Visual Basic .NET Strings

Visual Basic has a built in data type called String, which is a sequence of Unicode characters. Unicode characters are each 2 bytes in length. Remember that when a computer stores a character and deals with a character it is actually using a number that represents that character, i.e. the character ‘A’ has a numeric value of 65, and the character ‘2’ has a numeric value of 50.

String variables are stored as sequences of unsigned 16-bit (2-byte) numbers ranging in value from 0 through 65535. Each number represents a single Unicode character. A string can contain up to approximately 2 billion (2^31) Unicode characters.

The first 128 numbers (0–127) of Unicode correspond to the letters and symbols on a standard U.S. keyboard. These first 128 numbers are the same as those defined by the ASCII character set. The second 128 numbers (128–255) represent special characters, such as Latin-based alphabet letters, accents, currency symbols, and fractions. The remaining numbers are used for a wide variety of symbols, including worldwide textual characters, diacritics, and mathematical and technical symbols.

Appending the identifier type character $ to any identifier forces it to the String data type, similar to the way one appends the letter D after a number to ensure it is treated as a Decimal data type.

Property Members of String Class, 2 of real importance
Length Holds the numbers of characters in the string object.
Chars(index) References the character at the given index, first character is always 0.

Standard String Operators
VB has a concatenation operator denoted by the “&”, which will allow two items to be combined together to form one string.
Strings can be compared with all the relational operators, <, <=, >, >= , = , <>

Methods (or procedures) of the String Class
The following is a partial list of some of the methods in the string class. For the shared methods, which are given last, you don’t need to specify a String Object first, but the data type String. For nonshared methods, you must attach the method to the String Object by using the dot . operator. For all the following examples the following definitions are used and some others as well:

Dim strA, Source, Changed, Test As String

Example of shared method:
If String.Compare(strA, strB) > 0 then
        
Example of Nonshared method
If SourceStr.EndsWith("ed") then
        

**ToUpper()**  
Returns a copy of the string in uppercase, only affects lower case alphabetic characters.  
Example:  
```vbnet  
Dim Source As String = "hello"  
Dim Changed As String  
Changed = Source.ToUpper() 'Changed holds "HELLO"  
```

**ToLower()**  
Returns a copy of the string in lowercase, only affects upper case alphabetic characters.  
Example:  
```vbnet  
Dim Source As String = "HELLO"  
Dim Changed As String  
Changed = Source.ToLower() 'Changed holds "hello"  
```

**Trim()**  
Removes all white-space characters from the beginning and end of string.  
Example:  
```vbnet  
Dim Source As String = "      Hello        "  
Dim Changed As String  
Changed = Source.Trim() 'Changed holds "Hello"  
```

**TrimEnd()**  
Removes all white-space characters from the end of string.  

**TrimStart()**  
Removes all white-space characters from the beginning of string.  

**PadLeft(len, char)**  
Adds a specified character to the beginning of a string until the string is a specified length.  
*len* is the padded length of the string and *char* is a character in quotation marks or a Char variable. If no *char* is given then pad with leading blanks.  
Example:  
```vbnet  
Dim Source As String = "hello"  
Dim Changed As String  
Changed = Source.PadLeft(10, "***c") 'Changed holds "*****hello"  
' need 'c' after the character *** to  
'tell it is a character  
```

**PadRight(len, char)**  
Adds a specified character to the end of a string until the string is a specified length.  
*len* is the padded length of the string and *char* is a character in quotation marks or a Char variable. If no *char* is given then pad with trailing blanks.  
Example:  
```vbnet  
Dim Source As String = "hello"  
Dim Changed As String  
Changed = Source.PadRight(10) 'Changed holds "hello     "  
```
**EndsWith**(*strA*)
Returns a True if the end of the string object matches the specified String, *strA*. *strA* may be a string literal or a string variable. Is case sensitive.
Example:

```vbnet
Dim Source As String = "hello"
Dim Answer As Boolean
Answer = Source.EndsWith("llo") 'Answer holds "True"
```

**StartsWith**(*strA*)
Returns a True if the start of the string object matches the specified String, *strA*. *strA* may be a string literal or a string variable. Is case sensitive.
Example:

```vbnet
Dim Source As String = "hello"
Dim Test As String = "ab"
Dim Answer As Boolean
Answer = Source.StartsWith(Test) 'Answer holds "False"
```

**Equals**(*strA*)
Returns a True if the string object matches the specified String, *strA*. Is case Sensitive.
Example:

```vbnet
Dim Source As String = "hello"
Dim Test As String = "ab"
Dim Answer As Boolean
Answer = Source.Equals(Test) 'Answer holds "False"
```

**IndexOf**(*char, StartPosition*)
Searches the string object from *StartPosition* and looks for the *char* in string. When finds the first occurrence of the *char* it returns the index number, if not found returns a negative value. If *StartPosition* is left out, then starts searching from the beginning of string. *StartPosition* is an Integer and *char* is a character or a Char variable.
Example:

```vbnet
Dim Source As String = "hello"
Dim IndexPosition As Integer
IndexPosition = Source.IndexOf("l") 'IndexPosition holds 2
```

**IndexOf**(*strA, StartPosition*)
Does the same thing as above except that it searches the string object for another string.

**LastIndexOf**(*x,y*)
Does the same as ones above, but starts searching from the end of the string object. *x* and *y* would correspond to ones above as well.

**Insert**(*StartPosition*, *strA*)
Inserts the specified string, *strA*, into the string object at the given *StartPosition* as index number.
Example:

```vbnet
Dim Source As String = "hello"
Dim Test As String
Test = Source.Insert(2,"bb") 'Test holds "hebblo"
```
**Remove(StartPosition, NumberCharacters)**

Deletes specified number of characters from the string starting at given location. *StartPosition* and *NumberCharacters* are both Integers.

Example:

```vba
Dim Source As String = “abcdefg”
Dim Test As String
Test = Source.Remove(3,2) ‘Test holds “abcfg”
```

**Replace(OldString, NewString)**

Searches through the string and replaces each occurrence of the specified string, *OldString*, with the new string specified with *NewString*, will also work for character replacement. *OldString* or *NewString* may be a string variable or a string literal.

Example:

```vba
Dim Source As String = “abcabesf”
Dim NewString As String = “34”
Dim Test As String
Test = Source.Replace(“ab”, NewString) ‘Test holds “34c34esf”
```

**Substring(StartPosition, NumberCharacters)**

Returns a substring of the string object. *StartPosition* indicates the position of the character that starts the substring, with the first character of a string at position 0. *NumberCharacters* indicates the length of the substring, if left off, then substring would start at *StartPosition* location and go until the end of string.

Example:

```vba
Dim Source As String = “abcabesf”
Dim Test As String
Test = Source.Substring(3,2) ‘Test holds “ab”
```

The following are considered Shared Methods:

**Compare(strA, strB, CaseInsensitive)**

Returns a 0 if the two strings, *strA* and *strB*, are the same. A positive number is returned if *strA* is greater then *strB* and a negative number is returned if *strA* is less then *strB*. *CaseInsensitive* should be True if the case of the strings should not be considered, a False will compare the case of the strings as well, if leave off then defaults to False.

Example:

```vba
Dim Source As String = “abcd”
Dim Test As String = “abf”
Dim Answer As String
Answer = String.Compare(Source, Test) ‘Answer holds -1 because “abcd” < “abf”
```

Example 2:

```vba
Dim Source As String = “AbC”
Dim Test As String = “abc”
Dim Answer As String
Answer = String.Compare(Source, Test, True) ‘Answer holds 0 because “AbC” = “abc”
```

When using the Compare method it should be used when you want to alphabetically compare strings. The relational operators work as well, but they compare the Unicode (ASCII) values of the characters. Where a capital ‘A’ has a value of 65 and a lower ‘a’ has a value of 97.

There are other forms of the Compare method as well, by use of the help features one would able to look these up.