System.IFormattable Interface

```
[ILAsm]
.class interface public abstract IFormattable

[C#]
public interface IFormattable
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
    - CLSCompliantAttribute(true)

Summary

Implemented by classes that construct customizable string representations of objects.

Library: BCL

Description

[*Note:* `System.IFormattable` contains the `System.IFormattable.ToString` method. The consumer of an object calls this method to obtain a formatted string representation of the value of the object.]

A *format* is a string that describes the appearance of an object when it is converted to a string. Either standard or custom formats can be used. A standard format takes the form *Axx*, where *A* is a single alphabetic character called the *format specifier*, and *xx* is an integer between zero and 99 inclusive, called the *precision specifier*. The format specifier controls the type of formatting applied to the value being represented as a string. The *precision specifier* controls the number of significant digits or decimal places in the string, if applicable. [*Note:* For the list of standard format specifiers, see the table below. Note that a given data type, such as `System.Int32`, might not support one or more of the standard format specifiers.]

[*Note:* When a format includes symbols that vary by culture, such as the currency symbol included by the "C" and "c" formats, a formatting object supplies the actual characters used in the string representation. A method might include a parameter to pass a `System.IFormatProvider` object that supplies a formatting object, or the method might use the default formatting object, which contains the symbol definitions for the current culture. The current culture typically uses the same set of symbols used system-wide by default. In the Base Class Library, the formatting object for system-supplied numeric types is a `System.Globalization.NumberFormatInfo` instance. For `System.DateTime` instances, a `System.Globalization.DateTimeFormatInfo` is used.]

The following table describes the standard format specifiers and associated formatting object members that are used with numeric data types in the Base Class Library.

Format Specifier	**Description**
C c	`Currency Format`: Used for strings containing a monetary value. The `System.Globalization.NumberFormatInfo.CurrencySymbol`, `System.Globalization.NumberFormatInfo.CurrencyGroupSizes`, `System.Globalization.NumberFormatInfo.CurrencyGroupSeparator`, and `System.Globalization.NumberFormatInfo.CurrencyDecimalSeparator` members of a `System.Globalization.NumberFormatInfo` supply the currency symbol, size and separator for digit groupings, and decimal separator, respectively. `System.Globalization.NumberFormatInfo.CurrencyNegativePattern` and `System.Globalization.NumberFormatInfo.CurrencyPositivePattern` determine the symbols used to represent negative and positive values. For example, a negative value can be prefixed with a minus sign, or enclosed in parentheses. If the precision specifier is omitted, `System.Globalization.NumberFormatInfo.CurrencyDecimalDigits` determines the number of decimal places in the string. Results are rounded to the nearest representable value when necessary.
D d	`Decimal Format`: (This format is valid only when specified with integral data types.) Used for strings containing integer values. Negative numbers are prefixed with the negative number symbol specified by the `System.Globalization.NumberFormatInfo.NegativeSign` property. The precision specifier determines the minimum number of digits that appear in the string. If the specified precision requires more digits than the value contains, the string is left-padded with zeros. If the precision specifier specifies fewer digits than are in the value, the precision specifier is ignored.
E e	`Scientific (Engineering) Format`: Used for strings in one of the following forms: [-]*m.ddddddE+xxx* [-]*m.ddddddE-xxx* [-]*m.dddddd*e*+xxx*

	[-]*m.dddddd*e-*xxx* The negative number symbol ('-') appears only if the value is negative, and is supplied by the `System.Globalization.NumberFormatInfo.NegativeSign` property. Exactly one non-zero decimal digit *(m)* precedes the decimal separator ('.'), which is supplied by the `System.Globalization.NumberFormatInfo.NumberDecimalSeparator` property. The precision specifier determines the number of decimal places (*dddddd*) in the string. If the precision specifier is omitted, six decimal places are included in the string. The exponent *(+/-xxx*) consists of either a positive or negative number symbol followed by a minimum of three digits (*xxx*). The exponent is left-padded with zeros, if necessary. The case of the format specifier ('E' or 'e') determines the case used for the exponent prefix (E or e) in the string. Results are rounded to the nearest representable value when necessary. The positive number symbol is supplied by the `System.Globalization.NumberFormatInfo.PositiveSign` property.
F f	`Fixed-Point Format`: Used for strings in the following form: "[-]*m.dd...d*" At least one non-zero decimal digit (*m*) precedes the decimal separator ('.'), which is supplied by the `System.Globalization.NumberFormatInfo.NumberDecimalSeparator` property. A negative number symbol sign ('-') precedes *m* only if the value is negative. This symbol is supplied by the `System.Globalization.NumberFormatInfo.NegativeSign` property. The precision specifier determines the number of decimal places (*dd...d*) in the string. If the precision specifier is omitted, `System.Globalization.NumberFormatInfo.NumberDecimalDigits` determines the number of decimal places in the string. Results are rounded to the nearest representable value when necessary.
G g	`General Format`: The string is formatted in either fixed-point format ('F' or 'f') or scientific format ('E' or 'e'). For integral types:

	Values are formatted using fixed-point format if *exponent* < precision specifier, where *exponent* is the exponent of the value in scientific format. For all other values, scientific format is used. If the precision specifier is omitted, a default precision equal to the field width required to display the maximum value for the data type is used, which results in the value being formatted in fixed-point format. The default precisions for integral types are as follows: `System.Int16`, `System.UInt16` - 5 `System.Int32`, `System.UInt32`- 10 `System.Int64`, `System.UInt64` - 19 For Single, Decimal and Double types: Values are formatted using fixed-point format if *exponent* >= -4 and *exponent* < precision specifier, where *exponent* is the exponent of the value in scientific format. For all other values, scientific format is used. Results are rounded to the nearest representable value when necessary. If the precision specifier is omitted, the following default precisions are used: `System.Single`: 7 `System.Double`: 15 `System.Decimal`: 29 For all types: - The number of digits that appear in the result (not including the exponent) will not exceed the value of the precision specifier; values are rounded as necessary. - The decimal point and any trailing zeros after the decimal point are removed whenever possible. - The case of the format specifier ('G' or 'g') determines whether 'E' or 'e' prefixes the scientific format exponent.

`N`
 	
`n`
	`Number Format`: Used for strings in the following form:
	
	[-]*d,ddd,ddd.dd...d*

	The representation of negative values is determined by the `System.Globalization.NumberFormatInfo.NumberNegativePattern`property. If the pattern includes a negative number symbol ('-'), this symbol is supplied by the `System.Globalization.NumberFormatInfo.NegativeSign` property. At least one non-zero decimal digit (*d*) precedes the decimal separator ('.'), which is supplied by the `System.Globalization.NumberFormatInfo.NumberDecimalSeparator` property. Digits between the decimal point and the most significant digit in the value are grouped using the group size specified by the `System.Globalization.NumberFormatInfo.NumberGroupSizes` property. The group separator (',') is inserted between each digit group, and is supplied by the `System.Globalization.NumberFormatInfo.NumberGroupSeparator` property. The precision specifier determines the number of decimal places (*dd...d*). If the precision specifier is omitted, `System.Globalization.NumberFormatInfo.NumberDecimalDigits` determines the number of decimal places in the string. Results are rounded to the nearest representable value when necessary.
P p	`Percent Format`: Used for strings containing a percentage. The `System.Globalization.NumberFormatInfo.PercentSymbol`, `System.Globalization.NumberFormatInfo.PercentGroupSizes`, `System.Globalization.NumberFormatInfo.PercentGroupSeparator`, and `System.Globalization.NumberFormatInfo.PercentDecimalSeparator` members of a `System.Globalization.NumberFormatInfo` supply the percent symbol, size and separator for digit groupings, and decimal separator, respectively. `System.Globalization.NumberFormatInfo.PercentNegativePattern` and `System.Globalization.NumberFormatInfo.PercentPositivePattern` determine the symbols used to represent negative and positive values. For example, a negative value can be prefixed with a minus sign, or enclosed in parentheses. If no precision is specified, the number of decimal places in the result is determined by `System.Globalization.NumberFormatInfo.PercentDecimalDigits`. Results are rounded to the nearest representable value when necessary. The result is scaled by 100 (.99 becomes 99%).
R	`Round trip Format`: (This format is valid only when specified with `System.Double` or `System.Single`.) Used to ensure that the precision of the

r	string representation of a floating-point value is such that parsing the string does not result in a loss of precision when compared to the original value. If the maximum precision of the data type (7 for System.Single, and 15 for System.Double) would result in a loss of precision, the precision is increased by two decimal places. If a precision specifier is supplied with this format specifier, it is ignored. This format is otherwise identical to the fixed-point format.
X x	Hexadecimal Format: (This format is valid only when specified with integral data types.) Used for string representations of numbers in Base 16. The precision determines the minimum number of digits in the string. If the precision specifies more digits than the number contains, the number is left-padded with zeros. The case of the format specifier ('X' or 'x') determines whether upper case or lower case letters are used in the hexadecimal representation.

If the numerical value is a System.Single or System.Double with a value of NaN, PositiveInfinity, or NegativeInfinity, the format specifier is ignored, and one of the following is returned: System.Globalization.NumberFormatInfo.NaNSymbol, System.Globalization.NumberFormatInfo.PositiveInfinitySymbol, or System.Globalization.NumberFormatInfo.NegativeInfinitySymbol.

A custom format is any string specified as a format that is not in the form of a standard format string (Axx) described above. The following table describes the characters that are used in constructing custom formats.

Format Specifier	**Description**
0 (zero)	Zero placeholder: If the value being formatted has a digit in the position where a '0' appears in the custom format, then that digit is copied to the output string; otherwise a zero is stored in that position in the output string. The position of the leftmost '0' before the decimal separator and the rightmost '0' after the decimal separator determine the range of digits that are always present in the output string. The number of Zero and/or Digit placeholders after the decimal separator determines the number of digits that appear after the decimal separator. Values are rounded as necessary.
#	Digit placeholder: If the value being formatted has a digit in the position where a '#' appears in the custom format, then that digit is copied to the output string; otherwise, nothing is stored in that position in the output string. Note that this specifier never stores the '0' character if it is not a significant digit, even if '0' is the only digit in the string. (It does display the '0' character in the output string if it is a significant digit.)

	The number of Zero and/or Digit placeholders after the decimal separator determines the number of digits that appear after the decimal separator. Values are rounded as necessary.
. (period)	`Decimal separator:` The left most '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The `System.Globalization.NumberFormatInfo.NumberDecimalSeparator` property determines the symbol used as the decimal separator.
, (comma)	`Group separator and number scaling:` The ',' character serves two purposes. First, if the custom format contains this character between two Zero or Digit placeholders (0 or #) and to the left of the decimal separator if one is present, then the output will have group separators inserted between each group of digits to the left of the decimal separator. The `System.Globalization.NumberFormatInfo.NumberGroupSeparator` and `System.Globalization.NumberFormatInfo.NumberGroupSizes` properties determine the symbol used as the group separator and the number of digits in each group, respectively. If the format string contains one or more ',' characters immediately to the left of the decimal separator, then the number will be scaled. The scale factor is determined by the number of group separator characters immediately to the left of the decimal separator. If there are x characters, then the value is divided by 1000^{X} before it is formatted. For example, the format string '0,,' will divide a value by one million. Note that the presence of the ',' character to indicate scaling does not insert group separators in the output string. Thus, to scale a number by 1 million and insert group separators, use a custom format similar to "#,##0,,".
% (percent)	`Percentage placeholder:` The presence of a '%' character in a custom format causes a number to be multiplied by 100 before it is formatted. The percent symbol is inserted in the output string at the location where the '%' appears in the format string. The `System.Globalization.NumberFormatInfo.PercentSymbol` property determines the percent symbol.
E0 E+0 E-0 e0	`Engineering format:` If any of the strings 'E', 'E+', 'E-', 'e', 'e+', or 'e-' are present in a custom format and is followed immediately by at least one '0' character, then the value is formatted using scientific notation. The number of '0' characters following the exponent prefix (E or e) determines the minimum number of digits in the exponent. The 'E+' and 'e+' formats indicate that a positive or negative number symbol always precedes the exponent. The 'E', 'E-', 'e', or 'e-' formats indicate that a negative number symbol precedes negative

e+0 e-0	exponents; no symbol is precedes positive exponents. The positive number symbol is supplied by the `System.Globalization.NumberFormatInfo.PositiveSign` property. The negative number symbol is supplied by the `System.Globalization.NumberFormatInfo.NegativeSign` property.
\ (backslash)	`Escape character:` In some languages, such as C#, the backslash character causes the next character in the custom format to be interpreted as an escape sequence. It is used with C language formatting sequences, such as "\n" (newline). In some languages, the escape character itself is required to be preceded by an escape character when used as a literal. Otherwise, the compiler interprets the character as an escape sequence. This escape character is not required to be supported in all programming languages.
'ABC' "ABC"	`Literal string:` Characters enclosed in single or double quotes are copied to the output string literally, and do not affect formatting.
; (semicolon)	`Section separator:` The ';' character is used to separate sections for positive, negative, and zero numbers in the format string. (This feature is described in detail below.)
Other	`All other characters:` All other characters are stored in the output string as literals in the position in which they appear.

Note that for fixed-point format strings (strings not containing an 'E0', E+0', 'E-0', 'e0', 'e+0', or 'e-0'), numbers are rounded to as many decimal places as there are Zero or Digit placeholders to the right of the decimal separator. If the custom format does not contain a decimal separator, the number is rounded to the nearest integer. If the number has more digits than there are Zero or Digit placeholders to the left of the decimal separator, the extra digits are copied to the output string immediately before the first Zero or Digit placeholder.

A custom format can contain up to three sections separated by section separator characters, to specify different formatting for positive, negative, and zero values. The sections are interpreted as follows:

- `One section:` The custom format applies to all values (positive, negative and zero). Negative values include a negative sign.
- `Two sections:` The first section applies to positive values and zeros, and the second section applies to negative values. If the value to be formatted is negative, but becomes zero after rounding according to the format in the second section, then the resulting zero is formatted according to the first section. Negative values do not include a negative sign to allow full control over representations of negative values.

For example, a negative can be represented in parenthesis using a custom format similar to "####.####;(####.####)".

- `Three sections:` The first section applies to positive values, the second section applies to negative values, and the third section applies to zeros. The second section can be empty (nothing appears between the semicolons), in which case the first section applies to all nonzero values, and negative values include a negative sign. If the number to be formatted is nonzero, but becomes zero after rounding according to the format in the first or second section, then the resulting zero is formatted according to the third section.

The `System.Enum` and `System.DateTime` types also support using format specifiers to format string representations of values. The meaning of a specific format specifier varies according to the kind of data (numeric, date/time, enumeration) being formatted. See `System.Enum` and `System.Globalization.DateTimeFormatInfo` for a comprehensive list of the format specifiers supported by each type.

IFormattable.ToString(System.String, System.IFormatProvider) Method

```
[ILAsm]
.method public hidebysig virtual abstract string ToString(string format,
class System.IFormatProvider formatProvider)

[C#]
string ToString(string format, IFormatProvider formatProvider)
```

Summary

Returns a `System.String` representation of the value of the current instance.

Parameters

Parameter	Description
format	A `System.String` that specifies the format of the returned string. If *format* is a null reference or the empty string, the default format defined for the type of the current instance is used.
formatProvider	A `System.IFormatProvider` that supplies a formatting object containing culture-specific formatting information, or `null`.

Return Value

A `System.String` containing the value of the current instance formatted in accordance with *format* and *formatProvider*.

Behaviors

Conforming implementations do not throw an exception when *format* and/or *formatProvider* are null references. If *formatProvider* is a null reference, the string is constructed using a system-supplied formatting object containing information for the current system culture. If *format* is `null`, the string is constructed using a system-supplied default format appropriate for the type of the current instance.

If the object returned by *formatProvider* supplies a culture-specific representation of symbols or patterns included in *format*, the returned string is required to use the information supplied by *formatProvider*.

How and When to Override

Implement to allow consumers of a class to use format strings and formatting objects to control the way in which the class is represented as a string.

Exceptions

Exception	Condition
System.FormatException	The specified *format* is invalid or cannot be used with the type of the current instance.

Example

The following example demonstrates using the `System.IFormattable.ToString` method to display values in a variety of formats. The current system culture is U.S. English, which provides the default values for the *formatProvider* parameter of `System.IFormattable.ToString`.

[C#]

```
using System;
class FormattableExample {
    public static void Main() {
    double d = 123.12345678901234;
    string[] formats = {"C","E","e","F","G","N","P","R"};
    for (int i = 0; i< formats.Length;i++)
        Console.WriteLine("{0:R} as {1}:
{2}",d,formats[i],d.ToString(formats[i],null));

    string[]intFormats = {"D","x","X"};
    int val = 255;
    for (int i = 0; i< intFormats.Length;i++)
        Console.WriteLine("{0} as {1}:
{2}",val,intFormats[i],val.ToString(intFormats[i],null));

    }
}
```

The output is

```
123.12345678901234 as C: $123.12


123.12345678901234 as E: 1.231235E+002


123.12345678901234 as e: 1.231235e+002


123.12345678901234 as F: 123.12


123.12345678901234 as G: 123.123456789012
```

```
123.12345678901234 as N: 123.12


123.12345678901234 as P: 12,312.35 %


123.12345678901234 as R: 123.12345678901234


255 as D: 255


255 as x: ff


255 as X: FF
```