

Regex Constructors

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

Assembly: System.Text.RegularExpressions.dll

Initializes a new instance of the [Regex](#) class.

In this article


- Definition
- Overloads
- [Regex\(\)](#)
- [Regex\(String\)](#)
- [Regex\(SerializationInfo, StreamingContext\)](#)
- [Regex\(String, RegexOptions\)](#)
- [Regex\(String, RegexOptions, TimeSpan\)](#)
- Applies to

Overloads

Regex()	Initializes a new instance of the Regex class.
Regex(String)	Initializes a new instance of the Regex class for the specified regular expression.
Regex(SerializationInfo, StreamingContext)	Initializes a new instance of the Regex class by using serialized data.
Regex(String, RegexOptions)	Initializes a new instance of the Regex class for the specified regular expression, with options that modify the pattern.
Regex(String, RegexOptions, TimeSpan)	Initializes a new instance of the Regex class for the specified regular expression, with options that modify the pattern and a value that specifies how long a pattern matching method should attempt a match before it times out.

Regex()

Initializes a new instance of the [Regex](#) class.


C#	 Copy
<pre>protected Regex ();</pre>	

Remarks

Note that this constructor is protected; it can only be called by classes derived from the [Regex](#) class.

Regex(String)

Initializes a new instance of the [Regex](#) class for the specified regular expression.

C#	 Copy
<pre>public Regex (string pattern);</pre>	

Parameters

pattern [String](#)

The regular expression pattern to match.

Exceptions

[ArgumentException](#)



A regular expression parsing error occurred.

[ArgumentNullException](#)

pattern is null.

Examples

The following example illustrates how to use this constructor to instantiate a regular expression that matches any word that begins with the letters "a" or "t".

C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\b[at]\w+"; string text = "The threaded application ate up the thread pool as it executed."; MatchCollection matches; Regex defaultRegex = new Regex(pattern); // Get matches of pattern in text matches = defaultRegex.Matches(text); Console.WriteLine("Parsing '{0}'", text); // Iterate matches for (int ctr = 0; ctr < matches.Count; ctr++) Console.WriteLine("{0}. {1}", ctr, matches[ctr].Value); } } // The example displays the following output: // Parsing 'The threaded application ate up the thread pool as it // executed.' // 0. threaded // 1. application // 2. ate // 3. the // 4. thread // 5. as</pre>		

Note that the regular expression pattern cannot match the word "The" at the beginning of the text, because comparisons are case-sensitive by default. For an example of case-insensitive comparison, see the [Regex\(String, RegexOptions\)](#) constructor.

Remarks

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

Calling the [Regex\(String\)](#) constructor is equivalent to calling the [Regex\(String, RegexOptions\)](#) constructor with a value of [None](#) for the options argument.

A [Regex](#) object is immutable, which means that it can be used only for the match pattern you define when you create it. However, it can be used any number of times without being recompiled.

This constructor instantiates a regular expression object that attempts a case-sensitive match of any alphabetical characters defined in pattern. For a case-insensitive match, use the [Regex.Regex\(String, RegexOptions\)](#) constructor.

Notes to Callers


This constructor creates a [Regex](#) object that uses the default time-out value of the application domain in which it is created. If a time-out value has not been defined for the application domain, the [Regex](#) object uses the value [InfiniteMatchTimeout](#), which prevents the operation from timing out. The recommended constructor for creating a [Regex](#) object is [Regex\(String, RegexOptions, TimeSpan\)](#), which lets you set the time-out interval.

See also

- [Regular Expression Language Elements](#)

Regex(SerializationInfo, StreamingContext)

Initializes a new instance of the [Regex](#) class by using serialized data.

C#	 Copy
<pre>protected Regex (System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serialization.StreamingContext context);</pre>	

Parameters

info [SerializationInfo](#)

The object that contains a serialized pattern and [RegexOptions](#) information.

context [StreamingContext](#)

The destination for this serialization. (This parameter is not used; specify `null`.)

Exceptions

[ArgumentException](#)

A regular expression parsing error occurred.

[ArgumentNullException](#)


The pattern that `info` contains is `null`.

[ArgumentOutOfRangeException](#)

`info` contains an invalid [RegexOptions](#) flag.

Regex(String, RegexOptions)

Initializes a new instance of the [Regex](#) class for the specified regular expression, with options that modify the pattern.

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

Parameters

pattern [String](#)

The regular expression pattern to match.

options [RegexOptions](#)

A bitwise combination of the enumeration values that modify the regular expression.

Exceptions

[ArgumentException](#)

A regular expression parsing error occurred.

[ArgumentNullException](#)



pattern is null.

ArgumentOutOfRangeException

options contains an invalid flag.

Examples

The following example illustrates how to use this constructor to instantiate a regular expression that matches any word that begins with the letters "a" or "t".

C#	 Copy	 Run
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = @"\b[at]\w+"; RegexOptions options = RegexOptions.IgnoreCase RegexOptions.Compiled; string text = "The threaded application ate up the thread pool as it executed."; MatchCollection matches; Regex optionRegex = new Regex(pattern, options); Console.WriteLine("Parsing '{0}' with options {1}:", text, options.ToString()); // Get matches of pattern in text matches = optionRegex.Matches(text); // Iterate matches for (int ctr = 0; ctr < matches.Count; ctr++) Console.WriteLine("{0}. {1}", ctr, matches[ctr].Value); } } // The example displays the following output: // Parsing 'The threaded application ate up the thread pool as it executed.' // with options IgnoreCase, Compiled: // 0. The // 1. threaded // 2. application // 3. ate // 4. the // 5. thread // 6. as</pre>		

Note that the match collection includes the word "The" that begins the text because the `options` parameter has defined case-insensitive comparisons.

Remarks

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

A [Regex](#) object is immutable, which means that it can be used only for the match parameters you define when you create it. However, it can be used any number of times without being recompiled.

Notes to Callers


This constructor creates a [Regex](#) object that uses the default time-out value of the application domain in which it is created. If a time-out value has not been defined for the application domain, the [Regex](#) object uses the value [InfiniteMatchTimeout](#), which prevents the operation from timing out. The recommended constructor for creating a [Regex](#) object is [Regex\(String, RegexOptions, TimeSpan\)](#), which lets you set the time-out interval.

See also

- [Regular Expression Language Elements](#)

Regex(String, RegexOptions, TimeSpan)

Initializes a new instance of the [Regex](#) class for the specified regular expression, with options that modify the pattern and a value that specifies how long a pattern matching method should attempt a match before it times out.

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

Parameters

pattern [String](#)

The regular expression pattern to match.

options [RegexOptions](#)

A bitwise combination of the enumeration values that modify the regular expression.

matchTimeout [TimeSpan](#)

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

Exceptions

[ArgumentException](#)

A regular expression parsing error occurred.

[ArgumentNullException](#)

pattern is null.

[ArgumentOutOfRangeException](#)



options is not a valid [RegexOptions](#) value.

-or-

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

Examples

The following example calls the [Regex\(String, RegexOptions, TimeSpan\)](#) constructor to instantiate a [Regex](#) object with a time-out value of one second. The regular expression pattern `(a+)+$`, which matches one or more sequences of one or more "a" characters at the end of a line, is subject to excessive backtracking. If a [RegexMatchTimeoutException](#) is thrown, the example increases the time-out value up to the maximum value of three seconds. Otherwise, it abandons the attempt to match the pattern.

C#	 Copy	 Run
<pre>using System; using System.ComponentModel; using System.Diagnostics; using System.Security;</pre>		


```

using System.Text.RegularExpressions;
using System.Threading;

public class Example
{
    const int MaxTimeoutInSeconds = 3;

    public static void Main()
    {
        string pattern = @"(a+)+$";    // DO NOT REUSE THIS PATTERN.
        Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase,
        TimeSpan.FromSeconds(1));
        Stopwatch sw = null;

        string[] inputs= { "aa", "aaaa>",
                           "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
                           "aaaaaaaaaaaaaaaaaaaaaaaa>",
                           "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>" };

        foreach (var inputValue in inputs) {
            Console.WriteLine("Processing {0}", inputValue);
            bool timedOut = false;
            do {
                try {
                    sw = Stopwatch.StartNew();
                    // Display the result.
                    if (rgx.IsMatch(inputValue)) {
                        sw.Stop();
                        Console.WriteLine(@"Valid: '{0}' ({1:ss\.ffffff}
seconds)",
                                     inputValue, sw.Elapsed);
                    }
                    else {
                        sw.Stop();
                        Console.WriteLine(@"' '{0}' is not a valid string.
({1:ss\.ffffff} seconds)",
                                     inputValue, sw.Elapsed);
                    }
                }
                catch (RegexMatchTimeoutException e) {
                    sw.Stop();
                    // Display the elapsed time until the exception.
                    Console.WriteLine(@"Timeout with '{0}' after {1:ss\.ffffff}",
                                     inputValue, sw.Elapsed);
                    Thread.Sleep(1500);    // Pause for 1.5 seconds.

                    // Increase the timeout interval and retry.
                    TimeSpan timeout =
e.MatchTimeout.Add(TimeSpan.FromSeconds(1));
                    if (timeout.TotalSeconds > MaxTimeoutInSeconds) {
                        Console.WriteLine("Maximum timeout interval of {0} seconds

```

```

exceeded.",
                                MaxTimeoutInSeconds);
        timedOut = false;
    }
    else {
        Console.WriteLine("Changing the timeout interval to {0}",
                            timeout);
        rgx = new Regex(pattern, RegexOptions.IgnoreCase,
timeout);
        timedOut = true;
    }
    }
    } while (timedOut);
    Console.WriteLine();
}
}
}

// The example displays output like the following :
//   Processing aa
//   Valid: 'aa' (00.0000779 seconds)
//
//   Processing aaaa>
//   'aaaa>' is not a valid string. (00.00005 seconds)
//
//   Processing aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
//   Valid: 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' (00.0000043
seconds)
//
//   Processing aaaaaaaaaaaaaaaaaaaaaa>
//   Timeout with 'aaaaaaaaaaaaaaaaaaaaa>' after 01.00469
//   Changing the timeout interval to 00:00:02
//   Timeout with 'aaaaaaaaaaaaaaaaaaaaa>' after 02.01202
//   Changing the timeout interval to 00:00:03
//   Timeout with 'aaaaaaaaaaaaaaaaaaaaa>' after 03.01043
//   Maximum timeout interval of 3 seconds exceeded.
//
//   Processing aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
//   Timeout with 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa>' after
03.01018
//   Maximum timeout interval of 3 seconds exceeded.

```

Remarks

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

A [Regex](#) object is immutable, which means that it can be used only for the match pattern that you define when you create it. However, it can be used any number of times without being recompiled.

The `matchTimeout` parameter specifies how long a pattern-matching method should try to find a match before it times out. If no match is found in that time interval, the pattern-matching method throws a [RegexMatchTimeoutException](#) exception.

`matchTimeout` overrides any default time-out value defined for the application domain in which the [Regex](#) object is created. The instance pattern-matching methods that observe the `matchTimeout` time-out interval include the following:

- [IsMatch](#)
- [Match](#)
- [Matches](#)
- [Replace](#)
- [Split](#)
- [Match.NextMatch](#)

Setting a time-out interval prevents regular expressions that rely on excessive backtracking from appearing to stop responding when they process input that contains near matches. For more information, see [Best Practices for Regular Expressions](#) and [Backtracking](#). To set a reasonable time-out interval, consider the following factors:

- The length and complexity of the regular expression pattern. Longer and more complex regular expressions require more time than shorter and simpler ones.
- The expected machine load. Processing takes more time on systems that have high CPU and memory utilization.

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

- [Backtracking in Regular Expressions](#)
- [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
