String interpolation in C#

Article • 08/29/2023

This tutorial shows you how to use string interpolation to format and include expression results in a result string. The examples assume that you are familiar with basic C# concepts and .NET type formatting. If you are new to string interpolation or .NET type formatting, check out the interactive string interpolation tutorial first. For more information about formatting types in .NET, see Formatting types in .NET.

Introduction

To identify a string literal as an interpolated string, prepend it with the `$` symbol. You can embed any valid C# expression that returns a value in an interpolated string. In the following example, as soon as an expression is evaluated, its result is converted into a string and included in a result string:

C#

```
double a = 3;
double b = 4;
Console.WriteLine($"Area of the right triangle with legs of {a} and {b} is
{0.5 * a * b}");
Console.WriteLine($"Length of the hypotenuse of the right triangle with legs
of {a} and {b} is {CalculateHypotenuse(a, b)}");
double CalculateHypotenuse(double leg1, double leg2) => Math.Sqrt(leg1 *
leg1 + leg2 * leg2);
// Output:
// Area of the right triangle with legs of 3 and 4 is 6
// Length of the hypotenuse of the right triangle with legs of 3 and 4 is 5
```

As the example shows, you include an expression in an interpolated string by enclosing it with braces:

C#

```
{<interpolationExpression>}
```

Interpolated strings support all the capabilities of the string composite formatting feature. That makes them a more readable alternative to the use of the String.Format method.

How to specify a format string for an interpolation expression

To specify a format string that is supported by the type of the expression result, follow the interpolation expression with a colon (":") and the format string:

```
C#

{<interpolationExpression>:<formatString>}
```

The following example shows how to specify standard and custom format strings for expressions that produce date and time or numeric results:

```
C#

var date = new DateTime(1731, 11, 25);
Console.WriteLine($"On {date:dddd, MMMM dd, yyyy} L. Euler introduced the
letter e to denote {Math.E:F5}.");
// Output:
// On Sunday, November 25, 1731 L. Euler introduced the letter e to denote
2.71828.
```

For more information, see the Format string component section of the Composite formatting article.

How to control the field width and alignment of the formatted interpolation expression

To specify the minimum field width and the alignment of the formatted expression result, follow the interpolation expression with a comma (",") and the constant expression:

```
C#

{<interpolationExpression>,<alignment>}
```

If the *alignment* value is positive, the formatted expression result is right-aligned; if negative, it's left-aligned.

If you need to specify both alignment and a format string, start with the alignment component:

```
```

C#

```
{<interpolationExpression>,<alignment>:<formatString>}
```

The following example shows how to specify alignment and uses pipe characters ("|") to delimit text fields:

C#

```
const int NameAlignment = -9;
const int ValueAlignment = 7;
double a = 3;
double b = 4;
Console.WriteLine($"Three classical Pythagorean means of {a} and {b}:");
Console.WriteLine($"|{"Arithmetic",NameAlignment}|{0.5 * (a +
b),ValueAlignment:F3}|");
Console.WriteLine($"|{"Geometric",NameAlignment}|{Math.Sqrt(a *
b),ValueAlignment:F3}|");
Console.WriteLine($"|{"Harmonic",NameAlignment}|{2 / (1 / a + 1 /
b),ValueAlignment:F3}|");
// Output:
// Three classical Pythagorean means of 3 and 4:
// |Arithmetic|   3.500|
// |Geometric|   3.464|
// |Harmonic |   3.429|
```

As the example output shows, if the length of the formatted expression result exceeds specified field width, the *alignment* value is ignored.

For more information, see the Alignment component section of the Composite formatting article.

How to use escape sequences in an interpolated string

Interpolated strings support all escape sequences that can be used in ordinary string literals. For more information, see String escape sequences.

To interpret escape sequences literally, use a verbatim string literal. An interpolated verbatim string starts with the both `$` and `@` characters. You can use `$` and `@` in any order: both `$@"..."` and `@$"..."` are valid interpolated verbatim strings.

To include a brace, "{" or "}", in a result string, use two braces, "{{" or "}}". For more information, see the Escaping braces section of the Composite formatting article.

The following example shows how to include braces in a result string and construct a verbatim interpolated string:

```
C#
```

```
var xs = new int[] { 1, 2, 7, 9 };
var ys = new int[] { 7, 9, 12 };
Console.WriteLine($"Find the intersection of the {{{string.Join(", ",xs)}}}
and {{{string.Join(", ",ys)}}} sets.");
// Output:
// Find the intersection of the {1, 2, 7, 9} and {7, 9, 12} sets.

var userName = "Jane";
var stringWithEscapes = $"C:\\Users\\{userName}\\Documents";
var verbatimInterpolated = $@"C:\Users\{userName}\Documents";
Console.WriteLine(stringWithEscapes);
Console.WriteLine(verbatimInterpolated);
// Output:
// C:\Users\Jane\Documents
// C:\Users\Jane\Documents
```

Beginning with C# 11, you can use interpolated raw string literals.

How to use a ternary conditional operator `?:` in an interpolation expression

As the colon (":") has special meaning in an item with an interpolation expression, in order to use a conditional operator in an expression, enclose it in parentheses, as the following example shows:

```
C#
```

```
var rand = new Random();
for (int i = 0; i < 7; i++)
{
    Console.WriteLine($"Coin flip: {(rand.NextDouble() < 0.5 ? "heads" :
"tails")}");
}
```

How to create a culture-specific result string with string interpolation

By default, an interpolated string uses the current culture defined by the CultureInfo.CurrentCulture property for all formatting operations.

Beginning with .NET 6, you can use the String.Create(IFormatProvider, DefaultInterpolatedStringHandler) method to resolve an interpolated string to a culture-specific result string, as the following example shows:

C#

```
var cultures = new System.Globalization.CultureInfo[]
{
    System.Globalization.CultureInfo.GetCultureInfo("en-US"),
    System.Globalization.CultureInfo.GetCultureInfo("en-GB"),
    System.Globalization.CultureInfo.GetCultureInfo("nl-NL"),
    System.Globalization.CultureInfo.InvariantCulture
};
var date = DateTime.Now;
var number = 31_415_926.536;
foreach (var culture in cultures)
{
    var cultureSpecificMessage = string.Create(culture, $"{date,23}
{number,20:N3}");
    Console.WriteLine($"{culture.Name,-10}{cultureSpecificMessage}");
}
// Output is similar to:
// en-US       8/27/2023 12:35:31 PM      31,415,926.536
// en-GB         27/08/2023 12:35:31      31,415,926.536
// nl-NL         27-08-2023 12:35:31      31.415.926,536
//               08/27/2023 12:35:31      31,415,926.536
```

In earlier versions of .NET, use implicit conversion of an interpolated string to a System.FormattableString instance and call its ToString(IFormatProvider) method to create a culture-specific result string. The following example shows how to do that:

C#

```
var cultures = new System.Globalization.CultureInfo[]
{
    System.Globalization.CultureInfo.GetCultureInfo("en-US"),
    System.Globalization.CultureInfo.GetCultureInfo("en-GB"),
    System.Globalization.CultureInfo.GetCultureInfo("nl-NL"),
    System.Globalization.CultureInfo.InvariantCulture
};
var date = DateTime.Now;
var number = 31_415_926.536;
FormattableString message = $"{date,23}{number,20:N3}";
foreach (var culture in cultures)
{
    var cultureSpecificMessage = message.ToString(culture);
    Console.WriteLine($"{culture.Name,-10}{cultureSpecificMessage}");
}
// Output is similar to:
// en-US       8/27/2023 12:35:31 PM      31,415,926.536
// en-GB         27/08/2023 12:35:31      31,415,926.536
```

```
// nl-NL          27-08-2023 12:35:31      31.415.926,536
//                08/27/2023 12:35:31      31,415,926.536
```

As the example shows, you can use one FormattableString instance to generate multiple result strings for various cultures.

How to create a result string using the invariant culture

Beginning with .NET 6, use the String.Create(IFormatProvider, DefaultInterpolatedStringHandler) method to resolve an interpolated string to a result string for the InvariantCulture, as the following example shows:

C#

```
string message = string.Create(CultureInfo.InvariantCulture, $"Date and time
in invariant culture: {DateTime.Now}");
Console.WriteLine(message);
// Output is similar to:
// Date and time in invariant culture: 05/17/2018 15:46:24
```

In earlier versions of .NET, along with the FormattableString.ToString(IFormatProvider) method, you can use the static FormattableString.Invariant method, as the following example shows:

C#

```
string message = FormattableString.Invariant($"Date and time in invariant
culture: {DateTime.Now}");
Console.WriteLine(message);
// Output is similar to:
// Date and time in invariant culture: 05/17/2018 15:46:24
```

Conclusion

This tutorial describes common scenarios of string interpolation usage. For more information about string interpolation, see String interpolation. For more information about formatting types in .NET, see the Formatting types in .NET and Composite formatting articles.

See also

- String.Format
- System.FormattableString
- System.IFormattable
- Strings