String interpolation using `$`

Article • 08/29/2023

The `$` character identifies a string literal as an *interpolated string*. An interpolated string is a string literal that might contain *interpolation expressions*. When an interpolated string is resolved to a result string, items with interpolation expressions are replaced by the string representations of the expression results.

String interpolation provides a more readable, convenient syntax to format strings. It's easier to read than string composite formatting. The following example uses both features to produce the same output:

C#

```
var name = "Mark";
var date = DateTime.Now;

// Composite formatting:
Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.",
name, date.DayOfWeek, date);
// String interpolation:
Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, it's
{date:HH:mm} now.");
// Both calls produce the same output that is similar to:
// Hello, Mark! Today is Wednesday, it's 19:40 now.
```

Beginning with C# 10, you can use an interpolated string to initialize a constant string. You can do that only if all interpolation expressions within the interpolated string are constant strings as well.

Structure of an interpolated string

To identify a string literal as an interpolated string, prepend it with the `$` symbol. You can't have any white space between the `$` and the `"` that starts a string literal.

The structure of an item with an interpolation expression is as follows:

C#

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

Elements in square brackets are optional. The following table describes each element:

Element	Description
`interpolationExpression`	The expression that produces a result to be formatted. String representation of `null` is String.Empty.
`alignment`	The constant expression whose value defines the minimum number of characters in the string representation of the expression result. If positive, the string representation is right-aligned; if negative, it's left-aligned. For more information, see the Alignment component section of the Composite formatting article.
`formatString`	A format string that is supported by the type of the expression result. For more information, see the Format string component section of the Composite formatting article.

The following example uses optional formatting components described above:

C#

```
Console.WriteLine($"|{"Left",-7}|{"Right",7}|");

const int FieldWidthRightAligned = 20;
Console.WriteLine($"{Math.PI,FieldWidthRightAligned} - default for-
matting of the pi number");
Console.WriteLine($"{Math.PI,FieldWidthRightAligned:F3} - display
only three decimal digits of the pi number");
// Output is:
// |Left   |  Right|
//     3.14159265358979 - default formatting of the pi number
//                3.142 - display only three decimal digits of the pi
number
```

Beginning with C# 11, you can use new-lines within an interpolation expression to make the expression's code more readable. The following example shows how new-lines can improve the readability of an expression involving pattern matching:

C#

```
string message = $"The usage policy for {safetyScore} is {
    safetyScore switch
    {
        > 90 => "Unlimited usage",
        > 80 => "General usage, with daily safety check",
        > 70 => "Issues must be addressed within 1 week",
        > 50 => "Issues must be addressed within 1 day",
        _ => "Issues must be addressed before continued use",
    }
    }";
```

Interpolated raw string literals

Beginning with C# 11, you can use an interpolated raw string literal, as the following example shows:

C#

```
int X = 2;
int Y = 3;

var pointMessage = $"""The point "{X}, {Y}" is {Math.Sqrt(X * X + Y *
Y):F3} from the origin""";

Console.WriteLine(pointMessage);
// Output is:
// The point "2, 3" is 3.606 from the origin
```

To embed `{` and `}` characters in the result string, start an interpolated raw string literal with multiple `$` characters. When you do that, any sequence of `{` or `}` characters shorter than the number of `$` characters is embedded in the result string. To enclose any interpolation expression within that string, you need to use the same number of braces as the number of `$` characters, as the the following example shows:

C#

```
int X = 2;
int Y = 3;

var pointMessage = $$"""{The point {{{X}}, {{Y}}} is {{Math.Sqrt(X *
X + Y * Y):F3}} from the origin}""";
Console.WriteLine(pointMessage);
// Output is:
// {The point {2, 3} is 3.606 from the origin}
```

In the preceding example, an interpolated raw string literal starts with two `$` characters. That's why you need to put every interpolation expression between double braces, `{{` and `}}`. A single brace is embedded into a result string. If you need to embed repeated `{` or `}` characters into a result string, use an appropriately greater number of `$` characters to designate an interpolated raw string literal.

Special characters

To include a brace, "{" or "}", in the text produced by an interpolated string, use two braces, "{{" or "}}". For more information, see the Escaping braces section of the

Composite formatting article.

As the colon (":") has special meaning in an interpolation expression item, to use a conditional operator in an interpolation expression, enclose that expression in parentheses.

The following example shows how to include a brace in a result string. It also shows how to use a conditional operator:

C#

```
string name = "Horace";
int age = 34;
Console.WriteLine($"He asked, \"Is your name {name}?\", but didn't
wait for a reply :-{{");
Console.WriteLine($"{name} is {age} year{(age == 1 ? "" : "s")}
old.");
// Output is:
// He asked, "Is your name Horace?", but didn't wait for a reply :-{
// Horace is 34 years old.
```

An interpolated verbatim string starts with the both `$` and `@` characters. You can use `$` and `@` in any order: both `$@"..."` and `@$"..."` are valid interpolated verbatim strings. For more information about verbatim strings, see the string and verbatim identifier articles.

Culture-specific formatting

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

To resolve an interpolated string to a culture-specific result string, use the String.Create(IFormatProvider, DefaultInterpolatedStringHandler) method, which is available beginning with .NET 6. The following example shows how to do that:

C#

```
double speedOfLight = 299792.458;

System.Globalization.CultureInfo.CurrentCulture =
System.Globalization.CultureInfo.GetCultureInfo("nl-NL");
string messageInCurrentCulture = $"The speed of light is {speedOf-
Light:N3} km/s.";

var specificCulture =
System.Globalization.CultureInfo.GetCultureInfo("en-IN");
string messageInSpecificCulture = string.Create(
```

```
    specificCulture, $"The speed of light is {speedOfLight:N3}
km/s.");

string messageInInvariantCulture = string.Create(
    System.Globalization.CultureInfo.InvariantCulture, $"The speed of
light is {speedOfLight:N3} km/s.");

Console.WriteLine($"
{System.Globalization.CultureInfo.CurrentCulture,-10} {messageInCur-
rentCulture}");
Console.WriteLine($"{specificCulture,-10}
{messageInSpecificCulture}");
Console.WriteLine($"{"Invariant",-10} {messageInInvariantCulture}");
// Output is:
// nl-NL      The speed of light is 299.792,458 km/s.
// en-IN      The speed of light is 2,99,792.458 km/s.
// Invariant  The speed of light is 299,792.458 km/s.
```

In .NET 5 and earlier versions of .NET, use implicit conversion of an interpolated string to a FormattableString instance. Then, you can use an instance FormattableString.ToString(IFormatProvider) method or a static FormattableString.Invariant method to produce a culture-specific result string. The following example shows how to do that:

C#

```
double speedOfLight = 299792.458;
FormattableString message = $"The speed of light is
{speedOfLight:N3} km/s.";

var specificCulture =
System.Globalization.CultureInfo.GetCultureInfo("en-IN");
string messageInSpecificCulture = message.ToString(specificCulture);
Console.WriteLine(messageInSpecificCulture);
// Output:
// The speed of light is 2,99,792.458 km/s.

string messageInInvariantCulture =
FormattableString.Invariant(message);
Console.WriteLine(messageInInvariantCulture);
// Output is:
// The speed of light is 299,792.458 km/s.
```

For more information about custom formatting, see the Custom formatting with ICustomFormatter section of the Formatting types in .NET article.

Other resources

If you're new to string interpolation, see the String interpolation in C# interactive tutorial. You can also check another String interpolation in C# tutorial. That tutorial demonstrates how to use interpolated strings to produce formatted strings.

Compilation of interpolated strings

Beginning with C# 10 and .NET 6, the compiler checks if an interpolated string is assigned to a type that satisfies the *interpolated string handler pattern*. An *interpolated string handler* is a type that converts the interpolated string into a result string. When an interpolated string has the type `string`, it's processed by the System.Runtime.CompilerServices.DefaultInterpolatedStringHandler. For the example of a custom interpolated string handler, see the Write a custom string interpolation handler tutorial. Use of an interpolated string handler is an advanced scenario, typically required for performance reasons.

> **Note**
>
> One side effect of interpolated string handlers is that a custom handler, including **System.Runtime.CompilerServices.DefaultInterpolatedStringHandler**, may not evaluate all the interpolation expressions within the interpolated string under all conditions. That means side effects of those expressions may not occur.

In C# 9 and earlier, if an interpolated string has the type `string`, it's typically transformed into a String.Format method call. The compiler may replace String.Format with String.Concat if the analyzed behavior would be equivalent to concatenation.

If an interpolated string has the type IFormattable or FormattableString, the compiler generates a call to the FormattableStringFactory.Create method.

C# language specification

For more information, see the Interpolated string expressions section of the C# language specification and the following new feature specifications:

- C# 10 - Improved interpolated strings
- C# 11 - Raw string literals
- C# 11 - New-lines in string interpolations

See also

- C# reference
- C# special characters
- Strings
- Standard numeric format strings
- Composite formatting
- String.Format
- Simplify interpolation (style rule IDE0071)
- String interpolation in C# 10 and .NET 6 (.NET blog)