Match failed.");
        }
    }
}
// The example displays the following output:
//    Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
//       Match succeeded.
//    Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
//       Match succeeded.
//    Detroit\ Tigers,\ American\ League,\ 1901-present\n
//       Match succeeded.
//    New\ York\ Giants,\ National\ League,\ 1885-1957
//       Match succeeded.
//    Washington\ Senators,\ American\ League,\ 1901-1960\r\n
//       Match succeeded.
```

End of String Only: \z

The `\z` anchor specifies that a match must occur at the end of the input string. Like the `$` language element, `\z` ignores the RegexOptions.Multiline option. Unlike the `\Z` language element, `\z` does not match a `\n` character at the end of a string. Therefore, it can only match the last line of the input string.

The following example uses the `\z` anchor in a regular expression that is otherwise identical to the example in the previous section, which extracts information about the years during which some professional baseball teams existed. The example tries to match each of five elements in a string array with the regular expression pattern `^((\w+(\s?))`

`{2,}),\s(\w+\s\w+),(\s\d{4}(-(\d{4}|present))?,?)+\r?\z`. Two of the strings end with carriage return and line feed characters, one ends with a line feed character, and two end with neither a carriage return nor a line feed character. As the output shows, only the strings without a carriage return or line feed character match the pattern.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string[] inputs = { "Brooklyn Dodgers, National League, 1911, 1912,
1932-1957",
                            "Chicago Cubs, National League, 1903-present" +
Environment.NewLine,
                            "Detroit Tigers, American League, 1901-present\n",
                            "New York Giants, National League, 1885-1957",
                            "Washington Senators, American League, 1901-1960" +
Environment.NewLine };
        string pattern = @"^((\w+(\s?)){2,}),\s(\w+\s\w+),(\s\d{4}(-
(\d{4}|present))?,?)+\r?\z";

        foreach (string input in inputs)
        {
            Console.WriteLine(Regex.Escape(input));
            Match match = Regex.Match(input, pattern);
            if (match.Success)
                Console.WriteLine("   Match succeeded.");
            else
                Console.WriteLine("   Match failed.");
        }
    }
}
// The example displays the following output:
//    Brooklyn\ Dodgers,\ National\ League,\ 1911,\ 1912,\ 1932-1957
//       Match succeeded.
//    Chicago\ Cubs,\ National\ League,\ 1903-present\r\n
//       Match failed.
//    Detroit\ Tigers,\ American\ League,\ 1901-present\n
//       Match failed.
//    New\ York\ Giants,\ National\ League,\ 1885-1957
//       Match succeeded.
//    Washington\ Senators,\ American\ League,\ 1901-1960\r\n
//       Match failed.
```

Contiguous Matches: \G

The `\G` anchor specifies that a match must occur at the point where the previous match ended. When you use this anchor with the Regex.Matches or Match.NextMatch method, it ensures that all matches are contiguous.

The following example uses a regular expression to extract the names of rodent species from a comma-delimited string.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "capybara,squirrel,chipmunk,porcupine,gopher," +
                       "beaver,groundhog,hamster,guinea pig,gerbil," +
                       "chinchilla,prairie dog,mouse,rat";
        string pattern = @"\G(\w+\s?\w*),?";
        Match match = Regex.Match(input, pattern);
        while (match.Success)
        {
            Console.WriteLine(match.Groups[1].Value);
            match = match.NextMatch();
        }
    }
}
// The example displays the following output:
//       capybara
//       squirrel
//       chipmunk
//       porcupine
//       gopher
//       beaver
//       groundhog
//       hamster
//       guinea pig
//       gerbil
//       chinchilla
//       prairie dog
//       mouse
//       rat
```

The regular expression `\G(\w+\s?\w*),?` is interpreted as shown in the following table.

Pattern	Description
`\G`	Begin where the last match ended.
`\w+`	Match one or more word characters.
`\s?`	Match zero or one space.
`\w*`	Match zero or more word characters.
`(\w+\s?\w*)`	Match one or more word characters followed by zero or one space, followed by zero or more word characters. This is the first capturing group.
`,?`	Match zero or one occurrence of a literal comma character.

Word Boundary: \b

The `\b` anchor specifies that the match must occur on a boundary between a word character (the `\w` language element) and a non-word character (the `\W` language element). Word characters consist of alphanumeric characters and underscores; a non-word character is any character that is not alphanumeric or an underscore. (For more information, see Character Classes.) The match may also occur on a word boundary at the beginning or end of the string.

The `\b` anchor is frequently used to ensure that a subexpression matches an entire word instead of just the beginning or end of a word. The regular expression `\bare\w*\b` in the following example illustrates this usage. It matches any word that begins with the substring "are". The output from the example also illustrates that `\b` matches both the beginning and the end of the input string.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "area bare arena mare";
        string pattern = @"\bare\w*\b";
        Console.WriteLine("Words that begin with 'are':");
        foreach (Match match in Regex.Matches(input, pattern))
```

```
            Console.WriteLine("'{0}' found at position {1}",
                              match.Value, match.Index);
    }
}
// The example displays the following output:
//       Words that begin with 'are':
//       'area' found at position 0
//       'arena' found at position 10
```

The regular expression pattern is interpreted as shown in the following table.

Pattern	Description
`\b`	Begin the match at a word boundary.
`are`	Match the substring "are".
`\w*`	Match zero or more word characters.
`\b`	End the match at a word boundary.

Non-Word Boundary: \B

The `\B` anchor specifies that the match must not occur on a word boundary. It is the opposite of the `\b` anchor.

The following example uses the `\B` anchor to locate occurrences of the substring "qu" in a word. The regular expression pattern `\Bqu\w+` matches a substring that begins with a "qu" that does not start a word and that continues to the end of the word.

C# Copy

```
using System;
using System.Text.RegularExpressions;

public class Example
{
    public static void Main()
    {
        string input = "equity queen equip acquaint quiet";
        string pattern = @"\Bqu\w+";
        foreach (Match match in Regex.Matches(input, pattern))
            Console.WriteLine("'{0}' found at position {1}",
                              match.Value, match.Index);
    }
```

```
}
// The example displays the following output:
//       'quity' found at position 1
//       'quip' found at position 14
//       'quaint' found at position 21
```

The regular expression pattern is interpreted as shown in the following table.

Pattern	Description
`\B`	Do not begin the match at a word boundary.
`qu`	Match the substring "qu".
`\w+`	Match one or more word characters.

See also

- Regular Expression Language - Quick Reference
- Regular Expression Options

Recommended content

Character Classes in .NET Regular Expressions

Learn how to use character classes to represent a set of characters in .NET regular expressions.

Character Escapes in .NET Regular Expressions

Learn about special characters and escaped characters in .NET regular expressions.

Regular Expression Options

Learn how to use regular expression options in .NET, such as case-insensitive matching, multiline mode, and right-to-left mode.

RegexOptions Enum (System.Text.RegularExpressions)

Provides enumerated values to use to set regular expression options.

Grouping Constructs in Regular Expressions

Learn to use grouping constructs in .NET. Grouping constructs delineate subexpressions of a regular expression and capture substrings of an input string.

Regex.Matches Method (System.Text.RegularExpressions)

Searches an input string for all occurrences of a regular expression and returns all the matches.

MatchCollection Class (System.Text.RegularExpressions)

Represents the set of successful matches found by iteratively applying a regular expression pattern to the input string. The collection is immutable (read-only) and has no public constructor. The Matches(String) method returns a MatchCollection object.

Regex.Match Method (System.Text.RegularExpressions)

Searches an input string for a substring that matches a regular expression pattern and returns the first occurrence as a single Match object.

Show more