**The String and StringBuilder types**

Although StringBuilder and [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) both represent sequences of characters, they are implemented differently. [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) is an immutable type. That is, each operation that appears to modify a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object actually creates a new string.

For example, the call to the [String.Concat](http://msdn.microsoft.com/en-us/library/system.string.concat%28v=vs.110%29.aspx) method in the following C# example appears to change the value of a string variable named value. In fact, the [Concat](http://msdn.microsoft.com/en-us/library/system.string.concat%28v=vs.110%29.aspx) method returns a value object that has a different value and address from the value object that was passed to the method. Note that the example must be compiled using the /unsafe compiler option.

C#

using System;

public class Example

{

public unsafe static void Main()

{

string value = "This is the first sentence" + ".";

fixed (char\* start = value)

{

value = String.Concat(value, "This is the second sentence. ");

fixed (char\* current = value)

{

Console.WriteLine(start == current);

}

}

}

}

// The example displays the following output:

// False

For routines that perform extensive string manipulation (such as apps that modify a string numerous times in a loop), modifying a string repeatedly can exact a significant performance penalty. The alternative is to use StringBuilder, which is a mutable string class. Mutability means that once an instance of the class has been created, it can be modified by appending, removing, replacing, or inserting characters. A StringBuilder object maintains a buffer to accommodate expansions to the string. New data is appended to the buffer if room is available; otherwise, a new, larger buffer is allocated, data from the original buffer is copied to the new buffer, and the new data is then appended to the new buffer.

|  |
| --- |
| **Important noteImportant** |
| Although the StringBuilder class generally offers better performance than the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) class, you should not automatically replace [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) with StringBuilder whenever you want to manipulate strings. Performance depends on the size of the string, the amount of memory to be allocated for the new string, the system on which your app is executing, and the type of operation. You should be prepared to test your app to determine whether StringBuilder actually offers a significant performance improvement. |

Consider using the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) class under these conditions:

* When the number of changes that your app will make to a string is small. In these cases, StringBuilder might offer negligible or no performance improvement over [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx).
* When you are performing a fixed number of concatenation operations, particularly with string literals. In this case, the compiler might combine the concatenation operations into a single operation.
* When you have to perform extensive search operations while you are building your string. The StringBuilder class lacks search methods such as IndexOf or StartsWith. You'll have to convert the StringBuilder object to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) for these operations, and this can negate the performance benefit from using StringBuilder. For more information, see the [Searching the text in a StringBuilder object](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder%28v=vs.110%29.aspx#Searching) section.

Consider using the StringBuilder class under these conditions:

* When you expect your app to make an unknown number of changes to a string at design time (for example, when you are using a loop to concatenate a random number of strings that contain user input).
* When you expect your app to make a significant number of changes to a string.

**How StringBuilder works**

The [StringBuilder.Length](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.length%28v=vs.110%29.aspx) property indicates the number of characters the StringBuilder object currently contains. If you add characters to the StringBuilder object, its length increases until it equals the size of the [StringBuilder.Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property, which defines the number of characters that the object can contain. If the number of added characters causes the length of the StringBuilder object to exceed its current capacity, new memory is allocated, the value of the [Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property is doubled, new characters are added to the StringBuilder object, and its [Length](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.length%28v=vs.110%29.aspx) property is adjusted. Additional memory for the StringBuilder object is allocated dynamically until it reaches the value defined by the [StringBuilder.MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) property. When the maximum capacity is reached, no further memory can be allocated for the StringBuilder object, and trying to add characters or expand it beyond its maximum capacity throws either an [ArgumentOutOfRangeException](http://msdn.microsoft.com/en-us/library/system.argumentoutofrangeexception%28v=vs.110%29.aspx) or an [OutOfMemoryException](http://msdn.microsoft.com/en-us/library/system.outofmemoryexception%28v=vs.110%29.aspx) exception.

The following example illustrates how a StringBuilder object allocates new memory and increases its capacity dynamically as the string assigned to the object expands. The code creates a StringBuilder object by calling its default (parameterless) constructor. The default capacity of this object is 16 characters, and its maximum capacity is more than 2 billion characters. Appending the string "This is a sentence." results in a new memory allocation because the string length (19 characters) exceeds the default capacity of the StringBuilder object. The capacity of the object doubles to 32 characters, the new string is added, and the length of the object now equals 19 characters. The code then appends the string "This is an additional sentence." to the value of the StringBuilder object 11 times. Whenever the append operation causes the length of the StringBuilder object to exceed its capacity, its existing capacity is doubled and the [Append](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.append%28v=vs.110%29.aspx) operation succeeds.

C#

using System;

using System.Reflection;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder sb = new StringBuilder();

ShowSBInfo(sb);

sb.Append("This is a sentence.");

ShowSBInfo(sb);

for (int ctr = 0; ctr <= 10; ctr++) {

sb.Append("This is an additional sentence.");

ShowSBInfo(sb);

}

}

private static void ShowSBInfo(StringBuilder sb)

{

foreach (var prop in sb.GetType().GetProperties()) {

if (prop.GetIndexParameters().Length == 0)

Console.Write("{0}: {1:N0} ", prop.Name, prop.GetValue(sb));

}

Console.WriteLine();

}

}

// The example displays the following output:

// Capacity: 16 MaxCapacity: 2,147,483,647 Length: 0

// Capacity: 32 MaxCapacity: 2,147,483,647 Length: 19

// Capacity: 64 MaxCapacity: 2,147,483,647 Length: 50

// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 81

// Capacity: 128 MaxCapacity: 2,147,483,647 Length: 112

// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 143

// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 174

// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 205

// Capacity: 256 MaxCapacity: 2,147,483,647 Length: 236

// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 267

// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 298

// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 329

// Capacity: 512 MaxCapacity: 2,147,483,647 Length: 360

**Memory allocation**

The default capacity of a StringBuilder object is 16 characters, and its default maximum capacity is [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx). These default values are used if you call the [StringBuilder()](http://msdn.microsoft.com/en-us/library/dsd3ey60%28v=vs.110%29.aspx) and [StringBuilder(String)](http://msdn.microsoft.com/en-us/library/5fxz4s6t%28v=vs.110%29.aspx) constructors.

You can explicitly define the initial capacity of a StringBuilder object in the following ways:

* By calling any of the StringBuilder constructors that includes a capacity parameter when you create the object.
* By explicitly assigning a new value to the [StringBuilder.Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property to expand an existing StringBuilder object. Note that the property throws an exception if the new capacity is less than the existing capacity or greater than the StringBuilder object's maximum capacity.
* By calling the [StringBuilder.EnsureCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.ensurecapacity%28v=vs.110%29.aspx) method with the new capacity. The new capacity must not be greater than the StringBuilder object's maximum capacity. However, unlike an assignment to the [Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property, [EnsureCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.ensurecapacity%28v=vs.110%29.aspx) does not throw an exception if the desired new capacity is less than the existing capacity; in this case, the method call has no effect.

If the length of the string assigned to the StringBuilder object in the constructor call exceeds either the default capacity or the specified capacity, the [Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property is set to the length of the string specified with the value parameter.

You can explicitly define the maximum capacity of a StringBuilder object by calling the [StringBuilder(Int32, Int32)](http://msdn.microsoft.com/en-us/library/hbb08bby%28v=vs.110%29.aspx) constructor. You can't change the maximum capacity by assigning a new value to the [MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) property, because it is read-only.

As the previous section shows, whenever the existing capacity is inadequate, additional memory is allocated and the capacity of a StringBuilder object doubles up to the value defined by the [MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) property.

In general, the default capacity and maximum capacity are adequate for most apps. You might consider setting these values under the following conditions:

* If the eventual size of the StringBuilder object is likely to grow exceedingly large, typically in excess of several megabytes. In this case, there may be some performance benefit from setting the initial [Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) property to a significantly high value to eliminate the need for too many memory reallocations.
* If your app is running on a system with limited memory. In this case, you may want to consider setting the [MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) property to less than [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx) if your app is handling large strings that may cause it to execute in a memory-constrained environment.

**Instantiating a StringBuilder object**

You instantiate a StringBuilder object by calling one of its six overloaded class constructors, which are listed in the following table. Three of the constructors instantiate a StringBuilder object whose value is an empty string, but set its [Capacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.capacity%28v=vs.110%29.aspx) and [MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) values differently. The remaining three constructors define a StringBuilder object that has a specific string value and capacity. Two of the three constructors use the default maximum capacity of [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx), whereas the third allows you to set the maximum capacity.

|  |  |  |  |
| --- | --- | --- | --- |
| **Constructor** | **String value** | **Capacity** | **Maximum capacity** |
| [StringBuilder()](http://msdn.microsoft.com/en-us/library/dsd3ey60%28v=vs.110%29.aspx) | [String.Empty](http://msdn.microsoft.com/en-us/library/system.string.empty%28v=vs.110%29.aspx) | 16 | [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx) |
| [StringBuilder(Int32)](http://msdn.microsoft.com/en-us/library/h1h0a5sy%28v=vs.110%29.aspx) | [String.Empty](http://msdn.microsoft.com/en-us/library/system.string.empty%28v=vs.110%29.aspx) | Defined by the capacity parameter | [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx) |
| [StringBuilder(Int32, Int32)](http://msdn.microsoft.com/en-us/library/hbb08bby%28v=vs.110%29.aspx) | [String.Empty](http://msdn.microsoft.com/en-us/library/system.string.empty%28v=vs.110%29.aspx) | Defined by the capacity parameter | Defined by the maxCapacity parameter |
| [StringBuilder(String)](http://msdn.microsoft.com/en-us/library/5fxz4s6t%28v=vs.110%29.aspx) | Defined by the value parameter | 16 or value. [Length](http://msdn.microsoft.com/en-us/library/system.string.length%28v=vs.110%29.aspx), whichever is greater | [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx) |
| [StringBuilder(String, Int32)](http://msdn.microsoft.com/en-us/library/zb91weab%28v=vs.110%29.aspx) | Defined by the value parameter | Defined by the capacity parameter or value. [Length](http://msdn.microsoft.com/en-us/library/system.string.length%28v=vs.110%29.aspx), whichever is greater. | [Int32.MaxValue](http://msdn.microsoft.com/en-us/library/system.int32.maxvalue%28v=vs.110%29.aspx) |
| [StringBuilder(String, Int32, Int32, Int32)](http://msdn.microsoft.com/en-us/library/y6zc419d%28v=vs.110%29.aspx) | Defined by value. [Substring](http://msdn.microsoft.com/en-us/library/system.string.substring%28v=vs.110%29.aspx)(startIndex, length) | Defined by the capacity parameter or value. [Length](http://msdn.microsoft.com/en-us/library/system.string.length%28v=vs.110%29.aspx), whichever is greater. | Defined by the maxCapacity parameter |

The following example uses three of these constructor overloads to instantiate StringBuilder objects.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

string value = "An ordinary string";

int index = value.IndexOf("An ") + 3;

int capacity = 0xFFFF;

// Instantiate a StringBuilder from a string.

StringBuilder sb1 = new StringBuilder(value);

ShowSBInfo(sb1);

// Instantiate a StringBuilder from string and define a capacity.

StringBuilder sb2 = new StringBuilder(value, capacity);

ShowSBInfo(sb2);

// Instantiate a StringBuilder from substring and define a capacity.

StringBuilder sb3 = new StringBuilder(value, index,

value.Length - index,

capacity );

ShowSBInfo(sb3);

}

public static void ShowSBInfo(StringBuilder sb)

{

Console.WriteLine("\nValue: {0}", sb.ToString());

foreach (var prop in sb.GetType().GetProperties()) {

if (prop.GetIndexParameters().Length == 0)

Console.Write("{0}: {1:N0} ", prop.Name, prop.GetValue(sb));

}

Console.WriteLine();

}

}

// The example displays the following output:

// Value: An ordinary string

// Capacity: 18 MaxCapacity: 2,147,483,647 Length: 18

//

// Value: An ordinary string

// Capacity: 65,535 MaxCapacity: 2,147,483,647 Length: 18

//

// Value: ordinary string

// Capacity: 65,535 MaxCapacity: 2,147,483,647 Length: 15

**Calling StringBuilder methods**

Most of the methods that modify the string in a StringBuilder instance return a reference to that same instance. This enables you to call StringBuilder methods in two ways:

* You can make individual method calls and ignore the return value, as the following example does.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder sb = new StringBuilder();

sb.Append("This is the beginning of a sentence, ");

sb.Replace("the beginning of ", "");

sb.Insert(sb.ToString().IndexOf("a ") + 2, "complete ");

sb.Replace(",", ".");

Console.WriteLine(sb.ToString());

}

}

// The example displays the following output:

// This is a complete sentence.

* You can make a series of method calls in a single statement. This can be convenient if you want to write a single statement that chains successive operations. The following example consolidates three method calls from the previous example into a single line of code.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder sb = new StringBuilder("This is the beginning of a sentence, ");

sb.Replace("the beginning of ", "").Insert(sb.ToString().IndexOf("a ") + 2,

"complete ").Replace(",", ".");

Console.WriteLine(sb.ToString());

}

}

// The example displays the following output:

// This is a complete sentence.

**Performing StringBuilder operations**

You can use the methods of the StringBuilder class to iterate, add, delete, or modify characters in a StringBuilder object.

[Iterating StringBuilder characters](javascript:void(0))

You can access the characters in a StringBuilder object by using the [StringBuilder.Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property. In C#, [Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) is an indexer; in Visual Basic, it is the default property of the StringBuilder class. This enables you to set or retrieve individual characters by using their index only, without explicitly referencing the [Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property. Characters in a StringBuilder object begin at index 0 (zero) and continue to index [Length](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.length%28v=vs.110%29.aspx) - 1.

The following example illustrates the [Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property. It appends ten random numbers to a StringBuilder object, and then iterates each character. If the character's Unicode category is [UnicodeCategory.DecimalDigitNumber](http://msdn.microsoft.com/en-us/library/system.globalization.unicodecategory%28v=vs.110%29.aspx), it decreases the number by 1 (or changes the number to 9 if its value is 0). The example displays the contents of the StringBuilder object both before and after the values of individual characters were changed.

C#

using System;

using System.Globalization;

using System.Text;

public class Example

{

public static void Main()

{

Random rnd = new Random();

StringBuilder sb = new StringBuilder();

// Generate 10 random numbers and store them in a StringBuilder.

for (int ctr = 0; ctr <= 9; ctr++)

sb.Append(rnd.Next().ToString("N5"));

Console.WriteLine("The original string:");

Console.WriteLine(sb.ToString());

// Decrease each number by one.

for (int ctr = 0; ctr < sb.Length; ctr++) {

if (Char.GetUnicodeCategory(sb[ctr]) == UnicodeCategory.DecimalDigitNumber) {

int number = (int) Char.GetNumericValue(sb[ctr]);

number--;

if (number < 0) number = 9;

sb[ctr] = number.ToString()[0];

}

}

Console.WriteLine("\nThe new string:");

Console.WriteLine(sb.ToString());

}

}

// The example displays the following output:

// The original string:

// 1,457,531,530.00000940,522,609.000001,668,113,564.000001,998,992,883.000001,792,660,834.00

// 000101,203,251.000002,051,183,075.000002,066,000,067.000001,643,701,043.000001,702,382,508

// .00000

//

// The new string:

// 0,346,420,429.99999839,411,598.999990,557,002,453.999990,887,881,772.999990,681,559,723.99

// 999090,192,140.999991,940,072,964.999991,955,999,956.999990,532,690,932.999990,691,271,497

// .99999

[Adding text to a StringBuilder object](javascript:void(0))

The StringBuilder class includes the following methods for expanding the contents of a StringBuilder object:

* The [Append](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.append%28v=vs.110%29.aspx) method appends a string, a substring, a character array, a portion of a character array, a single character repeated multiple times, or the string representation of a primitive data type to a StringBuilder object.
* The [AppendLine](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.appendline%28v=vs.110%29.aspx) method appends a line terminator or a string along with a line terminator to a StringBuilder object.
* The [AppendFormat](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.appendformat%28v=vs.110%29.aspx) method appends a [composite format string](http://msdn.microsoft.com/en-us/library/txafckwd%28v=vs.110%29.aspx) to a StringBuilder object. The string representations of objects included in the result string can reflect the formatting conventions of the current system culture or a specified culture.
* The [Insert](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.insert%28v=vs.110%29.aspx) method inserts a string, a substring, multiple repetitions of a string, a character array, a portion of a character array, or the string representation of a primitive data type at a specified position in the StringBuilder object. The position is defined by a zero-based index.

The following example uses the [Append](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.append%28v=vs.110%29.aspx), [AppendLine](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.appendline%28v=vs.110%29.aspx), [AppendFormat](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.appendformat%28v=vs.110%29.aspx), and [Insert](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.insert%28v=vs.110%29.aspx) methods to expand the text of a StringBuilder object.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

// Create a StringBuilder object with no text.

StringBuilder sb = new StringBuilder();

// Append some text.

sb.Append('\*', 10).Append(" Adding Text to a StringBuilder Object ").Append('\*', 10);

sb.AppendLine("\n");

sb.AppendLine("Some code points and their corresponding characters:");

// Append some formatted text.

for (int ctr = 50; ctr <= 60; ctr++) {

sb.AppendFormat("{0,12:X4} {1,12}", ctr, Convert.ToChar(ctr));

sb.AppendLine();

}

// Find the end of the introduction to the column.

int pos = sb.ToString().IndexOf("characters:") + 11 +

Environment.NewLine.Length;

// Insert a column header.

sb.Insert(pos, String.Format("{2}{0,12:X4} {1,12}{2}", "Code Unit",

"Character", "\n"));

// Convert the StringBuilder to a string and display it.

Console.WriteLine(sb.ToString());

}

}

// The example displays the following output:

// \*\*\*\*\*\*\*\*\*\* Adding Text to a StringBuilder Object \*\*\*\*\*\*\*\*\*\*

//

// Some code points and their corresponding characters:

//

// Code Unit Character

// 0032 2

// 0033 3

// 0034 4

// 0035 5

// 0036 6

// 0037 7

// 0038 8

// 0039 9

// 003A :

// 003B ;

// 003C <

[Deleting text from a StringBuilder object](javascript:void(0))

The StringBuilder class includes methods that can reduce the size of the current StringBuilder instance. The [Clear](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.clear%28v=vs.110%29.aspx) method removes all characters and sets the [Length](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.length%28v=vs.110%29.aspx) property to zero. The [Remove](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.remove%28v=vs.110%29.aspx) method deletes a specified number of characters starting at a particular index position. In addition, you can remove characters from the end of a StringBuilder object by setting its [Length](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.length%28v=vs.110%29.aspx) property to a value that is less than the length of the current instance.

The following example removes some of the text from a StringBuilder object, displays its resulting capacity, maximum capacity, and length property values, and then calls the [Clear](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.clear%28v=vs.110%29.aspx) method to remove all the characters from the StringBuilder object.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder sb = new StringBuilder("A StringBuilder object");

ShowSBInfo(sb);

// Remove "object" from the text.

string textToRemove = "object";

int pos = sb.ToString().IndexOf(textToRemove);

if (pos >= 0) {

sb.Remove(pos, textToRemove.Length);

ShowSBInfo(sb);

}

// Clear the StringBuilder contents.

sb.Clear();

ShowSBInfo(sb);

}

public static void ShowSBInfo(StringBuilder sb)

{

Console.WriteLine("\nValue: {0}", sb.ToString());

foreach (var prop in sb.GetType().GetProperties()) {

if (prop.GetIndexParameters().Length == 0)

Console.Write("{0}: {1:N0} ", prop.Name, prop.GetValue(sb));

}

Console.WriteLine();

}

}

// The example displays the following output:

// Value: A StringBuilder object

// Capacity: 22 MaxCapacity: 2,147,483,647 Length: 22

//

// Value: A StringBuilder

// Capacity: 22 MaxCapacity: 2,147,483,647 Length: 16

//

// Value:

// Capacity: 22 MaxCapacity: 2,147,483,647 Length: 0

[Modifying the text in a StringBuilder object](javascript:void(0))

The [StringBuilder.Replace](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.replace%28v=vs.110%29.aspx) method replaces all occurrences of a character or a string in the entire StringBuilder object or in a particular character range. The following example uses the [Replace](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.replace%28v=vs.110%29.aspx) method to replace all exclamation points (!) with question marks (?) in the StringBuilder object.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

StringBuilder MyStringBuilder = new StringBuilder("Hello World!");

MyStringBuilder.Replace('!', '?');

Console.WriteLine(MyStringBuilder);

}

}

// The example displays the following output:

// Hello World?

**Searching the text in a StringBuilder object**

The StringBuilder class does not include methods similar to the [String.Contains](http://msdn.microsoft.com/en-us/library/dy85x1sa%28v=vs.110%29.aspx), [String.IndexOf](http://msdn.microsoft.com/en-us/library/system.string.indexof%28v=vs.110%29.aspx), and [String.StartsWith](http://msdn.microsoft.com/en-us/library/system.string.startswith%28v=vs.110%29.aspx) methods provided by the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) class, which allow you to search the object for a particular character or a substring. Determining the presence or starting character position of a substring requires that you search a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) value by using either a string search method or a regular expression method. There are four ways to implement such searches, as the following table shows.

|  |  |  |
| --- | --- | --- |
| **Technique** | **Pros** | **Cons** |
| Search string values before adding them to the StringBuilder object. | Useful for determining whether a substring exists. | Cannot be used when the index position of a substring is important. |
| Call [ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) and search the returned [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object. | Easy to use if you assign all the text to a StringBuilder object, and then begin to modify it. | Cumbersome to repeatedly call [ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) if you must make modifications before all text is added to the StringBuilder object.  You must remember to work from the end of the StringBuilder object's text if you're making changes. |
| Use the [Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property to sequentially search a range of characters. | Useful if you're concerned with individual characters or a small substring. | Cumbersome if the number of characters to search is large or if the search logic is complex. |
| Convert the StringBuilder object to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object, and perform modifications on the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object. | Useful if the number of modifications is small. | Negates the performance benefit of the StringBuilder class if the number of modifications is large. |

Let's examine these techniques in greater detail.

* If the goal of the search is to determine whether a particular substring exists (that is, if you aren't interested in the position of the substring), you can search strings before storing them in the StringBuilder object. The following example provides one possible implementation. It defines a StringBuilderFinder class whose constructor is passed a reference to a StringBuilder object and the substring to find in the string. In this case, the example tries to determine whether recorded temperatures are in Fahrenheit or Celsius, and adds the appropriate introductory text to the beginning of the StringBuilder object. A random number generator is used to select an array that contains data in either degrees Celsius or degrees Fahrenheit.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

Random rnd = new Random();

string[] tempF = { "47.6F", "51.3F", "49.5F", "62.3F" };

string[] tempC = { "21.2C", "16.1C", "23.5C", "22.9C" };

string[][] temps = { tempF, tempC };

StringBuilder sb = new StringBuilder();

var f = new StringBuilderFinder(sb, "F");

var baseDate = new DateTime(2013, 5, 1);

String[] temperatures = temps[rnd.Next(2)];

bool isFahrenheit = false;

foreach (var temperature in temperatures) {

if (isFahrenheit)

sb.AppendFormat("{0:d}: {1}\n", baseDate, temperature);

else

isFahrenheit = f.SearchAndAppend(String.Format("{0:d}: {1}\n",

baseDate, temperature));

baseDate = baseDate.AddDays(1);

}

if (isFahrenheit) {

sb.Insert(0, "Average Daily Temperature in Degrees Fahrenheit");

sb.Insert(47, "\n\n");

}

else {

sb.Insert(0, "Average Daily Temperature in Degrees Celsius");

sb.Insert(44, "\n\n");

}

Console.WriteLine(sb.ToString());

}

}

public class StringBuilderFinder

{

private StringBuilder sb;

private String text;

public StringBuilderFinder(StringBuilder sb, String textToFind)

{

this.sb = sb;

this.text = textToFind;

}

public bool SearchAndAppend(String stringToSearch)

{

sb.Append(stringToSearch);

return stringToSearch.Contains(text);

}

}

// The example displays output similar to the following:

// Average Daily Temperature in Degrees Celsius

//

// 5/1/2013: 21.2C

// 5/2/2013: 16.1C

// 5/3/2013: 23.5C

// 5/4/2013: 22.9C

* Call the [StringBuilder.ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) method to convert the StringBuilder object to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object. You can search the string by using methods such as [String.LastIndexOf](http://msdn.microsoft.com/en-us/library/system.string.lastindexof%28v=vs.110%29.aspx) or [String.StartsWith](http://msdn.microsoft.com/en-us/library/system.string.startswith%28v=vs.110%29.aspx), or you can use regular expressions and the [Regex](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex%28v=vs.110%29.aspx) class to search for patterns. Because both StringBuilder and [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) objects use UTF-16 encoding to store characters, the index positions of characters, substrings, and regular expression matches are the same in both objects. This enables you to use StringBuilder methods to make changes at the same position at which that text is found in the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object.

|  |
| --- |
| **NoteNote** |
| If you adopt this approach, you should work from the end of the StringBuilder object to its beginning so that you don't have to repeatedly convert the StringBuilder object to a string. |

* The following example illustrates this approach. It stores four occurrences of each letter of the English alphabet in a StringBuilder object. It then converts the text to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object and uses a regular expression to identify the starting position of each four-character sequence. Finally, it adds an underscore before each four-character sequence except for the first sequence, and converts the first character of the sequence to uppercase.
* C#
* using System;
* using System.Text;
* using System.Text.RegularExpressions;
* public class Example
* {
* public static void Main()
* {
* // Create a StringBuilder object with 4 successive occurrences
* // of each character in the English alphabet.
* StringBuilder sb = new StringBuilder();
* for (ushort ctr = (ushort)'a'; ctr <= (ushort) 'z'; ctr++)
* sb.Append(Convert.ToChar(ctr), 4);
* // Create a parallel string object.
* String sbString = sb.ToString();
* // Determine where each new character sequence begins.
* String pattern = @"(\w)\1+";
* MatchCollection matches = Regex.Matches(sbString, pattern);
* // Uppercase the first occurrence of the sequence, and separate it
* // from the previous sequence by an underscore character.
* for (int ctr = matches.Count - 1; ctr >= 0; ctr--) {
* Match m = matches[ctr];
* sb[m.Index] = Char.ToUpper(sb[m.Index]);
* if (m.Index > 0) sb.Insert(m.Index, "\_");
* }
* // Display the resulting string.
* sbString = sb.ToString();
* int line = 0;
* do {
* int nChars = line \* 80 + 79 <= sbString.Length ?
* 80 : sbString.Length - line \* 80;
* Console.WriteLine(sbString.Substring(line \* 80, nChars));
* line++;
* } while (line \* 80 < sbString.Length);
* }
* }
* // The example displays the following output:
* // Aaaa\_Bbbb\_Cccc\_Dddd\_Eeee\_Ffff\_Gggg\_Hhhh\_Iiii\_Jjjj\_Kkkk\_Llll\_Mmmm\_Nnnn\_Oooo\_Pppp\_
* // Qqqq\_Rrrr\_Ssss\_Tttt\_Uuuu\_Vvvv\_Wwww\_Xxxx\_Yyyy\_Zzzz
* Use the [StringBuilder.Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property to sequentially search a range of characters in a StringBuilder object. This approach may not be practical if the number of characters to be searched is large or the search logic is particularly complex.

The following example is identical in functionality to the previous example but differs in implementation. It uses the [Chars](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.chars%28v=vs.110%29.aspx) property to detect when a character value has changed, inserts an underscore at that position, and converts the first character in the new sequence to uppercase.

C#

using System;

using System.Text;

public class Example

{

public static void Main()

{

// Create a StringBuilder object with 4 successive occurrences

// of each character in the English alphabet.

StringBuilder sb = new StringBuilder();

for (ushort ctr = (ushort) 'a'; ctr <= (ushort) 'z'; ctr++)

sb.Append(Convert.ToChar(ctr), 4);

// Iterate the text to determine when a new character sequence occurs.

int position = 0;

Char current = '\u0000';

do {

if (sb[position] != current) {

current = sb[position];

sb[position] = Char.ToUpper(sb[position]);

if (position > 0)

sb.Insert(position, "\_");

position += 2;

}

else {

position++;

}

} while (position <= sb.Length - 1);

// Display the resulting string.

String sbString = sb.ToString();

int line = 0;

do {

int nChars = line \* 80 + 79 <= sbString.Length ?

80 : sbString.Length - line \* 80;

Console.WriteLine(sbString.Substring(line \* 80, nChars));

line++;

} while (line \* 80 < sbString.Length);

}

}

// The example displays the following output:

// Aaaa\_Bbbb\_Cccc\_Dddd\_Eeee\_Ffff\_Gggg\_Hhhh\_Iiii\_Jjjj\_Kkkk\_Llll\_Mmmm\_Nnnn\_Oooo\_Pppp\_

// Qqqq\_Rrrr\_Ssss\_Tttt\_Uuuu\_Vvvv\_Wwww\_Xxxx\_Yyyy\_Zzzz

* Store all the unmodified text in the StringBuilder object, call the [StringBuilder.ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) method to convert the StringBuilder object to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object, and perform the modifications on the [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object. You can use this approach if you have only a few modifications; otherwise, the cost of working with immutable strings may negate the performance benefits of using a StringBuilder object.

The following example is identical in functionality to the previous two examples but differs in implementation. It creates a StringBuilder object, converts it to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object, and then uses a regular expression to perform all remaining modifications on the string. The [Regex.Replace(String, String, MatchEvaluator)](http://msdn.microsoft.com/en-us/library/ht1sxswy%28v=vs.110%29.aspx) method uses a lambda expression to perform the replacement on each match.

C#

using System;

using System.Text;

using System.Text.RegularExpressions;

public class Example

{

public static void Main()

{

// Create a StringBuilder object with 4 successive occurrences

// of each character in the English alphabet.

StringBuilder sb = new StringBuilder();

for (ushort ctr = (ushort)'a'; ctr <= (ushort) 'z'; ctr++)

sb.Append(Convert.ToChar(ctr), 4);

// Convert it to a string.

String sbString = sb.ToString();

// Use a regex to uppercase the first occurrence of the sequence,

// and separate it from the previous sequence by an underscore.

string pattern = @"(\w)(\1+)";

sbString = Regex.Replace(sbString, pattern,

m => (m.Index > 0 ? "\_" : "") +

m.Groups[1].Value.ToUpper() +

m.Groups[2].Value);

// Display the resulting string.

int line = 0;

do {

int nChars = line \* 80 + 79 <= sbString.Length ?

80 : sbString.Length - line \* 80;

Console.WriteLine(sbString.Substring(line \* 80, nChars));

line++;

} while (line \* 80 < sbString.Length);

}

}

// The example displays the following output:

// Aaaa\_Bbbb\_Cccc\_Dddd\_Eeee\_Ffff\_Gggg\_Hhhh\_Iiii\_Jjjj\_Kkkk\_Llll\_Mmmm\_Nnnn\_Oooo\_Pppp\_

// Qqqq\_Rrrr\_Ssss\_Tttt\_Uuuu\_Vvvv\_Wwww\_Xxxx\_Yyyy\_Zzzz

**Converting the StringBuilder object to a string**

You must convert the StringBuilder object to a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) object before you can pass the string represented by the StringBuilder object to a method that has a [String](http://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx) parameter or display it in the user interface. You perform this conversion by calling the [StringBuilder.ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) method. For an illustration, see the previous example, which calls the [ToString](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.tostring%28v=vs.110%29.aspx) method to convert a StringBuilder object to a string so that it can be passed to a regular expression method.

Notes to Callers

In the .NET Framework 4 and the .NET Framework 4.5, when you instantiate the StringBuilder object by calling the [StringBuilder(Int32, Int32)](http://msdn.microsoft.com/en-us/library/hbb08bby%28v=vs.110%29.aspx) constructor, both the length and the capacity of the StringBuilder instance can grow beyond the value of its [MaxCapacity](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.maxcapacity%28v=vs.110%29.aspx) property. This can occur particularly when you call the [Append](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.append%28v=vs.110%29.aspx) and [AppendFormat](http://msdn.microsoft.com/en-us/library/system.text.stringbuilder.appendformat%28v=vs.110%29.aspx) methods to append small strings.

[Examples](javascript:void(0))

The following example shows how to call many of the methods defined by the StringBuilder class.

C#

using System;

using System.Text;

public sealed class App

{

static void Main()

{

// Create a StringBuilder that expects to hold 50 characters.

// Initialize the StringBuilder with "ABC".

StringBuilder sb = new StringBuilder("ABC", 50);

// Append three characters (D, E, and F) to the end of the StringBuilder.

sb.Append(new char[] { 'D', 'E', 'F' });

// Append a format string to the end of the StringBuilder.

sb.AppendFormat("GHI{0}{1}", 'J', 'k');

// Display the number of characters in the StringBuilder and its string.

Console.WriteLine("{0} chars: {1}", sb.Length, sb.ToString());

// Insert a string at the beginning of the StringBuilder.

sb.Insert(0, "Alphabet: ");

// Replace all lowercase k's with uppercase K's.

sb.Replace('k', 'K');

// Display the number of characters in the StringBuilder and its string.

Console.WriteLine("{0} chars: {1}", sb.Length, sb.ToString());

}

}

// This code produces the following output.

//

// 11 chars: ABCDEFGHIJk

// 21 chars: Alphabet: ABCDEFGHIJK