C# String Format

String interpolation using $

The $ special 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. Compare the following example that uses both features to produce the same output:

```
string 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.
```

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. To concatenate multiple interpolated strings, add the $ special character to each string literal.

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

```
{<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 Alignment Component.
`formatString`	A format string that is supported by the type of the expression result. For more information, see Format String Component.

The following example uses optional formatting components described above:

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

const int FieldWidthRightAligned = 20;

Console.WriteLine($"{Math.PI,FieldWidthRightAligned} - default formatting of the pi number");

Console.WriteLine($"{Math.PI,FieldWidthRightAligned:F3} - display only three decimal digits of the pi number");

// Expected 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# 10, you can use string interpolation to initialize a constant string. All expressions used for placeholders must be constant strings. In other words, every *interpolation expression* must be a string, and it must be a compile time constant.

Beginning with C# 11, the interpolated expressions can include newlines. The text between the { and } must be valid C#, therefore it can include newlines that improve readability. The following example shows how newlines can improve the readability of an expression involving pattern matching:

```
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",
    }
    }";
```

Also, beginning in C# 11, you can use a raw string literal for the format string:

```
int X = 2;
int Y = 3;
var pointMessage = $"""The point "{X}, {Y}" is {Math.Sqrt(X * X + Y * Y)} from the origin""";
Console.WriteLine(pointMessage);
// output:  The point "2, 3" is 3.605551275463989 from the origin.
```

You can use multiple $ characters in an interpolated raw string literal to embed { and } characters in the output string without escaping them:

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

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

Console.WriteLine(pointMessage);

// output:  The point {2, 3} is 3.605551275463989 from the origin.
```

If your output string should contain repeated { or } characters, you can add more $ to designate the interpolated string. Any sequence of { or } shorter than the number of $ will be embedded in the output string. As shown in the preceding example, sequences longer than the sequence of $ characters embed the additional { or } characters in the output. The compiler issues an error if the sequence of brace characters is equal to or greater than double the length of the sequence of $ characters.

Composite format string

A composite format string and object list are used as arguments of methods that support the composite formatting feature. A composite format string consists of zero or more runs of fixed text intermixed with one or more format items. The fixed text is any string that you choose, and each format item corresponds to an object or boxed structure in the list. The composite formatting feature returns a new result string where each format item is replaced by the string representation of the corresponding object in the list.

Consider the following Format code fragment:

```
string name = "Fred";

String.Format("Name = {0}, hours = {1:hh}", name, DateTime.Now);
```

The fixed text is Name = and , hours = . The format items are {0}, whose index of 0 corresponds to the object name, and {1:hh}, whose index of 1 corresponds to the object DateTime.Now.

Format item syntax

Each format item takes the following form and consists of the following components:

{index[,alignment][:formatString]}

The matching braces ({ and }) are required.