






.NET regular expressions

03/30/2017 • 10 minutes to read •      +7

In this article

[How regular expressions work](#)

[Regular expression examples](#)

[Related topics](#)

[Reference](#)

Regular expressions provide a powerful, flexible, and efficient method for processing text. The extensive pattern-matching notation of regular expressions enables you to quickly parse large amounts of text to:

- Find specific character patterns.
- Validate text to ensure that it matches a predefined pattern (such as an email address).
- Extract, edit, replace, or delete text substrings.
- Add extracted strings to a collection in order to generate a report.

For many applications that deal with strings or that parse large blocks of text, regular expressions are an indispensable tool.

How regular expressions work

The centerpiece of text processing with regular expressions is the regular expression engine, which is represented by the [System.Text.RegularExpressions.Regex](#) object in .NET. At a minimum, processing text using regular expressions requires that the regular expression engine be provided with the following two items of information:

- The regular expression pattern to identify in the text.

In .NET, regular expression patterns are defined by a special syntax or language, which is compatible with Perl 5 regular expressions and adds some additional features such as right-to-left matching. For more information, see [Regular Expression Language - Quick Reference](#).

- The text to parse for the regular expression pattern.

The methods of the [Regex](#) class let you perform the following operations:

- Determine whether the regular expression pattern occurs in the input text by calling the [Regex.IsMatch](#) method. For an example that uses the [IsMatch](#) method for validating text, see [How to: Verify that Strings Are in Valid Email Format](#).
- Retrieve one or all occurrences of text that matches the regular expression pattern by calling the [Regex.Match](#) or [Regex.Matches](#) method. The former method returns a [System.Text.RegularExpressions.Match](#) object that provides information about the matching text. The latter returns a [MatchCollection](#) object that contains one [System.Text.RegularExpressions.Match](#) object for each match found in the parsed text.
- Replace text that matches the regular expression pattern by calling the [Regex.Replace](#) method. For examples that use the [Replace](#) method to change date formats and remove invalid characters from a string, see [How to: Strip Invalid Characters from a String](#) and [Example: Changing Date Formats](#).

For an overview of the regular expression object model, see [The Regular Expression Object Model](#).

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](#)

Regular expression examples


The [String](#) class includes a number of string search and replacement methods that you can use when you want to locate literal strings in a larger string. Regular expressions are most useful either when you want to locate one of several substrings in a larger string, or when you want to identify patterns in a string, as the following examples illustrate.

Tip

The [System.Web.RegularExpressions](#) namespace contains a number of regular expression objects that implement predefined regular expression patterns for parsing strings from HTML, XML, and ASP.NET documents. For example, the [TagRegex](#) class identifies start tags in a string and the [CommentRegex](#) class identifies ASP.NET comments in a string.

Example 1: Replace substrings


Assume that a mailing list contains names that sometimes include a title (Mr., Mrs., Miss, or Ms.) along with a first and last name. If you do not want to include the titles when you generate envelope labels from the list, you can use a regular expression to remove the titles, as the following example illustrates.

C#	 Copy
<pre>using System; using System.Text.RegularExpressions; public class Example { public static void Main() { string pattern = "(Mr\\.?.? Mrs\\.?.? Miss Ms\\.?.?)"; string[] names = { "Mr. Henry Hunt", "Ms. Sara Samuels", "Abraham Adams", "Ms. Nicole Norris" }; foreach (string name in names) Console.WriteLine(Regex.Replace(name, pattern, String.Empty)); } } // The example displays the following output: // Henry Hunt // Sara Samuels // Abraham Adams // Nicole Norris</pre>	

The regular expression pattern `(Mr\\.?.? |Mrs\\.?.? |Miss |Ms\\.?.?)` matches any occurrence of "Mr ", "Mr. ", "Mrs ", "Mrs. ", "Miss ", "Ms or "Ms. ". The call to the [Regex.Replace](#) method replaces the matched string with [String.Empty](#); in other words, it removes it from the original string.

Example 2: Identify duplicated words

Accidentally duplicating words is a common error that writers make. A regular expression can be used to identify duplicated words, as the following example shows.

C#	 Copy
<pre>using System; using System.Text.RegularExpressions; public class Class1</pre>	

```

{
    public static void Main()
    {
        string pattern = @"\b(\w+)\s\1\b";
        string input = "This this is a nice day. What about this? This tastes
good. I saw a a dog.";
        foreach (Match match in Regex.Matches(input, pattern,
RegexOptions.IgnoreCase))
            Console.WriteLine("{0} (duplicates '{1}') at position {2}",
                               match.Value, match.Groups[1].Value, match.Index);
    }
}
// The example displays the following output:
//      This this (duplicates 'This') at position 0
//      a a (duplicates 'a') at position 66

```

The regular expression pattern `\b(\w+)\s\1\b` can be interpreted as follows:

Pattern	Interpretation
<code>\b</code>	Start at a word boundary.
<code>(\w+?)</code>	Match one or more word characters, but as few characters as possible. Together, they form a group that can be referred to as <code>\1</code> .
<code>\s</code>	Match a white-space character.
<code>\1</code>	Match the substring that is equal to the group named <code>\1</code> .
<code>\b</code>	Match a word boundary.


The [Regex.Matches](#) method is called with regular expression options set to [RegexOptions.IgnoreCase](#). Therefore, the match operation is case-insensitive, and the example identifies the substring "This this" as a duplication.

The input string includes the substring "this? This". However, because of the intervening punctuation mark, it is not identified as a duplication.

Example 3: Dynamically build a culture-sensitive regular expression

The following example illustrates the power of regular expressions combined with the flexibility offered by .NET's globalization features. It uses the [NumberFormatInfo](#) object to determine the format of currency values in the system's current culture. It then uses that information to dynamically construct a regular expression that extracts currency values

from the text. For each match, it extracts the subgroup that contains the numeric string only, converts it to a [Decimal](#) value, and calculates a running total.

C#	 Copy
<pre>using System; using System.Collections.Generic; using System.Globalization; using System.Text.RegularExpressions; public class Example { public static void Main() { // Define text to be parsed. string input = "Office expenses on 2/13/2008:\n" + "Paper (500 sheets) \$3.95\n" + "Pencils (box of 10) \$1.00\n" + "Pens (box of 10) \$4.49\n" + "Erasers \$2.19\n" + "Ink jet printer \$69.95\n\n" + "Total Expenses \$ 81.58\n"; // Get current culture's NumberFormatInfo object. NumberFormatInfo nfi = CultureInfo.CurrentCulture.NumberFormat; // Assign needed property values to variables. string currencySymbol = nfi.CurrencySymbol; bool symbolPrecedesIfPositive = nfi.CurrencyPositivePattern % 2 == 0; string groupSeparator = nfi.CurrencyGroupSeparator; string decimalSeparator = nfi.CurrencyDecimalSeparator; // Form regular expression pattern. string pattern = Regex.Escape(symbolPrecedesIfPositive ? currencySymbol : "") + @"\\s*[-+]?\" + "([0-9]{0,3}(" + groupSeparator + "[0-9] {3})*(" + Regex.Escape(decimalSeparator) + "[0-9]+)?)\" + (! symbolPrecedesIfPositive ? currencySymbol : ""); Console.WriteLine("The regular expression pattern is:"); Console.WriteLine(" " + pattern); // Get text that matches regular expression pattern. MatchCollection matches = Regex.Matches(input, pattern, RegexOptions.IgnorePatternWhitespace); Console.WriteLine("Found {0} matches.", matches.Count); // Get numeric string, convert it to a value, and add it to List object. List<decimal> expenses = new List<Decimal>();</pre>	

```

foreach (Match match in matches)
    expenses.Add(Decimal.Parse(match.Groups[1].Value));

// Determine whether total is present and if present, whether it is
correct.
decimal total = 0;
foreach (decimal value in expenses)
    total += value;

if (total / 2 == expenses[expenses.Count - 1])
    Console.WriteLine("The expenses total {0:C2}.",
expenses[expenses.Count - 1]);
else
    Console.WriteLine("The expenses total {0:C2}.", total);
}
}
// The example displays the following output:
//      The regular expression pattern is:
//      \s*[-+]?([0-9]{0,3}([0-9]{3})*(\.[0-9]+)?)
//      Found 6 matches.
//      The expenses total $81.58.

```

On a computer whose current culture is English - United States (en-US), the example dynamically builds the regular expression `\s*[-+]?([0-9]{0,3}([0-9]{3})*(\.[0-9]+)?)`. This regular expression pattern can be interpreted as follows:

Pattern	Interpretation
<code>\\$</code>	Look for a single occurrence of the dollar symbol (\$) in the input string. The regular expression pattern string includes a backslash to indicate that the dollar symbol is to be interpreted literally rather than as a regular expression anchor. (The \$ symbol alone would indicate that the regular expression engine should try to begin its match at the end of a string.) To ensure that the current culture's currency symbol is not misinterpreted as a regular expression symbol, the example calls the Regex.Escape method to escape the character.
<code>\s*</code>	Look for zero or more occurrences of a white-space character.
<code>[-+]?</code>	Look for zero or one occurrence of either a positive sign or a negative sign.
<code>([0-9]{0,3}([0-9]{3})*(\.[0-9]+)?)</code>	The outer parentheses around this expression define it as a capturing group or a subexpression. If a match is found, information about this part of the matching string can be retrieved from the second Group object in the GroupCollection object returned by the Match.Groups property. (The first element in the collection represents the entire match.)

Pattern	Interpretation
[0-9]{0,3}	Look for zero to three occurrences of the decimal digits 0 through 9.
(,[0-9]{3})*	Look for zero or more occurrences of a group separator followed by three decimal digits.
\.	Look for a single occurrence of the decimal separator.
[0-9]+	Look for one or more decimal digits.
(\.[0-9]+)?	Look for zero or one occurrence of the decimal separator followed by at least one decimal digit.

If each of these subpatterns is found in the input string, the match succeeds, and a [Match](#) object that contains information about the match is added to the [MatchCollection](#) object.

Related topics

Title	Description
Regular Expression Language - Quick Reference	Provides information on the set of characters, operators, and constructs that you can use to define regular expressions.
The Regular Expression Object Model	Provides information and code examples that illustrate how to use the regular expression classes.
Details of Regular Expression Behavior	Provides information about the capabilities and behavior of .NET regular expressions.

[Use regular expressions in Visual Studio](#)

Reference

- [System.Text.RegularExpressions](#)
- [System.Text.RegularExpressions.Regex](#)
- [Regular Expressions - Quick Reference \(download in Word format\)](#)

- [Regular Expressions - Quick Reference \(download in PDF format\)](#)

Is this page helpful?

 Yes  No
