Regex Class

Namespace: System.Text.RegularExpressions

Assembly: System.Text.RegularExpressions.dll

Represents an immutable regular expression.

In this article

C#

Copy

```
public class Regex : System.Runtime.Serialization.ISerializable
```

Inheritance Object → Regex

Derived System.Web.RegularExpressions.AspCodeRegex
System.Web.RegularExpressions.AspEncodedExprRegex
System.Web.RegularExpressions.AspExprRegex
System.Web.RegularExpressions.CommentRegex
System.Web.RegularExpressions.DatabindExprRegex
System.Web.RegularExpressions.DataBindRegex
System.Web.RegularExpressions.DirectiveRegex
System.Web.RegularExpressions.EndTagRegex
System.Web.RegularExpressions.GTRegex
System.Web.RegularExpressions.IncludeRegex
System.Web.RegularExpressions.LTRegex

System.Web.RegularExpressions.RunatServerRegex
System.Web.RegularExpressions.ServerTagsRegex
System.Web.RegularExpressions.SimpleDirectiveRegex
System.Web.RegularExpressions.TagRegex
System.Web.RegularExpressions.TagRegex35
System.Web.RegularExpressions.TextRegex

Implements ISerializable

Examples

The following example uses a regular expression to check for repeated occurrences of words in a string. The regular expression `\b(?<word>\w+)\s+(\k<word>)\b` can be interpreted as shown in the following table.

Pattern	Description
`\b`	Start the match at a word boundary.
`(?<word>\w+)`	Match one or more word characters up to a word boundary. Name this captured group `word`.
`\s+`	Match one or more white-space characters.
`(\k<word>)`	Match the captured group that is named `word`.
`\b`	Match a word boundary.

C# Copy Run

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

public class Test
{

    public static void Main ()
    {

        // Define a regular expression for repeated words.
        Regex rx = new Regex(@"\b(?<word>\w+)\s+(\k<word>)\b",
          RegexOptions.Compiled | RegexOptions.IgnoreCase);
        // Define a test string.
        string text = "The the quick brown fox  fox jumps over the lazy dog
dog.";

        // Find matches.
        MatchCollection matches = rx.Matches(text);

        // Report the number of matches found.
        Console.WriteLine("{0} matches found in:\n   {1}",
                          matches.Count,
                          text);

        // Report on each match.
        foreach (Match match in matches)
        {
            GroupCollection groups = match.Groups;
            Console.WriteLine("'{0}' repeated at positions {1} and {2}",
                              groups["word"].Value,
                              groups[0].Index,
                              groups[1].Index);
        }
    }
}
// The example produces the following output to the console:
//       3 matches found in:
//          The the quick brown fox  fox jumps over the lazy dog dog.
//       'The' repeated at positions 0 and 4
//       'fox' repeated at positions 20 and 25
//       'dog' repeated at positions 50 and 54
```

The following example illustrates the use of a regular expression to check whether a string either represents a currency value or has the correct format to represent a currency value. In this case, the regular expression is built dynamically from the NumberFormatInfo.CurrencyDecimalSeparator, CurrencyDecimalDigits, NumberFormatInfo.CurrencySymbol, NumberFormatInfo.NegativeSign, and NumberFormatInfo.PositiveSign properties for the user's current culture. If the system's current culture is en-US, the resulting regular expression is `^\s*[\+-]?\s?\$?\s?(\d*\.?\d{2}?){1}$`. This regular expression can be interpreted as shown in the following table.

Pattern	Description
`^`	Start at the beginning of the string.
`\s*`	Match zero or more white-space characters.
`[\+-]?`	Match zero or one occurrence of either the positive sign or the negative sign.

Pattern	Description
`\s?`	Match zero or one white-space character.
`\$?`	Match zero or one occurrence of the dollar sign.
`\s?`	Match zero or one white-space character.
`\d*`	Match zero or more decimal digits.
`\.?`	Match zero or one decimal point symbol.
`\d{2}?`	Match two decimal digits zero or one time.
`(\d*\.?\d{2}?){1}`	Match the pattern of integral and fractional digits separated by a decimal point symbol at least one time.
`$`	Match the end of the string.

In this case, the regular expression assumes that a valid currency string does not contain group separator symbols, and that it has either no fractional digits or the number of fractional digits defined by the current culture's CurrencyDecimalDigits property.

C# Copy Run

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

public class Example
{
   public static void Main()
   {
      // Get the current NumberFormatInfo object to build the regular
      // expression pattern dynamically.
      NumberFormatInfo nfi = NumberFormatInfo.CurrentInfo;

      // Define the regular expression pattern.
      string pattern;
      pattern = @"^\s*[";
      // Get the positive and negative sign symbols.
      pattern += Regex.Escape(nfi.PositiveSign + nfi.NegativeSign) + @"]?\s?";
      // Get the currency symbol.
      pattern += Regex.Escape(nfi.CurrencySymbol) + @"?\s?";
      // Add integral digits to the pattern.
      pattern += @"(\d*";
      // Add the decimal separator.
      pattern += Regex.Escape(nfi.CurrencyDecimalSeparator) + "?";
      // Add the fractional digits.
      pattern += @"\d{";
      // Determine the number of fractional digits in currency values.
      pattern += nfi.CurrencyDecimalDigits.ToString() + "}?){1}$";

      Regex rgx = new Regex(pattern);

      // Define some test strings.
      string[] tests = { "-42", "19.99", "0.001", "100 USD",
                         ".34", "0.34", "1,052.21", "$10.62",
                         "+1.43", "-$0.23" };

      // Check each test string against the regular expression.
      foreach (string test in tests)
      {
         if (rgx.IsMatch(test))
            Console.WriteLine("{0} is a currency value.", test);
         else
            Console.WriteLine("{0} is not a currency value.", test);
      }
   }
}
// The example displays the following output:
//       -42 is a currency value.
//       19.99 is a currency value.
//       0.001 is not a currency value.
//       100 USD is not a currency value.
//       .34 is a currency value.
//       0.34 is a currency value.
//       1,052.21 is not a currency value.
//       $10.62 is a currency value.
//       +1.43 is a currency value.
//       -$0.23 is a currency value.
```

Because the regular expression in this example is built dynamically, we do not know at design time whether the current culture's currency symbol, decimal sign, or positive and negative signs might be misinterpreted by the regular expression engine as regular expression language operators. To prevent any misinterpretation, the example passes each dynamically generated string to the Escape method.

Remarks

The Regex class represents the .NET Framework's regular expression engine. It can be used to quickly parse large amounts of text to find specific character patterns; to extract, edit, replace, or delete text substrings; and to add the extracted strings to a collection to generate a report.

> ⓘ **Note**
>
> If your primary interest is to validate a string by determining whether it conforms to a particular pattern, you can use the System.Configuration.RegexStringValidator class.

To use regular expressions, you define the pattern that you want to identify in a text stream by using the syntax documented in Regular Expression Language - Quick Reference. Next, you can optionally instantiate a Regex object. Finally, you call a method that performs some operation, such as replacing text that matches the regular expression pattern, or identifying a pattern match.

> ⓘ **Note**
>
> For some common regular expression patterns, see Regular Expression Examples. There are also a number of online libraries of regular expression patterns, such as the one at Regular-Expressions.info.

For more information about using the Regex class, see the following sections in this topic:

- Regex vs. String Methods
- Static vs. Instance Methods
- Performing Regular Expression Operations
- Defining a Time-Out Value

For more information about the regular expression language, see Regular Expression Language - Quick Reference or download and print one of these brochures:

Quick Reference in Word (.docx) format
Quick Reference in PDF (.pdf) format

Regex vs. String Methods

The System.String class includes several search and comparison methods that you can use to perform pattern matching with text. For example, the String.Contains, String.EndsWith, and String.StartsWith methods determine whether a string instance contains a specified substring; and the String.IndexOf, String.IndexOfAny, String.LastIndexOf, and String.LastIndexOfAny methods return the starting position of a specified substring in a string. Use the methods of the System.String class when you are searching for a specific string. Use the Regex class when you are searching for a specific pattern in a string. For more information and examples, see .NET Framework Regular Expressions.

Back to Remarks

Static vs. Instance Methods

After you define a regular expression pattern, you can provide it to the regular expression engine in either of two ways:

- By instantiating a Regex object that represents the regular expression. To do this, you pass the regular expression pattern to a Regex constructor. A Regex object is immutable; when you instantiate a Regex object with a regular expression, that object's regular expression cannot be changed.

- By supplying both the regular expression and the text to search to a `static` (`Shared` in Visual Basic) Regex method. This enables you to use a regular expression without explicitly creating a Regex object.

All Regex pattern identification methods include both static and instance overloads.

The regular expression engine must compile a particular pattern before the pattern can be used. Because Regex objects are immutable, this is a one-time procedure that occurs when a Regex class constructor or a static method is called. To eliminate the need to repeatedly compile a single regular expression, the regular expression engine caches the compiled regular expressions used in static method calls. As a result, regular expression pattern-matching methods offer comparable performance for static and instance methods.

> **Important**
>
> In the .NET Framework versions 1.0 and 1.1, all compiled regular expressions, whether they were used in instance or static method calls, were cached. Starting with the .NET Framework 2.0, only regular expressions used in static method calls are cached.

However, caching can adversely affect performance in the following two cases:

- When you use static method calls with a large number of regular expressions. By default, the regular expression engine caches the 15 most recently used static regular expressions. If your application uses more than 15 static regular expressions, some regular expressions must be recompiled. To prevent this recompilation, you can increase the Regex.CacheSize property.
- When you instantiate new Regex objects with regular expressions that have previously been compiled. For example, the following code defines a regular expression to locate duplicated words in a text stream. Although the example uses a single regular expression, it instantiates a new Regex object to process each line of text. This results in the recompilation of the regular expression with each iteration of the loop.

C# Copy

```csharp
StreamReader sr = new StreamReader(filename);
string input;
string pattern = @"\b(\w+)\s\1\b";
while (sr.Peek() >= 0)
{
   input = sr.ReadLine();
   Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);
   MatchCollection matches = rgx.Matches(input);
   if (matches.Count > 0)
   {
      Console.WriteLine("{0} ({1} matches):", input, matches.Count);
      foreach (Match match in matches)
         Console.WriteLine("   " + match.Value);
   }
}
sr.Close();
```

To prevent recompilation, you should instantiate a single Regex object that is accessible to all code that requires it, as shown in the following rewritten example.

C# Copy

```csharp
StreamReader sr = new StreamReader(filename);
string input;
string pattern = @"\b(\w+)\s\1\b";
Regex rgx = new Regex(pattern, RegexOptions.IgnoreCase);

while (sr.Peek() >= 0)
{
   input = sr.ReadLine();
   MatchCollection matches = rgx.Matches(input);
```

```
      if (matches.Count > 0)
      {
         Console.WriteLine("{0} ({1} matches):", input, matches.Count);
         foreach (Match match in matches)
            Console.WriteLine("   " + match.Value);
      }
   }
   sr.Close();
```

Back to Remarks

Performing Regular Expression Operations

Whether you decide to instantiate a Regex object and call its methods or call static methods, the Regex class offers the following pattern-matching functionality:

- Validation of a match. You call the IsMatch method to determine whether a match is present.
- Retrieval of a single match. You call the Match method to retrieve a Match object that represents the first match in a string or in part of a string. Subsequent matches can be retrieved by calling the Match.NextMatch method.
- Retrieval of all matches. You call the Matches method to retrieve a System.Text.RegularExpressions.MatchCollection object that represents all the matches found in a string or in part of a string.
- Replacement of matched text. You call the Replace method to replace matched text. The replacement text can also be defined by a regular expression. In addition, some of the Replace methods include a MatchEvaluator parameter that enables you to programmatically define the replacement text.
- Creation of a string array that is formed from parts of an input string. You call the Split method to split an input string at positions that are defined by the regular expression.

In addition to its pattern-matching methods, the Regex class includes several special-purpose methods:

- The Escape method escapes any characters that may be interpreted as regular expression operators in a regular expression or input string.
- The Unescape method removes these escape characters.

- The CompileToAssembly method creates an assembly that contains predefined regular expressions. The .NET Framework contains examples of these special-purpose assemblies in the System.Web.RegularExpressions namespace.

Back to Remarks

Defining a Time-Out Value

The .NET Framework supports a full-featured regular expression language that provides substantial power and flexibility in pattern matching. However, the power and flexibility come at a cost: the risk of poor performance. Regular expressions that perform poorly are surprisingly easy to create. In some cases, regular expression operations that rely on excessive backtracking can appear to stop responding when they process text that nearly matches the regular expression pattern. For more information about the .NET Framework regular expression engine, see Details of Regular Expression Behavior. For more information about excessive backtracking, see Backtracking.

Starting with the .NET Framework 4.5, you can define a time-out interval for regular expression matches. If the regular expression engine cannot identify a match within this time interval, the matching operation throws a RegexMatchTimeoutException exception. In most cases, this prevents the regular expression engine from wasting processing power by trying to match text that nearly matches the regular expression pattern. It also could indicate, however, that the timeout interval has been set too low, or that the current machine load has caused an overall degradation in performance.

How you handle the exception depends on the cause of the exception. If the exception occurs because the time-out interval is set too low or because of excessive machine load, you can increase the time-out interval and retry the matching operation. If the exception occurs because the regular expression relies on excessive backtracking, you can assume that a match does not exist, and, optionally, you can log information that will help you modify the regular expression pattern.

You can set a time-out interval by calling the Regex(String, RegexOptions, TimeSpan) constructor when you instantiate a regular expression object. For static methods, you can set a time-out interval by calling an overload of a matching method that has a `matchTimeout` parameter. If you do not set a time-out value explicitly, the default time-out value is determined as follows:

- By using the application-wide time-out value, if one exists. This can be any time-out value that applies to the application domain in which the Regex object is instantiated or the static method call is made. You can set the application-wide time-out value by calling the AppDomain.SetData method to assign the string representation of a TimeSpan value to the "REGEX_DEFAULT_MATCH_TIMEOUT" property.

- By using the value InfiniteMatchTimeout, if no application-wide time-out value has been set.

> ⓘ **Important**
>
> We recommend that you set a time-out value in all regular expression pattern-matching operations. For more information, see Best Practices for Regular Expressions.

Back to Remarks

Constructors

Regex()	Initializes a new instance of the Regex class.
Regex(SerializationInfo, StreamingContext)	Initializes a new instance of the Regex class by using serialized data.
Regex(String)	Initializes a new instance of the Regex class for the specified regular expression.
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.

Fields

capnames	Used by a Regex object generated by the CompileToAssembly method.
caps	Used by a Regex object generated by the CompileToAssembly method.

capsize	Used by a Regex object generated by the CompileToAssembly method.
capslist	Used by a Regex object generated by the CompileToAssembly method.
factory	Used by a Regex object generated by the CompileToAssembly method.
InfiniteMatchTimeout	Specifies that a pattern-matching operation should not time out.
internalMatchTimeout	The maximum amount of time that can elapse in a pattern-matching operation before the operation times out.
pattern	Used by a Regex object generated by the CompileToAssembly method.
roptions	Used by a Regex object generated by the CompileToAssembly method.

Properties

CacheSize	Gets or sets the maximum number of entries in the current static cache of compiled regular expressions.
CapNames	Gets or sets a dictionary that maps named capturing groups to their index values.
Caps	Gets or sets a dictionary that maps numbered capturing groups to their index values.
MatchTimeout	Gets the time-out interval of the current instance.
Options	Gets the options that were passed into the Regex constructor.
RightToLeft	Gets a value that indicates whether the regular expression searches from right to left.

Methods

CompileToAssembly(RegexCompilationInfo[], AssemblyName)	Compiles one or more specified Regex objects to a named assembly.
CompileToAssembly(RegexCompilationInfo[], AssemblyName, CustomAttributeBuilder[])	Compiles one or more specified Regex objects to a named assembly with the specified attributes.
CompileToAssembly(RegexCompilationInfo[], AssemblyName, CustomAttributeBuilder[], String)	Compiles one or more specified Regex objects and a specified resource file to a named assembly with the specified attributes.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object)
Escape(String)	Escapes a minimal set of characters (\, *, +, ?, |, {, [, (,), ^, $, ., #, and white space) by replacing them with their escape codes. This instructs the regular expression engine to interpret these characters literally rather than as metacharacters.
GetGroupNames()	Returns an array of capturing group names for the regular expression.
GetGroupNumbers()	Returns an array of capturing group numbers that correspond to group names in an array.
GetHashCode()	Serves as the default hash function. (Inherited from Object)
GetType()	Gets the Type of the current instance. (Inherited from Object)
GroupNameFromNumber(Int32)	Gets the group name that corresponds to the specified group number.
GroupNumberFromName(String)	Returns the group number that corresponds to the specified group name.
InitializeReferences()	Used by a Regex object generated by the CompileToAssembly method.

IsMatch(String)	Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.
IsMatch(String, Int32)	Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string.
IsMatch(String, String)	Indicates whether the specified regular expression finds a match in the specified input string.
IsMatch(String, String, RegexOptions)	Indicates whether the specified regular expression finds a match in the specified input string, using the specified matching options.
IsMatch(String, String, RegexOptions, TimeSpan)	Indicates whether the specified regular expression finds a match in the specified input string, using the specified matching options and time-out interval.
Match(String)	Searches the specified input string for the first occurrence of the regular expression specified in the Regex constructor.
Match(String, Int32)	Searches the input string for the first occurrence of a regular expression, beginning at the specified starting position in the string.
Match(String, Int32, Int32)	Searches the input string for the first occurrence of a regular expression, beginning at the specified starting position and searching only the specified number of characters.
Match(String, String)	Searches the specified input string for the first occurrence of the specified regular expression.
Match(String, String, RegexOptions)	Searches the input string for the first occurrence of the specified regular expression, using the specified matching options.
Match(String, String, RegexOptions, TimeSpan)	Searches the input string for the first occurrence of the 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.
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, Regex Options)	Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options.
Matches(String, String, Regex Options, TimeSpan)	Searches the specified input string for all occurrences of a specified regular expression, using the specified matching options and time-out interval.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
Replace(String, MatchEvaluator)	In a specified input string, replaces all strings that match a specified regular expression with a string returned by a MatchEvaluator delegate.
Replace(String, MatchEvaluator, Int32)	In a specified input string, replaces a specified maximum number of strings that match a regular expression pattern with a string returned by a MatchEvaluator delegate.
Replace(String, MatchEvaluator, Int32, Int32)	In a specified input substring, replaces a specified maximum number of strings that match a regular expression pattern with a string returned by a MatchEvaluator delegate.
Replace(String, String)	In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.
Replace(String, String, Int32)	In a specified input string, replaces a specified maximum number of strings that match a regular expression pattern with a specified replacement string.
Replace(String, String, Int32, Int32)	In a specified input substring, replaces a specified maximum number of strings that match a regular expression pattern with a specified replacement string.
Replace(String, String, Match Evaluator)	In a specified input string, replaces all strings that match a specified regular expression with a string returned by a MatchEvaluator delegate.
Replace(String, String, Match	In a specified input string, replaces all strings that match a

Evaluator, RegexOptions)	specified regular expression with a string returned by a MatchEvaluator delegate. Specified options modify the matching operation.
Replace(String, String, MatchEvaluator, RegexOptions, TimeSpan)	In a specified input string, replaces all substrings that match a specified regular expression with a string returned by a MatchEvaluator delegate. Additional parameters specify options that modify the matching operation and a time-out interval if no match is found.
Replace(String, String, String)	In a specified input string, replaces all strings that match a specified regular expression with a specified replacement string.
Replace(String, String, String, RegexOptions)	In a specified input string, replaces all strings that match a specified regular expression with a specified replacement string. Specified options modify the matching operation.
Replace(String, String, String, RegexOptions, TimeSpan)	In a specified input string, replaces all strings that match a specified regular expression with a specified replacement string. Additional parameters specify options that modify the matching operation and a time-out interval if no match is found.
Split(String)	Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified in the Regex constructor.
Split(String, Int32)	Splits an input string a specified maximum number of times into an array of substrings, at the positions defined by a regular expression specified in the Regex constructor.
Split(String, Int32, Int32)	Splits an input string a specified maximum number of times into an array of substrings, at the positions defined by a regular expression specified in the Regex constructor. The search for the regular expression pattern starts at a specified character position in the input string.
Split(String, String)	Splits an input string into an array of substrings at the positions defined by a regular expression pattern.
Split(String, String, RegexOptions)	Splits an input string into an array of substrings at the positions defined by a specified regular expression pattern. Specified options modify the matching operation.
Split(String, String, Regex	Splits an input string into an array of substrings at the positions

Options, TimeSpan)	defined by a specified regular expression pattern. Additional parameters specify options that modify the matching operation and a time-out interval if no match is found.
ToString()	Returns the regular expression pattern that was passed into the `Regex` constructor.
Unescape(String)	Converts any escaped characters in the input string.
UseOptionC()	Used by a Regex object generated by the CompileToAssembly method.
UseOptionR()	Used by a Regex object generated by the CompileToAssembly method.
ValidateMatchTimeout(TimeSpan)	Checks whether a time-out interval is within an acceptable range.

Explicit Interface Implementations

ISerializable.GetObjectData(SerializationInfo, StreamingContext)	Populates a SerializationInfo object with the data necessary to deserialize the current Regex object.

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

Thread Safety

The Regex class is immutable (read-only) and thread safe. Regex objects can be created on any thread and shared between threads. For more information, see Thread Safety.

See also

- RegexStringValidator
- .NET Framework Regular Expressions
- Regular Expression Language Elements
- Regular Expressions - Quick Reference (download in Word format)
- Regular Expressions - Quick Reference (download in PDF format)