Standard Numeric Format Strings [Visual Basic]

Standard numeric format strings are used to return common numeric string types. A standard format string takes the form *Axx* where *A* is an alphabetic character called the format specifier, and *xx* is a sequence of digits called the precision specifier. The format specifier must be one of the built-in format characters that define the most commonly used numeric format types. The precision specifier controls the number of significant digits or zeros to the right of a decimal.

All numeric format strings that consist of a single alphabetic character (without a leading or following white space), optionally followed by a digit from 0 to 99, are interpreted as standard numeric format strings. If a string is interpreted as a standard numeric format string and contains one of the standard numeric format specifiers, then the numeric value is formatted accordingly. However, if a string is interpreted as a standard format string but does not contain one of the standard format specifiers, then a FormatException is thrown. For example, the format string, "z", is interpreted as a standard numeric format string because it contains one alphabetic character, but this string causes a **FormatException** to be thrown because it is not one of the standard numeric format specifiers. Any numeric format string that does not fit the definition of a standard numeric format string is interpreted as a custom numeric format string. The format string "c!" is interpreted as a custom format string because it contains two alphabetic characters, even though the character "c" is a standard numeric format specifier.

The following table describes the standard numeric format strings. Note that the patterns produced by these format specifiers are influenced by the settings in the Regional Options control panel. Computers using different cultures or different currency settings will display different patterns.

Format specifer	Name	Description
C or c	Currency	The number is converted to a string that represents a currency amount. The conversion is controlled by the currency format information of the NumberFormatInfo object used to format the number. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default currency precision given by the **NumberFormatInfo** is used.
D or d	Decimal	This format is supported for integral types only. The number is converted to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.
E or e	Scientific (exponential)	The number is converted to a string of the form "-d.ddd…E+ddd" or "-d.ddd…e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used. The case of the format specifier indicates whether to prefix the exponent with an 'E' or an 'e'. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
F or f	Fixed-point	The number is converted to a string of the form "-ddd.ddd…" where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the **NumberFormatInfo** is used.
G or g	General	The number is converted to the most compact decimal form, using fixed or scientific notation. The precision specifier determines the number of significant digits in the resulting string. If the precision specifier is omitted, the number of significant digits is determined by the type of number being converted: • **Int16** or **UInt16**: 5 digits • **Int32** or **UInt32**: 10 digits • **Int64** or **UInt64**: 19 digits • **Single**: 7 digits • **Double**: 15 digits • **Decimal**: 29 digits Trailing zeros after the decimal point are removed, and the resulting string contains a decimal point only if required. The resulting string uses fixed-point format if the exponent of the number (as produced by the 'E' format) is less than the number of significant digits, and greater than or equal to –4. Otherwise, the resulting string uses scientific format, and the case of the format specifier controls whether the format is prefixed with an 'E' or an 'e'.
N or n	Number	The number is converted to a string of the form "-d,ddd,ddd.ddd…", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. Thousand separators are inserted between each group of three digits to the left of the decimal point. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the **NumberFormatInfo** is used.
P or p	Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property or the NumberFormatInfo.PercentPositivePattern property. If the number is negative, the string produced is defined by the **PercentNegativePattern** and starts with a minus sign. The converted number is multiplied by 100 in order to be presented as a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by **NumberFormatInfo** is used.
R or r	Round-trip	The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same numeric value. When a numeric value is formatted using this specifier, it is first tested using the general format, with 15 spaces of precision for a **Double** and 7 spaces of precision for a **Single**. If the value is successfully parsed back to the same numeric value, then it is formatted using the general format specifer. However, if the value is not successfully parsed back to the same numeric value, then the value is formatted using 17 digits of precision for a **Double** and 9 digits of precision for a **Single**. Although a precision specifier can be appended to the round-trip format specifier, it is ignored. Round trips are given precedence over precision when using this specifier. This format is supported by floating-point types only.
X or x	Hexadecimal	The number is converted to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for the hexadecimal digits greater than 9. For example, use 'X' to produce 'ABCDEF', and 'x' to produce 'abcdef'. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. This format is supported for integral types only.

The following table illustrates the output created by the standard numeric format strings when applied to specific data types and values. While not explicitly shown, all examples are applied to a **ToString** method, where the Format column indicates the passed format string. The Culture column indicates which culture associated with the current thread is assumed when formatting the output. The Data type column indicates the data type used, while the Value column indicates the value of the number being formatted. Finally, the Output column indicates the default result of the formatting using the specified culture. This table is meant to be a quick guide to understand how the standard numeric format specifiers affect formatting. See the section below the table for selected code examples.

Format	Culture	Data type	Value	Output
C	en-US	**Double**	12345.6789	$12,345.68
C	de-DE	**Double**	12345.678	12.345,68 DM
D	en-US	**Int32**	12345	12345
D8	en-US	**Int32**	12345	00012345
E	en-US	**Double**	12345.6789	1.234568E+004
E10	en-US	**Double**	12345.6789	1.2345678900E+004
E	fr-FR	**Double**	12345.6789	1,234568E+004
e4	en-US	**Double**	12345.6789	1.2346e+004
F	en-US	**Double**	12345.6789	12345.68
F	es-ES	**Double**	12345.6789	12345,68
F0	en-US	**Double**	12345.6789	123456
F6	en-US	**Double**	12345.6789	12345.678900
G	en-US	**Double**	12345.6789	12345.6789
G7	en-US	**Double**	12345.6789	12345.68
G	en-US	**Double**	0.0000023	2.3E-6
G	en-US	**Double**	0.0023	0.0023
G2	en-US	**Double**	1234	1.2E3
G	en-US	**Double**	Math.PI	3.14159265358979
N	en-US	**Double**	12345.6789	12,345.68
N	sv-SE	**Double**	12345.6789	12 345,68
N4	en-US	**Double**	123456789	123,456,789.0000
P	en-US	**Double**	.126	12.60 %
r	en-US	**Double**	Math.PI	3.141592653589793
x	en-US	**Int32**	0x2c45e	2c45e
X	en-US	**Int32**	0x2c45e	2C45E
X8	en-US	**Int32**	0x2c45e	0002C45E
x	en-US	**Int32**	123456789	75bcd15

The following code example illustrates how to use the standard numeric format specifiers to format numeric base types.

```
Imports System
Imports System.Globalization
Imports System.Threading

Module Module1
   Sub Main()

      Thread.CurrentThread.CurrentCulture = New CultureInfo("en-us")
      Dim MyDouble As Double = 123456789

      Console.WriteLine("The examples in en-US culture:")
      Console.WriteLine(MyDouble.ToString("C"))
      Console.WriteLine(MyDouble.ToString("E"))
      Console.WriteLine(MyDouble.ToString("P"))
      Console.WriteLine(MyDouble.ToString("N"))
      Console.WriteLine(MyDouble.ToString("F"))

      Thread.CurrentThread.CurrentCulture = New CultureInfo("de-DE")
      Console.WriteLine("The examples in de-DE culture:")
      Console.WriteLine(MyDouble.ToString("C"))
      Console.WriteLine(MyDouble.ToString("E"))
      Console.WriteLine(MyDouble.ToString("P"))
      Console.WriteLine(MyDouble.ToString("N"))
      Console.WriteLine(MyDouble.ToString("F"))
   End Sub
End Module
```

The preceding code example displays the following to the console.

```
The examples in en-US culture:
$123,456,789.00
1.234568E+008
12,345,678,900.00%
123,456,789.00
123456789.00
The examples in de-DE culture:
123.456.789,00 DM
1,234568E+008
12,345,678,900.00%
123.456.789,00
123456789,00
```

See Also

Formatting Types | System.Globalization.NumberFormatInfo | Custom Numeric Format Strings