

# 1 System.String Class

```
2 [ILAsm]
3 .class public sealed serializable String extends System.Object implements
4 System.ICloneable, System.IComparable, System.Collections.IEnumerable,
5 System.IComparable`1<string>, System.IEquatable`1<string>,
6 System.Collections.Generic.IEnumerable`1<char>
7
8 [C#]
9 public sealed class String: ICloneable, IComparable, IEnumerable,
10 IComparable<String>, IEquatable<String>, IEnumerable<Char>
```

## 10 Assembly Info:

- 11 • *Name:* mscorlib
- 12 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 13 • *Version:* 2.0.x.x
- 14 • *Attributes:*
  - 15 ○ CLSCompliantAttribute(true)

## 16 Type Attributes:

- 17 • DefaultMemberAttribute("Chars") [*Note:* This attribute requires the
- 18 RuntimeInfrastructure library.]

## 19 Implements:

- 20 • System.IComparable
- 21 • System.ICloneable
- 22 • System.IComparable<System.String>
- 23 • System.IEquatable<System.String>
- 24 • System.Collections.IEnumerable
- 25 • System.Collections.Generic.IEnumerable<System.Char>

## 26 Summary

27 Represents an immutable series of characters.

## 28 Inherits From: System.Object

29

30 **Library:** BCL

31

32 **Thread Safety:** This type is safe for multithreaded operations.

33

## 34 Description

35 An *index* is the position of a character within a string. The first character in the string is  
36 at index 0. The length of a string is the number of characters it is made up of. The last  
37 accessible *index* of a string instance is `System.String.Length - 1`.

1  
2 Strings are immutable; once created, the contents of a `System.String` do not change.  
3 Combining operations, such as `System.String.Replace`, cannot alter existing strings.  
4 Instead, such operations return a new string that contains the results of the operation,  
5 an unchanged string, or the null value. To perform modifications to a `System.String`  
6 use the `System.Text.StringBuilder`.

7  
8 Implementations of `System.String` are required to contain a variable-length character  
9 buffer positioned a fixed number of bytes after the beginning of the String object. [*Note:*  
10 The `System.Runtime.CompilerServices.RuntimeHelpers.OffsetToStringData`  
11 method returns the number of bytes between the start of the String object and the  
12 character buffer. This information is intended primarily for use by compilers, not  
13 application programmers. For additional information, see  
14 `System.Runtime.CompilerServices.RuntimeHelpers.OffsetToStringData`.]

15  
16  
17  
18 [*Note:* Comparisons and searches are case-sensitive by default, and unless otherwise  
19 specified, use the culture defined (if any) for the current thread to determine the order  
20 of the alphabet used by the strings. This information is then used to compare the two  
21 strings on a character-by-character basis. Upper case letters evaluate greater than their  
22 lowercase equivalents.

23  
24 The following characters are considered white space when present in a `System.String`  
25 instance: `0x9`, `0xA`, `0xB`, `0xC`, `0xD`, `0x20`, `0xA0`, `0x2000`, `0x2001`, `0x2002`, `0x2003`,  
26 `0x2004`, `0x2005`, `0x2006`, `0x2007`, `0x2008`, `0x2009`, `0x200A`, `0x200B`, `0x3000`, and  
27 `0xFEFF`. The null character is defined as hexadecimal `0x00`.

28  
29 The `System.String(System.String)` constructor is omitted for performance reasons. If  
30 you need a copy of a `System.String`, consider using `System.String.Copy` or the  
31 `System.Text.StringBuilder` class.

32  
33 To insert a formatted string representation of an object into a string, use the  
34 `System.String.Format` methods. These methods take one or more arguments to be  
35 formatted, and a format string. The format string contains literals and zero or more  
36 format specifications of the form `{ N [, M] [: formatSpecifier] }`, where:

- 37 • *N* is a zero-based integer indicating the argument to be formatted. If the actual  
38 argument is a null reference, then an empty string is used in its place.
- 39 • *M* is an optional integer indicating the minimum width of the region to contain the  
40 formatted value of argument *N*. If the length of the string representation of the value  
41 is less than *M*, then the region is padded with spaces. If *M* is negative, the formatted  
42 value is left justified in the region; if *M* is positive, then the value is right justified. If  
43 *M* is not specified, it is assumed to be zero indicating that neither padding nor  
44 alignment is customized. Note that if the length of the formatted value is greater  
45 than *M*, then *M* is ignored.
- 46 • *formatSpecifier* is an optional string that determines the representation used for  
47 arguments. For example, an integer can be represented in hexadecimal or decimal  
48 format, or as a monetary value. If *formatSpecifier* is omitted and an argument  
49 implements the `System.IFormattable` interface, then a null reference is used as the

1        `System.IFormattable.ToString` format specifier. Therefore, all implementations of  
2        `System.IFormattable.ToString` are required to allow a null reference as a format  
3        specifier, and return a string containing the default representation of the object as  
4        determined by the object type. For additional information on format specifiers, see  
5        `System.IFormattable`.

6    If an object referenced in the format string implements `System.IFormattable`, then the  
7    `System.IFormattable.ToString` method of the object provides the formatting. If the  
8    argument does not implement `System.IFormattable`, then the `System.Object.ToString`  
9    method of the object provides default formatting, and *formatSpecifier*, if present, is ignored.  
10   For an example that demonstrates this, see Example 2.

11  
12   To include a curly bracket in a formatted string, specify the bracket twice; for example,  
13   specify "{{" to include "{" in the formatted string. See Example 1.

14  
15   The `System.Console` class exposes the same functionality as the `System.String.Format`  
16   methods via `System.Console.Write` and `System.Console.WriteLine`. The primary  
17   difference is that the `System.String.Format` methods return the formatted string, while the  
18   `System.Console` methods write the formatted string to a stream.

19  
20   ]

21  
22   When a non-empty string is searched for the first or last occurrence of an empty string, the  
23   empty string is found at the search start position.

## 24   **Example**

25        Example 1

26  
27        The following example demonstrates formatting numeric data types and inserting literal  
28        curly brackets into strings.

29        [C#]  
30  
31        `using System;`  
32        `class StringFormatTest {`  
33            `public static void Main() {`  
34                `decimal dec = 1.99999m;`  
35                `double doub = 1.0000000001;`  
36  
37                `string somenums = String.Format("Some formatted numbers: dec={0,15:E}`  
38        `doub={1,20}", dec, doub);`  
39                `Console.WriteLine(somenums);`  
40  
41                `string curlies = "Literal curly brackets: {{ and }} and {{0}}";`  
42                `Console.WriteLine(curlies);`  
43  
44                `object nullObject = null;`  
45                `string embeddedNull = String.Format("A null argument looks like:`  
46        `{0}", nullObject);`  
47                `Console.WriteLine(embeddedNull);`

```
1     }
2 }
3
```

4 The output is

```
5
6 Some formatted numbers: dec= 1.999990E+000 doub= 1.0000000001
7 Literal curly brackets: {{ and }} and {{0}}
8 A null argument looks like:
9
```

10 Example 2

11

12 The following example demonstrates how formatting works if `System.IFormattable` is or is  
13 not implemented by an argument to the `System.String.Format` method. Note that the  
14 format specifier is ignored if the argument does not implement `System.IFormattable`.

15

16 [C#]

```
17 using System;
18 class StringFormatTest {
19     public class DefaultFormatEleven {
20         public override string ToString() {
21             return "11 string";
22         }
23     }
24     public class FormattableEleven:IFormattable {
25         // The IFormattable ToString implementation.
26         public string ToString(string format, IFormatProvider formatProvider)
27     {
28         Console.Write("[IFormattable called] ");
29         return 11.ToString(format, formatProvider);
30     }
31     // Override Object.ToString to show that it is not called.
32     public override string ToString() {
33         return "Formatted 11 string";
34     }
35 }
36
37     public static void Main() {
38         DefaultFormatEleven def11 = new DefaultFormatEleven ();
39         FormattableEleven for11 = new FormattableEleven();
40         string def11string = String.Format("{0}",def11);
41         Console.WriteLine(def11string);
42         // The format specifier x is ignored.
43         def11string = String.Format("{0,15:x}", def11);
44         Console.WriteLine(def11string);
45
46         string form11string = String.Format("{0}",for11);
47         Console.WriteLine(form11string );
48         form11string = String.Format("{0,15:x}",for11);
49         Console.WriteLine(form11string);
50     }
51 }
```

52 The output is

```
1
2 11 string
3     11 string
4 [IFormattable called] 11
5 [IFormattable called]          b
6
```

### 7 Example 3

8  
9 The following example demonstrates searching for an empty string in a non-empty string.

```
10
11 [C#]
12 using System;
13 class EmptyStringSearch {
14     public static void Main() {
15         Console.WriteLine("ABCDEF".IndexOf(""));
16         Console.WriteLine("ABCDEF".IndexOf("", 2));
17         Console.WriteLine("ABCDEF".IndexOf("", 3, 2));
18         Console.WriteLine("ABCDEF".LastIndexOf(""));
19         Console.WriteLine("ABCDEF".LastIndexOf("", 1));
20         Console.WriteLine("ABCDEF".LastIndexOf("", 4, 2));
21     }
22 }
```

23 The output is

```
24 0
25 2
26 3
27 5
28 1
29 4
30
```

# 1 String(System.Char, System.Int32)

## 2 Constructor

```
3 [ILAsm]  
4 public rtspecialname specialname instance void .ctor(valuetype System.Char  
5 c, int32 count)  
  
6 [C#]  
7 public String(char c, int count)
```

### 8 Summary

9 Constructs and initializes a new instance of System.String.

### 10 Parameters

Parameter	Description
<i>c</i>	A System.Char.
<i>count</i>	A System.Int32 containing the number of occurrences of <i>c</i> .

### 11 Description

13 If the specified number is 0, System.String.Empty is created.

### 14 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>count</i> is less than zero.

### 15 Example

17 The following example demonstrates using this constructor.

```
18 [C#]  
19  
20 using System;  
21  
22 public class StringExample {  
23     public static void Main() {  
24  
25         string s = new String('a', 10);  
26     }  
}
```

```
1 Console.WriteLine(s);
2 }
3 }
4
5 The output is
6
7 aaaaaaaaaa
8
```

# 1 String(System.Char\*) Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor(char* value)  
4 [C#]  
5 unsafe public String(char* value)
```

## 6 Summary

7 Constructs and initializes a new instance of `System.String` using a specified pointer to a  
8 sequence of Unicode characters.

## 9 Type Attributes:

- 10 • `CLSCompliantAttribute(false)`

## 11 Parameters

Parameter	Description
<i>value</i>	A pointer to a null-terminated array of Unicode characters. If <i>value</i> is a null pointer, <code>System.String.Empty</code> is created.

12

## 13 Description

14 This member is not CLS-compliant. For a CLS-compliant alternative, use the  
15 `System.String(System.Char[])` constructor.

16

17 This constructor copies the sequence of Unicode characters at the specified pointer until  
18 a null character (hexadecimal 0x00) is reached.

19

20 If the specified array is not null-terminated, the behavior of this constructor is system  
21 dependent. For example, such a situation might cause an access violation.

22

23 [*Note:* In C# this constructor is defined only in the context of unmanaged code.]

24

25

26

# 1 String(System.Char[]) Constructor

```
2 [ILAsm]  
3 public rtspecialname specialname instance void .ctor(char[] value)  
4 [C#]  
5 public String(char[] value)
```

## 6 Summary

7 Constructs and initializes a new instance of `System.String` by copying the specified  
8 array of Unicode characters.

## 9 Parameters

Parameter	Description
<i>value</i>	An array of Unicode characters.

10

## 11 Description

12 If the specified array is a null reference or contains no elements, `System.String.Empty`  
13 is created.

14

# 1 String(System.Char\*, System.Int32, 2 System.Int32) Constructor

```
3 [ILAsm]  
4 public rtspecialname specialname instance void .ctor(char* value, int32  
5 startIndex, int32 length)  
  
6 [C#]  
7 unsafe public String(char* value, int startIndex, int length)
```

## 8 Summary

9 Constructs and initializes a new instance of `System.String` using a specified pointer to a  
10 sequence of Unicode characters, the index within that sequence at which to start  
11 copying characters, and the number of characters to be copied to construct the  
12 `System.String`.

## 13 Type Attributes:

- 14 • `CLSCompliantAttribute(false)`

## 15 Parameters

Parameter	Description
<i>value</i>	A pointer to an array of Unicode characters.
<i>startIndex</i>	A <code>System.Int32</code> containing the index within the array referenced by <i>value</i> from which to start copying.
<i>length</i>	A <code>System.Int32</code> containing the number of characters to copy from <i>value</i> to the new <code>System.String</code> . If <i>length</i> is zero, <code>System.String.Empty</code> is created.

16

## 17 Description

18 This member is not CLS-compliant. For a CLS-compliant alternative, use the  
19 `System.String(System.Char, System.Int32, System.Int32)` constructor.

20  
21 This constructor copies Unicode characters from *value*, starting at *startIndex* and ending  
22 at (*startIndex* + *length* - 1).

23  
24 If the specified range is outside of the memory allocated for the sequence of characters,  
25 the behavior of this constructor is system dependent. For example, such a situation  
26 might cause an access violation.

27  
28 [*Note:* In C# this constructor is defined only in the context of unmanaged code.]

1  
2

### 3 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>length</i> is less than zero. -or- <i>value</i> is a null pointer and <i>length</i> is not zero.

4  
5

1 **The following member must be implemented if the RuntimeInfrastructure library is**  
2 **present in the implementation.**

## 3 String(System.SByte\*, System.Int32, 4 System.Int32, System.Text.Encoding) 5 Constructor

```
6 [ILAsm]  
7 public rtspecialname specialname instance void .ctor(class System.SByte*  
8 value, int32 startIndex, int32 length, class System.Text.Encoding enc)  
9  
10 [C#]  
11 unsafe public String(sbyte* value, int startIndex, int length, Encoding  
enc)
```

### 12 Summary

13 Constructs and initializes a new instance of the `String` class to the value indicated by a  
14 specified pointer to an array of 8-bit signed integers, a starting character position within  
15 that array, a length, and an `Encoding` object.

### 16 Type Attributes:

- 17 • `CLSCompliantAttribute(false)`

### 18 Parameters

Parameter	Description
<i>value</i>	A pointer to a <code>System.SByte</code> array.
<i>startIndex</i>	A <code>System.Int32</code> containing the starting position within <i>value</i> .
<i>length</i>	A <code>System.Int32</code> containing the number of characters within <i>value</i> to use. If <i>length</i> is zero, <code>System.String.Empty</code> is created.
<i>enc</i>	A <code>System.Text.Encoding</code> object that specifies how the array referenced by <i>value</i> is encoded.

### 19 Description

20 If *value* is a null pointer, a `System.String.Empty` instance is constructed.

### 22 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>length</i> is less than zero.  -or-  <i>value</i> is a null pointer and <i>length</i> is not zero.

1

2

# 1 String(System.Char[], System.Int32, 2 System.Int32) Constructor

```
3 [ILAsm]  
4 public rtspecialname specialname instance void .ctor(char[] value, int32  
5 startIndex, int32 length)  
  
6 [C#]  
7 public String(char[] value, int startIndex, int length)
```

## 8 Summary

9 Constructs and initializes a new instance of `System.String` using an array of Unicode  
10 characters, the index within array at which to start copying characters, and the number  
11 of characters to be copied.

## 12 Parameters

Parameter	Description
<i>value</i>	An array of Unicode characters.
<i>startIndex</i>	A <code>System.Int32</code> containing the index within the array referenced by <i>value</i> from which to start copying.
<i>length</i>	A <code>System.Int32</code> containing the number of characters to copy from the <i>value</i> array. If <i>length</i> is zero, <code>System.String.Empty</code> is created.

13

## 14 Description

15 This constructor copies the sequence Unicode characters found at *value* between  
16 indexes *startIndex* and *startIndex* + *length* - 1.

## 17 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>value</i> is a null reference.
<code>System.ArgumentOutOfRangeException</code>	<i>startIndex</i> or <i>length</i> is less than zero. -or- The sum of <i>startIndex</i> and <i>length</i> is greater

than the number of elements in *value*.

1

2

# 1 String.Empty Field

```
2 [ILAsm]  
3 .field public static initOnly string Empty  
4 [C#]  
5 public static readonly string Empty
```

## 6 Summary

7 A constant string representing the empty string.

## 8 Description

9 This field is read-only.

10

11 This field is a string of length zero, "".

12

# 1 String.Clone() Method

```
2 [ILAsm]  
3 .method public final hidebysig virtual object Clone()  
4 [C#]  
5 public object Clone()
```

## 6 Summary

7 Returns a reference to the current instance of `System.String`.

## 8 Return Value

9 A reference to the current instance of `System.String`.

## 10 Description

11 [*Note:* `System.String.Clone` does not generate a new `System.String` instance. Use  
12 the `System.String.Copy` or `System.String.CopyTo` method to create a separate  
13 `System.String` object with the same value as the current instance.

14 This method is implemented to support the `System.ICloneable` interface.

15 ]  
16 ]  
17 ]

18

# 1 String.Compare(System.String, 2 System.Int32, System.String, System.Int32, 3 System.Int32, System.Boolean) Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 Compare(string strA, int32 indexA,  
6 string strB, int32 indexB, int32 length, bool ignoreCase)  
  
7 [C#]  
8 public static int Compare(string strA, int indexA, string strB, int  
9 indexB, int length, bool ignoreCase)
```

## 10 Summary

11 Compares substrings of two strings, ignoring or honoring their case.

## 12 Parameters

Parameter	Description
<i>strA</i>	The first <code>System.String</code> containing a substring to compare. Can be a null reference.
<i>indexA</i>	A <code>System.Int32</code> containing the starting index of the substring within <i>strA</i> .
<i>strB</i>	The second <code>System.String</code> containing a substring to compare. Can be a null reference.
<i>indexB</i>	A <code>System.Int32</code> containing the starting index of the substring within <i>strB</i> .
<i>length</i>	A <code>System.Int32</code> containing the maximum number of characters in the substrings to compare. If <i>length</i> is zero, then zero is returned.
<i>ignoreCase</i>	A <code>System.Boolean</code> indicating if the comparison is case-insensitive. If <i>ignoreCase</i> is <code>true</code> , the comparison is case-insensitive. If <i>ignoreCase</i> is <code>false</code> , the comparison is case-sensitive, and uppercase letters evaluate greater than their lowercase equivalents.

13

## 14 Return Value

15 The return value is a negative number, zero, or a positive number reflecting the sort  
16 order of the specified substrings. For non-zero return values, the exact value returned  
17 by this method is unspecified. The following table defines the return value:

Value Type	Condition
A negative number	The substring in <i>strA</i> is < the substring in <i>strB</i> .
Zero	The substring in <i>strA</i> == the substring in <i>strB</i> , or <i>length</i> is zero.
A positive number	The substring in <i>strA</i> is > the substring in <i>strB</i> .

1

## 2 Description

3 [Note: The result of comparing any `System.String` (including the empty string) to a null  
4 reference is greater than zero. The result of comparing two null references is zero.  
5 Uppercase letters evaluate greater than their lower case equivalents.

6  
7 The maximum number of characters compared is the lesser of the length of *strA* less  
8 *indexA*, the length of *strB* less *indexB*, and *length*.

9  
10 When a culture is available, the method uses the culture of the current thread to  
11 determine the ordering of individual characters. The two strings are compared on a  
12 character-by-character basis.

13  
14 ]

## 15 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>indexA</i> is greater than <i>strA.Length</i> -or- <i>indexB</i> is greater than <i>strB.Length</i> -or- <i>indexA</i> , <i>indexB</i> , or <i>length</i> is negative.

16

## 17 Example

18 The following example demonstrates comparing substrings with and without case  
19 sensitivity.

20  
21 [C#]

```
1 using System;
2 public class StringCompareExample {
3     public static void Main() {
4         string strA = "STRING A";
5         string strB = "string b";
6         int first = String.Compare( strA, strB, true );
7         int second = String.Compare( strA, 0, strB, 0, 4, true );
8         int third = String.Compare( strA, 0, strB, 0, 4, false );
9         Console.WriteLine( "When the string 'STRING A' is compared to the string
10 'string b' in a case-insensitive manner, the return value is {0}.", first );
11         Console.WriteLine( "When the substring 'STRI' of 'STRING A' is compared to
12 the substring 'stri' of 'string b' in a case-insensitive manner, the return
13 value is {0}.", second );
14         Console.WriteLine( "When the substring 'STRI' of 'STRING A' is compared to
15 the substring 'stri' of 'string b' in a case-sensitive manner, the return
16 value is {0}.", third );
17     }
18 }
19
```

20 The output is

21  
22 When the string 'STRING A' is compared to the string 'string b' in a case-  
23 insensitive manner, the return value is -1.

24  
25  
26 When the substring 'STRI' of 'STRING A' is compared to the substring 'stri'  
27 of 'string b' in a case-insensitive manner, the return value is 0.

28  
29  
30 When the substring 'STRI' of 'STRING A' is compared to the substring 'stri'  
31 of 'string b' in a case-sensitive manner, the return value is 1.

32

33

# 1 String.Compare(System.String, 2 System.Int32, System.String, System.Int32, 3 System.Int32) Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 Compare(string strA, int32 indexA,  
6 string strB, int32 indexB, int32 length)  
  
7 [C#]  
8 public static int Compare(string strA, int indexA, string strB, int  
9 indexB, int length)
```

## 10 Summary

11 Compares substrings of two strings.

## 12 Parameters

Parameter	Description
<i>strA</i>	The first <code>System.String</code> to compare. Can be a null reference.
<i>indexA</i>	A <code>System.Int32</code> containing the starting index of the substring within <i>strA</i> .
<i>strB</i>	The second <code>System.String</code> to compare. Can be a null reference.
<i>indexB</i>	A <code>System.Int32</code> containing the starting index of the substring within <i>strB</i> .
<i>length</i>	A <code>System.Int32</code> containing the maximum number of characters in the substrings to compare. If <i>length</i> is zero, then zero is returned.

13

## 14 Return Value

15 The return value is a negative number, zero, or a positive number reflecting the sort  
16 order of the specified substrings. For non-zero return values, the exact value returned  
17 by this method is unspecified. The following table defines the return value:

Value	Meaning
A negative number	The substring in <i>strA</i> is < the substring in <i>strB</i> .
Zero	The substring in <i>strA</i> == the substring in <i>strB</i> , or <i>length</i> is zero.
A positive number	The substring in <i>strA</i> is > the substring in <i>strB</i> .

1

## 2 Description

3 [Note: The result of comparing any `System.String` (including the empty string) to a null  
 4 reference is greater than zero. The result of comparing two null references is zero.  
 5 Uppercase letters evaluate greater than their lowercase equivalents.

6  
 7 The method uses the culture (if any) of the current thread to determine the ordering of  
 8 individual characters. The two strings are compared on a character-by-character basis.

9  
 10 ]

## 11 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	The sum of <i>indexA</i> and <i>length</i> is greater than <i>strA.Length</i> .  -or-  The sum of <i>indexB</i> and <i>length</i> is greater than <i>strB.Length</i> .  -or-  <i>indexA</i> , <i>indexB</i> , or <i>length</i> is negative.

12

## 13 Example

14 The following example demonstrates comparing substrings.

15 [C#]  
 16

```

17 using System;
18 public class StringCompareExample {
19     public static void Main() {
20         string strA = "A string";
21         string strB = "B ring";
22         int first = String.Compare( strA, 4, strB, 2, 3 );
23         int second = String.Compare( strA, 3, strB, 3, 3 );
24         Console.WriteLine( "When the substring 'rin' of 'A string' is compared to
25 the substring 'rin' of 'B ring', the return value is {0}.", first );
26         Console.WriteLine( "When the substring 'tri' of 'A string' is compared to
27 the substring 'ing' of 'B ring', the return value is {0}.", second );
28     }
29 }
30
```

1 The output is  
2  
3 When the substring 'rin' of 'A string' is compared to the substring 'rin' of  
4 'B ring', the return value is 0.  
5  
6  
7 When the substring 'tri' of 'A string' is compared to the substring 'ing' of  
8 'B ring', the return value is 1.  
9  
10

# 1 String.Compare(System.String, 2 System.String, System.Boolean) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 Compare(string strA, string strB,  
5 bool ignoreCase)  
  
6 [C#]  
7 public static int Compare(string strA, string strB, bool ignoreCase)
```

## 8 Summary

9 Returns sort order of two System.String objects, ignoring or honoring their case.

## 10 Parameters

Parameter	Description
<i>strA</i>	The first System.String to compare. Can be a null reference.
<i>strB</i>	The second System.String to compare. Can be a null reference.
<i>ignoreCase</i>	A System.Boolean indicating whether the comparison is case-insensitive. If <i>ignoreCase</i> is true, the comparison is case-insensitive. If <i>ignoreCase</i> is false, the comparison is case-sensitive, and uppercase letters evaluate greater than their lowercase equivalents.

11

## 12 Return Value

13 The return value is a negative number, zero, or a positive number reflecting the sort  
14 order of the specified substrings. For non-zero return values, the exact value returned  
15 by this method is unspecified. The following table defines the return value:

Value	Meaning
A negative number	<i>strA</i> is < <i>strB</i> .
Zero	<i>strA</i> == <i>strB</i> .
A positive number	<i>strA</i> is > <i>strB</i> .

16

## 17 Description

1 [Note: The result of comparing any `System.String` (including the empty string) to a null  
2 reference is greater than zero. The result of comparing two null references is zero.  
3 Uppercase letters evaluate greater than their lowercase equivalents.  
4  
5 The method uses the culture (if any) of the current thread to determine the ordering of  
6 individual characters. The two strings are compared on a character-by-character basis.  
7  
8 `String.Compare(strA, strB, false)` is equivalent to `String.Compare(strA, strB)`.  
9  
10 ]

## 11 Example

12 The following example demonstrates comparing strings with and without case  
13 sensitivity.

```
14 [C#  
15  
16 using System;  
17 public class StringComparisonExample {  
18     public static void Main() {  
19         string strA = "A STRING";  
20         string strB = "a string";  
21         int first = String.Compare( strA, strB, true );  
22         int second = String.Compare( strA, strB, false );  
23         Console.WriteLine( "When 'A STRING' is compared to 'a string' in a case-  
24 insensitive manner, the return value is {0}.", first );  
25         Console.WriteLine( "When 'A STRING' is compared to 'a string' in a case-  
26 sensitive manner, the return value is {0}.", second );  
27     }  
28 }  
29
```

30 The output is

31  
32 When 'A STRING' is compared to 'a string' in a case-insensitive manner, the  
33 return value is 0.  
34

35  
36 When 'A STRING' is compared to 'a string' in a case-sensitive manner, the  
37 return value is 1.  
38

39

# 1 String.Compare(System.String, 2 System.String) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 Compare(string strA, string strB)  
5 [C#]  
6 public static int Compare(string strA, string strB)
```

## 7 Summary

8 Compares two System.String objects in a case-sensitive manner.

## 9 Parameters

Parameter	Description
<i>strA</i>	The first System.String to compare. Can be a null reference.
<i>strB</i>	The second System.String to compare. Can be a null reference.

## 10 11 Return Value

12 The return value is a negative number, zero, or a positive number reflecting the sort  
13 order of the specified strings. For non-zero return values, the exact value returned by  
14 this method is unspecified. The following table defines the return value:

Value	Meaning
A negative number	<i>strA</i> is lexicographically < <i>strB</i> .
Zero	<i>strA</i> is lexicographically == <i>strB</i> .
A positive number	<i>strA</i> is lexicographically > <i>strB</i> .

## 15 16 Description

17 This method performs a case-sensitive operation.

18  
19 [Note: The result of comparing any System.String (including the empty string) to a null  
20 reference is greater than zero. The result of comparing two null references is zero.  
21 Uppercase letters evaluate greater than their lowercase equivalents.  
22

1 The method uses the culture (if any) of the current thread to determine the ordering of  
2 individual characters. The two strings are compared on a character-by-character basis.  
3  
4 ]  
5

# 1 String.CompareOrdinal(System.String, 2 System.String) Method

```
3 [ILAsm]  
4 .method public hidebysig static int32 CompareOrdinal(string strA, string  
5 strB)  
  
6 [C#]  
7 public static int CompareOrdinal(string strA, string strB)
```

## 8 Summary

9 Compares two specified `System.String` objects based on the code points of the  
10 contained Unicode characters.

## 11 Parameters

Parameter	Description
<i>strA</i>	The first <code>System.String</code> to compare.
<i>strB</i>	The second <code>System.String</code> to compare.

## 12 13 Return Value

14 The return value is a negative number, zero, or a positive number reflecting the sort  
15 order of the specified strings. For non-zero return values, the exact value returned by  
16 this method is unspecified. The following table defines the return value:

Value	Description
A negative number	<i>strA</i> is < <i>strB</i> , or <i>strA</i> is a null reference.
Zero	<i>strA</i> == <i>strB</i> , or both <i>strA</i> and <i>strB</i> are null references.
A positive number	<i>strA</i> is > <i>strB</i> , or <i>strB</i> is a null reference.

## 17 18 Description

19 [Note: The result of comparing any `System.String` (including the empty string) to a null  
20 reference is greater than zero. The result of comparing two null references is zero.  
21 Uppercase letters evaluate greater than their lowercase equivalents.  
22

1 The method uses the culture (if any) of the current thread to determine the ordering of  
2 individual characters. The two strings are compared on a character-by-character basis.  
3  
4 ]  
5

# 1 String.CompareOrdinal(System.String, 2 System.Int32, System.String, System.Int32, 3 System.Int32) Method

```
4 [ILAsm]  
5 .method public hidebysig static int32 CompareOrdinal(string strA, int32  
6 indexA, string strB, int32 indexB, int32 length)  
  
7 [C#]  
8 public static int CompareOrdinal(string strA, int indexA, string strB, int  
9 indexB, int length)
```

## 10 Summary

11 Compares substrings of two specified `System.String` objects based on the code points  
12 of the contained Unicode characters.

## 13 Parameters

Parameter	Description
<i>strA</i>	The first <code>System.String</code> to compare.
<i>indexA</i>	A <code>System.Int32</code> containing the starting index of the substring in <i>strA</i> .
<i>strB</i>	The second <code>System.String</code> to compare.
<i>indexB</i>	A <code>System.Int32</code> containing the starting index of the substring in <i>strB</i> .
<i>length</i>	A <code>System.Int32</code> containing the number of characters in the substrings to compare.

## 14 15 Return Value

16 The return value is a negative number, zero, or a positive number reflecting the sort  
17 order of the specified strings. For non-zero return values, the exact value returned by  
18 this method is unspecified. The following table defines the return value:

Value Type	Condition
A negative number	The substring in <i>strA</i> is < the substring in <i>strB</i> , or <i>strA</i> is a null reference.
Zero	The substring in <i>strA</i> == the substring in <i>strB</i> , or both <i>strA</i> and <i>strB</i> are

	null references.
A positive number	The substring in <i>strA</i> is > the substring in <i>strB</i> , or <i>strB</i> is a null reference.

1

2 **Description**

3 When either of the String arguments is the null reference an  
 4 `System.ArgumentOutOfRangeException` shall be thrown if the corresponding index is  
 5 non-zero.

6  
 7 [Note: The maximum number of characters compared is the lesser of the length of *strA*  
 8 less *indexA*, the length of *strB* less *indexB*, and *length*.  
 9

10 The result of comparing any `System.String` (including the empty string) to a null  
 11 reference is greater than zero. The result of comparing two null references is zero.  
 12 Upper case letters evaluate greater than their lowercase equivalents.

13  
 14 The method uses the culture (if any) of the current thread to determine the ordering of  
 15 individual characters. The two strings are compared on a character-by-character basis.  
 16

17 ]

18 **Exceptions**

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>indexA</i> is greater than <i>strA.Length</i> -or- <i>indexB</i> is greater than <i>strB.Length</i> -or- <i>indexA</i> , <i>indexB</i> , or <i>length</i> is negative.

19

20

# 1 String.CompareTo(System.Object) Method

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

## 6 Summary

7 Returns the sort order of the current instance compared to the specified object.

## 8 Parameters

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

9

## 10 Return Value

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

Value	Condition
A negative number	The current instance is lexicographically $<$ <i>value</i> .
Zero	The current instance is lexicographically $==$ <i>value</i> .
A positive number	The current instance is lexicographically $>$ <i>value</i> , or <i>value</i> is a null reference.

15

## 16 Description

17 *value* is required to be a `System.String` object.

18

19 [Note: The result of comparing any `System.String` (including the empty string) to a null  
20 reference is greater than zero. Uppercase letters evaluate greater than their lowercase  
21 equivalents.

22

1 The method uses the culture (if any) of the current thread to determine the ordering of  
2 individual characters. The two strings are compared on a character-by-character basis.

3

4 This method is implemented to support the `System.IComparable` interface.

5

6 ]

## 7 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	<i>value</i> is not a <code>System.String</code> .

8

9

# String.CompareTo(System.String) Method

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

## Summary

Returns the sort order of the current instance compared to the specified string.

## Parameters

Parameter	Description
<i>strB</i>	The <code>System.String</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 *strB*. For non-zero return values, the exact value returned by this method is unspecified. The following table defines the return value:

Value	Condition
A negative number	The current instance is lexicographically $<$ <i>strB</i> .
Zero	The current instance is lexicographically $=$ <i>strB</i> .
A positive number	The current instance is lexicographically $>$ <i>strB</i> , or <i>strB</i> is a null reference.

## Description

[Note: Uppercase letters evaluate greater than their lowercase equivalents.

The method uses the culture (if any) of the current thread to determine the ordering of individual characters. The two strings are compared on a character-by-character basis.

This method is implemented to support the `System.IComparable<System.String>`

```
1 interface.  
2  
3 ]  
4
```

# 1 String.Concat(System.Object, System.Object)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string Concat(object arg0, object arg1)  
5 [C#]  
6 public static string Concat(object arg0, object arg1)
```

### 7 Summary

8 Concatenates the `System.String` representations of two specified objects.

### 9 Parameters

Parameter	Description
<i>arg0</i>	The first <code>System.Object</code> to concatenate.
<i>arg1</i>	The second <code>System.Object</code> to concatenate.

10

### 11 Return Value

12 The concatenated `System.String` representation of the values of *arg0* and *arg1*.

### 13 Description

14 `System.String.Empty` is used in place of any null argument.

15

16 This version of `System.String.Concat` is equivalent to `System.String.Concat(  
17 arg0.ToString(), arg1.ToString() )`.

18

19 [*Note:* If either of the arguments is an array reference, the method concatenates a  
20 string representing that array, instead of its members (for example, `System.String  
21 )[]`].

22

23

### 24 Example

25 The following example demonstrates concatenating two objects.

26

27 [C#]

```
28 using System;  
29 public class StringConcatExample {  
30     public static void Main() {  
31         string str = String.Concat( 'c', 32 );
```

```
1 Console.WriteLine( "The concatenated Objects are: {0}", str );
2 }
3 }
4
5 The output is
6
7 The concatenated Objects are: c32
8
```

# 1 String.Concat(System.Object, System.Object, 2 System.Object) Method

```
3 [ILAsm]  
4 .method public hidebysig static string Concat(object arg0, object arg1,  
5 object arg2)  
  
6 [C#]  
7 public static string Concat(object arg0, object arg1, object arg2)
```

## 8 Summary

9 Concatenates the `System.String` representations of three specified objects, in order  
10 provided.

## 11 Parameters

Parameter	Description
<i>arg0</i>	The first <code>System.Object</code> to concatenate.
<i>arg1</i>	The second <code>System.Object</code> to concatenate.
<i>arg2</i>	The third <code>System.Object</code> to concatenate.

12

## 13 Return Value

14 The concatenated `System.String` representations of the values of *arg0*, *arg1*, and *arg2*.

## 15 Description

16 This method concatenates the values returned by the `System.String.ToString`  
17 methods on every argument. `System.String.Empty` is used in place of any null  
18 argument.

19

20 This version of `System.String.Concat` is equivalent to `String.Concat(  
21 arg0.ToString(), arg1.ToString(), arg2.ToString() )`.

## 22 Example

23 The following example demonstrates concatenating three objects.

24

25 [C#]

```
26 using System;  
27 public class StringConcatExample {  
28     public static void Main() {
```

```
1  string str = String.Concat( 'c', 32, "String" );
2  Console.WriteLine( "The concatenated Objects are: {0}", str );
3  }
4  }
5
6  The output is
7
8  The concatenated Objects are: c32String
9
```

# 1 String.Concat(System.Object[]) Method

```
2 [ILAsm]  
3 .method public hidebysig static string Concat(object[] args)  
4 [C#]  
5 public static string Concat(params object[] args)
```

## 6 Summary

7 Concatenates the `System.String` representations of the elements in an array of  
8 `System.Object` instances.

## 9 Parameters

Parameter	Description
<i>args</i>	An array of <code>System.Object</code> instances to concatenate.

10

## 11 Return Value

12 The concatenated `System.String` representations of the values of the elements in *args*.

## 13 Description

14 This method concatenates the values returned by the `System.String.ToString`  
15 methods on every object in the *args* array. `System.String.Empty` is used in place of  
16 any null reference in the array.

## 17 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>args</i> is a null reference.

18

## 19 Example

20 The following example demonstrates concatenating an array of objects.

21

22 [C#]

```
23 using System;  
24 public class StringConcatExample {  
25     public static void Main() {  
26         string str = String.Concat( 'c', 32, "String" );  
27         Console.WriteLine( "The concatenated Object array is: {0}", str );  
28     }  
}
```

```
1  }
2
3  The output is
4
5  The concatenated Object array is: c32String
6
```

# 1 String.Concat(System.String, System.String)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string Concat(string str0, string str1)  
5 [C#]  
6 public static string Concat(string str0, string str1)
```

### 7 Summary

8 Concatenates two specified instances of System.String.

### 9 Parameters

Parameter	Description
<i>str0</i>	The first System.String to concatenate.
<i>str1</i>	The second System.String to concatenate.

### 10 Return Value

12 A System.String containing the concatenation of *str0* and *str1*.

### 13 Description

14 System.String.Empty is used in place of any null argument.

### 15 Example

16 The following example demonstrates concatenating two strings.

```
17 [C#]  
18  
19 using System;  
20 public class StringConcatExample {  
21     public static void Main() {  
22         string str = String.Concat( "one", "two" );  
23         Console.WriteLine( "The concatenated strings are: {0}", str );  
24     }  
25 }  
26
```

27 The output is

28  
29 The concatenated strings are: onetwo

30

# 1 String.Concat(System.String, System.String, System.String) Method

```
3 [ILAsm]
4 .method public hidebysig static string Concat(string str0, string str1,
5 string str2)
6
7 [C#]
8 public static string Concat(string str0, string str1, string str2)
```

## 8 Summary

9 Concatenates three specified instances of System.String.

## 10 Parameters

Parameter	Description
<i>str0</i>	The first System.String to concatenate.
<i>str1</i>	The second System.String to concatenate.
<i>str2</i>	The third System.String to concatenate.

11

## 12 Return Value

13 A System.String containing the concatenation of *str0*, *str1*, and *str2*.

## 14 Description

15 System.String.Empty is used in place of any null argument.

## 16 Example

17 The following example demonstrates concatenating three strings.

18  
19

```
19 [C#]
20 using System;
21 public class StringConcatExample {
22     public static void Main() {
23         string str = String.Concat( "one", "two", "three" );
24         Console.WriteLine( "The concatenated strings are: {0}", str );
25     }
26 }
27
```

- 1 The output is
- 2
- 3 The concatenated strings are: onetwothree
- 4

# 1 String.Concat(System.String[]) Method

```
2 [ILAsm]  
3 .method public hidebysig static string Concat(string[] values)  
4 [C#]  
5 public static string Concat(params string[] values)
```

## 6 Summary

7 Concatenates the elements of a specified array.

## 8 Parameters

Parameter	Description
<i>values</i>	An array of <i>System.String</i> instances to concatenate.

## 9 Return Value

11 A *System.String* containing the concatenated elements of *values*.

## 12 Description

13 *System.String.Empty* is used in place of any null reference in the array.

## 14 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>values</i> is a null reference.

## 15 Example

17 The following example demonstrates concatenating an array of strings.

```
18 [C#]  
19 using System;  
20 public class StringConcatExample {  
21     public static void Main() {  
22         string str = String.Concat( "one", "two", "three", "four", "five" );  
23         Console.WriteLine( "The concatenated String array is: {0}", str );  
24     }  
25 }  
26 }  
27
```

- 1 The output is
- 2
- 3 The concatenated String array is: onetwothreefourfive
- 4

# 1 String.Copy(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig static string Copy(string str)  
4 [C#]  
5 public static string Copy(string str)
```

## 6 Summary

7 Creates a new instance of `System.String` with the same value as a specified instance of  
8 `System.String`.

## 9 Parameters

Parameter	Description
<i>str</i>	The <code>System.String</code> to be copied.

10

## 11 Return Value

12 A new `System.String` with the same value as *str*.

## 13 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>str</i> is a null reference.

14

## 15 Example

16 The following example demonstrates copying strings.

17

18 [C#]

```
19 using System;  
20 public class StringCopyExample {  
21     public static void Main() {  
22         string strA = "string";  
23         Console.WriteLine( "The initial string, strA, is '{0}'.", strA );  
24         string strB = String.Copy( strA );  
25         strA = strA.ToUpper();  
26         Console.WriteLine( "The copied string, strB, before strA.ToUpper, is  
27 '{0}'.", strB );  
28         Console.WriteLine( "The initial string after StringCopy and ToUpper, is  
29 '{0}'.", strA );
```

```
1 Console.WriteLine( "The copied string, strB, after strA.ToUpper, is '{0}'.",
2 strB );
3 }
4 }
5
6 The output is
7
8 The initial string, strA, is 'string'.
9
10
11 The copied string, strB, before strA.ToUpper, is 'string'.
12
13
14 The initial string after StringCopy and ToUpper, is 'STRING'.
15
16
17 The copied string, strB, after strA.ToUpper, is 'string'.
18
19
```

# 1 String.CopyTo(System.Int32, System.Char[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance void CopyTo(int32 sourceIndex, char[]  
5 destination, int32 destinationIndex, int32 count)  
  
6 [C#]  
7 public void CopyTo(int sourceIndex, char[] destination, int  
8 destinationIndex, int count)
```

## 9 Summary

10 Copies a specified number of characters from a specified position in the current  
11 System.String instance to a specified position in a specified array of Unicode  
12 characters.

## 13 Parameters

Parameter	Description
<i>sourceIndex</i>	A System.Int32 containing the index of the current instance from which to copy.
<i>destination</i>	An array of Unicode characters.
<i>destinationIndex</i>	A System.Int32 containing the index of an array element in <i>destination</i> to copy.
<i>count</i>	A System.Int32 containing the number of characters in the current instance to copy to <i>destination</i> .

## 14 15 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>destination</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>sourceIndex</i> , <i>destinationIndex</i> , or <i>count</i> is negative -or- <i>count</i> is greater than the length of the substring from <i>startIndex</i> to the end of the

	<p>current instance</p> <p>-or-</p> <p><i>count</i> is greater than the length of the subarray from <i>destinationIndex</i> to the end of <i>destination</i></p>
--	--

1

2 **Example**

3 The following example demonstrates copying characters from a string to a Unicode  
 4 character array.

5  
 6 [C#]

```

7 using System;
8 public class StringCopyToExample {
9     public static void Main() {
10        string str = "this is the new string";
11        Char[] cAry = {'t','h','e',' ','o','l','d'};
12        Console.WriteLine( "The initial string is '{0}'", str );
13        Console.Write( "The initial character array is: ' " );
14        foreach( Char c in cAry)
15            Console.Write( c );
16        Console.WriteLine( "' ' );
17        str.CopyTo( 12, cAry, 4, 3 );
18        Console.Write( "The character array after CopyTo is: ' " );
19        foreach( Char c in cAry)
20            Console.Write( c );
21        Console.WriteLine( "' ' );
22    }
23 }
24 
```

25 The output is

26

27 The initial string is 'this is the new string'

28

29

30 The initial character array is: 'the old'

31

32

33 The character array after CopyTo is: 'the new'

34

35

# 1 String.EndsWith(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig instance bool EndsWith(string value)  
4 [C#]  
5 public bool EndsWith(string value)
```

## 6 Summary

7 Returns a `System.Boolean` value that indicates whether the ending characters of the  
8 current instance match the specified `System.String`.

## 9 Parameters

Parameter	Description
<i>value</i>	A <code>System.String</code> to match.

10

## 11 Return Value

12 `true` if the end of the current instance is equal to *value*; `false` if *value* is not equal to  
13 the end of the current instance or is longer than the current instance.

## 14 Description

15 This method compares *value* with the substring at the end of the current instance that  
16 has a same length as *value*.

17  
18 The comparison is case-sensitive.

## 19 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>value</i> is a null reference.

20

## 21 Example

22 The following example demonstrates determining whether the current instance ends  
23 with a specified string.

```
24 [C#]  
25  
26 using System;  
27 public class StringEndsWithExample {
```

```
1 public static void Main() {
2     string str = "One string to compare";
3     Console.WriteLine( "The given string is '{0}'", str );
4     Console.Write( "The given string ends with 'compare'? " );
5     Console.WriteLine( str.EndsWith( "compare" ) );
6     Console.Write( "The given string ends with 'Compare'? " );
7     Console.WriteLine( str.EndsWith( "Compare" ) );
8 }
9 }
```

10  
11 The output is

```
12  
13 The given string is 'One string to compare'
14  
15  
16 The given string ends with 'compare'? True
17  
18  
19 The given string ends with 'Compare'? False
20
```

21

# 1 String.Equals(System.Object) Method

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

## 6 Summary

7 Determines whether the current instance and the specified object have the same value.

## 8 Parameters

Parameter	Description
<i>obj</i>	A System.Object.

## 9 Return Value

11 true if *obj* is a System.String and its value is the same as the value of the current  
12 instance; otherwise, false.

## 13 Description

14 This method checks for value equality. This comparison is case-sensitive.

15  
16 [Note: This method overrides System.Object.Equals.]  
17  
18

## 19 Exceptions

Exception	Condition
<b>System.NullReferenceException</b>	The current instance is a null reference.

## 20 Example

22 The following example demonstrates checking to see if an object is equal to the current  
23 instance.

```
24 [C#]  
25  
26 using System;  
27 public class StringEqualsExample {
```

```
1 public static void Main() {
2     string str = "A string";
3     Console.WriteLine( "The given string is '{0}'", str );
4     Console.Write( "The given string is equal to 'A string'? " );
5     Console.WriteLine( str.Equals( "A string" ) );
6     Console.Write( "The given string is equal to 'A String'? " );
7     Console.WriteLine( str.Equals( "A String" ) );
8 }
9 }
```

10  
11 The output is

```
12
13 The given string is 'A string'
14
15
16 The given string is equal to 'A string'? True
17
18
19 The given string is equal to 'A String'? False
20
21
```

# 1 String.Equals(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig virtual bool Equals(string value)  
4 [C#]  
5 public override bool Equals(string value)
```

## 6 Summary

7 Determines whether the current instance and the specified string have the same value.

## 8 Parameters

Parameter	Description
<i>value</i>	A System.String.

9

## 10 Return Value

11 true if the value of *value* is the same as the value of the current instance; otherwise,  
12 false.

## 13 Description

14 This method checks for value equality. This comparison is case-sensitive.

15

16 [Note: This method is implemented to support the  
17 System.IEquatable<System.String> interface.]

18

19

20

# 1 String.Equals(System.String, System.String)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static bool Equals(string a, string b)  
5 [C#]  
6 public static bool Equals(string a, string b)
```

### 7 Summary

8 Determines whether two specified `System.String` objects have the same value.

### 9 Parameters

Parameter	Description
<i>a</i>	A <code>System.String</code> or a null reference.
<i>b</i>	A <code>System.String</code> or a null reference.

10

### 11 Return Value

12 `true` if the value of *a* is the same as the value of *b*; otherwise, `false`.

### 13 Description

14 The comparison is case-sensitive.

### 15 Example

16 The following example demonstrates checking to see if two strings are equal.

```
17 [C#]  
18  
19 using System;  
20 public class StringEqualsExample {  
21     public static void Main() {  
22         string strA = "A string";  
23         string strB = "a string";  
24         string strC = "a string";  
25         Console.Write( "The string '{0}' is equal to the string '{1}'? ", strA, strB  
26     );  
27         Console.WriteLine( String.Equals( strA, strB ) );  
28         Console.Write( "The string '{0}' is equal to the string '{1}'? ", strC, strB  
29     );  
30         Console.WriteLine( String.Equals( strC, strB ) );  
31     }  
}
```

```
1  }
2
3  The output is
4
5  The string 'A string' is equal to the string 'a string'? False
6
7
8  The string 'a string' is equal to the string 'a string'? True
9
10
```

# 1 String.Format(System.String, 2 System.Object[]) Method

```
3 [ILAsm]  
4 .method public hidebysig static string Format(string format, object[]  
5 args)  
  
6 [C#]  
7 public static string Format(string format, params object[] args)
```

## 8 Summary

9 Replaces the format specification in a specified `System.String` with the textual  
10 equivalent of the value of a corresponding `System.Object` instance in a specified array.

## 11 Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> containing zero or more format specifications.
<i>args</i>	A <code>System.Object</code> array containing the objects to be formatted.

## 12 13 Return Value

14 A `System.String` containing a copy of *format* in which the format specifications have  
15 been replaced by the `System.String` equivalent of the corresponding instances of  
16 `System.Object` in *args*.

## 17 Description

18 If an object referenced in the format string is a null reference, an empty string is used in  
19 its place.

20  
21 [Note: This version of `System.String.Format` is equivalent to `System.String.Format`(  
22 `null, format, args`). For more information on the format specification see the  
23 `System.String` class overview.]  
24  
25

## 26 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>format</i> or <i>args</i> is a null reference.

## System.FormatException

*format* is invalid.

-or-

The number indicating an argument to be formatted is less than zero, or greater than or equal to the length of the *args* array.

1

## 2 Example

3 The following example demonstrates the System.String.Format method.

4

5 [C#]

6

7 using System;

8 public class StringFormat {

9 public static void Main() {

10 Console.WriteLine( String.Format("The winning numbers were {0:000}

11 {1:000} {2:000} {3:000} {4:000} today.", 5, 10, 11, 37, 42) );

12 Console.WriteLine( "The winning numbers were {0, -6}{1, -6}{2, -6}{3, -

13 6}{4, -6} today.", 5, 10, 11, 37, 42 );

14 }

15 }

16 The output is

17

18 The winning numbers were 005 010 011 037 042 today.

19 The winning numbers were 5 10 11 37 42 today.

20

# 1 String.Format(System.String, System.Object)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string Format(string format, object arg0)  
5 [C#]  
6 public static string Format(string format, object arg0)
```

### 7 Summary

8 Replaces the format specification in a provided `System.String` with a specified textual  
9 equivalent of the value of a specified `System.Object` instance.

### 10 Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> containing zero or more format specifications.
<i>arg0</i>	A <code>System.Object</code> to be formatted. Can be a null reference.

11

### 12 Return Value

13 A copy of *format* in which the first format specification has been replaced by the  
14 formatted `System.String` equivalent of the *arg0*.

### 15 Description

16 If an object referenced in the format string is a null reference, an empty string is used in  
17 its place.

18  
19 [Note: This version of `System.String.Format` is equivalent to `String.Format( null,`  
20 `format, new Object[] {arg0} )`. For more information on the format specification see  
21 the `System.String` class overview.]  
22  
23

### 24 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>format</i> is a null reference.

## System.FormatException

The format specification in *format* is invalid.

-or-

The number indicating an argument to be formatted is less than zero, or greater than or equal to the number of provided objects to be formatted (1).

1

## 2 Example

3 The following example demonstrates the System.String.Format method.

4

5 [C#]

6

```
7 using System;
```

```
8 public class StringFormat {
```

```
9     public static void Main() {
```

```
10     Console.WriteLine(String.Format("The high temperature today was {0:###}
```

```
11     degrees.", 88));
```

```
12     Console.WriteLine("The museum had {0,-6} visitors today.", 88);
```

```
13     }
```

```
14 }
```

15 The output is

16

17 The high temperature today was 88 degrees.

18 The museum had 88 visitors today.

19

# 1 String.Format(System.String, System.Object, 2 System.Object) Method

```
3 [ILAsm]  
4 .method public hidebysig static string Format(string format, object arg0,  
5 object arg1)  
  
6 [C#]  
7 public static string Format(string format, object arg0, object arg1)
```

## 8 Summary

9 Replaces the format specification in a specified `System.String` with the textual  
10 equivalent of the value of two specified `System.Object` instances.

## 11 Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> containing zero or more format specifications.
<i>arg0</i>	A <code>System.Object</code> to be formatted. Can be a null reference.
<i>arg1</i>	A <code>System.Object</code> to be formatted. Can be a null reference.

12

## 13 Return Value

14 A `System.String` containing a copy of *format* in which the format specifications have  
15 been replaced by the `System.String` equivalent of *arg0* and *arg1*.

## 16 Description

17 If an object referenced in the format string is a null reference, an empty string is used in  
18 its place.

19

20 [Note: This version of `System.String.Format` is equivalent to `String.Format( null,`  
21 `format, new Object [] {arg0, arg1} )`. For more information on the format specification  
22 see the `System.String` class overview.]

23

24

## 25 Exceptions

Exception	Condition
-----------	-----------

<b>System.ArgumentNullException</b>	<i>format</i> is a null reference.
<b>System.FormatException</b>	<i>format</i> is invalid. -or- The number indicating an argument to be formatted is less than zero, or greater than or equal to the number of provided objects to be formatted (2).

1

## 2 Example

3 The following example demonstrates the `System.String.Format` method.

4

5 [C#]

```

6 using System;
7 public class StringFormat {
8     public static void Main() {
9         Console.WriteLine( String.Format("The temperature today oscillated between
10 {0:####} and {1:####} degrees.", 78, 100) );
11         Console.WriteLine( String.Format("The temperature today oscillated between
12 {0:0000} and {1:0000} degrees.", 78, 100) );
13         Console.WriteLine( "The temperature today oscillated between {0, -4} and
14 {1, -4} degrees.", 78, 100 );
15     }
16 }

```

17 The output is

18

19 The temperature today oscillated between 78 and 100 degrees.

20 The temperature today oscillated between 0078 and 0100 degrees.

21 The temperature today oscillated between 78 and 100 degrees.

22

# String.Format(System.String, System.Object, System.Object, System.Object) Method

```
[ILAsm]
.method public hidebysig static string Format(string format, object arg0,
object arg1, object arg2)

[C#]
public static string Format(string format, object arg0, object arg1,
object arg2)
```

## Summary

Replaces the format specification in a specified `System.String` with the textual equivalent of the value of three specified `System.Object` instances.

## Parameters

Parameter	Description
<i>format</i>	A <code>System.String</code> containing zero or more format specifications.
<i>arg0</i>	The first <code>System.Object</code> to be formatted. Can be a null reference.
<i>arg1</i>	The second <code>System.Object</code> to be formatted. Can be a null reference.
<i>arg2</i>	The third <code>System.Object</code> to be formatted. Can be a null reference.

## Return Value

A `System.String` containing a copy of *format* in which the first, second, and third format specifications have been replaced by the `System.String` equivalent of *arg0*, *arg1*, and *arg2*.

## Description

If an object referenced in the format string is a null reference, an empty string is used in its place.

[*Note:* This version of `System.String.Format` is equivalent to `String.Format( null, format, new Object [] {arg0, arg1, arg2} )`. For more information on the format specification see the `System.String` class overview.]

## Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>format</i> is a null reference.
<b>System.FormatException</b>	<p><i>format</i> is invalid.</p> <p>-or-</p> <p>The number indicating an argument to be formatted is less than zero, or greater than or equal to the number of provided objects to be formatted (3).</p>

1

## 2 Example

3 The following example demonstrates the `System.String.Format` method.

4

5 [C#]

6

7 `using System;`

8 `public class StringFormat {`

9 `public static void Main() {`

10 `Console.WriteLine(String.Format("The temperature today oscillated`  
11 `between {0:###} and {1:###} degrees. The average temperature was {2:000}`  
12 `degrees.", 78, 100, 91));`

13 `Console.WriteLine("The temperature today oscillated between {0, 4} and`  
14 `{1, 4} degrees. The average temperature was {2, 4} degrees.", 78, 100, 91);`  
15 `}`

16 `}`

17 The output is

18

19 The temperature today oscillated between 78 and 100 degrees. The average  
20 temperature was 091 degrees.

21 The temperature today oscillated between 78 and 100 degrees. The average  
22 temperature was 91 degrees.

23

# String.Format(System.IFormatProvider, System.String, System.Object[]) Method

```
[ILAsm]
.method public hidebysig static string Format(class System.IFormatProvider
provider, string format, object[] args)

[C#]
public static string Format(IFormatProvider provider, string format,
params object[] args)
```

## Summary

Replaces the format specification in a specified `System.String` with the culture-specific textual equivalent of the value of a corresponding `System.Object` instance in a specified array.

## Parameters

Parameter	Description
<i>provider</i>	A <code>System.IFormatProvider</code> interface that supplies an object that provides culture-specific formatting information. Can be a null reference.
<i>format</i>	A <code>System.String</code> containing zero or more format specifications.
<i>args</i>	A <code>System.Object</code> array to be formatted.

## Return Value

A `System.String` containing a copy of *format* in which the format specifications have been replaced by the `System.String` equivalent of the corresponding instances of `System.Object` in *args*.

## Description

If an object referenced in the format string is a null reference, an empty string is used in its place.

The *format* parameter string is embedded with zero or more format specifications of the form, `{N[, M][: formatString]}`, where *N* is a zero-based integer indicating the argument to be formatted, *M* is an optional integer indicating the width of the region to contain the formatted value, and *formatString* is an optional string of formatting codes. [Note: For more information on the format specification see the `System.String` class overview.]

1  
2

### 3 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>format</i> or <i>args</i> is a null reference.
<b>System.FormatException</b>	<i>format</i> is invalid.  -or-  The number indicating an argument to be formatted ( <i>N</i> ) is less than zero, or greater than or equal to the length of the <i>args</i> array.

4

5

# 1 String.GetEnumerator() Method

```
2 [ILAsm]  
3 .method public hidebysig instance CharEnumerator GetEnumerator()  
4 [C#]  
5 public CharEnumerator GetEnumerator()
```

## 6 Summary

7 Retrieves an object that can iterate through the individual characters in the current  
8 instance.

## 9 Return Value

10 A System.CharEnumerator object.

## 11 Description

12 This method is required by programming languages that support the  
13 System.Collections.IEnumerator interface to iterate through members of a collection.

14

# 1 String.GetHashCode() Method

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

## 6 Summary

7 Generates a hash code for the current instance.

## 8 Return Value

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

## 10 Description

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

12

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

14

15

16

# 1 String.IndexOf(System.Char) Method

```
2 [ILAsm]  
3 .method public hidebysig instance int32 IndexOf(valuetype System.Char  
4 value)  
5 [C#]  
6 public int IndexOf(char value)
```

## 7 Summary

8 Returns the index of the first occurrence of a specified Unicode character in the current  
9 instance.

## 10 Parameters

Parameter	Description
<i>value</i>	A Unicode character.

11

## 12 Return Value

13 A `System.Int32` containing the zero-based index of the first occurrence of *value*  
14 character in the current instance; otherwise, -1 if *value* was not found.

## 15 Description

16 This method is case-sensitive.

17

# 1 String.IndexOf(System.Char, System.Int32) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 IndexOf(valuetype System.Char  
5 value, int32 startIndex)  
  
6 [C#]  
7 public int IndexOf(char value, int startIndex)
```

## 8 Summary

9 Returns the index of the first occurrence of a specified Unicode character in the current  
10 instance, with the search starting from a specified index.

## 11 Parameters

Parameter	Description
<i>value</i>	A Unicode character.
<i>startIndex</i>	A System.Int32 containing the index of the current instance from which to start searching.

12

## 13 Return Value

14 A System.Int32 containing the zero-based index of the first occurrence of *value* in the  
15 current instance starting from the specified index; otherwise, -1 if *value* was not found.

## 16 Description

17 This method is case-sensitive.

## 18 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is less than zero or greater than the length of the current instance.

19

## 20 Example

21 The following example demonstrates the System.String.IndexOf method.

22

23

```
[C#]
```

```
1 using System;
2 public class StringIndexOf {
3     public static void Main() {
4         String str = "This is the string";
5         Console.WriteLine( "Searching for the index of 'h' starting from index 0
6 yields {0}.", str.IndexOf( 'h', 0 ) );
7         Console.WriteLine( "Searching for the index of 'h' starting from index 10
8 yields {0}.", str.IndexOf( 'h', 10 ) );
9     }
10 }
```

11 The output is

12

13 Searching for the index of 'h' starting from index 0 yields 1.

14

15

16 Searching for the index of 'h' starting from index 10 yields -1.

17

18

# 1 String.IndexOf(System.Char, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 IndexOf(valuetype System.Char  
5 value, int32 startIndex, int32 count)  
  
6 [C#]  
7 public int IndexOf(char value, int startIndex, int count)
```

## 8 Summary

9 Returns the index of the first occurrence of a specified Unicode character in the current  
10 instance, with the search over the specified range starting at the provided index.

## 11 Parameters

Parameter	Description
<i>value</i>	A Unicode character.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.
<i>count</i>	A <code>System.Int32</code> containing the number of consecutive elements of the current instance to be searched starting at <i>startIndex</i> .

## 12 13 Return Value

14 A `System.Int32` containing the zero-based index of the first occurrence of *value* in the  
15 current instance in the specified range of indexes; otherwise, -1 if *value* was not found.

## 16 Description

17 The search begins at *startIndex* and continues until *startIndex* + *count* - 1 is reached.  
18 The character at *startIndex* + *count* is not included in the search.

19  
20 This method is case-sensitive.

## 21 Exceptions

Exception	Condition
<code>System.ArgumentOutOfRangeException</code>	<i>startIndex</i> or <i>count</i> is negative  -or-

	<i>startIndex</i> + <i>count</i> is greater than the length of the current instance.
--	--

1

2

# 1 String.IndexOf(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig instance int32 IndexOf(string value)  
4 [C#]  
5 public int IndexOf(string value)
```

## 6 Summary

7 Returns the index of the first occurrence of a specified `System.String` in the current  
8 instance.

## 9 Parameters

Parameter	Description
<i>value</i>	The <code>System.String</code> for which to search.

10

## 11 Return Value

12 A `System.Int32` that indicates the result of the search for *value* in the current instance  
13 as follows:

Return Value	Description
A zero-based number equal to the index of the start of the first substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found starting at the index returned.
-1	<i>value</i> was not found.

14

## 15 Description

16 The search begins at the first character of the current instance. The search is case-  
17 sensitive, culture-sensitive, and the culture (if any) of the current thread is used.

## 18 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>value</i> is a null reference.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

**Example**

The following example demonstrates the System.String.IndexOf method.

[C#]

```
using System;
public class StringIndexOf {
    public static void Main() {
        String str = "This is the string";
        Console.WriteLine( "Searching for the index of \"is\" yields {0,2}.",
str.IndexOf( "is" ) );
        Console.WriteLine( "Searching for the index of \"Is\" yields {0,2}.",
str.IndexOf( "Is" ) );
        Console.WriteLine( "Searching for the index of \"\" yields {0,2}.",
str.IndexOf( "" ) );
    }
}
```

The output is

Searching for the index of "is" yields 2.  
Searching for the index of "Is" yields -1.  
Searching for the index of "" yields 0.

# 1 String.IndexOf(System.String, System.Int32)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 IndexOf(string value, int32  
5 startIndex)  
  
6 [C#]  
7 public int IndexOf(string value, int startIndex)
```

### 8 Summary

9 Returns the index of the first occurrence of a specified `System.String` in the current  
10 instance, with the search starting from a specified index.

### 11 Parameters

Parameter	Description
<i>value</i>	The <code>System.String</code> for which to search.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.

12

### 13 Return Value

14 A `System.Int32` that indicates the result of the search for *value* in the current instance  
15 as follows:

Return Value	Description
A zero-based number equal to the index of the start of the first substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found starting at the index returned.
-1	<i>value</i> was not found.

16

### 17 Description

18 This method is case-sensitive.

### 19 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is greater than the length of the current instance.

1

2

# 1 String.IndexOf(System.String, System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 IndexOf(string value, int32  
5 startIndex, int32 count)  
  
6 [C#]  
7 public int IndexOf(string value, int startIndex, int count)
```

## 8 Summary

9 Returns the index of the first occurrence of a specified `System.String` in the current  
10 instance, with the search over the specified range starting at the provided index.

## 11 Parameters

Parameter	Description
<i>value</i>	The <code>System.String</code> for which to search
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.
<i>count</i>	A <code>System.Int32</code> containing the number of consecutive elements of the current instance to be searched starting at <i>startIndex</i> .

## 12 13 Return Value

14 A `System.Int32` that indicates the result of the search for *value* in the current instance  
15 as follows:

Return Value	Description
A zero-based number equal to the index of the start of the first substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found starting at the index returned.
-1	<i>value</i> was not found.

## 16 17 Description

- 1 The search begins at *startIndex* and continues until *startIndex + count - 1* is reached.
- 2 The character at *startIndex + count* is not included in the search.
- 3
- 4 This method is case-sensitive.

5 **Exceptions**

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>count</i> is negative  -or- <i>startIndex + count</i> is greater than the length of the current instance.

- 6
- 7

# 1 String.IndexOfAny(System.Char[]) Method

```
2 [ILAsm]  
3 .method public hidebysig instance int32 IndexOfAny(char[] anyOf)  
4 [C#]  
5 public int IndexOfAny(char[] anyOf)
```

## 6 Summary

7 Reports the index of the first occurrence in the current instance of any character in a  
8 specified array of Unicode characters.

## 9 Parameters

Parameter	Description
<i>anyOf</i>	An array of Unicode characters.

10

## 11 Return Value

12 The index of the first occurrence of an element of *anyOf* in the current instance;  
13 otherwise, -1 if no element of *anyOf* was found.

## 14 Description

15 This method is case-sensitive.

## 16 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.

17

18

# 1 String.IndexOfAny(System.Char[], 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 IndexOfAny(char[] anyOf, int32  
5 startIndex)  
  
6 [C#]  
7 public int IndexOfAny(char[] anyOf, int startIndex)
```

## 8 Summary

9 Returns the index of the first occurrence of any element in a specified array of Unicode  
10 characters in the current instance, with the search starting from a specified index.

## 11 Parameters

Parameter	Description
<i>anyOf</i>	An array of Unicode characters.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.

12

## 13 Return Value

14 A `System.Int32` containing a positive value equal to the index of the first occurrence of  
15 an element of *anyOf* in the current instance; otherwise, -1 if no element of *anyOf* was  
16 found.

## 17 Description

18 This method is case-sensitive.

## 19 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is greater than the length of the current instance

20

21

# String.IndexOfAny(System.Char[], System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig instance int32 IndexOfAny(char[] anyOf, int32
startIndex, int32 count)

[C#]
public int IndexOfAny(char[] anyOf, int startIndex, int count)
```

## Summary

Returns the index of the first occurrence of any element in a specified Array of Unicode characters in the current instance, with the search over the specified range starting from the provided index.

## Parameters

Parameter	Description
<i>anyOf</i>	An array containing the Unicode characters to seek.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.
<i>count</i>	A <code>System.Int32</code> containing the range of the current instance at which to end searching.

## Return Value

A `System.Int32` containing a positive value equal to the index of the first occurrence of an element of *anyOf* in the current instance; otherwise, -1 if no element of *anyOf* was found.

## Description

The search begins at *startIndex* and continues until *startIndex* + *count* - 1. The character at *startIndex* + *count* is not included in the search.

This method is case-sensitive.

## Exceptions

Exception	Condition
-----------	-----------

<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>count</i> is negative. -or- <i>startIndex</i> + <i>count</i> is greater than the length of the current instance.

1

2

# 1 String.Insert(System.Int32, System.String) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string Insert(int32 startIndex, string  
5 value)  
6 [C#]  
7 public string Insert(int startIndex, string value)
```

## 8 Summary

9 Returns a `System.String` equivalent to the current instance with a specified  
10 `System.String` inserted at the specified position.

## 11 Parameters

Parameter	Description
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the insertion.
<i>value</i>	The <code>System.String</code> to insert.

12

## 13 Return Value

14 A new `System.String` that is equivalent to the current string with *value* inserted at  
15 index *startIndex*.

## 16 Description

17 In the new string returned by this method, the first character of *value* is at *startIndex*,  
18 and all characters in the current string from *startIndex* to the end are inserted in the  
19 new string after the last character of *value*.

## 20 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is greater than the length of the current instance.

21

22

# 1 String.Intern(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig static string Intern(string str)  
4 [C#]  
5 public static string Intern(string str)
```

## 6 Summary

7 Retrieves the system's reference to a specified System.String.

## 8 Parameters

Parameter	Description
<i>str</i>	A System.String.

## 9 Return Value

10 The System.String reference to *str*.

## 12 Description

13 Instances of each unique literal string constant declared in a program, as well as any  
14 unique instance of System.String you add programmatically are kept in a table, called  
15 the "intern pool".

16 The intern pool conserves string storage. If a literal string constant is assigned to  
17 several variables, each variable is set to reference the same constant in the intern pool  
18 instead of referencing several different instances of System.String that have identical  
19 values.

20 This method looks up a specified string in the intern pool. If the string exists, a  
21 reference to it is returned. If it does not exist, an instance equal to the specified string is  
22 added to the intern pool and a reference that instance is returned.

## 25 Exceptions

Exception	Condition
System.ArgumentNullException	<i>str</i> is a null reference.

## 26 Example

1 The following example demonstrates the System.String.Intern method.

2

3 [C#]

4 using System;

5 using System.Text;

6 public class StringExample {

7 public static void Main() {

8

9 String s1 = "MyTest";

10 String s2 = new

11 StringBuilder().Append("My").Append("Test").ToString();

12 String s3 = String.Intern(s2);

13

14 Console.WriteLine(Object.ReferenceEquals(s1, s2)); //different

15 Console.WriteLine(Object.ReferenceEquals(s1, s3)); //the same

16 }

17 }

18 The output is

19

20 False

21

22

23 True

24

25

# 1 String.IsInterned(System.String) Method

```
2 [ILAsm]  
3 .method public hidebysig static string IsInterned(string str)  
4 [C#]  
5 public static string IsInterned(string str)
```

## 6 Summary

7 Retrieves a reference to a specified `System.String`.

## 8 Parameters

Parameter	Description
<i>str</i>	A <code>System.String</code> .

## 9 Return Value

11 A `System.String` reference to *str* if it is in the system's intern pool; otherwise, a null  
12 reference.

## 13 Description

14 Instances of each unique literal string constant declared in a program, as well as any  
15 unique instance of `System.String` you add programmatically are kept in a table, called  
16 the "intern pool".

17  
18 The intern pool conserves string storage. If a literal string constant is assigned to  
19 several variables, each variable is set to reference the same constant in the intern pool  
20 instead of referencing several different instances of `System.String` that have identical  
21 values.

22  
23 [*Note:* This method does not return a `System.Boolean` value, but can still be used  
24 where a `System.Boolean` is needed.]

## 27 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>str</i> is a null reference.

## 28 Example

1       The following example demonstrates the `System.String.IsInterned` method.

2

3       [C#]

4       using System;

5       using System.Text;

6

7       public class StringExample {

8             public static void Main() {

9

10                 String s1 = new

11             StringBuilder().Append("My").Append("Test").ToString();

12

13                 Console.WriteLine(String.IsInterned(s1) != null);

14             }

15       }

16       The output is

17

18       True

19

# 1 String.Join(System.String, System.String[])

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static string Join(string separator, string[]  
5 value)  
  
6 [C#]  
7 public static string Join(string separator, string[] value)
```

### 8 Summary

9 Concatenates the elements of a specified `System.String` array, inserting a separator  
10 string between each element pair and yielding a single concatenated string.

### 11 Parameters

Parameter	Description
<i>separator</i>	A <code>System.String</code> .
<i>value</i>	A <code>System.String</code> array.

### 12 Return Value

14 A `System.String` consisting of the elements of *value* separated by instances of the  
15 *separator* string.

### 16 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.

### 17 Example

19 The following example demonstrates the `System.String.Join` method.

```
20 [C#]  
21  
22 using System;  
23 public class StringJoin {  
24     public static void Main() {  
25         String[] strAry = { "Red", "Green", "Blue" };  
26         Console.WriteLine( String.Join( ":: ", strAry ) );  
27     }  
}
```

```
1 }
2 The output is
3
4 Red:: Green:: Blue
5
```

# 1 String.Join(System.String, System.String[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static string Join(string separator, string[]  
5 value, int32 startIndex, int32 count)  
  
6 [C#]  
7 public static string Join(string separator, string[] value, int  
8 startIndex, int count)
```

## 9 Summary

10 Concatenates a specified separator `System.String` between the elements of a specified  
11 `System.String` array, yielding a single concatenated string.

## 12 Parameters

Parameter	Description
<i>separator</i>	A <code>System.String</code> .
<i>value</i>	A <code>System.String</code> array.
<i>startIndex</i>	A <code>System.Int32</code> containing the first array element in <i>value</i> to join.
<i>count</i>	A <code>System.Int32</code> containing the number of elements in <i>value</i> to join.

## 13 14 Return Value

15 A `System.String` consisting of the specified strings in *value* joined by *separator*.  
16 Returns `System.String.Empty` if *count* is zero, *value* has no elements, or *separator* and  
17 all the elements of *value* are `Empty`.

## 18 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> plus <i>count</i> is greater than the number of elements in <i>value</i> .

## 19 20 Example

1 The following example demonstrates the System.String.Join method.

2

3 [C#]

4 using System;

5 public class StringJoin {

6 public static void Main() {

7 String[] strAry = { "Red", "Green", "Blue" };

8 Console.WriteLine( String.Join( ":: ", strAry, 1, 2 ) );

9 }

10 }

11 The output is

12

13 Green:: Blue

14

# 1 String.LastIndexOf(System.String, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOf(string value, int32  
5 startIndex)  
  
6 [C#]  
7 public int LastIndexOf(string value, int startIndex)
```

## 8 Summary

9 Returns the index of the last occurrence of a specified `System.String` within the current  
10 instance, starting at a given position.

## 11 Parameters

Parameter	Description
<i>value</i>	A <code>System.String</code> .
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.

12

## 13 Return Value

14 A `System.Int32` that indicates the result of the search for *value* in the current instance  
15 as follows:

Return Value	Description
A zero-based number equal to the index of the start of the last substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found.
-1	<i>value</i> was not found.

16

## 17 Description

18 This method searches for the last occurrence of the specified `System.String` between  
19 the start of the string and the indicated index.

20

21 This method is case-sensitive.

1 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is less than zero or greater than or equal to the length of the current instance.

2

3

# 1 String.LastIndexOf(System.String, 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOf(string value, int32  
5 startIndex, int32 count)  
  
6 [C#]  
7 public int LastIndexOf(string value, int startIndex, int count)
```

## 8 Summary

9 Returns the index of the last occurrence of a specified System.String in the provided  
10 range of the current instance.

## 11 Parameters

Parameter	Description
<i>value</i>	The substring to search for.
<i>startIndex</i>	A System.Int32 containing the index of the current instance from which to start searching.
<i>count</i>	A System.Int32 containing the range of the current instance at which to end searching.

## 12 13 Return Value

14 A System.Int32 that indicates the result of the search for *value* in the current instance as  
15 follows:

Return Value	Description
A zero-based number equal to the index of the start of the last substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found.
-1	<i>value</i> was not found.

## 16 17 Description

- 1 The search begins at *startIndex* and continues until *startIndex - count + 1*.
- 2
- 3 This method is case-sensitive.

4 **Exceptions**

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>count</i> is less than zero.
	-or- <i>startIndex - count</i> is smaller than -1.

- 5
- 6

# String.LastIndexOf(System.String) Method

```
[ILAsm]  
.method public hidebysig instance int32 LastIndexOf(string value)  
  
[C#]  
public int LastIndexOf(string value)
```

## Summary

Returns the index of the last occurrence of a specified `System.String` within the current instance.

## Parameters

Parameter	Description
<i>value</i>	A <code>System.String</code> .

## Return Value

A `System.Int32` that indicates the result of the search for *value* in the current instance as follows:

Return Value	Description
A zero-based number equal to the index of the start of the last substring in the current instance that is equal to <i>value</i> .	<i>value</i> was found.
-1	<i>value</i> was not found.

## Description

The search is case-sensitive.

## Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>value</i> is a null reference.

1

2

# 1 String.LastIndexOf(System.Char, 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOf(valuetype System.Char  
5 value, int32 startIndex, int32 count)  
  
6 [C#]  
7 public int LastIndexOf(char value, int startIndex, int count)
```

## 8 Summary

9 Returns the index of the last occurrence of a specified character in the provided range of  
10 the current instance.

## 11 Parameters

Parameter	Description
<i>value</i>	A Unicode character to locate.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.
<i>count</i>	A <code>System.Int32</code> containing the range of the current instance at which to end searching.

## 12 13 Return Value

14 A `System.Int32` containing the index of the last occurrence of *value* in the current  
15 instance if found between *startIndex* and (*startIndex* - *count* + 1); otherwise, -1.

## 16 Description

17 This method is case-sensitive.

## 18 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>count</i> is less than zero. -or-

	<i>startIndex</i> - <i>count</i> is less than -1.
--	---

1

2

# 1 String.LastIndexOf(System.Char, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOf(valuetype System.Char  
5 value, int32 startIndex)  
  
6 [C#]  
7 public int LastIndexOf(char value, int startIndex)
```

## 8 Summary

9 Returns the index of the last occurrence of a specified character within the current  
10 instance.

## 11 Parameters

Parameter	Description
<i>value</i>	A Unicode character to locate.
<i>startIndex</i>	A <code>System.Int32</code> containing the index in the current instance from which to begin searching.

12

## 13 Return Value

14 A `System.Int32` containing the index of the last occurrence of *value* in the current  
15 instance, if found; otherwise, -1.

## 16 Description

17 This method searches for the last occurrence of the specified character between the  
18 start of the string and the indicated index.

19

20 This method is case-sensitive.

## 21 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>value</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is less than zero or greater than the length of the current instance.

1

## 2 **Example**

3 The following example demonstrates the `System.String.LastIndexOf` method.

4

5 [C#]

6 `using System;`

7 `public class StringLastIndexOfTest {`

8  `public static void Main() {`

9  `String str = "aa bb cc dd";`

10

11  `Console.WriteLine( str.LastIndexOf('d', 8) );`

12  `Console.WriteLine( str.LastIndexOf('b', 8) );`

13  `}`

14  `}`

15 The output is

16

17 -1

18

19

20 4

21

22

# 1 String.LastIndexOf(System.Char) Method

```
2 [ILAsm]  
3 .method public hidebysig instance int32 LastIndexOf(valuetype System.Char  
4 value)  
5 [C#]  
6 public int LastIndexOf(char value)
```

## 7 Summary

8 Returns the index of the last occurrence of a specified character within the current  
9 instance.

## 10 Parameters

Parameter	Description
<i>value</i>	The Unicode character to locate.

11

## 12 Return Value

13 A `System.Int32` containing the index of the last occurrence of *value* in the current  
14 instance, if found; otherwise, -1.

## 15 Description

16 This method is case-sensitive.

17

# 1 String.LastIndexOfAny(System.Char[])

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOfAny(char[] anyOf)  
5 [C#]  
6 public int LastIndexOfAny(char[] anyOf)
```

### 7 Summary

8 Returns the index of the last occurrence of any element of a specified array of  
9 characters in the current instance.

### 10 Parameters

Parameter	Description
<i>anyOf</i>	An array of Unicode characters.

### 11 Return Value

13 A `System.Int32` containing the index of the last occurrence of any element of *anyOf* in  
14 the current instance, if found; otherwise, -1.

### 15 Description

16 This method is case-sensitive.

### 17 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.

18  
19

# 1 String.LastIndexOfAny(System.Char[], 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOfAny(char[] anyOf, int32  
5 startIndex)  
  
6 [C#]  
7 public int LastIndexOfAny(char[] anyOf, int startIndex)
```

## 8 Summary

9 Returns the index of the last occurrence of any element of a specified array of  
10 characters in the current instance.

## 11 Parameters

Parameter	Description
<i>anyOf</i>	An array of Unicode characters.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.

12

## 13 Return Value

14 A `System.Int32` containing the index of the last occurrence of any element of *anyOf* in  
15 the current instance, if found; otherwise, -1.

## 16 Description

17 This method searches for the last occurrence of the specified characters between the  
18 start of the string and the indicated index.

19

20 This method is case-sensitive.

## 21 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is less than zero or greater than or equal to the length of the current instance.

22



# 1 String.LastIndexOfAny(System.Char[], 2 System.Int32, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance int32 LastIndexOfAny(char[] anyOf, int32  
5 startIndex, int32 count)  
  
6 [C#]  
7 public int LastIndexOfAny(char[] anyOf, int startIndex, int count)
```

## 8 Summary

9 Returns the index of the last occurrence of any of specified characters in the provided  
10 range of the current instance.

## 11 Parameters

Parameter	Description
<i>anyOf</i>	An array of Unicode characters.
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start searching.
<i>count</i>	A <code>System.Int32</code> containing the range of the current instance at which to end searching.

## 12 13 Return Value

14 A `System.Int32` containing the index of the last occurrence of any element of *anyOf* if  
15 found between *startIndex* and (*startIndex* - *count* + 1); otherwise, -1.

## 16 Description

17 The search begins at *startIndex* and continues until *startIndex* - *count* + 1. The  
18 character at *startIndex* - *count* is not included in the search.

19  
20 This method is case-sensitive.

## 21 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>anyOf</i> is a null reference.

**System.ArgumentOutOfRangeException**

*startIndex* or *count* is less than zero.

-or-

*startIndex* - *count* is smaller than -1.

1

2

# 1 String.op\_Equality(System.String, 2 System.String) Method

```
3 [ILAsm]  
4 .method public hidebysig static specialname bool op_Equality(string a,  
5 string b)  
  
6 [C#]  
7 public static bool operator ==(String a, String b)
```

## 8 Summary

9 Returns a System.Boolean value indicating whether the two specified string values are  
10 equal to each other.

## 11 Parameters

Parameter	Description
<i>a</i>	The first System.String to compare.
<i>b</i>	The second System.String to compare.

12

## 13 Return Value

14 true if *a* and *b* represent the same string value; otherwise, false.

15

# 1 String.op\_Inequality(System.String, 2 System.String) Method

```
3 [ILAsm]  
4 .method public hidebysig static specialname bool op_Inequality(string a,  
5 string b)  
  
6 [C#]  
7 public static bool operator !=(String a, String b)
```

## 8 Summary

9 Returns a System.Boolean value indicating whether the two string values are not equal  
10 to each other.

## 11 Parameters

Parameter	Description
<i>a</i>	The first System.String to compare.
<i>b</i>	The second System.String to compare.

## 12 13 Return Value

14 true if *a* and *b* do not represent the same string value; otherwise, false.  
15

# 1 String.PadLeft(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string PadLeft(int32 totalWidth)  
4 [C#]  
5 public string PadLeft(int totalWidth)
```

## 6 Summary

7 Right-aligns the characters in the current instance, padding with spaces on the left, for a  
8 specified total length.

## 9 Parameters

Parameter	Description
<i>totalWidth</i>	A <code>System.Int32</code> containing the number of characters in the resulting string.

10

## 11 Return Value

12 A new `System.String` that is equivalent to the current instance right-aligned and  
13 padded on the left with as many spaces as needed to create a length of *totalWidth*. If  
14 *totalWidth* is less than the length of the current instance, returns a new `System.String`  
15 that is identical to the current instance.

## 16 Description

17 [Note: A space in Unicode format is defined as the hexadecimal value 0x20.]  
18  
19

## 20 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	<i>totalWidth</i> is less than zero.

21

22

# 1 String.PadLeft(System.Int32, System.Char)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string PadLeft(int32 totalWidth,  
5 valuetype System.Char paddingChar)  
  
6 [C#]  
7 public string PadLeft(int totalWidth, char paddingChar)
```

### 8 Summary

9 Right-aligns the characters in the current instance, padding on the left with a specified  
10 Unicode character, for a specified total length.

### 11 Parameters

Parameter	Description
<i>totalWidth</i>	A <code>System.Int32</code> containing the number of characters in the resulting string.
<i>paddingChar</i>	A <code>System.Char</code> that specifies the padding character to use.

12

### 13 Return Value

14 A new `System.String` that is equivalent to the current instance right-aligned and  
15 padded on the left with as many *paddingChar* characters as needed to create a length of  
16 *totalWidth*. If *totalWidth* is less than the length of the current instance, returns a new  
17 `System.String` that is identical to the current instance.

### 18 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	<i>totalWidth</i> is less than zero.

19

20

# 1 String.PadRight(System.Int32, System.Char)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string PadRight(int32 totalWidth,  
5 valuetype System.Char paddingChar)  
  
6 [C#]  
7 public string PadRight(int totalWidth, char paddingChar)
```

### 8 Summary

9 Left-aligns the characters in the current instance, padding on the right with a specified  
10 Unicode character, for a specified total number of characters.

### 11 Parameters

Parameter	Description
<i>totalWidth</i>	A <code>System.Int32</code> containing the number of characters in the resulting string.
<i>paddingChar</i>	A <code>System.Char</code> that specifies the padding character to use.

12

### 13 Return Value

14 A new `System.String` that is equivalent to the current instance left aligned and padded  
15 on the right with as many *paddingChar* characters as needed to create a length of  
16 *totalWidth*. If *totalWidth* is less than the length of the current instance, returns a new  
17 `System.String` that is identical to the current instance.

### 18 Exceptions

Exception	Condition
<code>System.ArgumentException</code>	<i>totalWidth</i> is less than zero.

19

20

# 1 String.PadRight(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string PadRight(int32 totalWidth)  
4 [C#]  
5 public string PadRight(int totalWidth)
```

## 6 Summary

7 Left-aligns the characters in the current instance, padding with spaces on the right, for a  
8 specified total number of characters.

## 9 Parameters

Parameter	Description
<i>totalWidth</i>	A <code>System.Int32</code> containing the number of characters in the resulting string.

10

## 11 Return Value

12 A new `System.String` that is equivalent to this instance left aligned and padded on the  
13 right with as many spaces as needed to create a length of *totalWidth*. If *totalWidth* is  
14 less than the length of the current instance, returns a new `System.String` that is  
15 identical to the current instance.

## 16 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	<i>totalWidth</i> is less than zero.

17

18

# 1 String.Remove(System.Int32, System.Int32)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string Remove(int32 startIndex, int32  
5 count)  
6 [C#]  
7 public string Remove(int startIndex, int count)
```

### 8 Summary

9 Deletes a specified number of characters from the current instance beginning at a  
10 specified index.

### 11 Parameters

Parameter	Description
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the current instance from which to start deleting characters.
<i>count</i>	A <code>System.Int32</code> containing the number of characters to delete.

12

### 13 Return Value

14 A new `System.String` that is equivalent to the current instance without the specified  
15 range characters.

### 16 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>count</i> is less than zero.  -or- <i>startIndex</i> plus <i>count</i> is greater than the length of the current instance.

17

18

# 1 String.Replace(System.String, System.String) 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string Replace(string oldValue, string  
5 newValue)  
6 [C#]  
7 public string Replace(string oldValue, string newValue)
```

## 8 Summary

9 Replaces all instances of a specified substring within the current instance with another  
10 specified string.

## 11 Parameters

Parameter	Description
<i>oldValue</i>	A <code>System.String</code> containing the string value to be replaced.
<i>newValue</i>	A <code>System.String</code> containing the string value to replace all occurrences of <i>oldValue</i> . Can be a null reference.

12

## 13 Return Value

14 A `System.String` equivalent to the current instance with all occurrences of *oldValue*  
15 replaced with *newValue*. If the replacement value is a null reference, the specified  
16 substring is removed.

17

# 1 String.Replace(System.Char, System.Char)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string Replace(valuetype System.Char  
5 oldChar, valuetype System.Char newChar)  
  
6 [C#]  
7 public string Replace(char oldChar, char newChar)
```

### 8 Summary

9 Replaces all instances of a specified Unicode character with another specified Unicode  
10 character.

### 11 Parameters

Parameter	Description
<i>oldChar</i>	The Unicode character to be replaced.
<i>newChar</i>	The Unicode character to replace all occurrences of <i>oldChar</i> .

### 12

### 13 Return Value

14 A `System.String` equivalent to the current instance with all occurrences of *oldChar*  
15 replaced with *newChar*.

16

# 1 String.Split(System.Char[]) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string[] Split(char[] separator)  
4 [C#]  
5 public string[] Split(params char[] separator)
```

## 6 Summary

7 Returns substrings of the current instance that are delimited by the specified characters.

## 8 Parameters

Parameter	Description
<i>separator</i>	A <code>System.Char</code> array of delimiters. Can be a null reference.

## 9 Return Value

11 A `System.String` array containing the results of the split operation as follows:

Return Value	Description
A single-element array containing the current instance.	None of the elements of <i>separator</i> are contained in the current instance.
A multi-element <code>System.String</code> array, each element of which is a substring of the current instance that was delimited by one or more characters in <i>separator</i> .	At least one element of <i>separator</i> is contained in the current instance.
A multi-element <code>System.String</code> array, each element of which is a substring of the current instance that was delimited by white space characters.	The current instance contains white space characters and <i>separator</i> is a null reference or an empty array.

## 12 Description

14 `System.String.Empty` is returned for any substring where two delimiters are adjacent  
15 or a delimiter is found at the beginning or end of the current instance.

16 Delimiter characters are not included in the substrings.

# 1 String.Split(System.Char[], System.Int32)

## 2 Method

```
3 [ILAsm]  
4 .method public hidebysig instance string[] Split(char[] separator, int32  
5 count)  
  
6 [C#]  
7 public string[] Split(char[] separator, int count)
```

### 8 Summary

9 Returns substrings of the current instance that are delimited by the specified characters.

### 10 Parameters

Parameter	Description
<i>separator</i>	An array of Unicode characters that delimit the substrings in the current instance, an empty array containing no delimiters, or a null reference.
<i>count</i>	A <code>System.Int32</code> containing the maximum number of array elements to return.

11

### 12 Return Value

13 A `System.String` array containing the results of the split operation as follows:

Return Value	Description
A single-element array containing the current instance.	None of the elements of <i>separator</i> are contained in the current instance.
A multi-element <code>System.String</code> array, each element of which is a substring of the current instance that was delimited by one or more characters in <i>separator</i>	At least one element of <i>separator</i> is contained in the current instance.
A multi-element <code>System.String</code> array, each element of which is a substring of the current instance that was delimited by white space characters.	The current instance contains white space characters and <i>separator</i> is a null reference or an empty array.

14

### 15 Description

1 `System.String.Empty` is returned for any substring where two delimiters are adjacent  
2 or a delimiter is found at the beginning or end of the current instance.  
3  
4 Delimiter characters are not included in the substrings.  
5  
6 If there are more substrings in the current instance than the maximum specified  
7 number, the first *count* -1 elements of the array contain the first *count* - 1 substrings.  
8 The remaining characters in the current instance are returned in the last element of the  
9 array.

10 **Exceptions**

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>count</i> is negative.

11

12

# String.StartsWith(System.String) Method

```
[ILAsm]  
.method public hidebysig instance bool StartsWith(string value)  
  
[C#]  
public bool StartsWith(string value)
```

## Summary

Returns a `System.Boolean` value that indicates whether the start of the current instance matches the specified `System.String`.

## Parameters

Parameter	Description
<i>value</i>	A <code>System.String</code> .

## Return Value

`true` if the start of the current instance is equal to *value*; `false` if *value* is not equal to the start of the current instance or is longer than the current instance.

## Description

This method compares *value* with the substring at the start of the current instance that has a length of *value.Length*. If *value.Length* is greater than the length of the current instance or the relevant substring of the current instance is not equal to *value*, this method returns `false`; otherwise, this method returns `true`.

The comparison is case-sensitive.

## Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	<i>value</i> is a null reference.

# 1 String.Substring(System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance string Substring(int32 startIndex, int32  
5 length)  
6 [C#]  
7 public string Substring(int startIndex, int length)
```

## 8 Summary

9 Retrieves a substring from the current instance, starting from a specified index,  
10 continuing for a specified length.

## 11 Parameters

Parameter	Description
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the start of the substring in the current instance.
<i>length</i>	A <code>System.Int32</code> containing the number of characters in the substring.

12

## 13 Return Value

14 A `System.String` containing the substring of the current instance with the specified  
15 length that begins at the specified position. Returns `System.String.Empty` if *startIndex*  
16 is equal to the length of the current instance and *length* is zero.

## 17 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>length</i> is greater than the length of the current instance. -or- <i>startIndex</i> or <i>length</i> is less than zero.

18

19

# 1 String.Substring(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string Substring(int32 startIndex)  
4 [C#]  
5 public string Substring(int startIndex)
```

## 6 Summary

7 Retrieves a substring from the current instance, starting from a specified index.

## 8 Parameters

Parameter	Description
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the start of the substring in the current instance.

9

## 10 Return Value

11 A `System.String` equivalent to the substring that begins at *startIndex* of the current  
12 instance. Returns `System.String.Empty` if *startIndex* is equal to the length of the  
13 current instance.

## 14 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> is less than zero or greater than the length of the current instance.

15

16

1

## 2 String.System.Collections.Generic.IEnumerable<System.Char>.GetEnumerator() Method

4

```
5 [ILAsm]  
6 .method private hidebysig virtual abstract class  
7 System.Collections.Generic.IEnumerator<char>  
8 System.Collections.Generic.IEnumerable<char>.GetEnumerator()
```

8

```
9 [C#]  
10 IEnumerator<Char> IEnumerable<Char>.GetEnumerator()
```

### 10 Summary

11 This method is implemented to support the  
12 System.Collections.Generic.IEnumerable<System.Char> interface.

13

1  
2 **String.System.Collections.IEnumerable.GetEnumerator() Method**  
3

```
4 [ILAsm]  
5 .method private final hidebysig virtual class  
6 System.Collections.IEnumerator  
7 System.Collections.IEnumerable.GetEnumerator()  
  
8 [C#]  
9 IEnumerator IEnumerable.GetEnumerator()
```

10 **Summary**

11 Implemented to support the System.Collections.IEnumerable interface. [Note: For  
12 more information, see System.Collections.IEnumerable.GetEnumerator.]

13

# 1 String.ToCharArray() Method

```
2 [ILAsm]  
3 .method public hidebysig instance char[] ToCharArray()  
  
4 [C#]  
5 public char[] ToCharArray()
```

## 6 Summary

7 Copies the characters in the current instance to a Unicode character array.

## 8 Return Value

9 A *System.Char* array whose elements are the individual characters of the current  
10 instance. If the current instance is an empty string, the array returned by this method is  
11 empty and has a zero length.

12

# 1 String.ToCharArray(System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig instance char[] ToCharArray(int32 startIndex,  
5 int32 length)  
  
6 [C#]  
7 public char[] ToCharArray(int startIndex, int length)
```

## 8 Summary

9 Copies the characters in a specified substring in the current instance to a Unicode  
10 character array.

## 11 Parameters

Parameter	Description
<i>startIndex</i>	A <code>System.Int32</code> containing the index of the start of a substring in the current instance.
<i>length</i>	A <code>System.Int32</code> containing the length of the substring in the current instance.

12

## 13 Return Value

14 A `System.Char` array whose elements are the *length* number of characters in the current  
15 instance, starting from the index *startIndex* in the current instance. If the specified  
16 length is zero, the entire string is copied starting from the beginning of the current  
17 instance, and ignoring the value of *startIndex*. If the current instance is an empty string,  
18 the returned array is empty and has a zero length.

## 19 Exceptions

Exception	Condition
<b>System.ArgumentOutOfRangeException</b>	<i>startIndex</i> or <i>length</i> is less than zero.  -or- <i>startIndex</i> plus <i>length</i> is greater than the length of the current instance.

20

21

# 1 String.ToLower() Method

```
2 [ILAsm]  
3 .method public hidebysig instance string ToLower()  
4 [C#]  
5 public string ToLower()
```

## 6 Summary

7 Returns a copy of this System.String in lowercase.

## 8 Return Value

9 A System.String in lowercase..

## 10 Description

11 This method takes into account the culture (if any) of the current thread.

12

# 1 String.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 instance.

## 8 Return Value

9 The current `System.String`.

## 10 Description

11 [*Note:* This method overrides `System.Object.ToString`.]  
12  
13  
14

# 1 String.ToString(System.IFormatProvider)

## 2 Method

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

### 8 Summary

9 Returns this instance of `String`; no actual conversion is performed.

### 10 Parameters

Parameter	Description
<i>provider</i>	(Reserved) A <code>System.IFormatProvider</code> interface implementation which supplies culture-specific formatting information.

### 11 Return Value

13 This `String`.

### 14 Description

15 *provider* is reserved, and does not currently participate in this operation.

16

# 1 String.ToUpper() Method

```
2 [ILAsm]  
3 .method public hidebysig instance string ToUpper()  
4 [C#]  
5 public string ToUpper()
```

## 6 Summary

7 Returns a copy of the current instance with all elements converted to uppercase, using  
8 default properties.

## 9 Return Value

10 A new `System.String` in uppercase.

11

# 1 String.Trim(System.Char[]) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string Trim(char[] trimChars)  
4 [C#]  
5 public string Trim(params char[] trimChars)
```

## 6 Summary

7 Removes all occurrences of a set of characters provided in a character `System.Array`  
8 from the beginning and end of the current instance.

## 9 Parameters

Parameter	Description
<i>trimChars</i>	An array of Unicode characters. Can be a null reference.

10

## 11 Return Value

12 A new `System.String` equivalent to the current instance with the characters in  
13 *trimChars* removed from its beginning and end. If *trimChars* is a null reference, all of  
14 the white space characters are removed from the beginning and end of the current  
15 instance.

16

# 1 String.Trim() Method

```
2 [ILAsm]  
3 .method public hidebysig instance string Trim()  
4 [C#]  
5 public string Trim()
```

## 6 Summary

7 Removes all occurrences of white space characters from the beginning and end of the  
8 current instance.

## 9 Return Value

10 A new *System.String* equivalent to the current instance after white space characters  
11 are removed from its beginning and end.

12

# 1 String.TrimEnd(System.Char[]) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string TrimEnd(char[] trimChars)  
4 [C#]  
5 public string TrimEnd(params char[] trimChars)
```

## 6 Summary

7 Removes all occurrences of a set of characters specified in a Unicode character  
8 System.Array from the end of the current instance.

## 9 Parameters

Parameter	Description
<i>trimChars</i>	An array of Unicode characters. Can be a null reference.

10

## 11 Return Value

12 A new System.String equivalent to the current instance with characters in *trimChars*  
13 removed from its end. If *trimChars* is a null reference, white space characters are  
14 removed.

15

# 1 String.TrimStart(System.Char[]) Method

```
2 [ILAsm]  
3 .method public hidebysig instance string TrimStart(char[] trimChars)  
4 [C#]  
5 public string TrimStart(params char[] trimChars)
```

## 6 Summary

7 Removes all occurrences of a set of characters specified in a Unicode character array  
8 from the beginning of the current instance.

## 9 Parameters

Parameter	Description
<i>trimChars</i>	An array of Unicode characters or a null reference.

10

## 11 Return Value

12 A new *System.String* equivalent to the current instance with the characters in  
13 *trimChars* removed from its beginning. If *trimChars* is a null reference, white space  
14 characters are removed.

15

# 1 String.Chars Property

```
2 [ILAsm]  
3 .property valuetype System.Char Chars[int32 index] { public hideby sig  
4 specialname instance valuetype System.Char get_Chars(int32 index) }  
  
5 [C#]  
6 public char this[int index] { get; }
```

## 7 Summary

8 Gets the character at a specified position in the current instance.

## 9 Property Value

10 A Unicode character at the location *index* in the current instance.

## 11 Description

12 This property is read-only.

13  
14 *index* is the position of a character within a string. The first character in the string is at  
15 index 0. The length of a string is the number of characters it is made up of. The last  
16 accessible *index* of a string instance is its length - 1.

## 17 Exceptions

Exception	Condition
<b>System.IndexOutOfRangeException</b>	<i>index</i> is greater than or equal to the length of the current instance or less than zero.

18

19

# 1 String.Length Property

```
2 [ILAsm]
3 .property int32 Length { public hidebysig specialname instance int32
4 get_Length() }
5 [C#]
6 public int Length { get; }
```

## 7 Summary

8 Gets the number of characters in the current instance.

## 9 Property Value

10 A System.Int32 containing the number of characters in the current instance.

## 11 Description

12 This property is read-only.

## 13 Example

14 The following example demonstrates the System.String.Length property.

```
15 [C#]
16
17 using System;
18 public class StringLengthExample {
19     public static void Main() {
20         string str = "STRING";
21         Console.WriteLine( "The length of string {0} is {1}", str, str.Length );
22     }
23 }
```

24 The output is

```
25
26 The length of string STRING is 6
```

27