

In this article

The "d" custom format specifier

The "dd" custom format specifier

The "ddd" custom format specifier

The "dddd" custom format specifier

The "f" custom format specifier

The "ff" custom format specifier

The "fff" custom format specifier

The "ffff" custom format specifier

The "fffff" custom format specifier

The "ffffff" custom format specifier

The "fffffff" custom format specifier

The "F" custom format specifier

The "FF" custom format specifier

The "FFF" custom format specifier

The "FFFF" custom format specifier

The "FFFFF" custom format specifier

The "FFFFFF" custom format specifier

The "FFFFFFF" custom format specifier

The "g" or "gg" custom format specifier

The "h" custom format specifier

The "hh" custom format specifier

The "H" custom format specifier

The "HH" custom format specifier

The "K" custom format specifier

The "m" custom format specifier

The "mm" custom format specifier

The "M" custom format specifier

- The "MM" custom format specifier
- The "MMM" custom format specifier
- The "MMMM" custom format specifier
- The "s" custom format specifier
- The "ss" custom format specifier
- The "t" custom format specifier
- The "tt" custom format specifier
- The "y" custom format specifier
- The "yy" custom format specifier
- The "yyy" custom format specifier
- The "yyyy" custom format specifier
- The "yyyyy" custom format specifier
- The "z" custom format specifier
- The "zz" custom format specifier
- The "zzz" custom format specifier
- The ":" custom format specifier
- The "/" custom format specifier
- Character literals
- Notes
- See also

A date and time format string defines the text representation of a [DateTime](#) or [DateTimeOffset](#) value that results from a formatting operation. It can also define the representation of a date and time value that is required in a parsing operation in order to successfully convert the string to a date and time. A custom format string consists of one or more custom date and time format specifiers. Any string that is not a [standard date and time format string](#) is interpreted as a custom date and time format string.



You can download the [Formatting Utility](#), an application that enables you to apply format strings to either date and time or numeric values and displays the result string.

Custom date and time format strings can be used with both [DateTime](#) and [DateTimeOffset](#) values.

ⓘ Note

Some of the C# examples in this article run in the [Try.NET](#) inline code runner and playground. Select the **Run** button to run an example in an interactive window. Once you execute the code, you can modify it and run the modified code by selecting **Run** again. The modified code either runs in the interactive window or, if compilation fails, the interactive window displays all C# compiler error messages.

The [local time zone](#) of the [Try.NET](#) inline code runner and playground is Coordinated Universal Time, or UTC. This may affect the behavior and the output of examples that illustrate the [DateTime](#), [DateTimeOffset](#), and [TimeZoneInfo](#) types and their members.

In formatting operations, custom date and time format strings can be used either with the `ToString` method of a date and time instance or with a method that supports composite formatting. The following example illustrates both uses.

C#



```
DateTime thisDate1 = new DateTime(2011, 6, 10);
Console.WriteLine("Today is " + thisDate1.ToString("MMMM dd, yyyy") + ".");

DateTimeOffset thisDate2 = new DateTimeOffset(2011, 6, 10, 15, 24, 16,
                                              TimeSpan.Zero);
Console.WriteLine("The current date and time: {0:MM/dd/yy H:mm:ss zzz}",
                  thisDate2);
// The example displays the following output:
// Today is June 10, 2011.
// The current date and time: 06/10/11 15:24:16 +00:00
```

In parsing operations, custom date and time format strings can be used with the [DateTime.ParseExact](#), [DateTime.TryParseExact](#), [DateTimeOffset.ParseExact](#), and [DateTimeOffset.TryParseExact](#) methods. These methods require that an input string conforms exactly to a particular pattern for the parse operation to succeed. The following example illustrates a call to the [DateTimeOffset.ParseExact\(String, String, IFormatProvider\)](#) method to parse a date that must include a day, a month, and a two-digit year.

C#

 Copy Run

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        string[] dateValues = { "30-12-2011", "12-30-2011",
                                "30-12-11", "12-30-11" };

        string pattern = "MM-dd-yy";
        DateTime parsedDate;

        foreach (var dateValue in dateValues) {
            if (DateTime.TryParseExact(dateValue, pattern, null,
                                      DateTimeStyles.None, out parsedDate))
                Console.WriteLine("Converted '{0}' to {1:d}.",
                                dateValue, parsedDate);
            else
                Console.WriteLine("Unable to convert '{0}' to a date and time.",
                                dateValue);
        }
    }
}

// The example displays the following output:
//     Unable to convert '30-12-2011' to a date and time.
//     Unable to convert '12-30-2011' to a date and time.
```

```
// Unable to convert '30-12-11' to a date and time.
// Converted '12-30-11' to 12/30/2011.
```

The following table describes the custom date and time format specifiers and displays a result string produced by each format specifier. By default, result strings reflect the formatting conventions of the en-US culture. If a particular format specifier produces a localized result string, the example also notes the culture to which the result string applies. For more information about using custom date and time format strings, see the [Notes](#) section.

Format specifier	Description	Examples
"d"	The day of the month, from 1 through 31.	2009-06-01T13:45:30 -> 1
	More information: The "d" Custom Format Specifier .	2009-06-15T13:45:30 -> 15
"dd"	The day of the month, from 01 through 31.	2009-06-01T13:45:30 -> 01
	More information: The "dd" Custom Format Specifier .	2009-06-15T13:45:30 -> 15
"ddd"	The abbreviated name of the day of the week.	2009-06-15T13:45:30 -> Mon (en-US)
	More information: The "ddd" Custom Format Specifier .	2009-06-15T13:45:30 -> Пн (ru-RU)
		2009-06-15T13:45:30 -> lun. (fr-FR)
Format specifier	Description	Examples
"dddd"	The full name of the day of the week.	2009-06-15T13:45:30 -> Monday (en-US)
	More information: The "dddd" Custom Format Specifier .	2009-06-15T13:45:30 -> понедельник (ru-RU)

		2009-06-15T13:45:30 -> lundi (fr-FR)
"f"	The tenths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 6
	More information: The "f" Custom Format Specifier.	2009-06-15T13:45:30.05 -> 0
"ff"	The hundredths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 61
	More information: The "ff" Custom Format Specifier.	2009-06-15T13:45:30.0050000 -> 00
"fff"	The milliseconds in a date and time value.	6/15/2009 13:45:30.617 -> 617
	More information: The "fff" Custom Format Specifier.	6/15/2009 13:45:30.0005 -> 000
"ffff"	The ten thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175000 -> 6175
	More information: The "ffff" Custom Format Specifier.	2009-06-15T13:45:30.0000500 -> 0000
"fffff"	The hundred thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175400 -> 61754
		6/15/2009 13:45:30.000005 -> 00000
	More information: The "fffff" Custom Format Specifier.	
Format specifier	Description	Examples
"ffffff"	The millionths of a second in a date and time value.	2009-06-15T13:45:30.6175420 -> 617542
	More information: The "ffffff" Custom Format Specifier.	2009-06-15T13:45:30.0000005 -> 000000

"ffffff"	The ten millionths of a second in a date and time value.	2009-06-15T13:45:30.6175425 -> 6175425
	More information: The "ffffff" Custom Format Specifier.	2009-06-15T13:45:30.0001150 -> 0001150
"F"	If non-zero, the tenths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 6
	More information: The "F" Custom Format Specifier.	2009-06-15T13:45:30.0500000 -> (no output)
"FF"	If non-zero, the hundredths of a second in a date and time value.	2009-06-15T13:45:30.6170000 -> 61
	More information: The "FF" Custom Format Specifier.	2009-06-15T13:45:30.0050000 -> (no output)
"FFF"	If non-zero, the milliseconds in a date and time value.	2009-06-15T13:45:30.6170000 -> 617
	More information: The "FFF" Custom Format Specifier.	2009-06-15T13:45:30.0005000 -> (no output)
"FFFF"	If non-zero, the ten thousandths of a second in a date and time value.	2009-06-15T13:45:30.5275000 -> 5275
	More information: The "FFFF" Custom Format Specifier.	2009-06-15T13:45:30.0000500 -> (no output)
Format specifier	Description	Examples
"FFFFF"	If non-zero, the hundred thousandths of a second in a date and time value.	2009-06-15T13:45:30.6175400 -> 61754
		2009-06-15T13:45:30.0000050 -> (no output)

More information: [The "FFFFF" Custom Format Specifier](#).

"FFFFFF"	If non-zero, the millionths of a second in a date and time value.	2009-06-15T13:45:30.6175420 -> 617542
	More information: The "FFFFFF" Custom Format Specifier .	2009-06-15T13:45:30.0000005 -> (no output)
"FFFFFFF"	If non-zero, the ten millionths of a second in a date and time value.	2009-06-15T13:45:30.6175425 -> 6175425
	More information: The "FFFFFFF" Custom Format Specifier .	2009-06-15T13:45:30.0001150 -> 000115
"g", "gg"	The period or era.	2009-06-15T13:45:30.6170000 -> A.D.
	More information: The "g" or "gg" Custom Format Specifier .	
"h"	The hour, using a 12-hour clock from 1 to 12.	2009-06-15T01:45:30 -> 1
	More information: The "h" Custom Format Specifier .	2009-06-15T13:45:30 -> 1
"hh"	The hour, using a 12-hour clock from 01 to 12.	2009-06-15T01:45:30 -> 01
	More information: The "hh" Custom Format Specifier .	2009-06-15T13:45:30 -> 01
Format specifier	Description	Examples
"H"	The hour, using a 24-hour clock from 0 to 23.	2009-06-15T01:45:30 -> 1
	More information: The "H" Custom Format Specifier .	2009-06-15T13:45:30 -> 13

"HH"	The hour, using a 24-hour clock from 00 to 23.	2009-06-15T01:45:30 -> 01
	More information: The "HH" Custom Format Specifier.	2009-06-15T13:45:30 -> 13
"K"	Time zone information.	With DateTime values:
	More information: The "K" Custom Format Specifier.	2009-06-15T13:45:30, Kind Unspecified ->
		2009-06-15T13:45:30, Kind Utc -> Z
		2009-06-15T13:45:30, Kind Local -> -07:00 (depends on local computer settings)
		With DateTimeOffset values:
		2009-06-15T01:45:30-07:00 --> -07:00
"m"	The minute, from 0 through 59.	2009-06-15T01:09:30 -> 9
	More information: The "m" Custom Format Specifier.	2009-06-15T13:29:30 -> 29
Format specifier	Description	Examples
"mm"	The minute, from 00 through 59.	2009-06-15T01:09:30 -> 09
	More information: The "mm" Custom Format Specifier.	2009-06-15T01:45:30 -> 45

"M"	The month, from 1 through 12.	2009-06-15T13:45:30 -> 6
	More information: The "M" Custom Format Specifier.	
"MM"	The month, from 01 through 12.	2009-06-15T13:45:30 -> 06
	More information: The "MM" Custom Format Specifier.	
"MMM"	The abbreviated name of the month.	2009-06-15T13:45:30 -> Jun (en-US)
	More information: The "MMM" Custom Format Specifier.	2009-06-15T13:45:30 -> juin (fr-FR)
		2009-06-15T13:45:30 -> Jun (zu-ZA)
"MMMM"	The full name of the month.	2009-06-15T13:45:30 -> June (en-US)
	More information: The "MMMM" Custom Format Specifier.	2009-06-15T13:45:30 -> juni (da-DK)
		2009-06-15T13:45:30 -> uJuni (zu-ZA)
"s"	The second, from 0 through 59.	2009-06-15T13:45:09 -> 9
	More information: The "s" Custom Format Specifier.	
Format specifier	Description	Examples
"ss"	The second, from 00 through 59.	2009-06-15T13:45:09 -> 09
	More information: The "ss" Custom Format Specifier.	

"t"	The first character of the AM/PM designator.	2009-06-15T13:45:30 -> P (en-US)
	More information: The "t" Custom Format Specifier.	2009-06-15T13:45:30 -> 午 (ja-JP)
		2009-06-15T13:45:30 -> (fr-FR)
"tt"	The AM/PM designator.	2009-06-15T13:45:30 -> PM (en-US)
	More information: The "tt" Custom Format Specifier.	2009-06-15T13:45:30 -> 午後 (ja-JP)
		2009-06-15T13:45:30 -> (fr-FR)
"y"	The year, from 0 to 99.	0001-01-01T00:00:00 -> 1
	More information: The "y" Custom Format Specifier.	0900-01-01T00:00:00 -> 0
		1900-01-01T00:00:00 -> 0
		2009-06-15T13:45:30 -> 9
		2019-06-15T13:45:30 -> 19

Format specifier	Description	Examples
"yy"	The year, from 00 to 99.	0001-01-01T00:00:00 -> 01
	More information: The "yy" Custom Format Specifier.	0900-01-01T00:00:00 -> 00

1900-01-01T00:00:00 -> 00

2019-06-15T13:45:30 -> 19

"yyy"

The year, with a minimum of three digits.

0001-01-01T00:00:00 -> 001

More information: [The "yyy" Custom Format Specifier.](#)

0900-01-01T00:00:00 -> 900

1900-01-01T00:00:00 -> 1900

2009-06-15T13:45:30 -> 2009

"yyyy"

The year as a four-digit number.

0001-01-01T00:00:00 -> 0001

More information: [The "yyyy" Custom Format Specifier.](#)

0900-01-01T00:00:00 -> 0900

1900-01-01T00:00:00 -> 1900

2009-06-15T13:45:30 -> 2009

"yyyyy"

The year as a five-digit number.

0001-01-01T00:00:00 -> 00001

More information: [The "yyyyy" Custom Format Specifier.](#)

2009-06-15T13:45:30 -> 02009

**Format
specifier****Description****Examples****"z"**

Hours offset from UTC, with no leading zeros.

2009-06-15T13:45:30-07:00 -> -7

More information: [The "z" Custom Format Specifier.](#)

"zz"	Hours offset from UTC, with a leading zero for a single-digit value. More information: The "zz" Custom Format Specifier.	2009-06-15T13:45:30-07:00 -> -07
"zzz"	Hours and minutes offset from UTC. More information: The "zzz" Custom Format Specifier.	2009-06-15T13:45:30-07:00 -> -07:00
":"	The time separator. More information: The ":" Custom Format Specifier.	2009-06-15T13:45:30 -> : (en-US) 2009-06-15T13:45:30 -> . (it-IT) 2009-06-15T13:45:30 -> : (ja-JP)
"/"	The date separator. More Information: The "/" Custom Format Specifier.	2009-06-15T13:45:30 -> / (en-US) 2009-06-15T13:45:30 -> - (ar-DZ) 2009-06-15T13:45:30 -> . (tr-TR)
"string"	Literal string delimiter.	2009-06-15T13:45:30 ("arr:" h:m t) -> arr: 1:45 P
'string'	More information: Character literals.	2009-06-15T13:45:30 ('arr:' h:m t) -> arr: 1:45 P
Format specifier	Description	Examples
%	Defines the following character as a custom format specifier.	2009-06-15T13:45:30 (%h) -> 1

More information: [Using Single Custom Format Specifiers](#).

\	The escape character. More information: Character literals and Using the Escape Character .	2009-06-15T13:45:30 (h \h) -> 1 h
Any other character	The character is copied to the result string unchanged. More information: Character literals .	2009-06-15T01:45:30 (arr hh:mm t) -> arr 01:45 A

The following sections provide additional information about each custom date and time format specifier. Unless otherwise noted, each specifier produces an identical string representation regardless of whether it's used with a [DateTime](#) value or a [DateTimeOffset](#) value.

The "d" custom format specifier

The "d" custom format specifier represents the day of the month as a number from 1 through 31. A single-digit day is formatted without a leading zero.

If the "d" format specifier is used without other custom format specifiers, it's interpreted as the "d" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "d" custom format specifier in several format strings.

C#	 Copy
<pre>DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15); Console.WriteLine(date1.ToString("d, M", CultureInfo.InvariantCulture)); // Displays 29, 8</pre>	


```
Console.WriteLine(date1.ToString("d MMMM",  
    CultureInfo.CreateSpecificCulture("en-US")));  
// Displays 29 August  
Console.WriteLine(date1.ToString("d MMMM",  
    CultureInfo.CreateSpecificCulture("es-MX")));  
// Displays 29 agosto
```

[Back to table](#)

The "dd" custom format specifier

The "dd" custom format string represents the day of the month as a number from 01 through 31. A single-digit day is formatted with a leading zero.

The following example includes the "dd" custom format specifier in a custom format string.

C#	 Copy
<pre>DateTime date1 = new DateTime(2008, 1, 2, 6, 30, 15); Console.WriteLine(date1.ToString("dd, MM", CultureInfo.InvariantCulture)); // 02, 01</pre>	

[Back to table](#)

The "ddd" custom format specifier

The "ddd" custom format specifier represents the abbreviated name of the day of the week. The localized abbreviated name of the day of the week is retrieved from the [DateTimeFormatInfo.AbbreviatedDayNames](#) property of the current or specified culture.

of the week is retrieved from the [DateTimeFormatInfo.AbbreviatedDayNames](#) property of the current or specified culture.

The following example includes the "ddd" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("ddd d MMM",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays ven. 29 août
```

[Back to table](#)

The "dddd" custom format specifier

The "dddd" custom format specifier (plus any number of additional "d" specifiers) represents the full name of the day of the week. The localized name of the day of the week is retrieved from the [DateTimeFormatInfo.DayNames](#) property of the current or specified culture.

The following example includes the "dddd" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("dddd dd MMMM",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Friday 29 August
```



```
Console.WriteLine(date1.ToString("dddd dd MMMM",  
                                CultureInfo.CreateSpecificCulture("it-IT")));  
// Displays venerdì 29 agosto
```

[Back to table](#)

The "f" custom format specifier

The "f" custom format specifier represents the most significant digit of the seconds fraction; that is, it represents the tenths of a second in a date and time value.

If the "f" format specifier is used without other format specifiers, it's interpreted as the "f" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

When you use "f" format specifiers as part of a format string supplied to the [ParseExact](#), [TryParseExact](#), [ParseExact](#), or [TryParseExact](#) method, the number of "f" format specifiers indicates the number of most significant digits of the seconds fraction that must be present to successfully parse the string.

The following example includes the "f" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);  
CultureInfo ci = CultureInfo.InvariantCulture;  
  
Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));  
// Displays 07:27:15.0  
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));  
// Displays 07:27:15  
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));  
// Displays 07:27:15.01
```

```
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));  
// Displays 07:27:15.018  
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));  
// Displays 07:27:15.018
```

[Back to table](#)

The "ff" custom format specifier

The "ff" custom format specifier represents the two most significant digits of the seconds fraction; that is, it represents the hundredths of a second in a date and time value.

following example includes the "ff" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);  
CultureInfo ci = CultureInfo.InvariantCulture;  
  
Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));  
// Displays 07:27:15.0  
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));  
// Displays 07:27:15  
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));  
// Displays 07:27:15.018  
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));  
// Displays 07:27:15.018
```

[Back to table](#)

The "fff" custom format specifier

The "fff" custom format specifier represents the three most significant digits of the seconds fraction; that is, it represents the milliseconds in a date and time value.

The following example includes the "fff" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018
```

[Back to table](#)

The "ffff" custom format specifier

The "ffff" custom format specifier represents the four most significant digits of the seconds fraction; that is, it represents the ten thousandths of a second in a date and time value.

thousandths of a second in a date and time value.

Although it's possible to display the ten thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT version 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "fffff" custom format specifier

The "fffff" custom format specifier represents the five most significant digits of the seconds fraction; that is, it represents the hundred thousandths of a second in a date and time value.

Although it's possible to display the hundred thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "ffffff" custom format specifier

The "ffffff" custom format specifier represents the six most significant digits of the seconds fraction; that is, it represents the millionths of a second in a date and time value.

Although it's possible to display the millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "ffffff" custom format specifier

The "ffffff" custom format specifier represents the seven most significant digits of the seconds fraction; that is, it represents the ten millionths of a second in a date and time value.

Although it's possible to display the ten millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "F" custom format specifier

The "F" custom format specifier represents the most significant digit of the seconds fraction; that is, it represents the tenths of a second in a date and time value. Nothing is displayed if the digit is zero.

If the "F" format specifier is used without other format specifiers, it's interpreted as the "F" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The number of "F" format specifiers used with the [ParseExact](#), [TryParseExact](#), [ParseExact](#), or [TryParseExact](#) method indicates the maximum number of most significant digits of the seconds fraction that can be present to successfully parse the string.

The following example includes the "F" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
```

```
Console.WriteLine(date1.ToString("nn:mm:ss.F", ci));  
// Displays 07:27:15  
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));  
// Displays 07:27:15.018  
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));  
// Displays 07:27:15.018
```

[Back to table](#)

The "FF" custom format specifier

The "FF" custom format specifier represents the two most significant digits of the seconds fraction; that is, it represents the hundredths of a second in a date and time value. However, trailing zeros or two zero digits aren't displayed.

The following example includes the "FF" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);  
CultureInfo ci = CultureInfo.InvariantCulture;  
  
Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));  
  
// Displays 07:27:15.0  
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));  
// Displays 07:27:15  
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));  
// Displays 07:27:15.01  
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
```

```
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018
```

[Back to table](#)

The "FFF" custom format specifier

The "FFF" custom format specifier represents the three most significant digits of the seconds fraction; that is, it represents the milliseconds in a date and time value. However, trailing zeros or three zero digits aren't displayed.

The following example includes the "FFF" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15, 18);
CultureInfo ci = CultureInfo.InvariantCulture;

Console.WriteLine(date1.ToString("hh:mm:ss.f", ci));
// Displays 07:27:15.0
Console.WriteLine(date1.ToString("hh:mm:ss.F", ci));
// Displays 07:27:15
Console.WriteLine(date1.ToString("hh:mm:ss.ff", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.FF", ci));
// Displays 07:27:15.01
Console.WriteLine(date1.ToString("hh:mm:ss.fff", ci));
// Displays 07:27:15.018
Console.WriteLine(date1.ToString("hh:mm:ss.FFF", ci));
// Displays 07:27:15.018
```

[Back to table](#)

The "FFFF" custom format specifier

The "FFFF" custom format specifier represents the four most significant digits of the seconds fraction; that is, it represents the ten thousandths of a second in a date and time value. However, trailing zeros or four zero digits aren't displayed.

Although it's possible to display the ten thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFF" custom format specifier

The "FFFFF" custom format specifier represents the five most significant digits of the seconds fraction; that is, it represents the hundred thousandths of a second in a date and time value. However, trailing zeros or five zero digits aren't displayed.

Although it's possible to display the hundred thousandths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFFF" custom format specifier

The "FFFFFF" custom format specifier represents the six most significant digits of the seconds fraction; that is, it represents the millionths of a second in a date and time value. However, trailing zeros or six zero digits aren't displayed.

Although it's possible to display the millionths of a second component of a time value, that value may not be meaningful. The

precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "FFFFFFF" custom format specifier

The "FFFFFFF" custom format specifier represents the seven most significant digits of the seconds fraction; that is, it represents the ten millionths of a second in a date and time value. However, trailing zeros or seven zero digits aren't displayed.

Although it's possible to display the ten millionths of a second component of a time value, that value may not be meaningful. The precision of date and time values depends on the resolution of the system clock. On the Windows NT 3.5 (and later) and Windows Vista operating systems, the clock's resolution is approximately 10-15 milliseconds.

[Back to table](#)

The "g" or "gg" custom format specifier

The "g" or "gg" custom format specifiers (plus any number of additional "g" specifiers) represents the period or era, such as A.D. The formatting operation ignores this specifier if the date to be formatted doesn't have an associated period or era string.

If the "g" format specifier is used without other custom format specifiers, it's interpreted as the "g" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "g" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(70, 08, 04);  
  
Console.WriteLine(date1.ToString("MM/dd/yyyy g",
```

```
CultureInfo.InvariantCulture));  
// Displays 08/04/0070 A.D.  
Console.WriteLine(date1.ToString("MM/dd/yyyy g",  
    CultureInfo.CreateSpecificCulture("fr-FR")));  
// Displays 08/04/0070 ap. J.-C.
```

[Back to table](#)

The "h" custom format specifier

The "h" custom format specifier represents the hour as a number from 1 through 12; that is, the hour is represented by a 12-hour clock that counts the whole hours since midnight or noon. A particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted without a leading zero. For example, given a time of 5:43 in the morning or afternoon, this custom format specifier displays "5".

If the "h" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "h" custom format specifier in a custom format string.

C#



```
DateTime date1;  
date1 = new DateTime(2008, 1, 1, 18, 9, 1);  
Console.WriteLine(date1.ToString("h:m:s.F t",  
    CultureInfo.InvariantCulture));  
// Displays 6:9:1 P  
Console.WriteLine(date1.ToString("h:m:s.F t",  
    CultureInfo.CreateSpecificCulture("el-GR")));
```

```
// Displays 6:9:1 µ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 µ
```

[Back to table](#)

The "hh" custom format specifier

The "hh" custom format specifier (plus any number of additional "h" specifiers) represents the hour as a number from 01 through 12; that is, the hour is represented by a 12-hour clock that counts the whole hours since midnight or noon. A particular hour after midnight is indistinguishable from the same hour after noon. The hour is not rounded, and a single-digit hour is formatted with a leading zero. For example, given a time of 5:43 in the morning or afternoon, this format specifier displays "05".

The following example includes the "hh" custom format specifier in a custom format string.

```
C#
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.InvariantCulture));
```

 Copy

```
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.
```

[Back to table](#)

The "H" custom format specifier

The "H" custom format specifier represents the hour as a number from 0 through 23; that is, the hour is represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted without a leading zero.

If the "H" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "H" custom format specifier in a custom format string.

```
C#
DateTime date1 = new DateTime(2008, 1, 1, 6, 9, 1);
Console.WriteLine(date1.ToString("H:mm:ss",
    CultureInfo.InvariantCulture));
// Displays 6:09:01
```

 Copy

[Back to table](#)

The "HH" custom format specifier

The "HH" custom format specifier (plus any number of additional "H" specifiers) represents the hour as a number from 00 through 23; that is, the hour is represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted

that is, the hour is represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted with a leading zero.

The following example includes the "HH" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 1, 1, 6, 9, 1);
Console.WriteLine(date1.ToString("HH:mm:ss",
    CultureInfo.InvariantCulture));
// Displays 06:09:01
```

[Back to table](#)

The "K" custom format specifier

The "K" custom format specifier represents the time zone information of a date and time value. When this format specifier is used with [DateTime](#) values, the result string is defined by the value of the [DateTime.Kind](#) property:

- For the local time zone (a [DateTime.Kind](#) property value of [DateTimeKind.Local](#)), this specifier is equivalent to the "zzz" specifier and produces a result string containing the local offset from Coordinated Universal Time (UTC); for example, "-07:00".
- For a UTC time (a [DateTime.Kind](#) property value of [DateTimeKind.Utc](#)), the result string includes a "Z" character to represent a UTC date.
- For a time from an unspecified time zone (a time whose [DateTime.Kind](#) property equals [DateTimeKind.Unspecified](#)), the result is equivalent to [String.Empty](#).

For [DateTimeOffset](#) values, the "K" format specifier is equivalent to the "zzz" format specifier, and produces a result string containing the [DateTimeOffset](#) value's offset from UTC.

If the "K" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example displays the string that results from using the "K" custom format specifier with various [DateTime](#) and [DateTimeOffset](#) values on a system in the U.S. Pacific Time zone.

C#

 Copy Run

```
Console.WriteLine(DateTime.Now.ToString("%K"));
// Displays -07:00
Console.WriteLine(DateTime.UtcNow.ToString("%K"));
// Displays Z
Console.WriteLine("{0}",
    DateTime.SpecifyKind(DateTime.Now,
        DateTimeKind.Unspecified).ToString("%K"));
// Displays ''
Console.WriteLine(DateTimeOffset.Now.ToString("%K"));
// Displays -07:00
Console.WriteLine(DateTimeOffset.UtcNow.ToString("%K"));
// Displays +00:00
Console.WriteLine(new DateTimeOffset(2008, 5, 1, 6, 30, 0,
    new TimeSpan(5, 0, 0)).ToString("%K"));
// Displays +05:00
```

[Back to table](#)

The "m" custom format specifier

The "m" custom format specifier represents the minute as a number from 0 through 59. The minute represents whole minutes that have passed since the last hour. A single-digit minute is formatted without a leading zero.

If the "m" format specifier is used without other custom format specifiers, it's interpreted as the "m" standard date and time format

specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "m" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ
```

[Back to table](#)

The "mm" custom format specifier

The "mm" custom format specifier (plus any number of additional "m" specifiers) represents the minute as a number from 00 through 59. The minute represents whole minutes that have passed since the last hour. A single-digit minute is formatted with a leading zero.

The following example includes the "mm" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1;  
date1 = new DateTime(2008, 1, 1, 18, 9, 1);  
Console.WriteLine(date1.ToString("hh:mm:ss tt",  
    CultureInfo.InvariantCulture));  
// Displays 06:09:01 PM  
Console.WriteLine(date1.ToString("hh:mm:ss tt",  
    CultureInfo.CreateSpecificCulture("hu-HU")));  
// Displays 06:09:01 du.  
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);  
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",  
    CultureInfo.InvariantCulture));  
// Displays 06:09:01.50 PM  
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",  
    CultureInfo.CreateSpecificCulture("hu-HU")));  
// Displays 06:09:01.50 du.
```


[Back to table](#)

The "M" custom format specifier

The "M" custom format specifier represents the month as a number from 1 through 12 (or from 1 through 13 for calendars that have 13 months). A single-digit month is formatted without a leading zero.

If the "M" format specifier is used without other custom format specifiers, it's interpreted as the "M" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "M" custom format specifier in a custom format string.

C#	 Copy
<pre>DateTime date1 = new DateTime(2008, 8, 18); Console.WriteLine(date1.ToString("(M) MMM, MMMM", CultureInfo.CreateSpecificCulture("en-US")));</pre>	


```
// Displays (8) Aug, August
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("nl-NL")));

// Displays (8) aug, augustus
Console.WriteLine(date1.ToString("(M) MMM, MMMM",
    CultureInfo.CreateSpecificCulture("lv-LV")));

// Displays (8) Aug, augusts
```

[Back to table](#)

The "MM" custom format specifier

The "MM" custom format specifier represents the month as a number from 01 through 12 (or from 1 through 13 for calendars that have 13 months). A single-digit month is formatted with a leading zero.

The following example includes the "MM" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1 = new DateTime(2008, 1, 2, 6, 30, 15);

Console.WriteLine(date1.ToString("dd, MM",
    CultureInfo.InvariantCulture));

// 02, 01
```

[Back to table](#)

The "MMM" custom format specifier

The "MMM" custom format specifier represents the abbreviated name of the month. The localized abbreviated name of the month is retrieved from the [DateTimeFormatInfo.AbbreviatedMonthNames](#) property of the current or specified culture.

retrieved from the [DateTimeFormatInfo.AbbreviatedMonthNames](#) property of the current or specified culture.

The following example includes the "MMM" custom format specifier in a custom format string.

C#



```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("ddd d MMM",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Fri 29 Aug
Console.WriteLine(date1.ToString("ddd d MMM",
    CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays ven. 29 août
```

[Back to table](#)

The "MMMM" custom format specifier

The "MMMM" custom format specifier represents the full name of the month. The localized name of the month is retrieved from the [DateTimeFormatInfo.MonthNames](#) property of the current or specified culture.

The following example includes the "MMMM" custom format specifier in a custom format string.

C#



```
DateTime date1 = new DateTime(2008, 8, 29, 19, 27, 15);

Console.WriteLine(date1.ToString("dddd dd MMMM",
    CultureInfo.CreateSpecificCulture("en-US")));
// Displays Friday 29 August
Console.WriteLine(date1.ToString("dddd dd MMMM",
    CultureInfo.CreateSpecificCulture("it-IT")));
```

```
// Displays venerdì 29 agosto
```

[Back to table](#)

The "s" custom format specifier

The "s" custom format specifier represents the seconds as a number from 0 through 59. The result represents whole seconds that have passed since the last minute. A single-digit second is formatted without a leading zero.

If the "s" format specifier is used without other custom format specifiers, it's interpreted as the "s" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "s" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
                                CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                                CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);

Console.WriteLine(date1.ToString("h:m:s.F t",
                                CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
                                CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ
```

[Back to table](#)

The "ss" custom format specifier

The "ss" custom format specifier (plus any number of additional "s" specifiers) represents the seconds as a number from 00 through 59. The result represents whole seconds that have passed since the last minute. A single-digit second is formatted with a leading zero.

The following example includes the "ss" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.
```

[Back to table](#)

The "t" custom format specifier

The "t" custom format specifier represents the first character of the AM/PM designator. The appropriate localized designator is retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#) property of the current or specific

retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#) property of the current or specific culture. The AM designator is used for all times from 0:00:00 (midnight) to 11:59:59.999. The PM designator is used for all times from 12:00:00 (noon) to 23:59:59.999.

If the "t" format specifier is used without other custom format specifiers, it's interpreted as the "t" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "t" custom format specifier in a custom format string.

C#



```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1 μ
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.InvariantCulture));
// Displays 6:9:1.5 P
Console.WriteLine(date1.ToString("h:m:s.F t",
    CultureInfo.CreateSpecificCulture("el-GR")));
// Displays 6:9:1.5 μ
```

[Back to table](#)

The "tt" custom format specifier

The "tt" custom format specifier (plus any number of additional "t" specifiers) represents the entire AM/PM designator. The appropriate localized designator is retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#).

appropriate localized designator is retrieved from the [DateTimeFormatInfo.AMDesignator](#) or [DateTimeFormatInfo.PMDesignator](#) property of the current or specific culture. The AM designator is used for all times from 0:00:00 (midnight) to 11:59:59.999. The PM designator is used for all times from 12:00:00 (noon) to 23:59:59.999.

Make sure to use the "tt" specifier for languages for which it's necessary to maintain the distinction between AM and PM. An example is Japanese, for which the AM and PM designators differ in the second character instead of the first character.

The following example includes the "tt" custom format specifier in a custom format string.

C#

 Copy

```
DateTime date1;
date1 = new DateTime(2008, 1, 1, 18, 9, 1);
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01 PM
Console.WriteLine(date1.ToString("hh:mm:ss tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01 du.
date1 = new DateTime(2008, 1, 1, 18, 9, 1, 500);
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.InvariantCulture));
// Displays 06:09:01.50 PM
Console.WriteLine(date1.ToString("hh:mm:ss.ff tt",
    CultureInfo.CreateSpecificCulture("hu-HU")));
// Displays 06:09:01.50 du.
```

[Back to table](#)

The "y" custom format specifier

The "y" custom format specifier represents the year as a one-digit or two-digit number. If the year has more than two digits, only the two low-order digits appear in the result. If the first digit of a two-digit year begins with a zero (for example, 2009), the number is

two low-order digits appear in the result. If the first digit of a two-digit year begins with a zero (for example, 2000), the number is formatted without a leading zero.

If the "y" format specifier is used without other custom format specifiers, it's interpreted as the "y" standard date and time format specifier. For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "y" custom format specifier in a custom format string.

C#

 Copy Run

```
DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010
```

[Back to table](#)

The "yy" custom format specifier

The "yy" custom format specifier represents the year as a two-digit number. If the year has more than two digits, only the two low-order digits appear in the result. If the two-digit year has fewer than two significant digits, the number is padded with leading zeros to produce two digits.

In a parsing operation, a two-digit year that is parsed using the "yy" custom format specifier is interpreted based on the [Calendar.TwoDigitYearMax](#) property of the format provider's current calendar. The following example parses the string representation of a date that has a two-digit year by using the default Gregorian calendar of the en-US culture, which, in this case, is the current culture. It then changes the current culture's [CultureInfo](#) object to use a [GregorianCalendar](#) object whose [TwoDigitYearMax](#) property has been modified.

C#

 Copy Run

```
using System;
using System.Globalization;
using System.Threading;

public class Example
{
    public static void Main()
    {
        string fmt = "dd-MMM-yy";
        string value = "24-Jan-49";

        Calendar cal = (Calendar) CultureInfo.CurrentCulture.Calendar.Clone();
        Console.WriteLine("Two Digit Year Range: {0} - {1}",
            cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax);

        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, null));
        Console.WriteLine();

        cal.TwoDigitYearMax = 2099;
        CultureInfo culture = (CultureInfo) CultureInfo.CurrentCulture.Clone();
```



```
        culture.DateTimeFormat.Calendar = cal;
        Thread.CurrentThread.CurrentCulture = culture;

        Console.WriteLine("Two Digit Year Range: {0} - {1}",
                           cal.TwoDigitYearMax - 99, cal.TwoDigitYearMax);
        Console.WriteLine("{0:d}", DateTime.ParseExact(value, fmt, null));
    }
}

// The example displays the following output:
//      Two Digit Year Range: 1930 - 2029
//      1/24/1949
//
//      Two Digit Year Range: 2000 - 2099
//      1/24/2049
```

The following example includes the "yy" custom format specifier in a custom format string.

C#

 Copy

 Run

```
DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("vvv"));
```

```
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010
```

[Back to table](#)

The "yyy" custom format specifier

The "yyy" custom format specifier represents the year with a minimum of three digits. If the year has more than three significant digits, they are included in the result string. If the year has fewer than three digits, the number is padded with leading zeros to produce three digits.

ⓘ Note

For the Thai Buddhist calendar, which can have five-digit years, this format specifier displays all significant digits.

The following example includes the "yyy" custom format specifier in a custom format string.

C#

 Copy

 Run

```
DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
```

```
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010
```

[Back to table](#)

The "yyyy" custom format specifier

The "yyyy" custom format specifier represents the year with a minimum of four digits. If the year has more than four significant digits, they are included in the result string. If the year has fewer than four digits, the number is padded with leading zeros to produce four digits.

ⓘ Note

For the Thai Buddhist calendar, which can have five-digit years, this format specifier displays a minimum of four digits.

The following example includes the "yyyy" custom format specifier in a custom format string.

C#

 Copy

 Run

```
DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010
```

[Back to table](#)

The "yyyyy" custom format specifier

The "yyyyy" custom format specifier (plus any number of additional "y" specifiers) represents the year with a minimum of five digits. If the year has more than five significant digits, they are included in the result string. If the year has fewer than five digits, the number is padded with leading zeros to produce five digits.

If there are additional "y" specifiers, the number is padded with as many leading zeros as necessary to produce the number of "y" specifiers.

The following example includes the "yyyy" custom format specifier in a custom format string.

C#

 Copy Run

```
DateTime date1 = new DateTime(1, 12, 1);
DateTime date2 = new DateTime(2010, 1, 1);
Console.WriteLine(date1.ToString("%y"));
// Displays 1
Console.WriteLine(date1.ToString("yy"));
// Displays 01
Console.WriteLine(date1.ToString("yyy"));
// Displays 001
Console.WriteLine(date1.ToString("yyyy"));
// Displays 0001
Console.WriteLine(date1.ToString("yyyyy"));
// Displays 00001
Console.WriteLine(date2.ToString("%y"));
// Displays 10
Console.WriteLine(date2.ToString("yy"));
// Displays 10
Console.WriteLine(date2.ToString("yyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyy"));
// Displays 2010
Console.WriteLine(date2.ToString("yyyyy"));
// Displays 02010
```

[Back to table](#)

The "z" custom format specifier

With [DateTime](#) values, the "z" custom format specifier represents the signed offset of the local operating system's time zone from Coordinated Universal Time (UTC), measured in hours. It doesn't reflect the value of an instance's [DateTime.Kind](#) property. For this



Coordinated Universal Time (UTC), measured in hours. It doesn't reflect the value of an instance's [DateTime.Kind](#) property. For this reason, the "z" format specifier is not recommended for use with [DateTime](#) values.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours.

The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours behind UTC. A single-digit offset is formatted without a leading zero.

If the "z" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

The following example includes the "z" custom format specifier in a custom format string.

C#	 Copy	 Run
<pre>DateTime date1 = DateTime.UtcNow; Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date1)); // Displays -7, -07, -07:00 DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0, new TimeSpan(6, 0, 0)); Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date2)); // Displays +6, +06, +06:00</pre>		

[Back to table](#)

The "zz" custom format specifier

With [DateTime](#) values, the "zz" custom format specifier represents the signed offset of the local operating system's time zone from



UTC, measured in hours. It doesn't reflect the value of an instance's [DateTime.Kind](#) property. For this reason, the "zz" format specifier is

UTC, measured in hours. It doesn't reflect the value of an instance's [DateTime.Kind](#) property. For this reason, the "zz" format specifier is not recommended for use with [DateTime](#) values.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours.

The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours behind UTC. A single-digit offset is formatted with a leading zero.

The following example includes the "zz" custom format specifier in a custom format string.

C#	 Copy	 Run
<pre>DateTime date1 = DateTime.UtcNow; Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date1)); // Displays -7, -07, -07:00 DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0, new TimeSpan(6, 0, 0)); Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date2)); // Displays +6, +06, +06:00</pre>		

[Back to table](#)

The "zzz" custom format specifier



With [DateTime](#) values, the "zzz" custom format specifier represents the signed offset of the local operating system's time zone from UTC, measured in hours and minutes. It doesn't reflect the value of an instance's [DateTime.Kind](#) property. For this reason, the "zzz" format specifier is not recommended for use with [DateTime](#) values.

With [DateTimeOffset](#) values, this format specifier represents the [DateTimeOffset](#) value's offset from UTC in hours and minutes.

The offset is always displayed with a leading sign. A plus sign (+) indicates hours ahead of UTC, and a minus sign (-) indicates hours

The offset is always displayed with a leading sign. A plus sign (`+`) indicates hours ahead of UTC, and a minus sign (`-`) indicates hours behind UTC. A single-digit offset is formatted with a leading zero.

The following example includes the "zzz" custom format specifier in a custom format string.

C#	 Copy	 Run
<pre>DateTime date1 = DateTime.UtcNow; Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date1)); // Displays -7, -07, -07:00 DateTimeOffset date2 = new DateTimeOffset(2008, 8, 1, 0, 0, 0, new TimeSpan(6, 0, 0)); Console.WriteLine(String.Format("{0:%z}, {0:zz}, {0:zzz}", date2)); // Displays +6, +06, +06:00</pre>		

[Back to table](#)

The ":" custom format specifier

The ":" custom format specifier represents the time separator, which is used to differentiate hours, minutes, and seconds. The appropriate localized time separator is retrieved from the [DateTimeFormatInfo.TimeSeparator](#) property of the current or specified culture.

ⓘ Note

To change the time separator for a particular date and time string, specify the separator character within a literal string delimiter. For example, the custom format string `hh'_'dd'_'ss` produces a result string in which "_" (an underscore) is always used as the time separator. To change the time separator for all dates for a culture, either change the value of the [DateTimeFormatInfo.TimeSeparator](#) property of the current culture, or instantiate a [DateTimeFormatInfo](#) object, assign the

[DateTimeFormatInfo.DateSeparator](#) property of the current culture, or instantiate a [DateTimeFormatInfo](#) object, assign the character to its [TimeSeparator](#) property, and call an overload of the formatting method that includes an [IFormatProvider](#) parameter.

If the ":" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

[Back to table](#)

The "/" custom format specifier

The "/" custom format specifier represents the date separator, which is used to differentiate years, months, and days. The appropriate localized date separator is retrieved from the [DateTimeFormatInfo.DateSeparator](#) property of the current or specified culture.

ⓘ Note

To change the date separator for a particular date and time string, specify the separator character within a literal string delimiter. For example, the custom format string `mm'/'dd'/'yyyy` produces a result string in which "/" is always used as the date separator. To change the date separator for all dates for a culture, either change the value of the [DateTimeFormatInfo.DateSeparator](#) property of the current culture, or instantiate a [DateTimeFormatInfo](#) object, assign the character to its [DateSeparator](#) property, and call an overload of the formatting method that includes an [IFormatProvider](#) parameter.

If the "/" format specifier is used without other custom format specifiers, it's interpreted as a standard date and time format specifier and throws a [FormatException](#). For more information about using a single format specifier, see [Using Single Custom Format Specifiers](#) later in this article.

[Back to table](#)

Character literals

The following characters in a custom date and time format string are reserved and are always interpreted as formatting characters or, in the case of ", ', /, and \, as special characters.

F	H	K	M	d
f	g	h	m	s
t	y	z	%	:
/	"	'	\	

All other characters are always interpreted as character literals and, in a formatting operation, are included in the result string unchanged. In a parsing operation, they must match the characters in the input string exactly; the comparison is case-sensitive.

The following example includes the literal characters "PST" (for Pacific Standard Time) and "PDT" (for Pacific Daylight Time) to represent the local time zone in a format string. Note that the string is included in the result string, and that a string that includes the local time zone string also parses successfully.

C#

 Copy

 Run

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
```

```
String[] formats = { "dd MMM yyyy hh:mm tt PST",  
                    "dd MMM yyyy hh:mm tt PDT" };  
var dat = new DateTime(2016, 8, 18, 16, 50, 0);  
// Display the result string.  
Console.WriteLine(dat.ToString(formats[1]));  
  
// Parse a string.  
String value = "25 Dec 2016 12:00 pm PST";  
DateTime newDate;  
if (DateTime.TryParseExact(value, formats, null,  
                          DateTimeStyles.None, out newDate))  
    Console.WriteLine(newDate);  
else  
    Console.WriteLine("Unable to parse '{0}'", value);  
}  
}  
// The example displays the following output:  
//      18 Aug 2016 04:50 PM PDT  
//      12/25/2016 12:00:00 PM
```

There are two ways to indicate that characters are to be interpreted as literal characters and not as reserve characters, so that they can be included in a result string or successfully parsed in an input string:

- By escaping each reserved character. For more information, see [Using the Escape Character](#).

The following example includes the literal characters "pst" (for Pacific Standard time) to represent the local time zone in a format string. Because both "s" and "t" are custom format strings, both characters must be escaped to be interpreted as character literals.

C#

 Copy Run

```
using System;  
using System.Globalization;  
  
public class Example  
{
```

```
public static void Main()
{
    String format = "dd MMM yyyy hh:mm tt p\\s\\t";
    var dat = new DateTime(2016, 8, 18, 16, 50, 0);
    // Display the result string.
    Console.WriteLine(dat.ToString(format));

    // Parse a string.
    String value = "25 Dec 2016 12:00 pm pst";
    DateTime newDate;
    if (DateTime.TryParseExact(value, format, null,
                              DateTimeStyles.None, out newDate))
        Console.WriteLine(newDate);
    else
        Console.WriteLine("Unable to parse '{0}'", value);
}
// The example displays the following output:
//      18 Aug 2016 04:50 PM PDT
//      12/25/2016 12:00:00 PM
```

- By enclosing the entire literal string in quotation marks or apostrophes. The following example is like the previous one, except that "pst" is enclosed in quotation marks to indicate that the entire delimited string should be interpreted as character literals.

C#

 Copy Run

```
using System;
using System.Globalization;

public class Example
{
    public static void Main()
    {
        String format = "dd MMM yyyy hh:mm tt \"pst\"";
        var dat = new DateTime(2016, 8, 18, 16, 50, 0);
        // Display the result string.
```

```
// Parse a string.
Console.WriteLine(dat.ToString(format));

// Parse a string.
String value = "25 Dec 2016 12:00 pm pst";
DateTime newDate;
if (DateTime.TryParseExact(value, format, null,
                           DateTimeStyles.None, out newDate))
    Console.WriteLine(newDate);
else
    Console.WriteLine("Unable to parse '{0}'", value);
}
}

// The example displays the following output:
//      18 Aug 2016 04:50 PM PDT
//      12/25/2016 12:00:00 PM
```

Notes

Using single custom format specifiers

A custom date and time format string consists of two or more characters. Date and time formatting methods interpret any single-character string as a standard date and time format string. If they don't recognize the character as a valid format specifier, they throw a [FormatException](#). For example, a format string that consists only of the specifier "h" is interpreted as a standard date and time format string. However, in this particular case, an exception is thrown because there is no "h" standard date and timeformat specifier.

To use any of the custom date and time format specifiers as the only specifier in a format string (that is, to use the "d", "f", "F", "g", "h", "H", "K", "m", "M", "s", "t", "y", "z", ":", or "/" custom format specifier by itself), include a space before or after the specifier, or include a percent ("%") format specifier before the single custom date and time specifier.

For example, "%h" is interpreted as a custom date and time format string that displays the hour represented by the current date and time value. You can also use the " h" or "h " format string, although this includes a space in the result string along with the hour. The

following example illustrates these three format strings.

C#

 Copy Run

```
DateTime dat1 = new DateTime(2009, 6, 15, 13, 45, 0);

Console.WriteLine("{0:%h}", dat1);
Console.WriteLine("{0: h}", dat1);
Console.WriteLine("{0:h }", dat1);
// The example displays the following output:
//      '1'
//      ' 1'
//      '1 '
```

Using the Escape Character

The "d", "f", "F", "g", "h", "H", "K", "m", "M", "s", "t", "y", "z", ":", or "/" characters in a format string are interpreted as custom format specifiers rather than as literal characters. To prevent a character from being interpreted as a format specifier, you can precede it with a backslash (\), which is the escape character. The escape character signifies that the following character is a character literal that should be included in the result string unchanged.

To include a backslash in a result string, you must escape it with another backslash (\\).

Note

Some compilers, such as the C++ and C# compilers, may also interpret a single backslash character as an escape character. To ensure that a string is interpreted correctly when formatting, you can use the verbatim string literal character (the @ character) before the string in C#, or add another backslash character before each backslash in C# and C++. The following C# example illustrates both approaches.

The following example uses the escape character to prevent the formatting operation from interpreting the `h` and `m` characters as format specifiers.

C#

 Copy Run

```
DateTime date = new DateTime(2009, 06, 15, 13, 45, 30, 90);
string fmt1 = "h \\h m \\m";
string fmt2 = @"h \h m \m";

Console.WriteLine("{0} ({1}) -> {2}", date, fmt1, date.ToString(fmt1));
Console.WriteLine("{0} ({1}) -> {2}", date, fmt2, date.ToString(fmt2));
// The example displays the following output:
//      6/15/2009 1:45:30 PM (h \h m \m) -> 1 h 45 m
//      6/15/2009 1:45:30 PM (h \h m \m) -> 1 h 45 m
```

Control Panel settings

The **Regional and Language Options** settings in Control Panel influence the result string produced by a formatting operation that includes many of the custom date and time format specifiers. These settings are used to initialize the [DateTimeFormatInfo](#) object associated with the current thread culture, which provides values used to govern formatting. Computers that use different settings generate different result strings.

In addition, if you use the [CultureInfo.CultureInfo\(String\)](#) constructor to instantiate a new [CultureInfo](#) object that represents the same culture as the current system culture, any customizations established by the **Regional and Language Options** item in Control Panel will

be applied to the new [CultureInfo](#) object. You can use the [CultureInfo.CultureInfo\(String, Boolean\)](#) constructor to create a [CultureInfo](#) object that doesn't reflect a system's customizations.

DateTimeFormatInfo properties

Formatting is influenced by properties of the current [DateTimeFormatInfo](#) object, which is provided implicitly by the current thread

culture or explicitly by the [IFormatProvider](#) parameter of the method that invokes formatting. For the [IFormatProvider](#) parameter, you should specify a [CultureInfo](#) object, which represents a culture, or a [DateTimeFormatInfo](#) object.

The result string produced by many of the custom date and time format specifiers also depends on properties of the current [DateTimeFormatInfo](#) object. Your application can change the result produced by some custom date and time format specifiers by changing the corresponding [DateTimeFormatInfo](#) property. For example, the "ddd" format specifier adds an abbreviated weekday name found in the [AbbreviatedDayNames](#) string array to the result string. Similarly, the "MMMM" format specifier adds a full month name found in the [MonthNames](#) string array to the result string.

See also

- [System.DateTime](#)
- [System.IFormatProvider](#)
- [Formatting Types](#)
- [Standard Date and Time Format Strings](#)
- [Sample: .NET Framework 4 Formatting Utility](#)