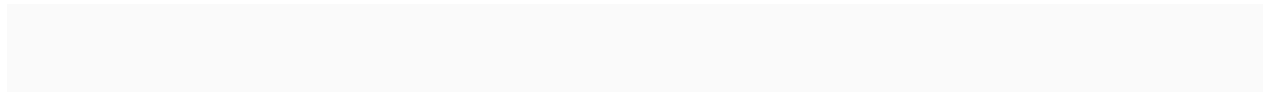


# String Manipulation In C# .Net

Quite often, strings of text need manipulating. Data from a textbox need to be tested and checked for things like blank strings, capital letters, extra spaces, incorrect formats, and a whole lots more besides. Data from text files often needs to be chopped and parsed before doing something with it, and the information your get from and put into databases routinely needs to be examined and worked on. All of this comes under the general heading of String Manipulation.

Later in this section, you're going to be creating your very own Hangman programme. The programme will make use of string manipulation techniques. Let's go through a few of the things that will help you deal with strings of text.



## C# String Variables

You've already worked with string variables a lot in this book. But there's a lot more to them than meets the eye. Strings come with their own Methods and Properties that you can make use of. To see which Methods and Properties are available, start a new C# Windows Application. Add a button and a textbox to your form. For the textbox, change the Text property to "some text" (make sure the text is in lowercase). Now Double click your button to get at the coding window. Then enter the following string declaration

```
string stringVar = textBox1.Text;
```

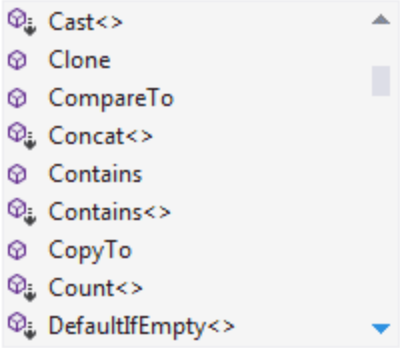
On a new line, type the following:

```
textBox1.Text = stringVar.
```

As soon as you type the full stop at the end, you'll see the IntelliSense list appear:

```
private void button1_Click(object sender, EventArgs e)
{
    string stringVar = textBox1.Text;

    textBox1.Text = stringVar.|
}
```

A screenshot of the Visual Studio code editor showing a C# method. The cursor is at the end of the line 'textBox1.Text = stringVar.', and an IntelliSense dropdown menu is open. The menu lists various methods and properties for the 'string' type, including Cast<>, Clone, CompareTo, Concat<>, Contains, Contains<>, CopyTo, Count<>, and DefaultIfEmpty<>.

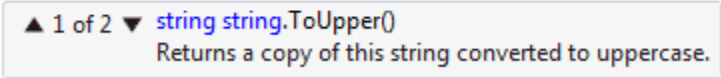
IntelliSense is showing you a list of Methods and Properties that are available for this string object you have called **stringVar**.

Most of the Methods on the list you won't use at all, and a lot of them are just plain baffling! Some are quite obvious in what they do, though.

Select **ToUpper** from the list by double clicking it. Because it's a Method, you need some round brackets. Type a left round bracket and you'll see a yellow box appear, giving you the available options for this Method. For the ToUpper Method, there are only two options available:

```
private void button1_Click(object sender, EventArgs e)
{
    string stringVar = textBox1.Text;

    textBox1.Text = stringVar.ToUpper(|)
}
```

A screenshot of the Visual Studio code editor showing the same C# method. The cursor is at the end of the line 'textBox1.Text = stringVar.ToUpper(|)', and an IntelliSense tooltip is open. The tooltip shows '▲ 1 of 2 ▼ string string.ToUpper()' and a description 'Returns a copy of this string converted to uppercase.'

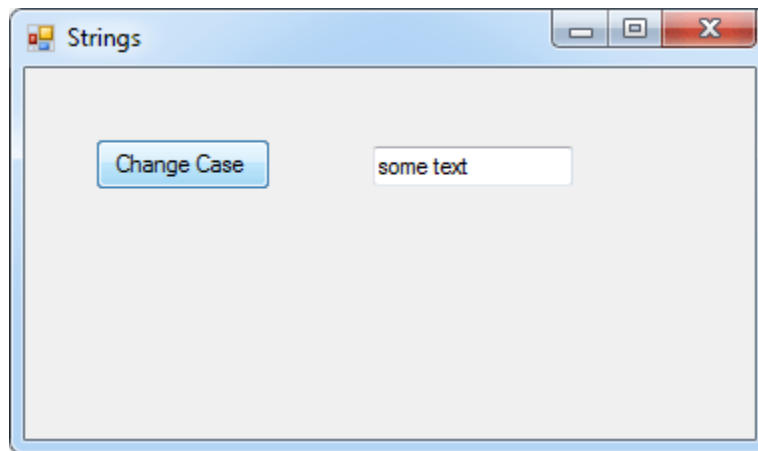
You can press the Down arrow on your keyboard to see the other one. But the first one, 1 of 2, is the one we need. As the tool tip is telling you, this Method converts the string to upper case.

The round brackets of the Method are empty, meaning it doesn't take any arguments. So just type the right round bracket, followed by a semicolon to end the line. Your code should look like this:

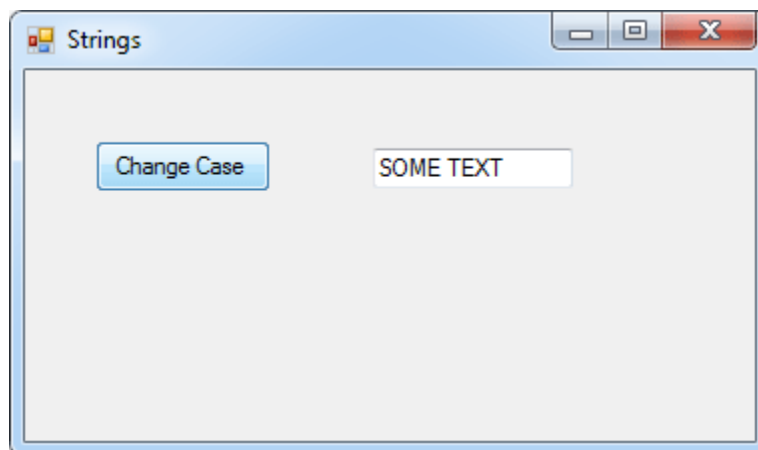
```
private void button1_Click(object sender, EventArgs e)
{
    string stringVar = textBox1.Text;

    textBox1.Text = stringVar.ToUpper();
}
```

Now run your programme. When the form starts it will look like this:



After you click the button, C# runs the ToUpper Method and converts the text in the text box to uppercase:



Another Method that changes case is the ToLower Method. This is the opposite of ToUpper, and is used in the same way.

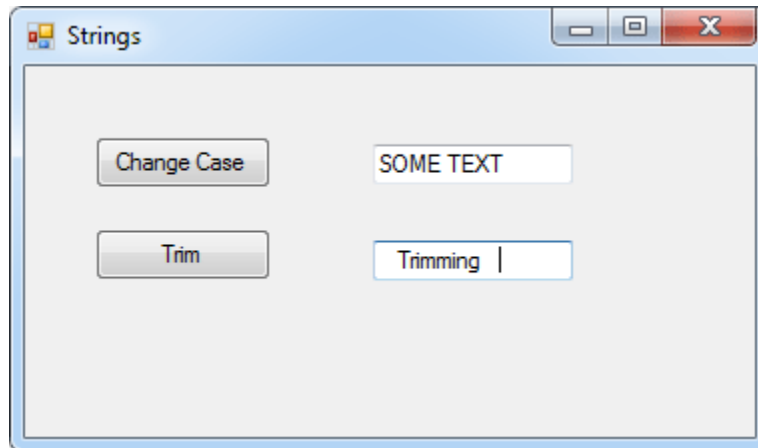
# Trim Unwanted Characters In C# .Net

If you have another look at [the Method list](#), you'll see that there are three that deal with Trimming: Trim, TrimEnd and TrimStart. These methods can be used to Trim unwanted characters from strings.

Add another button to your form. You can change the Text property of your buttons. Enter the text "Uppercase" for the first one. For the new button, enter Trim for the text property. Add another text box below the first one and set the Text property as follows:

**" Trimming "**

Leave out the double quotes but tap the spacebar on your keyboard three times before you type the text. At the end of the text, tap the spacebar three more times. Your Form should then look like this:



The line in the second text box is where the cursor is.

Now double click your second button to get at the code. We can count the number of characters a string has with the Length property. Enter the following code for your button:

```
private void button2_Click(object sender, EventArgs e)
{
    string stringTrim = textBox2.Text;

    int stringLength = stringTrim.Length;

    MessageBox.Show( stringLength.ToString() );
}
```

The first line just gets the text from the text box and puts it into a variable called **stringTrim**. Have a look at the second line, though:

```
int stringLength = stringTrim.Length;
```

We've set up a new integer variable called **stringLength**. To get the length of a string, type a dot after your string variable name. From the IntelliSense list, select the **Length** property. Note that you don't need any round brackets, because it's a property not a method. The Length of a string, by the way, refers to how many characters is in the string.

The third line uses a MessageBox to display the result:

```
MessageBox.Show( stringLength.ToString() );
```

You've seen the ToString method before. This can be used to convert numbers to a string a text. So "10" instead of 10. (The double quotes mean it's text. Without the quotes, it's a number. The variable called stringLength will hold a number.)

Run your programme and click the Trim button on your form. The message box should display an answer of 14. The word "Trimming", however, only has 8 characters in it. The other 6 are the three spaces we put at the beginning and end of the word.

To get rid of space at the beginning and end of text, you can use the Trim method. Add the following line of code to your button:

```
private void button2_Click(object sender, EventArgs e)
{
    string stringTrim = textBox2.Text;

    stringTrim = stringTrim.Trim();

    int stringLength = stringTrim.Length;

    MessageBox.Show( stringLength.ToString() );
}
```

The code to add is highlighted, in the image above. It's this:

```
stringTrim = stringTrim.Trim();
```

So after the dot of the **stringTrim** variable, you select the **Trim** method from the IntelliSense list, followed by a pair of empty round brackets. Run your programme and click the button again. You should find that the length is now 8. So Trim has trimmed the blank spaces from the beginning and the end of our word.

If you only wanted to trim the blank spaces at the end of the word, or just the blank spaces at the beginning of the word, you can use TrimEnd and TrimStart:

```
stringTrim = stringTrim.TrimStart( null );
```

TrimStart and TrimEnd are supposed to take a character array as a parameter. If you type the keyword **null** instead, it will trim the white space (blank characters).

Just as a reference for you, here's some code that strips unwanted hyphens off the end of a string:

```

private void button3_Click(object sender, EventArgs e)
{
    string stringTrim = "Some Text---";

    MessageBox.Show(stringTrim);

    char[] trimChars = {'-'};

    stringTrim = stringTrim.TrimEnd(trimChars);

    MessageBox.Show(stringTrim);
}

```

The **trimChar** line is a character array ( **char[ ]** ) with the hyphen in between curly brackets. This is then handed to the TrimEnd method as a parameter.

## The Contains Method In C# .Net

The contains method can be used if you want to check if a string contains certain characters. It's fairly simple to use. Here's an example:

```

private void button4_Click(object sender, EventArgs e)
{
    string stringVar = "Some Text---";

    if (stringVar.Contains("-"))
    {
        MessageBox.Show("TRUE");

        //CALL HYPHEN REMOVE METHOD HERE
    }
}

```

After the contains method, you type a pair of round brackets. In between the round brackets, you type the text you're checking for. In our code, we're using an if statement. If it's true that the string contains a "-" character, then some code can be executed.

# The IndexOf Method In C# .Net

The **IndexOf** method can be used to check if one character is inside of another. For example, suppose you want to check an email address to see if it contains the @ character. If it doesn't you can tell the user that it's an invalid email address.

Add a new button and a new text box to your form. For the Text property of the text box, enter an email address, complete with @ sign. Double click your button to get at the code. Enter the following:

```
private void button5_Click(object sender, EventArgs e)
{
    string stringEmail = textBox3.Text;

    int result = stringEmail.IndexOf("@");

    if (result == -1)
    {
        MessageBox.Show( "Invalid Email Address");
    }
    else
    {
        MessageBox.Show("@ found at position " + result.ToString());
    }
}
```

The first thing to examine is how **IndexOf** works. Here's the line of code:

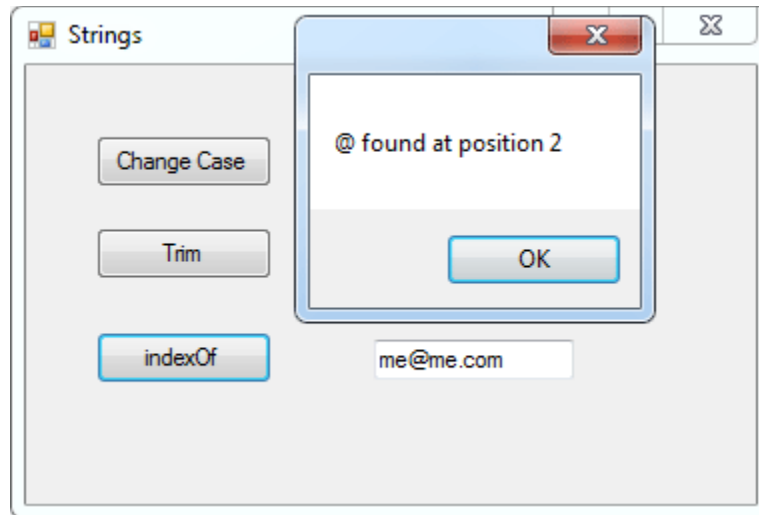
```
int result = stringEmail.IndexOf( "@" );
```

The **IndexOf** method returns an integer. This number is the character's position in the word you're trying to check. In the code above, we want to check the word that's inside of the variable we've called **stringEmail**. We want to see if it contains the "@" character. This goes between the round brackets of **IndexOf**. If C# finds the character, it will tell you where it was (at position number 3 in the word , for example). This number is then stored inside of the **int** variable we've called **result**. If the character you're looking for can't be found, **IndexOf** will return a value of -1 (minus 1). The IF statement in our code checks the value of the **result** variable, to see what's inside of it.

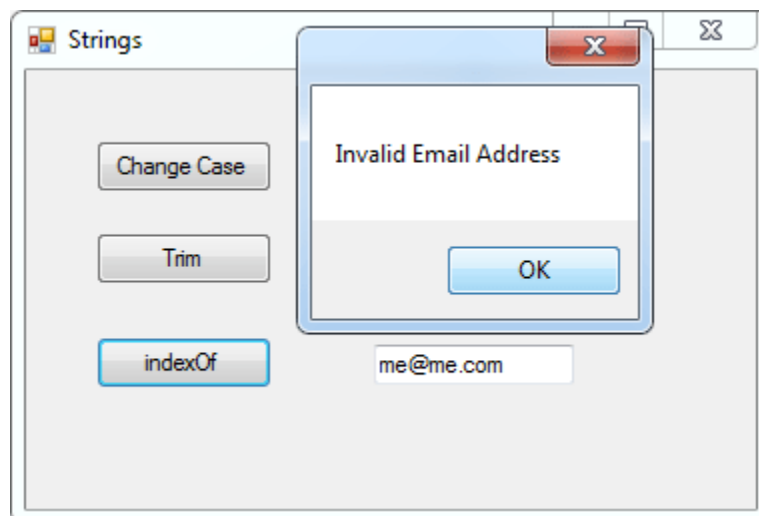


If it's -1 display an "Invalid Email Address message"; If it's not -1, a different message is displayed.

Run your programme and click the button. Here's the form with an @ character in the text box:



And here's what happens when we delete the @ character from the text box:



Note that the first message box displays "@ found at position 2". If you look at the email address in our text box, however, it's **me@me.com**.

So you might be thinking that the @ character is at position 3, not 2. If C# were to start counting at 1, you'd be right. But it doesn't. When you use the IndexOf method, the count starts at zero.

You can also specify a start position, and a character count for a search. This is useful if you want to do things like checking a longer string and counting how many occurrences there are of a particular character or characters. Or if you want a simple check to see if, say, a website entered in a text box on your form begins with http://www. Here's some code that does just that:

```
private void button6_Click(object sender, EventArgs e)
{
    string webAddress = "ttp://www.homeandlearn.co.uk";
    string checkWebAddress = "http://www";

    int start = 0;
    int numOfChars = 10;

    int result = webAddress.IndexOf(checkWebAddress, start, numOfChars);

    if (result != -1)
    {
        MessageBox.Show("Address OK");
    }
    else
    {
        MessageBox.Show("Address Bad");
    }
}
```

Have a look at this part of the highlighted line:

`webAddress.IndexOf( checkWebAddress, start, numOfChars )`

This time, we have three parameters inside of the round brackets of IndexOf. The first one is the string we want to check (**checkWebAddress**). Then we have **start**, and **numOfChars**. The start variable is where in your full string (webAddress) you want to start checking. The third parameter, numOfChars, is the number of characters you want to check from that starting position. In our code, the start is 0 and the number of characters is 10.

And finally, for IndexOf, here's some code that checks a long string of text and counts how many times the word true appears:

```
private void button3_Click(object sender, EventArgs e)
{
    string someText = "Is it true that true evaluates as true?";

    int strLen = someText.Length;
    int result = 0;
    int counter = 0;
    int foundLen = 0;

    for (int i = 0; i < strLen; i++)
    {
        result = someText.IndexOf("true", foundLen, strLen - foundLen);

        if (result != -1)
        {
            foundLen = result + 1;
            counter++;
        }
    }

    MessageBox.Show("true found " + counter.ToString() + " times");
}
```

The code is a bit complex, so don't worry if you don't understand it all. But it's just using IndexOf with three parameters: the word to search for, a starting position, and how many characters you want to check. The start position changes when the word is found; and the number of characters to count shrinks as you move through the word.

## The Insert Method In C# .Net

The Insert method is, not surprisingly, used to insert characters into a string of text. You use it like this:

```
string someText = "Some Text";
```

```
someText = someText.Insert( 5, "More " );
```

In between the round brackets of Insert, you need two things: A position in your text, and the text you want to insert. A comma

separates the two. In our code above, the position that we want to insert the new text is where the T of "Text" currently is. This is the fifth character in the string (the count starts at zero). The text that we want to Insert is the word "More".

### **Exercise**

Test out the code above, for the Insert( ) method. Have one message box to display the old text, and a second message box to display the new text.

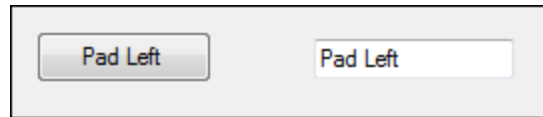
## **Padleft And Padright**

The PadLeft and PadRight methods in C# can also be used to insert characters. But these are used to add characters to the beginning and end of your text. As an example, add a new button to your form, and a new text box. For the text box, set the Text property to "Pad Left". Double click your button and enter the following code:

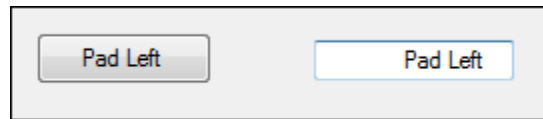
```
string paddingLeft = textBox5.Text;  
paddingLeft = paddingLeft.PadLeft( 20 );  
textBox5.Text = paddingLeft;
```

The PadLeft and PadRight methods can take 1 or two parameters. We're just using 1. This will insert blank space characters to the start of the string. The total number of characters will be 20. So, if you have four characters in the text box, PadLeft(20) will add 16 blank spaces, making a total of 20 characters in the text box after the button is clicked.

Run your programme and test it out. Type the text "Pad Left" in the text box. Your text box should look like this before the button is clicked:



And it will look like this after you click the button:



If you don't want to pad with blank spaces, you can use the second parameter. This is the character you want to pad with:

```
paddingLeft = paddingLeft.PadLeft(20 , '*');
```

In the code above, we're telling C# to pad with the asterisk character, instead of the default blank spaces. (Note the use of the single quotes surrounding the \* character. C# doesn't seem to like you using double quotes, if the type is char.)

If you change your code to the one above, then click the button on your form, the result will be this:



If you want to add characters to the end of the string, use PadRight instead of PadLeft.

## Remove And Replace In C# .Net

The Remove Method

As its name suggests, this method is used to Remove characters from a string of text. Here's a simple example of its use:

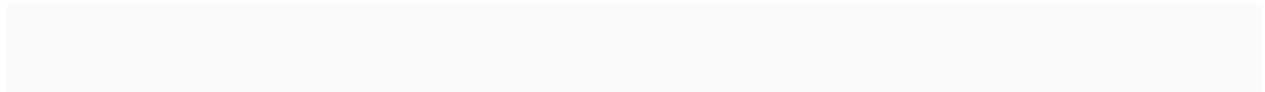
```
string oldString = "some text text text";
```

```
MessageBox.Show(oldString);
```

```
string newString = oldString.Remove(10, 9);
```

```
MessageBox.Show(newString);
```

Remove takes two parameters. The first one is what position in your string you want to start at. (The count starts at zero.) The second parameter is how many characters you want to delete, starting from the position you specified. Add another button to your form and try out the code above.



The Replace Method

The Replace method, you won't be surprised to hear, can be used to replace characters in your string. Here's an example of its use:

```
string spellingError = "mistak";
```

```
spellingError = spellingError.Replace(spellingError,  
"mistake");
```

The Replace method takes two parameters, the old word and the new word. In the code above, we're replacing "mistak" with "mistake".

## Substring In C# .Net

The Substring method is used to grab characters from a string of text. For example, suppose you were testing an email address. You want to test the last four characters to see they are .com. You can use Substring to return just those four characters, and see what's in them. (You can also use IndexOf to achieve the same result.)

Substring has this syntax:

```
the_word.Substring( start_position )
```

So the word you want to grab characters from goes first, followed by the Substring method. In between the round brackets, you have to tell C# where in the word to start grabbing characters from.

But Substring can also take a second parameter:

**the\_word.Substring( start\_position,  
num\_of\_chars\_to\_grab )**

The second parameter is how many characters you want to grab. If you leave this out, C# will grab all the characters to the end of your word. Here's some code to try, with a new button:

```
private void button11_Click(object sender, EventArgs e)
{
    string emailAddress = "me@me.com";
    string result = "";

    result = emailAddress.Substring(5, 4);

    if (result == ".com")
    {
        MessageBox.Show("Email Address OK");
    }
    else
    {
        MessageBox.Show("Bad Email Address");
    }
}
```

We're using Substring with two parameters (5, 4). But since we're grabbing to the end of the word, we could have left out the , 4 at the end.

To test it out, change the me@me.com to me@me.con. Run your programme and you should see "Bad Email Address". Change it back and the email address will be OK.

## Exercise M

Use Substring to check that an email address ends in .co.uk. For the email address to check, use enquiry@me.co.uk.

# Split And Join In C# .Net

## The Split Method

The Split method is used to split a string of text and put the words into an array. For example, you could grab a line of text from a text file. Each position in the array would then hold a word from the line of text. An example may clear things up.

Add another button to your form. Double click the button to get at the code, and add the following:

```
private void button12_Click(object sender, EventArgs e)
{
    string lineOfText = "item1, item2, item3";

    string[] wordArray = lineOfText.Split(',');

    MessageBox.Show( wordArray[0] );
    MessageBox.Show( wordArray[1] );
    MessageBox.Show( wordArray[2] );
}
```

Run the programme and click your button. You should see each word from the line of text display.

In the first line of the code, we're setting up a string with three items in it. Each item is separated by a comma. (Comma separated files from software like Excel are quite common, and so too is parsing each line of text.)

For the second line, we have this:

**string[] wordArray = lineOfText.Split( ',' );**

The first part sets up a string array that we've called wordArray. After the equals sign, we have this:



```
lineOfText.Split( ',' );
```

The variable called `lineOfText` is obviously the line of text we want to examine. For the round brackets of `Split`, we've typed a comma surrounded by single quotes. That's because C# needs to know what character in your line of text you are using to separate the words. This is known as the delimiter. If our line of text were this instead:

```
string lineOfText = "item1 item2 item3";
```

we'd use a blank space as a delimiter. Like this:

```
string[] wordArray = lineOfText.Split( ' ' );
```

This time, we've typed a blank space between the single quotes.

But C# will split the line, and put each part into the array we've set up. (It won't include the delimiter.) For our line of text we only have three words. So the Message box in our code displays what is at position 0, position 1, and position 2 in our array.

If you don't know how many position there are in the array (if you have lines of text that vary in size, for example), then you can loop through each position:

```
foreach (string s in wordArray)  
{  
  
    MessageBox.Show( s );  
  
}
```

The **Split** method can take other parameters, and get a bit complex. So we'll leave it there in this beginners course!

## The Join Method

You can join the pieces of your arrays back together again. Join, however, is not a method available to ordinary strings. Instead, you can access it through the String class. Like this:

```
private void button13_Click(object sender, EventArgs e)
{
    string lineOfText = "item1, item2, item3";
    string[] wordArray = lineOfText.Split(',');

    string joinedText;

    joinedText = String.Join("-", wordArray);

    MessageBox.Show(joinedText);
}
```

In the code above, we've used Split to split the line of text and put the words into an array. We've then used Join to create a single line of text again. This time, though, the words are separated with hyphens and not commas.

To use Join, first type the word **String** (with a capital letter). After a dot, you should then see the Join method appear on the IntelliSense list. In between the round brackets of Join, you first need the character you want to use as a delimiter. Note that this is surrounded by double quotes. If you use single quotes, C# will think it is the char variable type. But you need to use the string variable type, so you'll get an error. After a comma, you type the name of the array you want to Join together.