Read and write text files with Visual Basic .NET

With Visual Basic .NET, Microsoft introduced a new, object-oriented method for working with files. The System.IO namespace in the .NET framework provides several classes for working with text files, binary files, directories, and byte streams. I will look specifically at working with text files using classes from the System.IO namespace.

**Basic methods**

Before we jump into working with text files, we need to create a new file or open an existing one. That requires the System.IO.File class. This class contains methods for many common file operations, including copying, deleting, file attribute manipulation, and file existence. For our text file work, we will use the CreateText and OpenText methods.

**CreateText**

True to its name, the CreateText method creates a text file and returns a System.IO.StreamWriter object. With the StreamWriter object, you can then write to the file. The following code demonstrates how to create a text file:

Dim oFile as System.IO.File

Dim oWrite as System.IO.StreamWriter

oWrite = oFile.CreateText(“C:\sample.txt”)

**OpenText**

The OpenText method opens an existing text file for reading and returns a System.IO.StreamReader object. With the StreamReader object, you can then read the file. Let’s see how to open a text file for reading:

Dim oFile as System.IO.File ‘or do no not use, do below

Dim oRead as System.IO.StreamReader

oRead = oFile.OpenText(“C:\sample.txt”) ‘do not use ofile, change to IO.File.OpenText(…

**Writing to a text file**

The methods in the System.IO.StreamWriter class for writing to the text file are Write and WriteLine. The difference between these methods is that the WriteLine method appends a newline character at the end of the line while the Write method does not. Both of these methods are overloaded to write various data types and to write formatted text to the file. The following example demonstrates how to use the WriteLine method:

oWrite.WriteLine(“Write a line to the file”)

oWrite.WriteLine() ‘Write a blank line to the file

**Formatting the output**

The Write and WriteLine methods both support formatting of text during output. The ability to format the output has been significantly improved over previous versions of Visual Basic. There are several overloaded methods for producing formatted text. Let’s look at one of these methods:

oWrite.WriteLine(“{0,10}{1,10}{2,25}”, “Date”, “Time”, “Price”)

oWrite.WriteLine(“{0,10:dd MMMM}{0,10:hh:mm tt}{1,25:C}”, Now(), 13455.33)

oWrite.Close()

The overloaded method used in these examples accepts a string to be formatted and then a parameter array of values to be used in the formatted string. Let’s look at both lines more carefully.

The first line writes a header for our report. Notice the first string in this line is {0,10}{1,10}{2,25}. Each curly brace set consists of two numbers. The first number is the index of the item to be displayed in the parameter array. (Notice that the parameter array is zero based.) The second number represents the size of the field in which the parameter will be printed. Alignment of the field can also be defined; positive values are left aligned and negative values are right aligned.

The second line demonstrates how to format values of various data types. The first field is defined as {0,10:dd MMMM}. This will output today’s date (retrieved using the Now() function) in the format 02 July. The second field will output the current time formatted as 02:15 PM. The third field will format the value 13455.33 into the currency format as defined on the local machine. So if the local machine were set for U.S. Dollars, the value would be formatted as $13,455.33.

Listing A shows the output of our sample code.

|  |
| --- |
| Listing A |
| http://www.techrepublic.com/i/tr/spacer.gif |
| http://www.techrepublic.com/i/tr/spacer.gif |
| Date      Time                   Price     22 July  10:58 AM               $13,455.33 |

**Reading from a text file**

The System.IO.StreamReader class supports several methods for reading text files and offers a way of determining whether you are at the end of the file that's different from previous versions of Visual Basic.

**Line-by-line**

Reading a text file line-by-line is straightforward. We can read each line with a ReadLine method. To determine whether we have reached the end of the file, we call the Peek method of the StreamReader object. The Peek method reads the next character in the file without changing the place that we are currently reading. If we have reached the end of the file, Peek returns -1. Listing B provides an example for reading a file line-by-line until the end of the file.

|  |
| --- |
| Listing B |
| http://www.techrepublic.com/i/tr/spacer.gif |
| http://www.techrepublic.com/i/tr/spacer.gif |
| oRead = oFile.OpenText(“C:\sample.txt”) or IO.File.OpenText(“C:\sample.txt”)  While oRead.Peek <> -1        LineIn = oRead.ReadLine()  End While  oRead.Close() |

**An entire file**

You can also read an entire text file from the current position to the end of the file by using the ReadToEnd method, as shown in the following code snippet:

Dim EntireFile as String

oRead = oFile.OpenText(“C:\sample.txt”)

EntireFile = oRead.ReadToEnd()

This example reads the file into the variable EntireFile. Since reading an entire file can mean reading a large amount of data, be sure that the string can handle that much data.

**One character at a time**

If you need to read the file a character at a time, you can use the Read method. This method returns the integer character value of each character read. Listing C demonstrates how to use the Read method.

|  |
| --- |
| Listing C |
| http://www.techrepublic.com/i/tr/spacer.gif |
| http://www.techrepublic.com/i/tr/spacer.gif |
| Dim intSingleChar as Integer  Dim cSingleChar as String  oRead = oFile.OpenText(“C:\sample.txt”)  While oRead.Peek <> -1  intSingleChar = oRead.Read()  ‘ Convert the integer value into a character  cSingleChar = Chr(intSingleChar)  End While |