Chapter 7 – Other Data Types in Visual Basic.NET

# Objectives

* Enumerated Values
* Structure Types and the With Statement
* Strings and String Classes
* Char Class
* Dates
* Regular Expressions
* Random Numbers

This chapter is going to examine how to start working with more types of data in Visual Basic. Structures are a way that we can define our own new type of data. Enumerated types (or enums) allow us to create new sets of data. We'll conclude our discussion in this chapter with an examination of manipulating dates in VB.NET, searching strings for regular expressions and creating random numbers.

# Enumerated Values

Like Java, VB can create enumerations to make your code more maintainable and readable. Think of these as being like sets, but each value that makes up the set must map to some form of an integer-type number. Make sure that you declare your enums outside of any subprograms or you’ll encounter an error. Here’s an example where we specify the values (both positive and negative values can be used):

Enum Directions

Up = -1

Down = 1

Left = -2

Right = 2

End Enum

We could also let the system specify the values (these would default to integers starting at 0):

'Chapter 7 - Program 1

Module Module1

Enum Weather

Sunny

Cloudy

Partly\_Cloudy

Mostly\_Cloudy

Rainy

Snowy

Windy

End Enum

Sub Main()

Dim CurrentWeather As Weather

'Get weather from sensor...

CurrentWeather = Weather.Partly\_Cloudy

'Process based on the weather

Select Case CurrentWeather

Case Weather.Sunny

'Do whatever we need to when it's sunny -- open windows,

' pull shades down

Case Weather.Rainy

'Do whatever we need to when it's rainy -- close windows,

' open shades

End Select

'Print all names that make up the enumeration -- watch the syntax!

'Remember Enum is a keyword, so in order to see it as a type

'instead of a declaration, we wrap it in [ ] notation...

Dim names = [Enum].GetNames(GetType(Weather))

Debug.Write("Names : ")

For Each name In names

Debug.Write(name & " ")

Next

Debug.WriteLine("")

'Print all values that make up the enumeration -- watch the syntax!

Dim values = [Enum].GetValues(GetType(Weather))

Debug.Write("Values : ")

For Each value In values

Debug.Write(value & " ")

Next

Debug.WriteLine("")

Debug.WriteLine("The Value of Weather.Mostly\_Cloudy is " &

Weather.Mostly\_Cloudy)

End Sub

End Module

Output:

Names : Sunny Cloudy Partly\_Cloudy Mostly\_Cloudy Rainy Snowy Windy

Values : 0 1 2 3 4 5 6

The Value of Weather.Mostly\_Cloudy is 3

Now it might seem like a problem if our enums can only have integer-type numeric backings. Let’s say we’d like to be able to print error messages based on the error type that occurs. An enum of strings would be a much easier way to go, but VB doesn’t allow it. This actually isn’t really a problem if we’re smart and with the use of an array…

'Chapter 7 - Program 2

Module Module1

'Possible Errors that can occur in system

Enum OSErrors

WindowsNotReady

OSHasGainedFullAI

OSIntegratedIntoUser

End Enum

'Error messages we would like to return for each error that can occur

Public OSErrorMessageArray() As String =

{"The Windows Operating System is currently busy with other users",

"I'm afraid I can't do that Dave...",

"There is no escape. You will be assimilated"}

Sub Main()

Dim aSystemError As OSErrors

'Processing along and an error occurs

aSystemError = OSErrors.OSHasGainedFullAI

Debug.WriteLine("System reports error : " &

OSErrorMessageArray(aSystemError))

End Sub

End Module

When run, we get the following response:

System reports error : I'm afraid I can't do that Dave...

# Structure Types and the With Statement

Visual Basic gives programmers with the ability to create user-defined types much like structs in C or C++ or records in Pascal. (For you Javans, it would be like creating a class with no methods and all public attributes.) Remember that a Structure can hold different (heterogeneous) types of data.

About the only other piece of information you would need to know in order to work with structures, is how to access the fields, or individual data pieces, that you place in the structure. You specify them by using the . (dot) notation between the structure variable's name and the field name that you wish to access.

The basic Structure declarative syntax is:

Structure *typeName*

Dim *Field1* As *fieldtype*

Dim *Field2* As *fieldtype*

.

.

.

Dim *Fieldn* As *fieldtype*

End Structure

Structures, like enums, cannot be declared within a subprogram; they must be more global in scope! Here is some example code that shows how to declare, create structure variables and work with a structure.

'Here is the Structure declaration, which serves as a cookie cutter:

Structure CheckInfo

Dim CheckNumber As Integer

Dim CheckDate As Date

Dim CheckAmount As Single

Dim CheckPaidTo As String

End Structure

Sub Main()

'Now create a variable of the structure type

Dim Check1 As CheckInfo

'Place some data into the structure's fields – notice the . notation

Check1.CheckNumber = 123

Check1.CheckDate = #1/1/2001#

Check1.CheckAmount = 456.78

Check1.CheckPaidTo = "SVSU"

'We could avoid some typing by using VB's With statement. This statement

'essentially “opens” the structure up so that the fields contained within

'it can be accessed without resorting to typing the variable name.

'Open the Check structure up

With Check1

'Now that the structure is opened, just reference each field that

'we need to work with using the . and field name

.CheckNumber = 123

.CheckDate = #1/1/2001#

.CheckAmount = 456.78

.CheckPaidTo = "SVSU"

End With

End Sub

The With statement is a handy way to save yourself time and effort when typing in code. You can also use the With keyword on objects too! What I have showed so far is a very typical use of structures. It is important to note that structures can have methods and events in addition to attributes, implementing interfaces, constructors and making them behave much more like a class. Don’t do this! Why? (A) it’s not what structures were made for and (B) there are serious limitations/differences on structures versus true classes…

* classes are reference types and structures are value types
* You can use Structure only at the namespace or module level. This means the declaration context for a structure must be a source file, namespace, class, structure, module, or interface, and cannot be a procedure or block

The bottom line is that if you plan on trying to do anything beyond using the structure for anything other than a basic data container, you should be using classes! If you feel the need to write a method, use a class…

Here’s a more complex example showing a nested structure and an array contained with the structure. We then create an array to hold 3 checks in it. Can you visualize all of this?

'Chapter 7 - Program 3

Module Module1

Structure PayeeInfo

Dim PayeeName As String

Dim PayeeAddress As String

Dim PayeePhone As String

End Structure

Structure CheckInfo

Dim CheckNumber As Integer

Dim CheckDate As Date

Dim CheckAmount As Single

'Notice nesting

Dim CheckPaidTo As PayeeInfo

'Notice size cannot be specified on array

Dim AccountsToDrawFrom As Integer()

End Structure

Sub Main()

'Now create an array variable of the structure type

Dim Checks(2) As CheckInfo

'Place some data into the structure's fields – notice the () & . notation

Checks(0).CheckPaidTo.PayeeName = "SVSU"

'Save time with nested Withs...

With Checks(0)

.CheckAmount = 100.0

.CheckNumber = 1234

With Checks(0).CheckPaidTo

.PayeeName = "SVSU"

.PayeeAddress = "7400 Bay Road"

End With

.CheckDate = #1/1/2020#

'Set size of array inside of the structure at runtime to what we need

ReDim .AccountsToDrawFrom(3)

.AccountsToDrawFrom(0) = 141573

.AccountsToDrawFrom(1) = 141574

.AccountsToDrawFrom(2) = 224094

.AccountsToDrawFrom(3) = 224095

End With

'Let's print out the information

With Checks(0)

Debug.WriteLine("Check Number: " & .CheckNumber & " - Paid To: " &

.CheckPaidTo.PayeeName & " - Amount: " &

FormatCurrency(.CheckAmount))

Debug.WriteLine("Date: " & .CheckDate)

Debug.WriteLine("These are the accounts certified to draw funds" &

" from to cover check amount:")

For intI As Integer = 0 To .AccountsToDrawFrom.Count - 1

Debug.WriteLine(" Account Number: " & .AccountsToDrawFrom(intI))

Next

End With

End Sub

End Module

Here’s the output:

Check Number: 1234 - Paid To: SVSU - Amount: $100.00

Date: 1/1/2020

These are the accounts certified to draw funds from to cover check amount:

Account Number: 141573

Account Number: 141574

Account Number: 224094

Account Number: 224095

# Strings and String Classes

We have been using strings up to this point with little consideration for what they have been doing for us. Let’s start back with some basic concepts. First of all, there are two ways to declare strings:

Dim strName As String 'declares a string containing nothing

Dim strName2 As String = "Joe" 'declares string and initializes it

Older VB programs also used to be somewhat sloppy in the way that they performed concatenation. We have seen that the & is the proper choice – however the + symbol also will perform string concatenation as well (just like in the JavaScript & Java languages). The + symbol will get you into trouble if both operands are not strings, such as in:

strMsg = 22 + " Hello" 'An & would have forced the 22 to be

'converted to a string – will VB

'be smart enough to realize we’re not

'trying to add something to the integer

'22? The answer is NO – cast exception!

Strings can be compared to each other by simply using If statements and the relational operators (<, <=, >, >=, <>, =). VB provides you with a ton of string related functions to make your programming life easier when it comes to dealing with strings.

The first function to look at is the string comparison function, StrComp, in which two strings are given:

Syntax for StrComp function:

StrComp(*s1*,*s2*)

StrComp Returns:

0 if *s1* = *s2*; -1 if *s1* < *s2*; 1 if *s1* > *s2*

This function can become even more powerful if we set the Option Compare type. There are three choices for us to select from as shown in the next table:

|  |  |
| --- | --- |
| *Compare Type* | *Function* |
| Binary | (default) Case sensitive comparison; Hi <> hi |
| Text | Case insensitive comparison; Hi = hi |
| Database | Compare based on information from a database’s settings |

We also have previously mentioned the Like operator that will allow a programmer to compare patterns of characters as well as strings (this is similar to Perl pattern matching). The various patterns that can be matched in the Like operator are provided in the following table

Like function characters:

|  |  |
| --- | --- |
| *Character* | *Meaning* |
| \* | Any number of characters can follow |
| ? | Single character can be any type of character |
| # | Single character must be a digit |
| [] | Series of characters that can be used for pattern matching |
| [!] | Series of characters that are not to be used |

Here are some examples of Like comparisons:

"HBLT55DD" Like "HBLT55DD" 🡪 True

"HBLT55DD" Like "HBLT\*" 🡪 True

"HBLT55DD" Like "?#LT55DD" 🡪 False (second character is not a digit)

"HBLT55DD" Like "?BLT5#DD" 🡪 True

"HBLT55DD" Like "H[A-F]LT55DD" 🡪 True

"HBLT55DD" Like "H[A-F]LT[!4-7]5DD" 🡪 False (we have a 5 where we don’t want one)

We previously covered many of the string functions in Chapter 4, however it is appropriate to re-list those functions in this section. The following table lists the older VB string functions. Please note that there are some equivalent methods in the String class, while others are only available here:

|  |  |
| --- | --- |
| *Function* | *Purpose* |
| Asc(x) | returns the ASCII value of character *x* |
| Chr(x) | prints the character associated with ASCII value *x* |
| Hex(x) | returns the hexadecimal equivalent of numeric x |
| InStr(str1,str2) | tests if *str2* is a substring of *str1*: returns 0 if false, positive if true which is character position where *str2* appears in *str1* |
| InStrRev(str1,str2,[startat])) | just like InStr, but looks for last occurrence of a substring starting at the end of *str1* |
| Join | combines strings together with a specified delimiter |
| LCase(str) | converts *str* to lowercase |
| Left(str,x) | returns the *x* leftmost characters of *str*  (in the Microsoft.VisualBasic. namespace) |
| Len(str) | returns the length of *str* |
| LSet(str,len) | left justifies the *str* to a length of *len* characters and pads spaces to the right |
| LTrim(str) | removes all spaces from the beginning of *str* |
| Mid(str,m,n) | returns *n* characters from *str* beginning with character *m* |
| Oct(x) | returns the octal value of numeric *x* |
| Replace(str, whattofind, whattoputin) | returns a string that contains a version of *str* where all the *whattofind* have been replaced with *whattoputin* |
| Right(str,n) | returns the *n* rightmost characters from *str*  (in the Microsoft.VisualBasic. namespace) |
| RSet(str, len) | right justifies the *str* to a length of *len* characters and pads spaces to the left |
| Rtrim(str) | removes all spaces from the right side of *str* |
| Space(x) | returns a string consisting of *x* spaces |
| Split | returns an array of strings split by a common delimiter |
| Str(x) | attempts to convert numeric *x* to a string |
| StrComp(s1,s2) | compares two strings and returns a result based on comparison |
| StrDup(x,char) | returns a string composed of *x* number of character *char* |
| StrReverse(str) | returns the reversed string |
| Trim(str) | removes all spaces at the beginning and end of *str* |
| UCase(str) | converts *str* to all uppercase characters |

VB also includes some powerful string manipulator functions. Since I’m not sure if you’ve worked with these before in CS 146, I will provide examples of each these functions that I care about in this section. Most modern languages implement something very similar.

## Join

This function will take an array of strings as an argument and concatenate them all together into a single string using any delimiter you like. The default delimiter is a space. The really nice feature about this function is that it takes the work out of creating delimited values, which used to be a very tedious task.

Dim strWords(3) As String

Dim strLine As String

'Fill the original array with the individual strings

strWords(0) = "Saginaw"

strWords(1) = "Valley"

strWords(2) = "State"

strWords(3) = "University"

'Call the Join function to combine each of the array elements with the

'default space delimiter in between each value

strLine = Join(strWords)

'strLine would now contain "Saginaw Valley State University"

## Split

This function is the opposite of the Join function. In other words, it will take a string and split the string up into an array of substrings each of which was separated by some delimiter in the original string. By default, Split will default to a space delimiter unless the programmer specifies something different.

Dim strWords() As String

Dim strLine As String

Dim intLoop As Integer

'Here's the original source string

strLine = "Saginaw Valley State University"

'Call the Split function to tear the string apart

strWords = Split(strLine, " ") 'The " " is the default and I could have

'left it off of the example but I wanted

'to show you how to specify a delimiter

'Use the LBound and UBound functions to determine array size

'We could also have used 0 To strWords.Count-1

For intLoop = LBound(strWords) To UBound(strWords)

'Print out an item from the array of substrings

Debug.WriteLine(strWords(intLoop))

Next

You may have noticed that the preceding examples referred to string functions and not methods. These really are more leftovers from VB 6 and VBA that are here for compatibility purposes.

## Strings Class

There is a new Strings class in VB.NET that provides most of the same functionality as the old string functions but implemented instead as methods. You should get used to using these new methods since the Strings (note: not String) class is more aligned with the .NET architecture. Let's take a look at the list of the methods and properties and what each one does. You’ll notice many of our old functions are available in here.

Strings class Methods:

|  |  |
| --- | --- |
| *Name* | *Purpose* |
| Asc | Returns an Integer value representing the ASCII character code corresponding to a character. |
| AscW | Returns an Integer value representing the UNICODE character code corresponding to a character. |
| [Chr](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.chr.aspx) | Returns the character associated with the specified ASCII character code. |
| [ChrW](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.chrw.aspx) | Returns the character associated with the specified UNICODE character code. |
| [Equals(Object)](http://msdn.microsoft.com/en-us/library/bsc2ak47.aspx) | Determines whether the specified object is equal to the current object. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| [Filter(String(), String, Boolean, CompareMethod)](http://msdn.microsoft.com/en-us/library/h6ctk5wk.aspx) | Returns a zero-based array containing a subset of a String array based on specified filter criteria. |
| [Format](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.format.aspx) | Returns a string formatted according to instructions contained in a format String expression. |
| [FormatCurrency](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.formatcurrency.aspx) | Returns an expression formatted as a currency value using the currency symbol defined in the system control panel. |
| [FormatDateTime](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.formatdatetime.aspx) | Returns a string expression representing a date/time value. |
| [FormatNumber](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.formatnumber.aspx) | Returns an expression formatted as a number. |
| [FormatPercent](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.formatpercent.aspx) | Returns an expression formatted as a percentage (that is, multiplied by 100) with a trailing % character. |
| [GetChar](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.getchar.aspx) | Returns a Char value representing the character from the specified index in the supplied string. |
| [InStr(String, String, CompareMethod)](http://msdn.microsoft.com/en-us/library/47d6yh63.aspx) | Returns an integer specifying the start position of the first occurrence of one string within another. |
| [InStrRev](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.instrrev.aspx) | Returns the position of the first occurrence of one string within another, starting from the right side of the string. |
| Join | Returns a string created by joining a number of substrings contained in an array. |
| LCase | Returns a string or character converted to lowercase. |
| [Left](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.left.aspx) | Returns a string containing a specified number of characters from the left side of a string. |
| Len | Returns an integer containing either the number of characters in a string |
| [LSet](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.lset.aspx) | Returns a left-aligned string containing the specified string adjusted to the specified length. |
| [LTrim](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.ltrim.aspx) | Returns a string containing a copy of a specified string with no leading spaces (LTrim), no trailing spaces (RTrim), or no leading or trailing spaces (Trim). |
| [Mid](http://msdn.microsoft.com/en-us/library/8kyd8789.aspx) | Returns a string containing a specified number of characters from a string. |
| [Replace](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.replace.aspx) | Returns a string in which a specified substring has been replaced with another substring a specified number of times. |
| [Right](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.right.aspx) | Returns a string containing a specified number of characters from the right side of a string. |
| [RSet](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.rset.aspx) | Returns a right-aligned string containing the specified string adjusted to the specified length. |
| [RTrim](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.rtrim.aspx) | Returns a string containing a copy of a specified string with no leading spaces (LTrim), no trailing spaces (RTrim), or no leading or trailing spaces (Trim). |
| [Space](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.space.aspx) | Returns a string consisting of the specified number of spaces. |
| [Split](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.split.aspx) | Returns a zero-based, one-dimensional array containing a specified number of substrings. |
| [StrComp](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.strcomp.aspx) | Returns -1, 0, or 1, based on the result of a string comparison. |
| [StrConv](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.strconv.aspx) | Returns a string converted as specified. |
| StrDup | Returns a string consisting of the specified character repeated the specified number of times. |
| [StrReverse](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.strreverse.aspx) | Returns a string in which the character order of a specified string is reversed. |
| [ToString](http://msdn.microsoft.com/en-us/library/system.object.tostring.aspx) | Returns a string that represents the current object. (Inherited from [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx).) |
| [Trim](http://msdn.microsoft.com/en-us/library/microsoft.visualbasic.strings.trim.aspx) | Returns a string containing a copy of a specified string with no leading spaces (LTrim), no trailing spaces (RTrim), or no leading or trailing spaces (Trim). |
| [UCase](http://msdn.microsoft.com/en-us/library/e9zfk32z.aspx) | Returns a string or character containing the specified string converted to uppercase. |

In case you wonder about the likelihood of the old string functions disappearing, here’s an example: Filter. Filter used to exist as a function in earlier versions of .NET; no more – it at least was moved into the Strings class as a method.

## Filter

This function creates an array of strings that contain elements from a source array of strings in which the elements conform to a specified filter pattern.

Dim strClass(4) As String

Dim strResults() As String

'Here are the original strings

strClass(0) = "Bill"

strClass(1) = "Sue"

strClass(2) = "Bill"

strClass(3) = "Kim"

strClass(4) = "Dave"

'Look for any instances of Bill in the strClass array

strResults = Strings.Filter(strClass, "Bil") 'StrResults would be

'"Bill","Bill"

'This line is foreshadowing for the next chapter on arrays

'It deletes an array that exists...about as cool of a literary

'option that I have available to me

Erase strResults

'Notice the False to inverse the filtering on the next line, so

'we are saying we want any elements that do not contain "Bill"

strResults = Strings.Filter(strClass, "Bill", False) 'StrResults would be

'"Sue","Kim","Dave"

As I previously mentioned, you really should get in the habit of using these Strings methods in place of the aforementioned string functions. I think eventually Microsoft will do away with the string functions from the core language and put them all into the Strings namespace. By the way, just so you know, you can’t create variables of type Strings!

One final note…not everything is zero based in the Strings class! Again, a big shout out to Microsoft for the inconsistency. Check this out:

Debug.WriteLine(Strings.Mid("Scott", 2, 2))

With the output being:

co

This means S is 1 and c is 2, not S as 0 and c as 1 like we might think! This is a throwback to the pre-.NET days but is a real pain to remember now!!!! That’s the reason you ought to be playing around and learning the peculiarities of these methods/functions!

## String Class

All right, having said our piece on Strings, remember that this section started out by declaring variables of type **String**. String is a class belonging to the System namespace. Since it is a class, it shouldn’t surprise you that there are built-in methods there as well – in fact some of them duplicate what we have seen as intrinsic functions and as Strings method calls. Remember this list is just a quick summary – there are usually several overloaded methods for each one shown here.

String class Methods:

|  |  |
| --- | --- |
| *Name* | *Description* |
| [Clone](http://msdn.microsoft.com/en-us/library/system.string.clone.aspx) | Returns a reference to this instance of [String](http://msdn.microsoft.com/en-us/library/system.string.aspx). |
| [Compare(String, String, Boolean)](http://msdn.microsoft.com/en-us/library/zkcaxw5y.aspx) | Compares two specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) objects and returns an integer that indicates their relative position in the sort order. There are several versions of this method with different parameters. |
| [CompareOrdinal(String, String)](http://msdn.microsoft.com/en-us/library/af26w0wa.aspx) | Compares two specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) objects by evaluating the numeric values of the corresponding [Char](http://msdn.microsoft.com/en-us/library/system.char.aspx) objects in each string. |
| [CompareTo(String)](http://msdn.microsoft.com/en-us/library/35f0x18w.aspx) | Compares this instance with a specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx). |
| [Concat](http://msdn.microsoft.com/en-us/library/0wkb0y3w.aspx) | Concatenates the elements of a specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) array – again many variants of this method are available. |
| [Contains](http://msdn.microsoft.com/en-us/library/dy85x1sa.aspx) | Returns a value indicating whether the specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object occurs within this string. |
| [Copy](http://msdn.microsoft.com/en-us/library/system.string.copy.aspx) | Creates a new instance of [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) with the same value as a specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx). |
| [CopyTo](http://msdn.microsoft.com/en-us/library/system.string.copyto.aspx) | Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters. |
| [EndsWith(String)](http://msdn.microsoft.com/en-us/library/2333wewz.aspx) | Determines whether the end of this string instance matches the specified string. |
| [Equals(String)](http://msdn.microsoft.com/en-us/library/858x0yyx.aspx) | Determines whether this instance and another specified [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object have the same value – many variants |
| Format | Replaces one or more format items in a specified string with the string representation of a specified object. |
| [GetEnumerator](http://msdn.microsoft.com/en-us/library/system.string.getenumerator.aspx) | Retrieves an object that can iterate through the individual characters in this string. |
| [IndexOf(String)](http://msdn.microsoft.com/en-us/library/k8b1470s.aspx) | Reports the zero-based index of the first occurrence of the specified item in this instance – many variants |
| [IndexOfAny(Char())](http://msdn.microsoft.com/en-us/library/11w09h50.aspx) | Reports the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. |
| [Insert](http://msdn.microsoft.com/en-us/library/system.string.insert.aspx) | Returns a new string in which a specified string is inserted at a specified index position in this instance. |
| [IsNullOrEmpty](http://msdn.microsoft.com/en-us/library/system.string.isnullorempty.aspx) | Indicates whether the specified string is Nothing or an [Empty](http://msdn.microsoft.com/en-us/library/system.string.empty.aspx) string. |
| [IsNullOrWhiteSpace](http://msdn.microsoft.com/en-us/library/system.string.isnullorwhitespace.aspx) | Indicates whether a specified string is Nothing, empty, or consists only of white-space characters. |
| [Join(String, String())](http://msdn.microsoft.com/en-us/library/57a79xd0.aspx) | Concatenates all the elements of a string array, using the specified separator between each element. |
| LastIndexOf | Reports the zero-based index position of the last occurrence of a specified item within this instance – many variants |
| [PadLeft(Int32)](http://msdn.microsoft.com/en-us/library/0zk6ydzx.aspx) | Returns a new string that right-aligns the characters in this instance by padding them with spaces on the left, for a specified total length. |
| [PadRight(Int32)](http://msdn.microsoft.com/en-us/library/34d75d7s.aspx) | Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length. |
| [Remove(Int32)](http://msdn.microsoft.com/en-us/library/9ad138yc.aspx) | Returns a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted. |
| [Replace(String, String)](http://msdn.microsoft.com/en-us/library/fk49wtc1.aspx) | Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string. |
| Split | Returns a string array that contains the substrings in this instance that are delimited by elements of a specified array – many variants. |
| [StartsWith(String)](http://msdn.microsoft.com/en-us/library/baketfxw.aspx) | Determines whether the beginning of this string instance matches the specified string. |
| [Substring(Int32, Int32)](http://msdn.microsoft.com/en-us/library/aka44szs.aspx) | Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length. |
| [ToCharArray](http://msdn.microsoft.com/en-us/library/ezftk57x.aspx) | Copies the characters in this instance to a Unicode character array. |
| [ToLower](http://msdn.microsoft.com/en-us/library/e78f86at.aspx) | Returns a copy of this string converted to lowercase. |
| [ToLowerInvariant](http://msdn.microsoft.com/en-us/library/system.string.tolowerinvariant.aspx) | Returns a copy of this [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object converted to lowercase using the casing rules of the invariant culture. |
| [ToString](http://msdn.microsoft.com/en-us/library/8tc6ws5s.aspx) | Returns this instance of [String](http://msdn.microsoft.com/en-us/library/system.string.aspx); no actual conversion is performed. (Overrides [Object.ToString](http://msdn.microsoft.com/en-us/library/system.object.tostring.aspx).) |
| [ToUpper](http://msdn.microsoft.com/en-us/library/ewdd6aed.aspx) | Returns a copy of this string converted to uppercase. |
| [ToUpperInvariant](http://msdn.microsoft.com/en-us/library/system.string.toupperinvariant.aspx) | Returns a copy of this [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object converted to uppercase using the casing rules of the invariant culture. |
| [Trim](http://msdn.microsoft.com/en-us/library/t97s7bs3.aspx) | Removes all leading and trailing white-space characters from the current [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object. |
| [TrimEnd](http://msdn.microsoft.com/en-us/library/system.string.trimend.aspx) | Removes all trailing occurrences of a set of characters specified in an array from the current [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object. |
| [TrimStart](http://msdn.microsoft.com/en-us/library/system.string.trimstart.aspx) | Removes all leading occurrences of a set of characters specified in an array from the current [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) object. |

Here's a sample program that uses some of the String class methods.

Dim SchoolName As String = "Saginaw Valley State University"

Dim ComputerCompanies As String = \_

"HP;Dell;Gateway;IBM;Microsoft"

Dim SplitArray() As String

Dim aStr As String

Debug.WriteLine("Statistics on SchoolName String")

Debug.WriteLine("String: " & SchoolName)

Debug.WriteLine("Length is: " & SchoolName.Length)

Debug.WriteLine("Char at 0-based position 3 is: " & SchoolName.Chars(3))

Debug.WriteLine("Let's get all of the ComputerCompanies")

SplitArray = ComputerCompanies.Split(";")

For Each aStr In SplitArray

Debug.WriteLine(aStr)

Next

Debug.WriteLine("Some miscellaneous string methods")

Debug.WriteLine("Compare Goodbye & HELLO: " &

String.Compare("Goodbye", "HELLO"))

Debug.WriteLine("Concat Example: " & \_

String.Concat("This 'n", "' That!"))

Debug.WriteLine "0-based Index of Val is: " & \_

SchoolName.IndexOf("Val"))

Debug.WriteLine("Ends with Sag: " & SchoolName.EndsWith("Sag"))

Debug.WriteLine("Starts with Sag: " & \_

SchoolName.StartsWith("Sag"))

Debug.WriteLine("Left Pad With 8 1's: " & \_

SchoolName.PadLeft(39, "1"))

Debug.WriteLine("Upper: " & SchoolName.ToUpper())

Here's the output for the program. Remember when examining the output that objects of type String (the String class, not Strings) start at element 0.

Statistics on SchoolName String

String: Saginaw Valley State University

Length is: 31

Char at 0-based position 3 is: i

Let's get all of the ComputerCompanies

HP

Dell

Gateway

IBM

Microsoft

Some miscellaneous string methods

Compare Goodbye & HELLO: -1

Concat Example: This 'n' That!

0-based Index of Val is: 8

Ends with Sag: False

Starts with Sag: True

Left Pad With 8 1's: 11111111Saginaw Valley State University

Upper: SAGINAW VALLEY STATE UNIVERSITY

## StringBuilder Class

Just so you know, strings from the String class are considered immutable. What this means is that once you create a string, you cannot change it. This is true in many other languages such as Java. Just a minute, you say – you've made strings in Java and you've been able to change them. Well, yes and no. Technically you've been able to change what the string variable *pointed at*. The actual string itself wasn't changed: what happened instead is the original string object was thrown away and a new string object was created with the new value placed in it. Your string pointer was then updated to point at the new string object.

VB.NET includes an object called the StringBuilder and its contents are mutable. This means that you can change and modify the StringBuilder however as you like and you get better system performance since you are not continually creating and destroying strings every time you change one. Most of the necessary manipulation methods from String are available in StringBuilder, which you can look up in the MSDN help.

For this purposes of this class, the String class is fine to use for all of your work, but just remember that in a production environment, you will seriously want to consider the StringBuilder class!

The StringBuilder class has the following general methods:

* Append
* AppendFormat
* AppendLine
* Clear
* CopyTo
* EnsureCapacity
* Equals
* Insert
* Remove
* Replace
* ToString

Here’s a small sample application showing the StringBuilder:

'Chapter 7 - Program 4

'Need to import the System.Text namespace for StringBuilder

Imports System.Text

Module Module1

Sub Main()

' Create a StringBuilder that expects to hold 50 characters.

' Initialize the StringBuilder with "We are ".

Dim sb As New StringBuilder("We are ", 50)

' Append a character array of four characters (S, V, S and U)

' to the end of the StringBuilder.

' Remember that the c after each item indicates a character

sb.Append(New Char() {"S"c, "V"c, "S"c, "U"c})

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

' Placeholder 0 will be a character as will placeholder 1

sb.AppendFormat("20{0}{1}", "2"c, "1"c)

' Display the number of characters in the StringBuilder and its string

' using methods from the StringBuilder Class

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

' Insert a string at the beginning of the StringBuilder.

sb.Insert(0, "Who - ")

' Replace all 2's with 3's.

sb.Replace("2", "3")

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

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

End Sub

End Module

And its output:

15 chars: We are SVSU2021

21 chars: Who - We are SVSU3031

# Char Class

Just as there is a String class, there is a Char class that has some useful methods for manipulating characters. We are not going to spend any time going in depth on these methods and properties, since most are pretty obvious in what they do.

Char class methods and properties:

|  |  |
| --- | --- |
| *Method* | *Purpose* |
| CompareTo | Compares two characters |
| ConvertFrom | Converts from some object to char |
| ConvertTo | Converts a char class to some other object type |
| Equals | Returns a value that indicates whether this instance is equal to the specified Char object |
| IsControl | Returns a Boolean value telling whether a character is a control character |
| IsDigit | Returns a Boolean value telling whether a character is a digit |
| IsLetter | Returns a Boolean value telling whether a character is a letter |
| IsLetterOrDigit | Returns a Boolean value telling whether a character is a letter or digit |
| IsLower | Returns a Boolean value telling whether a character is lowercase |
| IsNumber | Returns a Boolean value telling whether a character is a number |
| IsPunctuation | Returns a Boolean value telling whether a character is punctuation |
| IsSeparator | Returns a Boolean value telling whether a character is a separator character |
| IsSymbol | Returns a Boolean value telling whether a character is a symbol |
| IsUpper | Returns a Boolean value telling whether a character is uppercase |
| IsWhiteSpace | Returns a Boolean value telling whether a character is whitespace |
| ToLower | Returns a lowercase version of the character |
| ToUpper | Returns an uppercase version of the character |

# 

# Date and Time Processing

One of the functions that we often need to perform when developing software is the handling of dates and times. Visual Basic provides a wealth of tools to allow you to process dates/times almost as easily as you can process strings.

You have the ability to add values to dates, subtract two dates to get the difference between them and so forth. You can manipulate days, months, years, hours, minutes, seconds and milliseconds with as much detail as you need. Just remember that when you are working with dates, you can represent date literals by enclosing the date in a set of #, such as #01/01/2001#.

Just as we observed that there were pre-.NET string functions, there are old leftover date and time related functions available for your use.

Intrinsic Date and Time Functions:

|  |  |
| --- | --- |
| *Function* | *Purpose* |
| Now() | Returns the current system date and time in the format xx/xx/xx xx:xx:xx XM |
| Day(*datetimevalue*) | Returns the day portion of a DateTime value |
| Month(*datetimevalue*) | Returns the month portion of a DateTime value |
| Year(*datetimevalue*) | Returns the year portion of a DateTime value |
| IsDate(*value*) | Checks to see if *value* represents a valid date or not |
| Hour(*datetimevalue*) | Returns the hour component of the specified DateTime value |
| Minute(*datetimevalue*) | Returns the minute component of the specified DateTime value |
| Second(*datetimevalue*) | Returns the second component of the specified DateTime value |
| Today | Returns the current system date |
| TimeOfDay | Returns the current system time |

Let's take a look at a sample program that showcases some of these date and time functions.

Dim myDT As Date = Now()

Debug.WriteLine("Hour: " & Hour(myDT))

Debug.WriteLine("Minute: " & Minute(myDT))

Debug.WriteLine("Second: " & Second(myDT))

Debug.WriteLine("Month: " & Month(myDT))

'We have to qualify the next line since there is a Day type in

'System.Windows.Forms namespace and we want to use the Day

'function which is in the Microsoft.VisualBasic namespace class

Debug.WriteLine("Day: " &

Microsoft.VisualBasic.Day(myDT).ToString)

Debug.WriteLine("TimeOfDay: " & TimeOfDay().ToString())

Here's the output:

Hour: 7

Minute: 53

Second: 18

Month: 12

Day: 21

TimeOfDay: 1/1/0001 7:53:18 AM

Notice that only the time value was valid in the TimeOfDay function above. For some unknown reason, Microsoft doesn't bother to return the correct date in that function. If you want the current date, call the Today() function.

As you might have guessed, there is a VB.NET way to handle dates and times through a class called the DateTime class. This class has all of the functionality of the old VB6 date and time functions that aren't directly accessible in .NET anymore.

DateTime class methods:

|  |  |
| --- | --- |
| *Method* | *Purpose* |
| Add | Adds the value contained in a TimeSpan object to the date that the object contains |
| AddDays | Adds the number of days specified to the date the object contains |
| AddHours | Adds the number of hours specified to the time the object contains |
| AddMilliseconds | Adds the number of milliseconds specified to the time the object contains |
| AddMinutes | Adds the number of minutes specified to the time the object contains |
| AddMonths | Adds the number of months specified to the date the object contains |
| AddSeconds | Adds the number of seconds specified to the time the object contains |
| AddTicks | Adds the number of ticks specified to the time the object contains |
| AddYears | Adds the number of years specified to the date the object contains |
| Compare | Compares two DateTime objects |
| CompareTo | Compares the object to another object |
| DaysInMonth | Returns the number of days in the month specified for a given year |
| Equals | Compares an instance of a DateTime object to a specified object to determine if the objects are equal |
| FromBinary | Deserializes a 64 bit value and recreates a serialized DateTime object |
| FromFileTime | Returns a DateTime object containing a file’s timestamp |
| FromFileTimeUtc | Returns a DateTime object containing a file’s timestamp shifted to UTC |
| GetDateTimeFormats | Converts the object's date and time vales to the string representation supported by the standard DateTime format specifiers |
| IsDaylightSavingTime | Returns a Boolean value that specifies if the time zone is currently in daylight savings |
| IsLeapYear | Returns a Boolean value that specifies if a given year is a leap year |
| Parse | Converts a string representation of a date and time value to a DateTime object |
| ParseExact | As above, but a string’s format must exactly match a specified date and time format |
| SpecifyKind | Creates a new DateTime that has the same number of ticks as the specified DateTime but is designated as either local, UTC or neither |
| Subtract | Subtracts the specified time from the object's time |
| ToBinary | Serializes the current DateTime object to a 64-bit value |
| ToFileTime | Converts a DateTime to a suitable timestamp |
| ToFileTimeUTC | Converts a DateTime to a UTC based suitable timestamp |
| ToLocalTime | Converts from UTC to local time |
| ToLongDateString | Converts to a long date string representation |
| ToLongTimeString | Converts to a long time string representation |
| ToShortDateString | Converts to a short date string representation |
| ToShortTimeString | Converts to a short time string representation |
| ToUniversalTime | Converts local time to UTC |

DateTime Class Properties:

|  |  |
| --- | --- |
| *Property* | *Purpose* |
| Date | Returns the object’s date component |
| Day | Returns the day of the month (1 to 31) |
| DayOfWeek | Returns the day of the week (1 Sunday to 7 Saturday) |
| DayOfYear | Returns the day of the year (1 to 365) |
| Hour | Returns the hour (0 – 23) |
| Kind | Returns whether local, UTC or neither |
| Millisecond | Returns the millisecond |
| Minute | Returns the minute (0 – 59) |
| Month | Returns the month (1 – 12) |
| Now | Returns a DateTime object with the current system date and time |
| Second | Returns the second (0 – 59) |
| Ticks | Returns the number of ticks that represent the date and time of this instance |
| TimeOfDay | Returns the object's time component |
| Today | Returns the current date |
| UtcNow | Returns a DateTime object the contains the computer's local time expressed as UTC time |
| Year | Returns the year that corresponds with the object's date |

Here's a sample program that illustrates quite a bit of the DateTime class:

Dim myDT As DateTime = Now()

Dim newDT As DateTime

Debug.WriteLine("Current date and time: " & myDT)

Debug.WriteLine("Long date: " & myDT.ToLongDateString)

Debug.WriteLine("Short date: " & myDT.ToShortDateString)

Debug.WriteLine("Long time: " & myDT.ToLongTimeString)

Debug.WriteLine("Short time: " & myDT.ToShortTimeString)

Debug.WriteLine("Universal time: " & myDT.ToUniversalTime)

Debug.WriteLine(" ")

Debug.WriteLine("Let's add some times and dates:")

newDT = Now()

Debug.WriteLine("3 Hours from now is: " & newDT.AddHours(3))

'This one will actually change the newDT value

newDT = newDT.AddDays(50)

Debug.WriteLine("50 days from now is: " & newDT)

Here's the output:

Current date and time: 12/21/2017 7:59:28 AM

Long date: Thursday, December 21, 2017

Short date: 12/21/2017

Long time: 7:59:28 AM

Short time: 7:59 AM

Universal time: 12/21/2017 12:59:28 PM

Let's add some times and dates:

3 Hours from now is: 12/21/2017 10:59:28 AM

50 days from now is: 2/9/2018 7:59:28 AM

# Regular Expressions (RegEx)

Regular expressions are used to find patterns in text and also to ensure that data is of a specific format. Regular expressions are very powerful to use, but fairly difficult to learn since the expressions are very symbolic, and in many cases, cryptic. Anyone that has worked much with the UNIX operating system or the programming language Perl usually learns how to manipulate regular expressions.

While regular expressions may be complicated to work with initially, the power that they pack into such compact syntax outweighs the learning curve you'll have to expend. I would highly recommend that you take a look at regular expressions and play around with them before you actually need to use them. The following tables show the different symbols that can be used to create regular expressions. This information comes directly from the MSDN help files.

## [Character Escapes](javascript:void(0))

The backslash character (\) in a regular expression indicates that the character that follows it either is a special character (as shown in the following table), or should be interpreted literally. For more information, see [Character Escapes in Regular Expressions](http://msdn.microsoft.com/en-us/library/4edbef7e.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Escaped character | Description | Pattern | Matches |
| \a | Matches a bell character, \u0007. | \a | "\u0007" in "Error!" + '\u0007' |
| \b | In a character class, matches a backspace, \u0008. | [\b]{3,} | "\b\b\b\b" in "\b\b\b\b" |
| \t | Matches a tab, \u0009. | (\w+)\t | "item1\t", "item2\t" in "item1\titem2\t" |
| \r | Matches a carriage return, \u000D. (\r is not equivalent to the newline character, \n.) | \r\n(\w+) | "\r\nThese" in "\r\nThese are\ntwo lines." |
| \v | Matches a vertical tab, \u000B. | [\v]{2,} | "\v\v\v" in "\v\v\v" |
| \f | Matches a form feed, \u000C. | [\f]{2,} | "\f\f\f" in "\f\f\f" |
| \n | Matches a new line, \u000A. | \r\n(\w+) | "\r\nThese" in "\r\nThese are\ntwo lines." |
| \e | Matches an escape, \u001B. | \e | "\x001B" in "\x001B" |
| \ nnn | Uses octal representation to specify a character (nnn consists of two or three digits). | \w\040\w | "a b", "c d" in  "a bc d" |
| \x nn | Uses hexadecimal representation to specify a character (nn consists of exactly two digits). | \w\x20\w | "a b", "c d" in  "a bc d" |
| \c X  \c x | Matches the ASCII control character that is specified by X or x, where X or x is the letter of the control character. | \cC | "\x0003" in "\x0003" (Ctrl-C) |
| \u nnnn | Matches a Unicode character by using hexadecimal representation (exactly four digits, as represented by nnnn). | \w\u0020\w | "a b", "c d" in  "a bc d" |
| \ | When followed by a character that is not recognized as an escaped character in this and other tables in this topic, matches that character. For example, \\* is the same as \x2A, and \. is the same as \x2E. This allows the regular expression engine to disambiguate language elements (such as \* or ?) and character literals (represented by \\* or \?). | \d+[\+-x\\*]\d+\d+[\+-x\\*\d+ | "2+2" and "3\*9" in "(2+2) \* 3\*9" |

## [Character Classes](javascript:void(0))

A character class matches any one of a set of characters. Character classes include the language elements listed in the following table. For more information, see [Character Classes in Regular Expressions](http://msdn.microsoft.com/en-us/library/20bw873z.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Character class | Description | Pattern | Matches |
| [character\_group] | Matches any single character in character\_group. By default, the match is case-sensitive. | [ae] | "a" in "gray"  "a", "e" in "lane" |
| [^character\_group] | Negation: Matches any single character that is not in character\_group. By default, characters incharacter\_group are case-sensitive. | [^aei] | "r", "g", "n" in "reign" |
| [ first - last ] | Character range: Matches any single character in the range from first to last. | [A-Z] | "A", "B" in "AB123" |
| . | Wildcard: Matches any single character except \n.  To match a literal period character (. or \u002E), you must precede it with the escape character (\.). | a.e | "ave" in "nave"  "ate" in "water" |
| \p{ name } | Matches any single character in the Unicode general category or named block specified by name. | \p{Lu}  \p{IsCyrillic} | "C", "L" in "City Lights"  "Д", "Ж" in "ДЖem" |
| \P{ name } | Matches any single character that is not in the Unicode general category or named block specified byname. | \P{Lu}  \P{IsCyrillic} | "i", "t", "y" in "City"  "e", "m" in "ДЖem" |
| \w | Matches any word character. | \w | "I", "D", "A", "1", "3" in "ID A1.3" |
| \W | Matches any non-word character. | \W | " ", "." in "ID A1.3" |
| \s | Matches any white-space character. | \w\s | "D " in "ID A1.3" |
| \S | Matches any non-white-space character. | \s\S | " \_" in "int \_\_ctr" |
| \d | Matches any decimal digit. | \d | "4" in "4 = IV" |
| \D | Matches any character other than a decimal digit. | \D | " ", "=", " ", "I", "V" in "4 = IV" |

## [Anchors](javascript:void(0))

Anchors, or atomic zero-width assertions, cause a match to succeed or fail depending on the current position in the string, but they do not cause regex to advance through the string or consume characters. The metacharacters listed in the following table are anchors. For more information, see [Anchors in Regular Expressions](http://msdn.microsoft.com/en-us/library/h5181w5w.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Assertion | Description | Pattern | Matches |
| ^ | The match must start at the beginning of the string or line. | ^\d{3} | "901" in  "901-333-" |
| $ | The match must occur at the end of the string or before \n at the end of the line or string. | -\d{3}$ | "-333" in  "-901-333" |
| \A | The match must occur at the start of the string. | \A\d{3} | "901" in  "901-333-" |
| \Z | The match must occur at the end of the string or before \n at the end of the string. | -\d{3}\Z | "-333" in  "-901-333" |
| \z | The match must occur at the end of the string. | -\d{3}\z | "-333" in  "-901-333" |
| \G | The match must occur at the point where the previous match ended. | \G\(\d\) | "(1)", "(3)", "(5)" in "(1)(3)(5)[7](9)" |
| \b | The match must occur on a boundary between a \w (alphanumeric) and a \W(nonalphanumeric) character. | \b\w+\s\w+\b | "them theme", "them them" in "them theme them them" |
| \B | The match must not occur on a \b boundary. | \Bend\w\*\b | "ends", "ender" in "end sends endure lender" |

## [Grouping Constructs](javascript:void(0))

Grouping constructs delineate subexpressions of a regular expression and typically capture substrings of an input string. Grouping constructs include the language elements listed in the following table. For more information, see [Grouping Constructs in Regular Expressions](http://msdn.microsoft.com/en-us/library/bs2twtah.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Grouping construct | Description | Pattern | Matches |
| ( subexpression ) | Captures the matched subexpression and assigns it a zero-based ordinal number. | (\w)\1 | "ee" in "deep" |
| (?< name >subexpression ) | Captures the matched subexpression into a named group. | (?<double>\w)\k<double> | "ee" in "deep" |
| (?< name1 -name2 >subexpression ) | Defines a balancing group definition. For more information, see the "Balancing Group Definition" section in [Grouping Constructs in Regular Expressions](http://msdn.microsoft.com/en-us/library/bs2twtah.aspx). | (((?'Open'\()[^\(\)]\*)+((?'Close-Open'\))[^\(\)]\*)+)\*(?(Open)(?!))$ | "((1-3)\*(3-1))" in "3+2^((1-3)\*(3-1))" |
| (?:subexpression ) | Defines a noncapturing group. | Write(?:Line)? | "WriteLine" in "Console.WriteLine()" |
| (?imnsx-imnsx:subexpression ) | Applies or disables the specified options within subexpression. For more information, see [Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx). | A\d{2}(?i:\w+)\b | "A12xl", "A12XL" in "A12xl A12XL a12xl" |
| (?=subexpression ) | Zero-width positive lookahead assertion. | \w+(?=\.) | "is", "ran", and "out" in "He is. The dog ran. The sun is out." |
| (?!subexpression ) | Zero-width negative lookahead assertion. | \b(?!un)\w+\b | "sure", "used" in "unsure sure unity used" |
| (?<=subexpression ) | Zero-width positive lookbehind assertion. | (?<=19)\d{2}\b | "99", "50", "05" in "1851 1999 1950 1905 2003" |
| (?<!subexpression ) | Zero-width negative lookbehind assertion. | (?<!19)\d{2}\b | "51", "03" in "1851 1999 1950 1905 2003" |
| (?>subexpression ) | Nonbacktracking (or "greedy") subexpression. | [13579](?>A+B+) | "1ABB", "3ABB", and "5AB" in "1ABB 3ABBC 5AB 5AC" |

## [Quantifiers](javascript:void(0))

A quantifier specifies how many instances of the previous element (which can be a character, a group, or a character class) must be present in the input string for a match to occur. Quantifiers include the language elements listed in the following table. For more information, see [Quantifiers in Regular Expressions](http://msdn.microsoft.com/en-us/library/3206d374.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Quantifier | Description | Pattern | Matches |
| \* | Matches the previous element zero or more times. | \d\*\.\d | ".0", "19.9", "219.9" |
| + | Matches the previous element one or more times. | "be+" | "bee" in "been", "be" in "bent" |
| ? | Matches the previous element zero or one time. | "rai?n" | "ran", "rain" |
| { n } | Matches the previous element exactly n times. | ",\d{3}" | ",043" in "1,043.6", ",876", ",543", and ",210" in "9,876,543,210" |
| { n ,} | Matches the previous element at least n times. | "\d{2,}" | "166", "29", "1930" |
| { n , m } | Matches the previous element at least n times, but no more than m times. | "\d{3,5}" | "166", "17668"  "19302" in "193024" |
| \*? | Matches the previous element zero or more times, but as few times as possible. | \d\*?\.\d | ".0", "19.9", "219.9" |
| +? | Matches the previous element one or more times, but as few times as possible. | "be+?" | "be" in "been", "be" in "bent" |
| ?? | Matches the previous element zero or one time, but as few times as possible. | "rai??n" | "ran", "rain" |
| { n }? | Matches the preceding element exactly n times. | ",\d{3}?" | ",043" in "1,043.6", ",876", ",543", and ",210" in "9,876,543,210" |
| { n ,}? | Matches the previous element at least n times, but as few times as possible. | "\d{2,}?" | "166", "29", "1930" |
| { n , m }? | Matches the previous element between n and m times, but as few times as possible. | "\d{3,5}?" | "166", "17668"  "193", "024" in "193024" |

## [Backreference Constructs](javascript:void(0))

A backreference allows a previously matched subexpression to be identified subsequently in the same regular expression. The following table lists the backreference constructs supported by regular expressions in the .NET Framework. For more information, see [Backreference Constructs in Regular Expressions](http://msdn.microsoft.com/en-us/library/thwdfzxy.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Backreference construct | Description | Pattern | Matches |
| \ number | Backreference. Matches the value of a numbered subexpression. | (\w)\1 | "ee" in "seek" |
| \k< name > | Named backreference. Matches the value of a named expression. | (?<char>\w)\k<char> | "ee" in "seek" |

## [Alternation Constructs](javascript:void(0))

Alternation constructs modify a regular expression to enable either/or matching. These constructs include the language elements listed in the following table. For more information, see [Alternation Constructs in Regular Expressions](http://msdn.microsoft.com/en-us/library/36xybswe.aspx).

|  |  |  |  |
| --- | --- | --- | --- |
| Alternation construct | Description | Pattern | Matches |
| | | Matches any one element separated by the vertical bar (|) character. | th(e|is|at) | "the", "this" in "this is the day. " |
| (?( expression) yes | no ) | Matches yes if expression matches; otherwise, matches the optional no part.expression is interpreted as a zero-width assertion. | (?(A)A\d{2}\b|\b\d{3}\b) | "A10", "910" in "A10 C103 910" |
| (?( name ) yes| no ) | Matches yes if the named capture name has a match; otherwise, matches the optional no. | (?<quoted>")?(?(quoted).+?"|\S+\s) | Dogs.jpg, "Yiska playing.jpg" in "Dogs.jpg "Yiska playing.jpg"" |

## [Substitutions](javascript:void(0))

Substitutions are regular expression language elements that are supported in replacement patterns. For more information, see [Substitutions in Regular Expressions](http://msdn.microsoft.com/en-us/library/ewy2t5e0.aspx). The metacharacters listed in the following table are atomic zero-width assertions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Character | Description | Pattern | Replacement pattern | Input string | Result string |
| $ number | Substitutes the substring matched by group number. | \b(\w+)(\s)(\w+)\b | $3$2$1 | "one two" | "two one" |
| ${ name } | Substitutes the substring matched by the named group name. | \b(?<word1>\w+)(\s)(?<word2>\w+)\b | ${word2} ${word1} | "one two" | "two one" |
| $$ | Substitutes a literal "$". | \b(\d+)\s?USD | $$$1 | "103 USD" | "$103" |
| $& | Substitutes a copy of the whole match. | (\$\*(\d\*(\.+\d+)?){1}) | \*\*$& | "$1.30" | "\*\*$1.30\*\*" |
| $` | Substitutes all the text of the input string before the match. | B+ | $` | "AABBCC" | "AAAACC" |
| $' | Substitutes all the text of the input string after the match. | B+ | $' | "AABBCC" | "AACCCC" |
| $+ | Substitutes the last group that was captured. | B+(C+) | $+ | "AABBCCDD" | AACCDD |
| $\_ | Substitutes the entire input string. | B+ | $\_ | "AABBCC" | "AAAABBCCCC" |

## [Regular Expression Options](javascript:void(0))

You can specify options that control how the regular expression engine interprets a regular expression pattern. Many of these options can be specified either inline (in the regular expression pattern) or as one or more [RegexOptions](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regexoptions.aspx) constants. This quick reference lists only the inline options. For more information about inline and [RegexOptions](http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regexoptions.aspx)options, see the article [Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx).

You can specify an inline option in two ways:

1 - By using the [miscellaneous construct](http://msdn.microsoft.com/en-us/library/x044wc7s.aspx)(?imnsx-imnsx), where a minus sign (-) before an option or set of options turns those options off. For example, (?i-mn) turns case-insensitive matching (i) on, turns multiline mode (m) off, and turns unnamed group captures (n) off. The option applies to the regular expression pattern from the point at which the option is defined, and is effective either to the end of the pattern or to the point where another construct reverses the option.

2 - By using the [grouping construct](http://msdn.microsoft.com/en-us/library/bs2twtah.aspx)(?imnsx-imnsx:subexpression), which defines options for the specified group only.

The .NET Framework regular expression engine supports the following inline options:

|  |  |  |  |
| --- | --- | --- | --- |
| Option | Description | Pattern | Matches |
| i | Use case-insensitive matching. | \b(?i)a(?-i)a\w+\b | "aardvark", "aaaAuto" in "aardvark AAAuto aaaAuto Adam breakfast" |
| m | Use multiline mode. ^ and $ match the beginning and end of a line, instead of the beginning and end of a string. | For an example, see the "Multiline Mode" section in [Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx). |  |
| n | Do not capture unnamed groups. | For an example, see the "Explicit Captures Only" section in [Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx). |  |
| s | Use single-line mode. | For an example, see the "Single-line Mode" section in [Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx). |  |
| x | Ignore unescaped white space in the regular expression pattern. | \b(?x) \d+ \s \w+ | "1 aardvark", "2 cats" in "1 aardvark 2 cats IV centurions" |

## [Miscellaneous Constructs](javascript:void(0))

Miscellaneous constructs either modify a regular expression pattern or provide information about it. The following table lists the miscellaneous constructs supported by the .NET Framework. For more information, see [Miscellaneous Constructs in Regular Expressions](http://msdn.microsoft.com/en-us/library/x044wc7s.aspx).

|  |  |  |
| --- | --- | --- |
| Construct | Definition | Example |
| (?imnsx-imnsx) | Sets or disables options such as case insensitivity in the middle of a pattern. For more information, see[Regular Expression Options](http://msdn.microsoft.com/en-us/library/yd1hzczs.aspx). | \bA(?i)b\w+\b matches "ABA", "Able" in "ABA Able Act" |
| (?#comment ) | Inline comment. The comment ends at the first closing parenthesis. | \bA(?#Matches words starting with A)\w+\b |
| # [to end of line] | X-mode comment. The comment starts at an unescaped # and continues to the end of the line. | (?x)\bA\w+\b#Matches words starting with A |

Here's an example program that parses a string to locate dates contained within the string even though the dates may be in different formats.

'Chapter 7 - Program 5

'\*\*\* MUST IMPORT REGULAREXPRESSIONS OURSELVES \*\*\*

Imports System.Text.RegularExpressions

Module Module1

Public Sub Main()

'Define the string containing some dates

Dim strDates = " 12-2-2020 10/24/2021 4/7/22 "

'Define a regular expression that defines how a valid

'date can look

Dim MyRegex As New \_

Regex("\s\*\d{1,2}(/|-)\d{1,2}\1(\d{4}|\d{2})")

'Here's where we'll store our valid dates located

Dim strOutput As String

'Attempt to locate a match in the string of dates

Dim strMatch As Match = MyRegex.Match(strDates)

strOutput = "Valid Dates Found In String:" & vbCrLf

'If we found a match...

Do While strMatch.Success

'...then print out the match

strOutput &= strMatch.ToString.Trim & vbCrLf

'and try to locate another match

strMatch = strMatch.NextMatch

Loop

Debug.WriteLine(strOutput)

End Sub

End Module

Output:

Valid Dates Found In String:

12-2-2020

10/24/2021

4/7/22

All three of the dates were successfully parsed out of the string. There is no doubt that the syntax of the regular expression parser is a bit unwieldy, yet there is far more power in it than using the simple Split/Join/Like functions. Again, look over regular expressions as it is more than likely that you will need to use them at some point in your programming career.

Here’s one more sample RegEx example that shows how to carry out some search and replace type work via regular expressions. This program will format a pure dotted phone number into one that uses ( ) and – where they belong.

'Chapter 7 - Program 6

'\*\*\* MUST IMPORT REGULAREXPRESSIONS OURSELVES \*\*\*

Imports System.Text.RegularExpressions

Module Module1

Public Sub Main()

'Define the string containing the original, unformatted phone number

Dim strBasicPhoneNumber = "989.964.4896"

'Define a regular expression that defines how the unformatted phone

'number looks

Dim MyRegex As New \_

Regex("(\d{3}).(\d{3}).(\d{4})")

'This is where the newly formatted phone number will go

Dim strNewNumber As String

'The second parameter in the Replace method has the literal ( ) -

'characters where they belong and then uses the backsubstitution

'of the matched parts of the phone number into the formatted string

strNewNumber = MyRegex.Replace(strBasicPhoneNumber, "($1) $2-$3")

Debug.WriteLine(strNewNumber)

End Sub

End Module

Here’s the formatted output:

(989) 964-4896

Obviously this was a simple example, but again it illustrates that not only can regular expressions be used to locate data that conforms to a particular format, they can also manipulate and modify that data. There is a learning curve to RegEx, but once it’s mastered you have a really powerful tool at your disposal. Think about how many times you’ve written code to mold data into the shape you wanted it…

## Random – The Random Number Generating Class

Since we are looking at some of the utility classes present in .NET, I’ll mention one more: Random. This class can be useful whenever we want to create random numbers

Here's a sample program than generates 10 random numbers between 1 and 5:

Public Sub Main()

'Create a new Random object

Dim rndObj As Random = New Random()

'Calling the Random object's Next method will keep

'producing new random numbers

Dim rndNum As Integer = rndObj.Next()

Dim aValue As Integer

Dim intLoop As Integer

Dim strOutput As String

strOutput = "Iteration" & vbTab & "Value" & vbCrLf

For intLoop = 1 To 10

aValue = 1 + rndObj.Next(5) 'next gives us zero to four

strOutput &= intLoop & vbTab & aValue & vbCrLf

Next

Debug.WriteLine(strOutput)

End Sub

Here’s the output:

Iteration Value

1 4

2 1

3 2

4 4

5 5

6 2

7 4

8 4

9 3

10 4

You can also ask for a range of numbers through the .Next method, as in .Next(x,y) where x <= .Next(x,y) < y. Now one other thing you should know about the Random number generator in .NET, it’s not that great. If you ask it to generate 100 random numbers between 1 and 10, you would expect to see a fairly straight distribution, but:

Public Sub Main()

'Create a new Random object

Dim rndObj As Random = New Random()

Dim arrCounts(10) As Integer

'Calling the Random object's Next method will keep

'producing new random numbers

Dim rndNum As Integer = rndObj.Next()

Dim aValue As Integer

Dim intLoop As Integer

Dim strOutput As String

For intLoop = 1 To 100

aValue = 1 + rndObj.Next(10) 'gives us one to ten when done

strOutput &= aValue & ","

arrCounts(aValue) += 1

Next

For intLoop = 1 To 10

Debug.WriteLine("Number: " & intLoop & " Count: " & arrCounts(intLoop)

& " Percentage: " & FormatPercent(arrCounts(intLoop) /

100))

Next

Debug.WriteLine("Sequence Generated: " & Strings.Left(strOutput,

Len(strOutput) - 1))

End Sub

Check out the output, where each number should approximate 10%:

Number: 1 Count: 13 Percentage: 13.00%

Number: 2 Count: 11 Percentage: 11.00%

Number: 3 Count: 5 Percentage: 5.00%

Number: 4 Count: 10 Percentage: 10.00%

Number: 5 Count: 6 Percentage: 6.00%

Number: 6 Count: 11 Percentage: 11.00%

Number: 7 Count: 12 Percentage: 12.00%

Number: 8 Count: 10 Percentage: 10.00%

Number: 9 Count: 9 Percentage: 9.00%

Number: 10 Count: 13 Percentage: 13.00%

Sequence Generated: 2,7,1,9,9,1,4,10,2,2,2,8,1,7,2,4,2,10,9,10,7,10,6,4,2,3,8,9,4,9,7,9,1,5,1,9,7,8,3,4,6,6,10,6,4,5,2,8,8,10,5,7,9,5,10,3,2,8,7,7,4,6,7,3,6,4,8,6,6,3,7,4,8,6,5,1,10,10,10,8,1,1,2,4,5,10,1,6,6,9,8,10,2,7,1,1,10,7,1,1

Yikes! The 3 showed up only 5%; not very good and look at the 10,10,10 sequence and the 1,1 sets… Not all that great! However, if we run it longer it tends to get better… If I rerun the same application but with 100,000 iterations, things tend to start getting more even:

Number: 1 Count: 9929 Percentage: 9.93%

Number: 2 Count: 9953 Percentage: 9.95%

Number: 3 Count: 9896 Percentage: 9.90%

Number: 4 Count: 9996 Percentage: 10.00%

Number: 5 Count: 10081 Percentage: 10.08%

Number: 6 Count: 10105 Percentage: 10.11%

Number: 7 Count: 9994 Percentage: 9.99%

Number: 8 Count: 10049 Percentage: 10.05%

Number: 9 Count: 10012 Percentage: 10.01%

Number: 10 Count: 9985 Percentage: 9.99%

Just consider yourself warned before getting too dependent on this class. Also there is a method (.Sample) that exists which you can override and modify to deliver your own pseudorandom numbers. Remember that the purpose is to deliver a random number set – that does not automatically equate to a normalized set, e.g. every value appears roughly the same number of times. Me personally, I usually want normalized random numbers!