Standard Numeric Format Strings

.NET Framework 4.6 and 4.5

Standard numeric format strings are used to format common numeric types. A standard numeric format string takes the form *Axx*, where:

- *A* is a single alphabetic character called the *format specifier*. Any numeric format string that contains more than one alphabetic character, including white space, is interpreted as a custom numeric format string. For more information, see Custom Numeric Format Strings.

- *xx* is an optional integer called the *precision specifier*. The precision specifier ranges from 0 to 99 and affects the number of digits in the result. Note that the precision specifier controls the number of digits in the string representation of a number. It does not round the number itself. To perform a rounding operation, use the Math.Ceiling, Math.Floor, or Math.Round method.

  When *precision specifier* controls the number of fractional digits in the result string, the result strings reflect numbers that are rounded away from zero (that is, using MidpointRounding.AwayFromZero).

Standard numeric format strings are supported by some overloads of the **ToString** method of all numeric types. For example, you can supply a numeric format string to the ToString(String) and ToString(String, IFormatProvider) methods of the Int32 type. Standard numeric format strings are also supported by the .NET Framework composite formatting feature, which is used by some **Write** and **WriteLine** methods of the Console and StreamWriter classes, the String.Format method, and the StringBuilder.AppendFormat method. The composite format feature allows you to include the string representation of multiple data items in a single string, to specify field width, and to align numbers in a field. For more information, see Composite Formatting.

Tip

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

The following table describes the standard numeric format specifiers and displays sample output produced by each format specifier. See the Notes section for additional information about using standard numeric format strings, and the Example section for a comprehensive illustration of their use.

Format specifier	Name	Description	Examples
"C" or "c"	Currency	Result: A currency value. Supported by: All numeric types. Precision specifier: Number of decimal digits. Default precision specifier:	123.456 ("C", en-US) -> $123.46 123.456 ("C", fr-FR) -> 123,46 € 123.456 ("C", ja-JP) -> ¥123 -123.456 ("C3", en-US) ->

		Default precision specifier: Defined byNumberFormatInfo.CurrencyDecimalDigits. More information: The Currency ("C") Format Specifier.	-123.456 ("C3", en-US) -> ($123.456) -123.456 ("C3", fr-FR) -> -123,456 € -123.456 ("C3", ja-JP) -> -¥123.456
"D" or "d"	Decimal	Result: Integer digits with optional negative sign. Supported by: Integral types only. Precision specifier: Minimum number of digits. Default precision specifier: Minimum number of digits required. More information: The Decimal("D") Format Specifier.	1234 ("D") -> 1234 -1234 ("D6") -> -001234
"E" or "e"	Exponential (scientific)	Result: Exponential notation. Supported by: All numeric types. Precision specifier: Number of decimal digits. Default precision specifier: 6. More information: The Exponential ("E") Format Specifier.	1052.0329112756 ("E", en-US) -> 1.052033E+003 1052.0329112756 ("e", fr-FR) -> 1,052033e+003 -1052.0329112756 ("e2", en-US) -> -1.05e+003 -1052.0329112756 ("E2", fr_FR) -> -1,05E+003
"F" or "f"	Fixed-point	Result: Integral and decimal digits with optional negative sign. Supported by: All numeric types. Precision specifier: Number of decimal digits. Default precision specifier: Defined byNumberFormatInfo.NumberDecimalDigits. More information: The Fixed-Point ("F") Format	1234.567 ("F", en-US) -> 1234.57 1234.567 ("F", de-DE) -> 1234,57 1234 ("F1", en-US) -> 1234.0 1234 ("F1", de-DE) -> 1234,0 -1234.56 ("F4", en-US) -> -1234.5600 -1234.56 ("F4", de-DE) -> -1234,5600

		Fixed-Point ("F") Format Specifier.	
"G" or "g"	General	Result: The most compact of either fixed-point or scientific notation. Supported by: All numeric types. Precision specifier: Number of significant digits. Default precision specifier: Depends on numeric type. More information: The General ("G") Format Specifier.	-123.456 ("G", en-US) -> -123.456 -123.456 ("G", sv-SE) -> -123,456 123.4546 ("G4", en-US) -> 123.5 123.4546 ("G4", sv-SE) -> 123,5 -1.234567890e-25 ("G", en-US) -> -1.23456789E-25 -1.234567890e-25 ("G", sv-SE) -> -1,23456789E-25
"N" or "n"	Number	Result: Integral and decimal digits, group separators, and a decimal separator with optional negative sign. Supported by: All numeric types. Precision specifier: Desired number of decimal places. Default precision specifier: Defined byNumberFormatInfo.NumberDecimalDigits. More information: The Numeric ("N") Format Specifier.	1234.567 ("N", en-US) -> 1,234.57 1234.567 ("N", ru-RU) -> 1 234,57 1234 ("N1", en-US) -> 1,234.0 1234 ("N1", ru-RU) -> 1 234,0 -1234.56 ("N3", en-US) -> -1,234.560 -1234.56 ("N3", ru-RU) -> -1 234,560
"P" or "p"	Percent	Result: Number multiplied by 100 and displayed with a percent symbol. Supported by: All numeric types. Precision specifier: Desired number of decimal places. Default precision specifier: Defined by NumberFormatInfo.PercentDecimalDigits. More information: The	1 ("P", en-US) -> 100.00 % 1 ("P", fr-FR) -> 100,00 % -0.39678 ("P1", en-US) -> -39.7 % -0.39678 ("P1", fr-FR) -> -39,7 %

		Percent ("P") Format Specifier.	
"R" or "r"	Round-trip	Result: A string that can round-trip to an identical number. Supported by: Single, Double, and BigInteger. Precision specifier: Ignored. More information: The Round-trip ("R") Format Specifier.	123456789.12345678 ("R") -> 123456789.12345678 -1234567890.12345678 ("R") -> -1234567890.1234567
"X" or "x"	Hexadecimal	Result: A hexadecimal string. Supported by: Integral types only. Precision specifier: Number of digits in the result string. More information: The HexaDecimal ("X") Format Specifier.	255 ("X") -> FF -1 ("x") -> ff 255 ("x4") -> 00ff -1 ("X4") -> 00FF
Any other single character	Unknown specifier	Result: Throws a FormatException at run time.	

Using Standard Numeric Format Strings

A standard numeric format string can be used to define the formatting of a numeric value in one of two ways:

- It can be passed to an overload of the **ToString** method that has a *format* parameter. The following example formats a numeric value as a currency string in the current (in this case, the en-US) culture.

  C#

  ```
  decimal value = 123.456m;
  Console.WriteLine(value.ToString("C2"));
  // Displays $123.46
  ```

- It can be supplied as the *formatString* parameter in a format item used with such methods as String.Format, Console.WriteLine, and StringBuilder.AppendFormat. For more information, see Composite Formatting. The following example uses a format item to insert a currency value in a string.

  C#

  ```
  decimal value = 123.456m;
  ```

```
decimal value = 123.456m;
Console.WriteLine("Your account balance is {0:C2}.", value);
// Displays "Your account balance is $123.46."
```

The following sections provide detailed information about each of the standard numeric format strings.

The Currency ("C") Format Specifier

The "C" (or currency) format specifier converts a number to a string that represents a currency amount. The precision specifier indicates the desired number of decimal places in the result string. If the precision specifier is omitted, the default precision is defined by the NumberFormatInfo.CurrencyDecimalDigits property.

If the value to be formatted has more than the specified or default number of decimal places, the fractional value is rounded in the result string. If the value to the right of the number of specified decimal places is 5 or greater, the last digit in the result string is rounded away from zero.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the NumberFormatInfo properties that control the formatting of the returned string.

NumberFormatInfo property	Description
CurrencyPositivePattern	Defines the placement of the currency symbol for positive values.
CurrencyNegativePattern	Defines the placement of the currency symbol for negative values, and specifies whether the negative sign is represented by parentheses or the NegativeSign property.
NegativeSign	Defines the negative sign used if CurrencyNegativePattern indicates that parentheses are not used.
CurrencySymbol	Defines the currency symbol.
CurrencyDecimalDigits	Defines the default number of decimal digits in a currency value. This value can be overridden by using the precision specifier.
CurrencyDecimalSeparator	Defines the string that separates integral and decimal digits.
CurrencyGroupSeparator	Defines the string that separates groups of integral numbers.
CurrencyGroupSizes	Defines the number of integer digits that appear in a group.

The following example formats a Double value with the currency format specifier.

C#

```
double value = 12345.6789;
Console.WriteLine(value.ToString("C", CultureInfo.CurrentCulture));

Console.WriteLine(value.ToString("C3", CultureInfo.CurrentCulture));

Console.WriteLine(value.ToString("C3",
                  CultureInfo.CreateSpecificCulture("da-DK")));
// The example displays the following output on a system whose
// current culture is English (United States):
//       $12,345.68
//       $12,345.679
//       kr 12.345,679
```

Back to table

The Decimal ("D") Format Specifier

The "D" (or decimal) format specifier converts a number to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. This format is supported only for integral types.

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. If no precision specifier is specified, the default is the minimum value required to represent the integer without leading zeros.

The result string is affected by the formatting information of the current NumberFormatInfo object. As the following table shows, a single property affects the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.

The following example formats an Int32 value with the decimal format specifier.

C#

```
int value;

value = 12345;
Console.WriteLine(value.ToString("D"));
// Displays 12345
Console.WriteLine(value.ToString("D8"));
// Displays 00012345

value = -12345;
Console.WriteLine(value.ToString("D"));
// Displays -12345
Console.WriteLine(value.ToString("D8"));
// Displays -00012345
```

Back to table

Back to table

The Exponential ("E") Format Specifier

The exponential ("E") format specifier converts a number 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. Exactly 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.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the NumberFormatInfo properties that control the formatting of the returned string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative for both the coefficient and exponent.
NumberDecimalSeparator	Defines the string that separates the integral digit from decimal digits in the coefficient.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats a Double value with the exponential format specifier.

C#

```
double value = 12345.6789;
Console.WriteLine(value.ToString("E", CultureInfo.InvariantCulture));
// Displays 1.234568E+004

Console.WriteLine(value.ToString("E10", CultureInfo.InvariantCulture));
// Displays 1.2345678900E+004

Console.WriteLine(value.ToString("e4", CultureInfo.InvariantCulture));
// Displays 1.2346e+004

Console.WriteLine(value.ToString("E",
                  CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 1,234568E+004
```

Back to table

The Fixed-Point ("F") Format Specifier

The fixed-point ("F) format specifier converts a number 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 current NumberFormatInfo.NumberDecimalDigits property supplies the numeric precision.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the properties of the NumberFormatInfo object that control the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.
NumberDecimalSeparator	Defines the string that separates integral digits from decimal digits.
NumberDecimalDigits	Defines the default number of decimal digits. This value can be overridden by using the precision specifier.

The following example formats a Double and an Int32 value with the fixed-point format specifier.

C#

```
int integerNumber;
integerNumber = 17843;
Console.WriteLine(integerNumber.ToString("F",
                  CultureInfo.InvariantCulture));
// Displays 17843.00

integerNumber = -29541;
Console.WriteLine(integerNumber.ToString("F3",
                  CultureInfo.InvariantCulture));
// Displays -29541.000

double doubleNumber;
doubleNumber = 18934.1879;
Console.WriteLine(doubleNumber.ToString("F", CultureInfo.InvariantCulture));
// Displays 18934.19

Console.WriteLine(doubleNumber.ToString("F0", CultureInfo.InvariantCulture));
// Displays 18934

doubleNumber = -1898300.1987;
Console.WriteLine(doubleNumber.ToString("F1", CultureInfo.InvariantCulture));
// Displays -1898300.2

Console.WriteLine(doubleNumber.ToString("F3",
                  CultureInfo.CreateSpecificCulture("es-ES")));
// Displays -1898300,199
```

Back to table

The General ("G") Format Specifier

The general ("G") format specifier converts a number to the most compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. The precision specifier defines the maximum number of significant digits that can appear in the result string. If the precision specifier is omitted or zero, the type of the number determines the default precision, as indicated in the following table.

Numeric type	Default precision
Byte or SByte	3 digits
Int16 or UInt16	5 digits
Int32 or UInt32	10 digits
Int64	19 digits
UInt64	20 digits
BigInteger	50 digits
Single	7 digits
Double	15 digits
Decimal	29 digits

Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required, and trailing zeros after the decimal point are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, the excess trailing digits are removed by rounding.

However, if the number is a Decimal and the precision specifier is omitted, fixed-point notation is always used and trailing zeros are preserved.

If scientific notation is used, the exponent in the result is prefixed with "E" if the format specifier is "G", or "e" if the format specifier is "g". The exponent contains a minimum of two digits. This differs from the format for scientific notation that is produced by the exponential format specifier, which includes a minimum of three digits in the exponent.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the NumberFormatInfo properties that control the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.
NumberDecimalSeparator	Defines the string that separates integral digits from

	decimal digits.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats assorted floating-point values with the general format specifier.

C#

```csharp
double number;

number = 12345.6789;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays  12345.6789
Console.WriteLine(number.ToString("G",
                  CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 12345,6789

Console.WriteLine(number.ToString("G7", CultureInfo.InvariantCulture));
// Displays 12345.68

number = .0000023;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays 2.3E-06
Console.WriteLine(number.ToString("G",
                  CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 2,3E-06

number = .0023;
Console.WriteLine(number.ToString("G", CultureInfo.InvariantCulture));
// Displays 0.0023

number = 1234;
Console.WriteLine(number.ToString("G2", CultureInfo.InvariantCulture));
// Displays 1.2E+03

number = Math.PI;
Console.WriteLine(number.ToString("G5", CultureInfo.InvariantCulture));
// Displays 3.1416
```

Back to table

The Numeric ("N") Format Specifier

The numeric ("N") format specifier converts a number to a string of the form "-d,ddd,ddd.ddd…", where "-" indicates a negative number symbol if required, "d" indicates a digit (0-9), "," indicates a group separator, and "." indicates a decimal point symbol. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, the number of decimal places is defined by the current NumberFormatInfo.NumberDecimalDigits property.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the NumberFormatInfo properties that control the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.
NumberNegativePattern	Defines the format of negative values, and specifies whether the negative sign is represented by parentheses or the NegativeSign property.
NumberGroupSizes	Defines the number of integral digits that appear between group separators.
NumberGroupSeparator	Defines the string that separates groups of integral numbers.
NumberDecimalSeparator	Defines the string that separates integral and decimal digits.
NumberDecimalDigits	Defines the default number of decimal digits. This value can be overridden by using a precision specifier.

The following example formats assorted floating-point values with the number format specifier.

C#

```
double dblValue = -12445.6789;
Console.WriteLine(dblValue.ToString("N", CultureInfo.InvariantCulture));
// Displays -12,445.68
Console.WriteLine(dblValue.ToString("N1",
                  CultureInfo.CreateSpecificCulture("sv-SE")));
// Displays -12 445,7

int intValue = 123456789;
Console.WriteLine(intValue.ToString("N1", CultureInfo.InvariantCulture));
// Displays 123,456,789.0
```

Back to table

The Percent ("P") Format Specifier

The percent ("P") format specifier multiplies a number by 100 and converts it to a string that represents a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision supplied by the current PercentDecimalDigits property is used.

The following table lists the NumberFormatInfo properties that control the formatting of the returned string.

NumberFormatInfo property	Description
PercentPositivePattern	Defines the placement of the percent symbol for positive values

	values.
PercentNegativePattern	Defines the placement of the percent symbol and the negative symbol for negative values.
NegativeSign	Defines the string that indicates that a number is negative.
PercentSymbol	Defines the percent symbol.
PercentDecimalDigits	Defines the default number of decimal digits in a percentage value. This value can be overridden by using the precision specifier.
PercentDecimalSeparator	Defines the string that separates integral and decimal digits.
PercentGroupSeparator	Defines the string that separates groups of integral numbers.
PercentGroupSizes	Defines the number of integer digits that appear in a group.

The following example formats floating-point values with the percent format specifier.

C#

```
double number = .2468013;
Console.WriteLine(number.ToString("P", CultureInfo.InvariantCulture));
// Displays 24.68 %
Console.WriteLine(number.ToString("P",
                  CultureInfo.CreateSpecificCulture("hr-HR")));
// Displays 24,68%
Console.WriteLine(number.ToString("P1", CultureInfo.InvariantCulture));
// Displays 24.7 %
```

Back to table

The Round-trip ("R") Format Specifier

The round-trip ("R") format specifier is used to ensure that a numeric value that is converted to a string will be parsed back into the same numeric value. This format is supported only for the Single, Double, and BigInteger types.

When a BigInteger value is formatted using this specifier, its string representation contains all the significant digits in the BigInteger value. When a Single or Double value is formatted using this specifier, it is first tested using the general format, with 15 digits of precision for a Double and 7 digits of precision for a Single. If the value is successfully parsed back to the same numeric value, it is formatted using the general format specifier. If the value is not successfully parsed back to the same numeric value, it is formatted using 17 digits of precision for a Double and 9 digits of precision for a Single.

Although you can include a precision specifier, it is ignored. Round trips are given precedence over precision when using this specifier.

The result string is affected by the formatting information of the current NumberFormatInfo object. The following table lists the NumberFormatInfo properties that control the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.
NumberDecimalSeparator	Defines the string that separates integral digits from decimal digits.
PositiveSign	Defines the string that indicates that an exponent is positive.

The following example formats Double values with the round-trip format specifier.

C#

```
double value;

value = Math.PI;
Console.WriteLine(value.ToString("r"));
// Displays 3.1415926535897931
Console.WriteLine(value.ToString("r",
                  CultureInfo.CreateSpecificCulture("fr-FR")));
// Displays 3,1415926535897931
value = 1.623e-21;
Console.WriteLine(value.ToString("r"));
// Displays 1.623E-21
```

Important

In some cases, Double values formatted with the "R" standard numeric format string do not successfully round-trip if compiled using the **/platform:x64** or **/platform:anycpu** switches and run on 64-bit systems. See the following paragraph for more information.

To work around the problem of Double values formatted with the "R" standard numeric format string not successfully round-tripping if compiled using the **/platform:x64** or **/platform:anycpu** switches and run on 64-bit systems., you can format Double values by using the "G17" standard numeric format string. The following example uses the "R" format string with a Double value that does not round-trip successfully, and also uses the "G17" format string to successfully round-trip the original value.

C#

```
using System;
using System.Globalization;

public class Example
{
   static void Main(string[] args)
   {
      Console.WriteLine("Attempting to round-trip a Double with 'R':");
      double initialValue = 0.6822871999174;
      double initialValue = 0.6822871999174;
      string valueString = initialValue.ToString("R",
                                                 CultureInfo.InvariantCulture);
      double roundTripped = double.Parse(valueString,
                                         CultureInfo.InvariantCulture);
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped, initialValue.Equals(roundTripped));

      Console.WriteLine("Attempting to round-trip a Double with 'G17':");
      string valueString17 = initialValue.ToString("G17",
                                                   CultureInfo.InvariantCulture);
      double roundTripped17 = double.Parse(valueString17,
                                           CultureInfo.InvariantCulture);
      Console.WriteLine("{0:R} = {1:R}: {2}\n",
                        initialValue, roundTripped17, initialValue.Equals(roundTripped17));
   }
}
// If compiled to an application that targets anycpu or x64 and run on an x64 system,
// the example displays the following output:
//       Attempting to round-trip a Double with 'R':
//       0.6822871999174 = 0.68228719991740006: False
//
//       Attempting to round-trip a Double with 'G17':
//       0.6822871999174 = 0.6822871999174: True
```

Back to table

The Hexadecimal ("X") Format Specifier

The hexadecimal ("X") format specifier converts a number to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for hexadecimal digits that are greater than 9. For example, use "X" to produce "ABCDEF", and "x" to produce "abcdef". This format is supported only for integral types.

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.

The result string is not affected by the formatting information of the current NumberFormatInfo object.

The following example formats Int32 values with the hexadecimal format specifier.

C#

```
int value;

value = 0x2045e;
Console.WriteLine(value.ToString("x"));
// Displays 2045e
Console.WriteLine(value.ToString("X"));
// Displays 2045E
Console.WriteLine(value.ToString("X8"));
// Displays 0002045E

value = 123456789;
Console.WriteLine(value.ToString("X"));
// Displays 75BCD15
Console.WriteLine(value.ToString("X2"));
Console.WriteLine(value.ToString("X2"));
// Displays 75BCD15
```

Back to table

Notes

Control Panel Settings

The settings in the **Regional and Language Options** item in Control Panel influence the result string produced by a formatting operation. Those settings are used to initialize the NumberFormatInfo 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 the CultureInfo.CultureInfo(String) constructor is used 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 does not reflect a system's customizations.

NumberFormatInfo Properties

Formatting is influenced by the properties of the current NumberFormatInfo object, which is provided implicitly by the current thread culture or explicitly by the IFormatProvider parameter of the method that invokes formatting. Specify a NumberFormatInfo or CultureInfo object for that parameter.

Note

For information about customizing the patterns or strings used in formatting numeric values, see the NumberFormatInfo class topic.

Integral and Floating-Point Numeric Types

Some descriptions of standard numeric format specifiers refer to integral or floating-point numeric types. The integral numeric types are Byte, SByte, Int16, Int32, Int64, UInt16, UInt32, UInt64, and BigInteger. The floating-point numeric types are Decimal, Single, and Double.

Floating-Point Infinities and NaN

Regardless of the format string, if the value of a Single or Double floating-point type is positive infinity, negative infinity, or not a number (NaN), the formatted string is the value of the respective PositiveInfinitySymbol, NegativeInfinitySymbol, or NaNSymbol property that is specified by the currently applicable NumberFormatInfo object.

Example

The following example formats an integral and a floating-point numeric value using the en-US culture and all the standard numeric format specifiers. This example uses two particular numeric types (Double and Int32), but would yield similar results for any of the other numeric base types (Byte, SByte, Int16, Int32, Int64, UInt16, UInt32, UInt64, BigInteger, Decimal, and Single).

C#

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

public class NumericFormats
{
   public static void Main()
   {
      // Display string representations of numbers for en-us culture
      CultureInfo ci = new CultureInfo("en-us");

      // Output floating point values
      double floating = 10761.937554;
      Console.WriteLine("C: {0}", 
              floating.ToString("C", ci));           // Displays "C: $10,761.94"
      Console.WriteLine("E: {0}", 
              floating.ToString("E03", ci));         // Displays "E: 1.076E+004"
      Console.WriteLine("F: {0}", 
              floating.ToString("F04", ci));         // Displays "F: 10761.9376"
      Console.WriteLine("G: {0}",  
              floating.ToString("G", ci));           // Displays "G: 10761.937554"
      Console.WriteLine("N: {0}", 
              floating.ToString("N03", ci));         // Displays "N: 10,761.938"
      Console.WriteLine("P: {0}", 
              (floating/10000).ToString("P02", ci)); // Displays "P: 107.62 %"
      Console.WriteLine("R: {0}", 
              floating.ToString("R", ci));           // Displays "R: 10761.937554"
      Console.WriteLine();

      // Output integral values
      int integral = 8395;
      Console.WriteLine("C: {0}", 
              integral.ToString("C", ci));           // Displays "C: $8,395.00"
      Console.WriteLine("D: {0}", 
              integral.ToString("D6", ci));          // Displays "D: 008395" 
      Console.WriteLine("E: {0}", 
              integral.ToString("E03", ci));         // Displays "E: 8.395E+003"
      Console.WriteLine("F: {0}", 
              integral.ToString("F01", ci));         // Displays "F: 8395.0"    
      Console.WriteLine("G: {0}",  
              integral.ToString("G", ci));           // Displays "G: 8395"
      Console.WriteLine("N: {0}", 
              integral.ToString("N01", ci));         // Displays "N: 8,395.0"
      Console.WriteLine("P: {0}", 
              (integral/10000.0).ToString("P02", ci)); // Displays "P: 83.95 %"
      Console.WriteLine("X: 0x{0}",
      Console.WriteLine("X: 0x{0}",
              integral.ToString("X", ci));            // Displays "X: 0x20CB"
      Console.WriteLine();
   }
}
```

See Also

Tasks

How to: Pad a Number with Leading Zeros

Reference

NumberFormatInfo

Concepts

Custom Numeric Format Strings

Composite Formatting

Other Resources

Formatting Types in the .NET Framework

Sample: .NET Framework 4 Formatting Utility