

# System.Double Structure

```
[ILASM]
.class public sequential sealed serializable Double extends
System.ValueType implements System.IComparable,
System.IFormattable

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

## Assembly Info:

- Name: mscorlib
- Public Key: [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- Version: 1.0.x.x
- Attributes:
  - CLSCompliantAttribute(true)

## Implements:

- System.IComparable
- System.IFormattable

## Summary

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

## Inherits From: System.ValueType

**Library:** ExtendedNumerics

**Thread Safety:** This type is 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:

- The finite set of non-zero values of the form  $s * m * 2^e$ , where  $s$  is 1 or -1, and  $0 < m < 2^{53}$  and  $-1075 \leq e \leq 970$ .

- Positive infinity and negative infinity. Infinities are produced by operations that produce results with a magnitude greater than that which can be represented by a **System.Double**, such as dividing a non-zero number by zero. For example, using **System.Double** operands,  $1.0 / 0.0$  yields positive infinity, and  $-1.0 / 0.0$  yields negative infinity. Operations include passing parameters and returning values.
- The *Not-a-Number* value (NaN). NaN values are produced by invalid floating-point operations, such as dividing zero by zero.

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

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

- If the result of a floating-point operation is too small for the destination format, the result of the operation is zero.
- If the magnitude of the result of a floating-point operation is too large for the destination format, the result of the operation is positive infinity or negative infinity, as appropriate for the sign of the result.
- If a floating-point operation is invalid, the result of the operation is NaN.
- If one or both operands of a floating-point operation are NaN, the result of the operation is NaN.

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

# 1 Double.Epsilon Field

```
2 [ILASM]  
3 .field public static literal float64 Epsilon =  
4 4.9406564584124654e-324  
  
5 [C#]  
6 public const double Epsilon = 4.9406564584124654e-324
```

## 7 Summary

8 Represents the smallest positive **System.Double** value greater than  
9 zero.

## 10 Description

11 The value of this constant is 4.9406564584124654E-324.

12

# 1 Double.MaxValue Field

```
2 [ILASM]  
3 .field public static literal float64 MaxValue =  
4 1.7976931348623157e+308  
  
5 [C#]  
6 public const double MaxValue = 1.7976931348623157e+308
```

## 7 Summary

8 Contains the maximum positive value for the **System.Double** type.

## 9 Description

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

11

# 1 Double.MinValue Field

```
2 [ILASM]  
3 .field public static literal float64 MinValue = -  
4 1.7976931348623157e+308  
  
5 [C#]  
6 public const double MinValue = -1.7976931348623157e+308
```

## 7 Summary

8 Contains the minimum (most negative) value for the **System.Double**  
9 type.

## 10 Description

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

12

# 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.]

# 1 Double.PositiveInfinity Field

```
2 [ILASM]
3 .field public static literal float64 PositiveInfinity =
4 (double)1.0 / (double)(0.0)

5 [C#]
6 public const double PositiveInfinity = (double)1.0 /
7 (double)(0.0)
```

## 8 Summary

9 Represents a positive infinity of type **System.Double**.

## 10 Description

11 The value of this constant is obtained by dividing a positive  
12 **System.Double** by zero.

13  
14 [Note: To test whether a **System.Double** value is a positive infinity  
15 value, use the **System.Double.IsPositiveInfinity** method.]

16



# 1 Double.CompareTo(System.Object)

## 2 Method

```
3 [ILASM]
4 .method public final hidebysig virtual int32
5 CompareTo(object value)
6
7 [C#]
8 public int CompareTo(object value)
```

### 8 Summary

9 Returns the sort order of the current instance compared to the  
10 specified **System.Object**.

### 11 Parameters

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

### 15 Return Value

17 A **System.Int32** containing a value that reflects the sort order of the  
18 current instance as compared to *value*. The following table defines the  
19 conditions under which the returned value is a negative number, zero,  
20 or a positive number.

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

## 1 Description

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

## 7 Exceptions

8  
9

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

10  
11  
12

# 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 <b>System.Object</b> 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**.]

# 1 Double.GetHashCode() Method

```
2 [ILASM]  
3 .method public hidebysig virtual int32 GetHashCode()  
4  
5 [C#]  
6 public override int GetHashCode()
```

## 6 Summary

7 Generates a hash code for the current instance.

## 8 Return Value

9

10 A **System.Int32** containing the hash code for this instance.

## 11 Description

12 The algorithm used to generate the hash code is unspecified.

13

14 [*Note:* This method overrides **System.Object.GetHashCode.**]

15

# 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 <b>System.Double</b> 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 <b>System.Double</b> 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.]

# 1 Double.IsNegativeInfinity(System.Double 2 ) Method

```
3 [ILASM]  
4 .method public hidebysig static bool  
5 IsNegativeInfinity(float64 d)  
  
6 [C#]  
7 public static bool IsNegativeInfinity(double d)
```

## 8 Summary

9 Determines whether the specified **System.Double** represents a  
10 negative infinity value.

## 11 Parameters

12  
13

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

14

## 15 Return Value

16

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

## 18 Description

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

21

# 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 <b>System.Double</b> 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 <b>System.String</b> containing the value to convert. The string is interpreted using the <b>System.Globalization.NumberStyles.Float</b> and/or <b>System.Globalization.NumberStyles.AllowThousands</b> style.

## Return Value

The **System.Double** value obtained from s. If the parsed value is less than **System.Double.MinValue**, this method returns **System.Double.NegativeInfinity**. If the parsed value is greater than **System.Double.MaxValue**, this method returns **System.Double.PositiveInfinity**. 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.

1  
2  
3

<b>System.FormatException</b>	s is not in the correct style.
-------------------------------	--------------------------------

# 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
<i>s</i>	A <b>System.String</b> containing the value to convert. The string is interpreted using the style specified by <i>style</i> .
<i>style</i>	Zero or more <b>System.Globalization.NumberStyles</b> 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 <b>System.Globalization.NumberStyles.Float</b> and <b>System.Globalization.NumberStyles.AllowThousands</b> styles.

## Return Value

The **System.Double** value obtained from *s*. If the parsed value is less than **System.Double.MinValue**, this method returns **System.Double.NegativeInfinity**. If the parsed value is greater than **System.Double.MaxValue**, this method returns **System.Double.PositiveInfinity**. 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

1 system culture. [Note: For more information, see  
2 **System.Globalization.NumberFormatInfo.CurrentInfo.**]

3 **Exceptions**  
4  
5

Exception	Condition
<b>System.ArgumentNullException</b>	s is a null reference.
<b>System.FormatException</b>	s is not in the correct style.

6  
7  
8

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

## Return Value

The **System.Double** value obtained from *s*. If the parsed value is less than **System.Double.MinValue**, this method returns **System.Double.NegativeInfinity**. If the parsed value is greater than **System.Double.MaxValue**, this method returns **System.Double.PositiveInfinity**. 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

1       **System.Globalization.NumberFormatInfo** cannot be obtained from  
2       *provider*, the formatting information for the current system culture is  
3       used.

4       **Exceptions**

5  
6

Exception	Condition
<b>System.ArgumentNullException</b>	s is a null reference.
<b>System.FormatException</b>	s is not in the correct style.

7  
8  
9

# 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 <b>System.String</b> containing the value to convert. The string is interpreted using the style specified by <i>style</i> .
<i>style</i>	Zero or more <b>System.Globalization.NumberStyles</b> 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 <b>System.Globalization.NumberStyles.Float</b> and <b>System.Globalization.NumberStyles.AllowThousands</b> styles.
<i>provider</i>	A <b>System.IFormatProvider</b> that supplies a <b>System.Globalization.NumberFormatInfo</b> containing culture-specific formatting information about <i>s</i> .

## Return Value

The **System.Double** value obtained from *s*. If the parsed value is less than **System.Double.MinValue**, this method returns **System.Double.NegativeInfinity**. If the parsed value is greater than **System.Double.MaxValue**, this method returns **System.Double.PositiveInfinity**. If *s* equals **System.Globalization.NumberFormatInfo.NaNSymbol**, this method returns **System.Double.NaN**.

## Description

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

7 **Exceptions**  
8  
9

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.

10  
11  
12



# 1 Double.ToString(System.IFormatProvider)

## 2 Method

```
3 [ILASM]
4 .method public final hidebysig virtual string
5 ToString(class System.IFormatProvider provider)
6
7 [C#]
8 public string ToString(IFormatProvider provider)
```

### 8 Summary

9 Returns a **System.String** representation of the value of the current  
10 instance.

### 11 Parameters

12  
13

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

14  
15  
16

### Return Value

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

### 22 Description

23 This version of **System.Double.ToString** is equivalent to  
24 **System.Double.ToString (null, provider)**.

25  
26 If *provider* is **null** or a **System.Globalization.NumberFormatInfo**  
27 cannot be obtained from *provider*, the formatting information for the  
28 current system culture is used.

29  
30 [Note: The general format specifier formats the number in either fixed-  
31 point or exponential notation form. For a detailed description of the  
32 general format, see the **System.IFormattable** interface.]

33

# 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 <b>System.String</b> 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 <b>System.String</b> .
<i>provider</i>	A <b>System.IFormatProvider</b> that supplies a <b>System.Globalization.NumberFormatInfo</b> 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.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

[*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
<b>System.FormatException</b>	<i>format</i> is invalid.

# 1 Double.ToString() Method

```
2 [ILASM]  
3 .method public hidebysig virtual string ToString()  
  
4 [C#]  
5 public override string ToString()
```

## 6 Summary

7 Returns a **System.String** representation of the value of the current  
8 instance.

## 9 Return Value

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

## 14 Description

15 This version of **System.Double.ToString** is equivalent to  
16 **System.Double.ToString (null, null)**.

17  
18 [Note: The general format specifier formats the number in either fixed-  
19 point or exponential notation form. For a detailed description of the  
20 general format, see the **System.IFormattable** interface.

21  
22 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 <b>System.String</b> that specifies the format of the returned string. [Note: For a list of valid values, see <b>System.Double.ToString (System.String, System.IFormatProvider)</b> .]

## 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
<b>System.FormatException</b>	<i>format</i> is invalid.

## Example

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

```

1
2    [C#]

3    using System;
4    class test {
5        public static void Main() {
6            double d = 1234.56789;
7            Console.WriteLine(d);
8            string[] fmts = {"C", "E", "e5", "F", "G", "N", "P", "R"};
9            for (int i=0;i<fmts.Length;i++)
10               Console.WriteLine("{0}: {1}",
11                                   fmts[i],d.ToString(fmts[i]));
12        }
13    }
14

```

15 The output is

16  
17 1234.56789

18  
19  
20 C: \$1,234.57

21  
22  
23 E: 1.234568E+003

24  
25  
26 e5: 1.23457e+003

27  
28  
29 F: 1234.57

30  
31  
32 G: 1234.56789

1	
2	
3	N: 1,234.57
4	
5	
6	P: 123,456.79 %
7	
8	
9	R: 1234.56789
10	
11	