The Fastest Way to Write Text Files to Disk in C#

December 5, 2016 / C# Tips and Tricks / By Jeremy Shanks / 3 COMMENTS

A few weeks ago I was tasked with creating a very large CSV file with some dynamic content. I decided that this would be an excellent opportunity to try out some of my C# skills and write the content of the file using a loop. I quickly realized that not all methods of writing files to the hard drive perform the same. In fact, some methods are just plain terrible. I decided to write this article to compare different ways to write text files and evaluate their performance.

Writing Many Times in a Loop

The first scenario we are going to investigate is when we need to write lines generated by a loop. A situation that this is relevant is like the one in the introduction. We need to loop a bunch of times and generate the lines of text from the loop.

File.AppendAllText

```
for (int i = 1; i < 10000; i++)
{
    File.AppendAllText(Path, "Some line of text");
}
```

In the example above, we see that the loop is going to write “Some line of text” to a text file 10,000 times. This will turn out to be very slow because for every iteration of the for loop we are opening the file, appending some text, and then closing the file. **Don’t do this.**

Using StreamWriter (Declared in For Loop)

```
for (int i = 1; i < 10000; i++)
{
    using (StreamWriter streamwriter = File.AppendText(Path))
    {
        streamwriter.WriteLine("Some line of text");
    }
}
```

In this example, we are basically doing the same thing as the previous example except we are utilizing the “Using” statement and then appending the line of text. This will perform no better than when using “File.AppendAllText” because we are opening the file and writing to it 10000 times. **Don’t do this either.**

Using StreamWriter (Declared Before For Loop)

```
using (StreamWriter streamwriter = File.AppendText(Path))
{
    for (int i = 1; i < 10000; i++)
    {
        streamwriter.WriteLine("Some line of text");
    }
}
```

In this code snippet, we are opening the file only once and then writing to it 10000 times. This will perform much better than the previous 2 examples. I still wouldn’t recommend this because it still has to write to the hard drive many times.

Using StreamWriter With Larger Buffer

```
using (StreamWriter streamwriter = new StreamWriter(Path, true, Encoding.UTF8, 65536))
{
    for (int i = 1; i < repeats; i++)
    {
        streamwriter.WriteLine("Some line of text");
    }
}
```

In this example, you will see that we are doing the same process as the previous example, but we are setting a larger buffer of 65k. This performed very well in my tests. In some tests, it performed faster than all the next two examples. This can be explained by having a larger buffer which means the StreamWriter is writing to the hard drive less often.

Appending to String Then Writing Once

```
string text = "";
for (int i = 1; i < 10000; i++)
{
    text += "Some line of text";
}
File.AppendAllText(Path, text);
```

In this snippet, we are creating a string and then adding to that string 10000 times. We will then write this large string one time to the hard drive. This will perform well and does not stress the hard drive, but there is still one slightly faster way to do this.

Using StringBuilder Then Writing Once

```
StringBuilder stringbuilder = new StringBuilder();
for (int i = 1; i < repeats; i++)
{
    stringbuilder.Append("Some line of text");
}
File.AppendAllText(Path, stringbuilder.ToString());
```

The only difference in this example and the previous is we are using a StringBuilder to create the larger string before writing one time to the hard drive. When doing lots of string operations the StringBuilder is faster than just working with a declared string. This code snippet performs better than the rest of the above examples except the StreamWriter with a larger buffer.

The Numbers

I set up a stopwatch to time how long it took to run the above sample of code. Here are the results:

Method	Time w/ Spinner Drive	Time w/ SSD
File.WriteAllText	2140	2757
File.AppendAllText	1447	2115
Using StreamWriter (Declared in For Loop)	2019	2189
Using StreamWriter with Larger Buffer	1579	1779

Writing Large Strings

The second scenario we are going to look at is when we need to write a very large string to a text file. A situation like this could be reading a text file changing the text in some way and writing it back to the file.

Generating a String

To test this scenario, we will need to generate a very large string to write to a file. We will do this with the code below.

```
var chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
var stringChars = new char[200000000];
var random = new Random();
```

```
4
5 for (int i = 0; i < stringChars.Length; i++)
6 {
7     stringChars[i] = chars[random.Next(chars.Length)];
8 }
9 var finalString = new String(stringChars);
```

In the above code, we are randomly choosing letters from a string of characters (chars) and then adding them to a character array. We are then creating a new string from that character array.

Now that we have a way to generate a random string of the size that we want we can proceed.

File.WriteAllText

```
1 File.WriteAllText(Path, finalString);
```

In this example, we are calling the static WriteAllText method which performs worse than all of the following examples. **Don't do this.**

File.AppendAllText

```
1 File.AppendAllText(Path, finalString);
```

The static method AppendAllText will create a new file if it does not exist but will also append to the file if it already exists. Surprisingly it performs better when creating a new file than WriteAllText does.

Using StreamWriter

```
1 using (StreamWriter sw = new StreamWriter(Path))
2 {
3     sw.WriteLine(finalString);
4 }
```

Here we are using a StreamWriter with the default buffer. In my tests I found it does **not** perform as well as File.AppendAllText but is faster than File.WriteAllText when it comes to writing a very large string.

Using StreamWriter With Larger Buffer

```
1 using (StreamWriter sw = new StreamWriter(Path, false, Encoding.UTF8, 65536))
2 {
3     sw.WriteLine(finalString);
4 }
```

In this example, we are using a StreamWriter but with a custom buffer size of 65k. This performs well but not always faster than File.AppendAllText.

The Numbers

Method	Time w/ Spinner Drive	Time w/ SSD
File.AppendAllText	2105	3148
Using StreamWriter (Declared in For Loop)	2381	3116
Using StreamWriter (Declared Before For Loop)	16	4
Using StreamWriter with Larger Buffer	3	3
Appending to String Then Writing Once	947	941
Using StringBuilder Then Writing Once	10	3

I hope you have found this article helpful. If you did, please share it with a fellow programmer so that they too can be helped. If you see something I missed, please let me know in the comments below. As always thanks for reading.