

Regex.Matches Method

Namespace: [System.Text.RegularExpressions](#)

Assembly: [System.Text.RegularExpressions.dll](#)

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

In this article


- [Definition](#)
- [Overloads](#)
- [Matches\(String\)](#)
- [Matches\(String, Int32\)](#)
- [Matches\(String, String\)](#)
- [Matches\(String, String, RegexOptions\)](#)
- [Matches\(String, String, RegexOptions, TimeSpan\)](#)
- [Applies to](#)

Overloads

Matches(String)	Searches the specified input string for all occurrences of a regular expression.
Matches(String, Int32)	Searches the specified input string for all occurrences of a regular expression, beginning at the specified starting position in the string.
Matches(String, String)	Searches the specified input string for all occurrences of a specified regular expression.
Matches(String, String, RegexOptions)	Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options.
Matches(String, String, RegexOptions, TimeSpan)	Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options and time-out interval.

Matches(String)

Searches the specified input string for all occurrences of a regular expression.

C#	 Copy
<pre>public System.Text.RegularExpressions.MatchCollection Matches (string input);</pre>	

Parameters

input String

The string to search for a match.

Returns

[MatchCollection](#)

A collection of the [Match](#) objects found by the search. If no matches are found, the method returns an empty collection object.

Exceptions

[ArgumentNullException](#)

input is null.

Examples

The following example uses the [Matches\(String\)](#) method to identify any words in a sentence that end in "es".

C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\"b\\w+es\\b"; Regex rgx = new Regex(pattern);</pre>		

```

string sentence = "Who writes these notes?";

foreach (Match match in rgx.Matches(sentence))
    Console.WriteLine("Found '{0}' at position {1}",
                      match.Value, match.Index);
}
// The example displays the following output:
//      Found 'writes' at position 4
//      Found 'notes' at position 17

```

The regular expression pattern `\b\w+es\b` is defined as shown in the following table.

Pattern	Description
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>es</code>	Match the literal string "es".
<code>\b</code>	End the match at a word boundary.

Remarks

The [Matches\(String\)](#) method is similar to the [Match\(String\)](#) method, except that it returns information about all the matches found in the input string, instead of a single match. It is equivalent to the following code:

C#	 Copy
<pre> Match match = regex.Match(input); while (match.Success) { // Handle match here... match = match.NextMatch(); } </pre>	

The collection includes only matches and terminates at the first non-match.

The regular expression pattern for which the [Matches\(String\)](#) method searches is defined by the call to one of the [Regex](#) class constructors. For more information about the elements that can form a regular expression pattern, see [Regular Expression Language - Quick Reference](#).

The [Matches](#) method uses lazy evaluation to populate the returned [MatchCollection](#) object. Accessing members of this collection such as [MatchCollection.Count](#) and [MatchCollection.CopyTo](#) causes the collection to be populated immediately. To take advantage of lazy evaluation, you should iterate the collection by using a construct such as `foreach` in C# and `For Each...Next` in Visual Basic.


Because of its lazy evaluation, calling the [Matches\(String\)](#) method does not throw a [RegexMatchTimeoutException](#) exception. However, the exception is thrown when an operation is performed on the [MatchCollection](#) object returned by this method, if the [MatchTimeout](#) property is not [Regex.InfiniteMatchTimeout](#) and a matching operation exceeds the time-out interval.

See also

- [Regular Expression Language Elements](#)

Matches(String, Int32)

Searches the specified input string for all occurrences of a regular expression, beginning at the specified starting position in the string.

C#	 Copy
<pre>public System.Text.RegularExpressions.MatchCollection Matches (string input, int startat);</pre>	

Parameters

input [String](#)

The string to search for a match.

startat [Int32](#)

The character position in the input string at which to start the search.

Returns

[MatchCollection](#)

A collection of the [Match](#) objects found by the search. If no matches are found, the method returns an empty collection object.

Exceptions

[ArgumentNullException](#)



input is null.

[ArgumentOutOfRangeException](#)

startat is less than zero or greater than the length of input.

Examples

The following example uses the [Match\(String\)](#) method to find the first word in a sentence that ends in "es", and then calls the [Matches\(String, Int32\)](#) method to identify any additional words that end in "es".

C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\b\w+es\b"; Regex rgx = new Regex(pattern); string sentence = "Who writes these notes and uses our paper?"; // Get the first match. Match match = rgx.Match(sentence); if (match.Success) { Console.WriteLine("Found first 'es' in '{0}' at position {1}", match.Value, match.Index); // Get any additional matches. foreach (Match m in rgx.Matches(sentence, match.Index + match.Length)) Console.WriteLine("Also found '{0}' at position {1}", m.Value, m.Index); } } } // The example displays the following output: // Found first 'es' in 'writes' at position 4</pre>		


```
//      Also found 'notes' at position 17
//      Also found 'uses' at position 27
```

The regular expression pattern `\b\w+es\b` is defined as shown in the following table.

Pattern	Description
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>es</code>	Match the literal string "es".
<code>\b</code>	End the match at a word boundary.

Remarks

The [Matches\(String, Int32\)](#) method is similar to the [Match\(String, Int32\)](#) method, except that it returns information about all the matches found in the input string, instead of a single match. It is equivalent to the following code:

C#	 Copy
<pre>Match match = regex.Match(input, startAt); while (match.Success) { // Handle match here... match = match.NextMatch(); }</pre>	

The regular expression pattern for which the [Matches\(String, Int32\)](#) method searches is defined by the call to one of the [Regex](#) class constructors. For more information about the elements that can form a regular expression pattern, see [Regular Expression Language - Quick Reference](#).

The [Matches](#) method uses lazy evaluation to populate the returned [MatchCollection](#) object. Accessing members of this collection such as [MatchCollection.Count](#) and [MatchCollection.CopyTo](#) causes the collection to be populated immediately. To take advantage of lazy evaluation, you should iterate the collection by using a construct such as `foreach` in C# and `For Each...Next` in Visual Basic.


Because of its lazy evaluation, calling the [Matches\(String, Int32\)](#) method does not throw a [RegexMatchTimeoutException](#) exception. However, the exception is thrown when an operation is performed on the [MatchCollection](#) object returned by this method, if the [MatchTimeout](#) property is not [Regex.InfiniteMatchTimeout](#) and a matching operation exceeds the time-out interval.

See also

- [Regular Expression Language Elements](#)

Matches(String, String)

Searches the specified input string for all occurrences of a specified regular expression.

C#	 Copy
<pre>public static System.Text.RegularExpressions.MatchCollection Matches (string input, string pattern);</pre>	

Parameters

input [String](#)

The string to search for a match.

pattern [String](#)

The regular expression pattern to match.

Returns

[MatchCollection](#)

A collection of the [Match](#) objects found by the search. If no matches are found, the method returns an empty collection object.

Exceptions

[ArgumentException](#)



A regular expression parsing error occurred.

ArgumentNullException

input OR pattern is null.

Examples

The following example uses the [Matches\(String, String\)](#) method to identify any word in a sentence that ends in "es".


C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\b\w+es\b"; string sentence = "Who writes these notes?"; foreach (Match match in Regex.Matches(sentence, pattern)) Console.WriteLine("Found '{0}' at position {1}", match.Value, match.Index); } } // The example displays the following output: // Found 'writes' at position 4 // Found 'notes' at position 17</pre>		

The regular expression pattern `\b\w+es\b` is defined as shown in the following table.

Pattern	Description
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>es</code>	Match the literal string "es".
<code>\b</code>	End the match at a word boundary.

Remarks

The [Matches\(String, String\)](#) method is similar to the [Match\(String, String\)](#) method, except that it returns information about all the matches found in the input string, instead of a single match. It is equivalent to the following code:

C#	 Copy
<pre>Match match = Regex.Match(input, pattern); while (match.Success) { // Handle match here... match = match.NextMatch(); }</pre>	

The static `Matches` methods are equivalent to constructing a [Regex](#) object with the specified regular expression pattern and calling the instance method `Matches`.

The `pattern` parameter consists of regular expression language elements that symbolically describe the string to match. For more information about regular expressions, see [.NET Framework Regular Expressions](#) and [Regular Expression Language - Quick Reference](#).

The [Matches](#) method uses lazy evaluation to populate the returned [MatchCollection](#) object. Accessing members of this collection such as [MatchCollection.Count](#) and [MatchCollection.CopyTo](#) causes the collection to be populated immediately. To take advantage of lazy evaluation, you should iterate the collection by using a construct such as `foreach` in C# and `For Each...Next` in Visual Basic.

Because of its lazy evaluation, calling the [Matches\(String, String\)](#) method does not throw a [RegexMatchTimeoutException](#) exception. However, the exception is thrown when an operation is performed on the [MatchCollection](#) object returned by this method, if a time-out interval is defined by the "REGEX_DEFAULT_MATCH_TIMEOUT" property of the current application domain and a matching operation exceeds this time-out interval.

Notes to Callers

This method times out after an interval that is equal to the default time-out value of the application domain in which it is called. If a time-out value has not been defined for the application domain, the value [InfiniteMatchTimeout](#), which prevents the method from timing out, is used. The recommended static method for retrieving multiple pattern


matches is [Matches\(String, String, RegexOptions, TimeSpan\)](#), which lets you specify the time-out interval.

See also

- [Regular Expression Language Elements](#)

Matches(String, String, RegexOptions)

Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options.

C#	 Copy
<pre>public static System.Text.RegularExpressions.MatchCollection Matches (string input, string pattern, System.Text.RegularExpressions.RegexOptions options);</pre>	

Parameters

input [String](#)

The string to search for a match.

pattern [String](#)

The regular expression pattern to match.

options [RegexOptions](#)

A bitwise combination of the enumeration values that specify options for matching.

Returns

[MatchCollection](#)

A collection of the [Match](#) objects found by the search. If no matches are found, the method returns an empty collection object.

Exceptions

[ArgumentException](#)

A regular expression parsing error occurred.

ArgumentNullException

input OR pattern is null.

ArgumentOutOfRangeException

options is not a valid bitwise combination of [RegexOptions](#) values.

Examples

The following example calls the [Matches\(String, String\)](#) method to identify any word in a sentence that ends in "es", and then calls the [Matches\(String, String, RegexOptions\)](#) method to perform a case-insensitive comparison of the pattern with the input string. As the output shows, the two methods return different results.


C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\b\w+es\b"; string sentence = "NOTES: Any notes or comments are optional."; // Call Matches method without specifying any options. foreach (Match match in Regex.Matches(sentence, pattern)) Console.WriteLine("Found '{0}' at position {1}", match.Value, match.Index); Console.WriteLine(); // Call Matches method for case-insensitive matching. foreach (Match match in Regex.Matches(sentence, pattern, RegexOptions.IgnoreCase)) Console.WriteLine("Found '{0}' at position {1}", match.Value, match.Index); } } // The example displays the following output: // Found 'notes' at position 11 // // Found 'NOTES' at position 0 // Found 'notes' at position 11</pre>		

The regular expression pattern `\b\w+es\b` is defined as shown in the following table.

Pattern	Description
\b	Begin the match at a word boundary.
\w+	Match one or more word characters.
es	Match the literal string "es".
\b	End the match at a word boundary.

Remarks

The [Matches\(String, String, RegexOptions\)](#) method is similar to the [Match\(String, String, RegexOptions\)](#) method, except that it returns information about all the matches found in the input string, instead of a single match. It is equivalent to the following code:

C#	 Copy
<pre>Match match = Regex.Match(input, pattern, options); while (match.Success) { // Handle match here... match = match.NextMatch(); }</pre>	

The static `Matches` methods are equivalent to constructing a [Regex](#) object with the specified regular expression pattern and calling the instance method `Matches`.

The `pattern` parameter consists of regular expression language elements that symbolically describe the string to match. For more information about regular expressions, see [.NET Framework Regular Expressions](#) and [Regular Expression Language - Quick Reference](#).

The [Matches](#) method uses lazy evaluation to populate the returned [MatchCollection](#) object. Accessing members of this collection such as [MatchCollection.Count](#) and [MatchCollection.CopyTo](#) causes the collection to be populated immediately. To take advantage of lazy evaluation, you should iterate the collection by using a construct such as `foreach` in C# and `For Each...Next` in Visual Basic.

Because of its lazy evaluation, calling the [Matches\(String, String\)](#) method does not throw a [RegexMatchTimeoutException](#) exception. However, the exception is thrown

when an operation is performed on the [MatchCollection](#) object returned by this method, if a time-out interval is defined by the "REGEX_DEFAULT_MATCH_TIMEOUT" property of the current application domain and a matching operation exceeds this time-out interval.

Notes to Callers

This method times out after an interval that is equal to the default time-out value of the application domain in which it is called. If a time-out value has not been defined for the application domain, the value [InfiniteMatchTimeout](#), which prevents the method from timing out, is used. The recommended static method for retrieving multiple pattern matches is [Matches\(String, String, RegexOptions, TimeSpan\)](#), which lets you set the time-out interval.

See also

- [Regular Expression Language Elements](#)

Matches(String, String, RegexOptions, TimeSpan)

Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options and time-out interval.

C#	 Copy
<pre>public static System.Text.RegularExpressions.MatchCollection Matches (string input, string pattern, System.Text.RegularExpressions.RegexOptions options, TimeSpan matchTimeout);</pre>	

Parameters

input [String](#)

The string to search for a match.

pattern [String](#)

The regular expression pattern to match.

options [RegexOptions](#)

A bitwise combination of the enumeration values that specify options for matching.

matchTimeout [TimeSpan](#)

A time-out interval, or [InfiniteMatchTimeout](#) to indicate that the method should not time out.

Returns

[MatchCollection](#)

A collection of the [Match](#) objects found by the search. If no matches are found, the method returns an empty collection object.

Exceptions

[ArgumentException](#)

A regular expression parsing error occurred.

[ArgumentNullException](#)

input OR pattern is null.

[ArgumentOutOfRangeException](#)


options is not a valid bitwise combination of [RegexOptions](#) values.

-or-

matchTimeout is negative, zero, or greater than approximately 24 days.

Examples

The following example calls the [Matches\(String, String, RegexOptions, TimeSpan\)](#) method to perform a case-sensitive comparison that matches any word in a sentence that ends in "es". It then calls the [Matches\(String, String, RegexOptions, TimeSpan\)](#) method to perform a case-insensitive comparison of the pattern with the input string. In both cases, the time-out interval is set to one second. As the output shows, the two methods return different results.

C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions;</pre>		

```

public class Example
{
    public static void Main()
    {
        string pattern = @"\b\w+es\b";
        string sentence = "NOTES: Any notes or comments are optional.";

        // Call Matches method without specifying any options.
        try {
            foreach (Match match in Regex.Matches(sentence, pattern,
                                                    RegexOptions.None,
                                                    TimeSpan.FromSeconds(1)))
                Console.WriteLine("Found '{0}' at position {1}",
                                   match.Value, match.Index);
        }
        catch (RegexMatchTimeoutException) {
            // Do Nothing: Assume that timeout represents no match.
        }
        Console.WriteLine();

        // Call Matches method for case-insensitive matching.
        try {
            foreach (Match match in Regex.Matches(sentence, pattern,
                                                    RegexOptions.IgnoreCase))
                Console.WriteLine("Found '{0}' at position {1}",
                                   match.Value, match.Index);
        }
        catch (RegexMatchTimeoutException) {}
    }
}

// The example displays the following output:
//      Found 'notes' at position 11
//
//      Found 'NOTES' at position 0
//      Found 'notes' at position 11


```

The regular expression pattern `\b\w+es\b` is defined as shown in the following table.

Pattern	Description
<code>\b</code>	Begin the match at a word boundary.
<code>\w+</code>	Match one or more word characters.
<code>es</code>	Match the literal string "es".
<code>\b</code>	End the match at a word boundary.

Remarks

The [Matches\(String, String, RegexOptions, TimeSpan\)](#) method is similar to the [Match\(String, String, RegexOptions, TimeSpan\)](#) method, except that it returns information about all the matches found in the input string, instead of a single match. It is equivalent to the following code:

C#	 Copy
<pre>try { Match match = Regex.Match(input, pattern, options, TimeSpan.FromSeconds(1)); while (match.Success) { // Handle match here... match = match.NextMatch(); } } catch (RegexMatchTimeoutException) { // Do nothing: assume that exception represents no match. }</pre>	

The static `Matches` methods are equivalent to constructing a [Regex](#) object with the specified regular expression pattern and calling the instance method `Matches`.

The `pattern` parameter consists of regular expression language elements that symbolically describe the string to match. For more information about regular expressions, see [.NET Framework Regular Expressions](#) and [Regular Expression Language - Quick Reference](#).

The [Matches](#) method uses lazy evaluation to populate the returned [MatchCollection](#) object. Accessing members of this collection such as [MatchCollection.Count](#) and [MatchCollection.CopyTo](#) causes the collection to be populated immediately. To take advantage of lazy evaluation, you should iterate the collection by using a construct such as `foreach` in C# and `For Each...Next` in Visual Basic.

Because of its lazy evaluation, calling the [Matches](#) method does not throw a [RegexMatchTimeoutException](#) exception. However, an exception is thrown when an operation is performed on the [MatchCollection](#) object returned by this method, if a matching operation exceeds this time-out interval specified by the `matchTimeout` parameter.

Notes to Callers

We recommend that you set the `matchTimeout` parameter to an appropriate value, such as two seconds. If you disable time-outs by specifying [InfiniteMatchTimeout](#), the regular expression engine offers slightly better performance. However, you should disable time-outs only under the following conditions:

- When the input processed by a regular expression is derived from a known and trusted source or consists of static text. This excludes text that has been dynamically input by users.
- When the regular expression pattern has been thoroughly tested to ensure that it efficiently handles matches, non-matches, and near matches.
- When the regular expression pattern contains no language elements that are known to cause excessive backtracking when processing a near match.

See also

- [Regular Expression Language Elements](#)

Applies to

.NET

5 Preview 1

.NET Core

3.1, 3.0, 2.2, 2.1, 2.0, 1.1, 1.0

.NET Framework

4.8, 4.7.2, 4.7.1, 4.7, 4.6.2, 4.6.1, 4.6, 4.5.2, 4.5.1, 4.5, 4.0, 3.5, 3.0, 2.0, 1.1

.NET Standard

2.1, 2.0, 1.6, 1.4, 1.3, 1.2, 1.1, 1.0

UWP

10.0

Xamarin.Android

7.1

Xamarin.iOS

10.8

Xamarin.Mac

3.0

Is this page helpful?

 Yes  No
