

# System.Double Structure

```
[ILAsm]
.class public sequential sealed serializable Double extends
System.ValueType implements System.IComparable, System.IFormattable,
System.IComparable`1<float64>, System.IEquatable`1<float64>

[C#]
public struct Double: IComparable, IFormattable, IComparable<Double>,
IEquatable<Double>
```

## 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)

## Implements:

- **System.IComparable**
- **System.IFormattable**
- **System.IComparable<System.Double>**
- **System.IEquatable<System.Double>**

## Summary

Represents a 64-bit double-precision floating-point number.

## Inherits From: System.ValueType

**Library:** ExtendedNumerics

**Thread Safety:** This type is not guaranteed to be safe for multithreaded operations.

## Description

`System.Double` is a 64-bit double precision floating-point type that represents values ranging from approximately 5.0E-324 to 1.7E+308 and from approximately -5.0E-324 to -1.7E+308 with a precision of 15-16 decimal digits. The `System.Double` type conforms to standard IEC 60559:1989, Binary Floating-point Arithmetic for Microprocessor Systems.

A `System.Double` can represent the following values:

- 1     • The finite set of non-zero values of the form  $s * m * 2^e$ , where  $s$  is 1 or -1, and  $0 <$   
2      $m < 2^{53}$  and  $-1075 \leq e \leq 970$ .
- 3     • Positive infinity and negative infinity. Infinities are produced by operations that  
4     produce results with a magnitude greater than that which can be represented by a  
5     `System.Double`, such as dividing a non-zero number by zero. For example, using  
6     `System.Double` operands, `1.0 / 0.0` yields positive infinity, and `-1.0 / 0.0` yields  
7     negative infinity. Operations include passing parameters and returning values.
- 8     • The *Not-a-Number* value (NaN). NaN values are produced by invalid floating-point  
9     operations, such as dividing zero by zero.

10 When performing binary operations, if one of the operands is a `System.Double`, then the  
11 other operand is required to be an integral type or a floating-point type (`System.Double` or  
12 `System.Single`). Prior to performing the operation, if the other operand is not a  
13 `System.Double`, it is converted to `System.Double`, and the operation is performed using at  
14 least `System.Double` range and precision. If the operation produces a numeric result, the  
15 type of the result is `System.Double`.

16  
17 The floating-point operators, including the assignment operators, do not throw exceptions.  
18 Instead, in exceptional situations, the result of a floating-point operation is zero, infinity, or  
19 NaN, as described below:

- 20     • If the result of a floating-point operation is too small for the destination format, the  
21     result of the operation is zero.
- 22     • If the magnitude of the result of a floating-point operation is too large for the  
23     destination format, the result of the operation is positive infinity or negative infinity,  
24     as appropriate for the sign of the result.
- 25     • If a floating-point operation is invalid, the result of the operation is NaN.
- 26     • If one or both operands of a floating-point operation are NaN, the result of the  
27     operation is NaN.

28 Conforming implementations of the CLI are permitted to perform floating-point operations  
29 using a precision that is higher than that required by the `System.Double` type. For example,  
30 hardware architectures that support an "extended" or "long double" floating-point type with  
31 greater range and precision than the `System.Double` type could implicitly perform all  
32 floating-point operations using this higher precision type. Expressions evaluated using a  
33 higher precision might cause a finite result to be produced instead of an infinity.

# Double.Epsilon Field

```
[ILAsm]  
.field public static literal float64 Epsilon = 4.9406564584124654e-324  
  
[C#]  
public const double Epsilon = 4.9406564584124654e-324
```

## Summary

Represents the smallest positive `System.Double` value greater than zero.

## Description

The value of this constant is 4.9406564584124654E-324.

# Double.MaxValue Field

```
[ILAsm]  
.field public static literal float64 MaxValue = 1.7976931348623157e+308  
  
[C#]  
public const double MaxValue = 1.7976931348623157e+308
```

## Summary

Contains the maximum positive value for the `System.Double` type.

## Description

The value of this constant is 1.7976931348623157E+308.

# Double.MinValue Field

```
[ILAsm]  
.field public static literal float64 MinValue = -1.7976931348623157e+308  
  
[C#]  
public const double MinValue = -1.7976931348623157e+308
```

## Summary

Contains the minimum (most negative) value for the `System.Double` type.

## Description

The value of this constant is -1.7976931348623157E+308.

# Double.NaN Field

```
[ILAsm]  
.field public static literal float64 NaN = (double)0.0 / (double)0.0  
  
[C#]  
public const double NaN = (double)0.0 / (double)0.0
```

## Summary

Represents an undefined result of operations involving `System.Double`.

## Description

Not-a-Number (NaN) values are returned when the result of a `System.Double` operation is undefined.

A NaN value is not equal to any other value, including another NaN value.

The value of this field is obtained by dividing `System.Double` zero by zero.

[*Note:* `System.Double.NaN` represents one of many possible NaN values. To test whether a `System.Double` value is a NaN, use the `System.Double.IsNaN` method.]

# Double.NegativeInfinity Field

```
[ILAsm]  
.field public static literal float64 NegativeInfinity = (double)-1.0 /  
(double)(0.0)
```

```
[C#]  
public const double NegativeInfinity = (double)-1.0 / (double)(0.0)
```

## Summary

Represents a negative infinity of type `System.Double`.

## Description

The value of this constant is obtained by dividing a negative `System.Double` by zero.

[*Note:* To test whether a `System.Double` value is a negative infinity value, use the `System.Double.IsNegativeInfinity` method.]

# Double.PositiveInfinity Field

```
[ILAsm]  
.field public static literal float64 PositiveInfinity = (double)1.0 /  
(double)(0.0)  
  
[C#]  
public const double PositiveInfinity = (double)1.0 / (double)(0.0)
```

## Summary

Represents a positive infinity of type `System.Double`.

## Description

The value of this constant is obtained by dividing a positive `System.Double` by zero.

[*Note:* To test whether a `System.Double` value is a positive infinity value, use the `System.Double.IsPositiveInfinity` method.]

# Double.CompareTo(System.Double) Method

```
[ILAsm]  
.method public final hidebysig virtual int32 CompareTo(float64 value)  
  
[C#]  
public int CompareTo(double value)
```

## Summary

Returns the sort order of the current instance compared to the specified `System.Double`.

## Parameters

Parameter	Description
<i>value</i>	The <code>System.Double</code> to compare to the current instance.

## Return Value

The return value is a negative number, zero, or a positive number reflecting the sort order of the current instance as compared to *value*. For non-zero return values, the exact value returned by this method is unspecified. The following table defines the return value:

Value	Description
Any negative number	Current instance < <i>value</i> .  -or-  Current instance is a NaN and <i>value</i> is not a NaN.
Zero	Current instance == <i>value</i> .  -or-  Current instance and <i>value</i> are both NaN, positive infinity, or negative infinity.
A positive number	Current instance > <i>value</i> .

	-or-  Current instance is not a NaN and <i>value</i> is a NaN.
--	--

1

2 **Description**

3 [Note: This method is implemented to support the `System.IComparable<Double>`  
4 interface.]

5

6

7

# Double.CompareTo(System.Object) Method

```
[ILAsm]  
.method public final hidebysig virtual int32 CompareTo(object value)  
  
[C#]  
public int CompareTo(object value)
```

## Summary

Returns the sort order of the current instance compared to the specified `System.Object`.

## Parameters

Parameter	Description
<i>value</i>	The <code>System.Object</code> to compare to the current instance.

## Return Value

The return value is a negative number, zero, or a positive number reflecting the sort order of the current instance as compared to *value*. For non-zero return values, the exact value returned by this method is unspecified. The following table defines the return value:

Value	Description
Any negative number	Current instance < <i>value</i> .  -or-  Current instance is a NaN and <i>value</i> is not a NaN and is not a null reference.
Zero	Current instance == <i>value</i> .  -or-  Current instance and <i>value</i> are both NaN, positive infinity, or negative infinity.

A positive number	Current instance > <i>value</i> .
	-or-
	<i>value</i> is a null reference.
	-or-
	Current instance is not a NaN and <i>value</i> is a NaN.

1

## 2 Description

3 [Note: This method is implemented to support the `System.IComparable` interface. Note  
4 that, although a NaN is not considered to be equal to another NaN (even itself), the  
5 `System.IComparable` interface requires that `A.CompareTo (A)` return zero.  
6  
7 ]

## 8 Exceptions

9

10

Exception	Condition
<b>System.ArgumentException</b>	<i>value</i> is not a null reference and is not of type <code>System.Double</code> .

11

12

13

# Double.Equals(System.Double) Method

```
[ILAsm]  
.method public hidebysig virtual bool Equals(float64 obj)  
  
[C#]  
public override bool Equals(double obj)
```

## Summary

Determines whether the current instance and the specified `System.Double` represent the same value.

## Parameters

Parameter	Description
<i>obj</i>	The <code>System.Double</code> to compare to the current instance.

## Return Value

`true` if *obj* has the same value as the current instance, otherwise `false`. If either *obj* or the current instance is a NaN and the other is not, returns `false`. If *obj* and the current instance are both NaN, positive infinity, or negative infinity, returns `true`.

## Description

[*Note:* This method is implemented to support the `System.IEquatable<Double>` interface.]

# Double.Equals(System.Object) Method

```
[ILAsm]  
.method public hidebysig virtual bool Equals(object obj)  
  
[C#]  
public override bool Equals(object obj)
```

## Summary

Determines whether the current instance and the specified `System.Object` represent the same type and value.

## Parameters

Parameter	Description
<i>obj</i>	The <code>System.Object</code> to compare to the current instance.

## Return Value

true if *obj* is a `System.Double` with the same value as the current instance, otherwise false. If *obj* is a null reference or is not an instance of `System.Double`, returns false. If either *obj* or the current instance is a NaN and the other is not, returns false. If *obj* and the current instance are both NaN, positive infinity, or negative infinity, returns true.

## Description

[Note: This method overrides `System.Object.Equals`.]

# Double.GetHashCode() Method

```
[ILAsm]  
.method public hidebysig virtual int32 GetHashCode()  
  
[C#]  
public override int GetHashCode()
```

## Summary

Generates a hash code for the current instance.

## Return Value

A `System.Int32` containing the hash code for this instance.

## Description

The algorithm used to generate the hash code is unspecified.

[*Note:* This method overrides `System.Object.GetHashCode()`.]

# Double.IsInfinity(System.Double) Method

```
[ILAsm]  
.method public hidebysig static bool IsInfinity(float64 d)  
  
[C#]  
public static bool IsInfinity(double d)
```

## Summary

Determines whether the specified `System.Double` represents an infinity, which can be either positive or negative.

## Parameters

Parameter	Description
<i>d</i>	The <code>System.Double</code> to be checked.

## Return Value

true if *d* represents a positive or negative infinity value; otherwise false.

## Description

[*Note:* Floating-point operations return positive or negative infinity values to signal an overflow condition.]

# Double.IsNaN(System.Double) Method

```
[ILAsm]  
.method public hidebysig static bool IsNaN(float64 d)  
  
[C#]  
public static bool IsNaN(double d)
```

## Summary

Determines whether the value of the specified `System.Double` is undefined (Not-a-Number).

## Parameters

Parameter	Description
<i>d</i>	The <code>System.Double</code> to be checked.

## Return Value

`true` if *d* represents a NaN value; otherwise `false`.

## Description

[*Note:* Floating-point operations return NaN values to signal that the result of the operation is undefined. For example, dividing (`Double`) 0.0 by 0.0 results in a NaN value.]

# Double.IsNegativeInfinity(System.Double)

## Method

```
[ILAsm]  
.method public hidebysig static bool IsNegativeInfinity(float64 d)  
  
[C#]  
public static bool IsNegativeInfinity(double d)
```

### Summary

Determines whether the specified `System.Double` represents a negative infinity value.

### Parameters

Parameter	Description
<i>d</i>	The <code>System.Double</code> to be checked.

### Return Value

true if *d* represents a negative infinity value; otherwise false.

### Description

[*Note:* Floating-point operations return negative infinity values to signal an overflow condition.]

# Double.IsPositiveInfinity(System.Double)

## Method

```
[ILAsm]  
.method public hidebysig static bool IsPositiveInfinity(float64 d)  
  
[C#]  
public static bool IsPositiveInfinity(double d)
```

### Summary

Determines whether the specified `System.Double` represents a positive infinity value.

### Parameters

Parameter	Description
<i>d</i>	The <code>System.Double</code> to be checked.

### Return Value

true if *d* represents a positive infinity value; otherwise false.

### Description

[*Note:* Floating-point operations return positive infinity values to signal an overflow condition.]

# Double.Parse(System.String) Method

```
[ILAsm]  
.method public hidebysig static float64 Parse(string s)  
  
[C#]  
public static double Parse(string s)
```

## Summary

Returns the specified System.String converted to a System.Double value.

## Parameters

Parameter	Description
s	A System.String containing the value to convert. The string is interpreted using the System.Globalization.NumberStyles.Float and/or System.Globalization.NumberStyles.AllowThousands style.

## Return Value

The System.Double value obtained from s. If s equals System.Globalization.NumberFormatInfo.NaNSymbol, this method returns System.Double.NaN.

## Description

This version of System.Double.Parse is equivalent to System.Double.Parse (s, System.Globalization.NumberStyles.Float | System.Globalization.NumberStyles.AllowThousands, null ).

The string s is parsed using the formatting information in a System.Globalization.NumberFormatInfo initialized for the current system culture.

[Note: For more information, see System.Globalization.NumberFormatInfo.CurrentInfo.]

## Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	s is a null reference.
<b>System.FormatException</b>	s is not in the correct style.
<b>System.OverflowException</b>	s represents a value that is less than <code>System.Double.MinValue</code> or greater than <code>System.Double.MaxValue</code> .

1  
2  
3

# Double.Parse(System.String, System.Globalization.NumberStyles) Method

```
[ILAsm]  
.method public hidebysig static float64 Parse(string s, valuetype  
System.Globalization.NumberStyles style)  
  
[C#]  
public static double Parse(string s, NumberStyles style)
```

## Summary

Returns the specified `System.String` converted to a `System.Double` value.

## Parameters

Parameter	Description
<code>s</code>	A <code>System.String</code> containing the value to convert. The string is interpreted using the style specified by <i>style</i> .
<code>style</code>	Zero or more <code>System.Globalization.NumberStyles</code> values that specify the style of <i>s</i> . Specify multiple values for <i>style</i> using the bitwise OR operator. If <i>style</i> is a null reference, the string is interpreted using the <code>System.Globalization.NumberStyles.Float</code> and <code>System.Globalization.NumberStyles.AllowThousands</code> styles.

## Return Value

The `System.Double` value obtained from *s*. If *s* equals `System.Globalization.NumberFormatInfo.NaNSymbol`, this method returns `System.Double.NaN`.

## Description

This version of `System.Double.Parse` is equivalent to `System.Double.Parse(s, style, null)`.

The string *s* is parsed using the formatting information in a `System.Globalization.NumberFormatInfo` initialized for the current system culture.

[*Note:* For more information, see `System.Globalization.NumberFormatInfo.CurrentInfo`.]

### Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	s is a null reference.
<b>System.FormatException</b>	s is not in the correct style.
<b>System.OverflowException</b>	s represents a value that is less than <code>System.Double.MinValue</code> or greater than <code>System.Double.MaxValue</code> .

# Double.Parse(System.String, System.IFormatProvider) Method

```
[ILAsm]  
.method public hidebysig static float64 Parse(string s, class  
System.IFormatProvider provider)  
  
[C#]  
public static double Parse(string s, IFormatProvider provider)
```

## Summary

Returns the specified `System.String` converted to a `System.Double` value.

## Parameters

Parameter	Description
<code>s</code>	A <code>System.String</code> containing the value to convert. The string is interpreted using the <code>System.Globalization.NumberStyles.Float</code> and/or <code>System.Globalization.NumberStyles.AllowThousands</code> style.
<code>provider</code>	A <code>System.IFormatProvider</code> that supplies a <code>System.Globalization.NumberFormatInfo</code> containing culture-specific formatting information about <code>s</code> .

## Return Value

The `System.Double` value obtained from `s`. If `s` equals `System.Globalization.NumberFormatInfo.NaNSymbol`, this method returns `System.Double.NaN`.

## Description

This version of `System.Double.Parse` is equivalent to `System.Double.Parse(s, System.Globalization.NumberStyles.Float | System.Globalization.NumberStyles.AllowThousands, provider)`.

The string `s` is parsed using the culture-specific formatting information from the `System.Globalization.NumberFormatInfo` instance supplied by `provider`. If `provider` is null or a `System.Globalization.NumberFormatInfo` cannot be obtained from `provider`, the formatting information for the current system culture is used.

## Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	s is a null reference.
<b>System.FormatException</b>	s is not in the correct style.
<b>System.OverflowException</b>	s represents a value that is less than <code>System.Double.MinValue</code> or greater than <code>System.Double.MaxValue</code> .

# Double.Parse(System.String, System.Globalization.NumberStyles, System.IFormatProvider) Method

```
[ILAsm]  
.method public hidebysig static float64 Parse(string s, valuetype  
System.Globalization.NumberStyles style, class System.IFormatProvider  
provider)  
  
[C#]  
public static double Parse(string s, NumberStyles style, IFormatProvider  
provider)
```

## Summary

Returns the specified `System.String` converted to a `System.Double` value.

## Parameters

Parameter	Description
<i>s</i>	A <code>System.String</code> containing the value to convert. The string is interpreted using the style specified by <i>style</i> .
<i>style</i>	Zero or more <code>System.Globalization.NumberStyles</code> values that specify the style of <i>s</i> . Specify multiple values for <i>style</i> using the bitwise OR operator. If <i>style</i> is a null reference, the string is interpreted using the <code>System.Globalization.NumberStyles.Float</code> and <code>System.Globalization.NumberStyles.AllowThousands</code> styles.
<i>provider</i>	A <code>System.IFormatProvider</code> that supplies a <code>System.Globalization.NumberFormatInfo</code> containing culture-specific formatting information about <i>s</i> .

## Return Value

The `System.Double` value obtained from *s*. If *s* equals `System.Globalization.NumberFormatInfo.NaNString`, this method returns `System.Double.NaN`.

## Description

The string *s* is parsed using the culture-specific formatting information from the `System.Globalization.NumberFormatInfo` instance supplied by *provider*. If *provider* is null or a `System.Globalization.NumberFormatInfo` cannot be obtained from *provider*, the formatting information for the current system culture is used.

**Exceptions**

Exception	Condition
<b>System.ArgumentNullException</b>	<i>s</i> is a null reference
<b>System.FormatException</b>	<i>s</i> is not in the correct style.
<b>System.OverflowException</b>	<i>s</i> represents a value that is less than <code>System.Double.MinValue</code> or greater than <code>System.Double.MaxValue</code> .

# Double.ToString(System.String, System.IFormatProvider) Method

```
[ILAsm]  
.method public final hidebysig virtual string ToString(string format,  
class System.IFormatProvider provider)  
  
[C#]  
public string ToString(string format, IFormatProvider provider)
```

## Summary

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

## Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> containing a character that specifies the format of the returned string, optionally followed by a non-negative integer that specifies the precision of the number in the returned <code>System.String</code> .
<i>provider</i>	A <code>System.IFormatProvider</code> that supplies a <code>System.Globalization.NumberFormatInfo</code> instance containing culture-specific formatting information.

## Return Value

A `System.String` representation of the current instance formatted as specified by *format*. The string takes into account the information in the `System.Globalization.NumberFormatInfo` instance supplied by *provider*.

## Description

If *provider* is null or a `System.Globalization.NumberFormatInfo` cannot be obtained from *provider*, the formatting information for the current system culture is used.

If *format* is a null reference, the general format specifier "G" is used.

The following table lists the format characters that are valid for the `System.Double` type.

Format Characters	Description
"C", "c"	Currency format.
"E", "e"	Exponential notation format.
"F", "f"	Fixed-point format.
"G", "g"	General format.
"N", "n"	Number format.
"P", "p"	Percent format.
"R", "r"	Round-trip format.

[*Note:* For a detailed description of formatting, see the `System.IFormattable` interface.

This method is implemented to support the `System.IFormattable` interface.

]

## Exceptions

Exception	Condition
<code>System.FormatException</code>	<i>format</i> is invalid.

# Double.ToString(System.IFormatProvider) Method

```
[ILAsm]  
.method public final hidebysig virtual string ToString(class  
System.IFormatProvider provider)  
  
[C#]  
public string ToString(IFormatProvider provider)
```

## Summary

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

## Parameters

Parameter	Description
<i>provider</i>	A <code>System.IFormatProvider</code> that supplies a <code>System.Globalization.NumberFormatInfo</code> containing culture-specific formatting information.

## Return Value

A `System.String` representation of the current instance formatted using the general format specifier, ("G"). The string takes into account the formatting information in the `System.Globalization.NumberFormatInfo` instance supplied by *provider*.

## Description

This version of `System.Double.ToString` is equivalent to `System.Double.ToString(null, provider)`.

If *provider* is `null` or a `System.Globalization.NumberFormatInfo` cannot be obtained from *provider*, the formatting information for the current system culture is used.

[*Note:* The general format specifier formats the number in either fixed-point or exponential notation form. For a detailed description of the general format, see the `System.IFormattable` interface.]

# Double.ToString() Method

```
[ILAsm]  
.method public hidebysig virtual string ToString()  
  
[C#]  
public override string ToString()
```

## Summary

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

## Return Value

A `System.String` representation of the current instance formatted using the general format specifier, ("G"). The string takes into account the current system culture.

## Description

This version of `System.Double.ToString` is equivalent to `System.Double.ToString (null, null )`.

[*Note:* The general format specifier formats the number in either fixed-point or exponential notation form. For a detailed description of the general format, see the `System.IFormattable` interface.

This method overrides `System.Object.ToString`.

]

# Double.ToString(System.String) Method

```
[ILAsm]  
.method public hidebysig instance string ToString(string format)  
  
[C#]  
public string ToString(string format)
```

## Summary

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

## Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> that specifies the format of the returned string. [ <i>Note:</i> For a list of valid values, see <code>System.Double.ToString (System.String, System.IFormatProvider )</code> .]

## Return Value

A `System.String` representation of the current instance formatted as specified by *format*. The string takes into account the current system culture.

## Description

This version of `System.Double.ToString` is equivalent to `System.Double.ToString (format, null )`.

If *format* is a null reference, the general format specifier "G" is used.

## Exceptions

Exception	Condition
<code>System.FormatException</code>	<i>format</i> is invalid.

## Example

The following example shows the effects of various formats on the string returned by `System.Double.ToString`.

[C#]

```
using System;
class test {
    public static void Main() {
        double d = 1234.56789;
        Console.WriteLine(d);
        string[] fmts = {"C", "E", "e5", "F", "G", "N", "P", "R"};
        for (int i=0; i<fmts.Length; i++)
            Console.WriteLine("{0}: {1}",
                               fmts[i], d.ToString(fmts[i]));
    }
}
```

The output is

1234.56789

C: \$1,234.57

E: 1.234568E+003

e5: 1.23457e+003

F: 1234.57

G: 1234.56789

N: 1,234.57

P: 123,456.79 %

R: 1234.56789

