How to read an entire file to a string using C#?

Ask Question

179

31

What is the quickest way to read a text file into a string variable?

I understand it can be done in several ways, such as read individual bytes and then convert those to string. I was looking for a method with minimal coding.

c# string file-io .net-3.5

edited Jan 25 '17 at 11:06
Breeze
1,467 2 20 34

asked Sep 12 '11 at 11:22
Shamim Hafiz
12.3k 30 90 152

Check this stackoverflow.com/questions/2855335/… – Sandeep G B Sep 12 '11 at 11:25

16 Answers

322

How about

```
string contents = File.ReadAllText(@"C:\temp\test.txt");
```

answered Sep 12 '11 at 11:24

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

✓

marc_s
584k 130 1124 1270

2 Not the best function to use, though. As Devendra D. Chavan points out in his answer, `StreamReader.ReadToEnd` is more efficient. – Owen Blacker Jun 3 '14 at 11:48

28 @OwenBlacker It depends on whether "quickest" means "least time to execute" or "least time to understand." – bonh Jun 22 '15 at 19:20

2 File.ReadAllText is definitively the easiest one to use, but as "Devendra D. Chavan" points out, it is not the fastest. So if you are reading small files, then it would be a better choice to use File.ReadAllText.it really depends on how big the textfiles are that you are reading. – Mana Sep 21 '15 at 12:04

To **read from the server** check this, hope helps someone. – stom Jul 7 '16 at 11:11

When developing and you need to load some data, this is good enough. It's easy to use for something temporary that will be removed eventually. – Jonny Aug 24 '18 at 8:54

2

You can use like this

```
public static string ReadFileAndFetchStringInSingleLine(string file)
    {
        StringBuilder sb;
        try
        {
            sb = new StringBuilder();
            using (FileStream fs = File.Open(file, FileMode.Open))
            {
                using (BufferedStream bs = new BufferedStream(fs))
                {
                    using (StreamReader sr = new StreamReader(bs))
                    {
                        string str;
                        while ((str = sr.ReadLine()) != null)
```

```
                            {
                                sb.Append(str);
                            }
                        }
                    }
                }
                return sb.ToString();
            }
            catch (Exception ex)
            {
                return "";
            }
        }
```

Hope this will help you.

edited Jun 23 '16 at 8:57

answered Jun 23 '16 at 7:51

Amit Kumawat
469 5 10

you can use :

2

```
 public static void ReadFileToEnd()
{
    try
    {
    //provide to reader your complete text file
        using (StreamReader sr = new StreamReader("TestFile.txt"))
        {
            String line = sr.ReadToEnd();
            Console.WriteLine(line);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("The file could not be read:");
        Console.WriteLine(e.Message);
    }
}
```

answered Mar 31 '15 at 9:51

Erwin Draconis
332 3 17

3

For the noobs out there who find this stuff fun and interesting, the fastest way to read an entire file into a string in most cases (according to these benchmarks) is by the following:

```
using (StreamReader sr = File.OpenText(fileName))
{
        string s = sr.ReadToEnd();
}
//you then have to process the string
```

However, the absolute fastest to read a text file overall appears to be the following:

```
using (StreamReader sr = File.OpenText(fileName))
{
        string s = String.Empty;
        while ((s = sr.ReadLine()) != null)
        {
               //do what you have to here
        }
}
```

Put up against several other techniques, it won out most of the time, including against the BufferedReader.

answered Dec 23 '14 at 7:46

Thrawn Wannabe

540 5 10

Comment is late I know, but a little confused on your benchmarks here and on the linked page. It appears to be testing read speeds only and not loading into an entire string. The second code snippet is reading a line at a time and not doing any appending so the "do what you have to here" would need to have a string builder or string to hold the data. At which point the memory used to add more data would change the test results. So s will usually be the same size assuming a fixed width file so the memory will be set for the size of a line and data won't need to be copied to new memory. – Charles Byrne Aug 18 '16 at 14:44

3

if you want to pick file from Bin folder of the application then you can try following and don't forget to do exception handling.

```
string content =
File.ReadAllText(Path.Combine(System.IO.Directory.GetCurrentDirectory
@"FilesFolder\Sample.txt"));
```

answered Jun 9 '14 at 10:24

Deeps
210 1 4 15

0

I made a comparison between a ReadAllText and StreamBuffer for a 2Mb csv and it seemed that the difference was quite small but ReadAllText seemed to take the upper hand from the times taken to complete functions.

answered Dec 6 '13 at 12:00

Hatitye Chindove
6 1

0

```
public partial class Testfile : System.Web.UI.Page
{
    public delegate void DelegateWriteToDB(string Inputstring);
    protected void Page_Load(object sender, EventArgs e)
    {
        getcontent(@"C:\Working\Teradata\New folder");
    }

      private void SendDataToDB(string data)
    {
        //InsertIntoData
          //Provider=SQLNCLI10.1;Integrated Security=SSPI;Persist Se
Info=False;User ID="";Initial Catalog=kannan;Data Source=jaya;
        SqlConnection Conn = new SqlConnection("Data Source=aras;Ini
Catalog=kannan;Integrated Security=true;");
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = Conn;
        cmd.CommandType = CommandType.Text;
        cmd.CommandText = "insert into test_file values('"+data+"')"
        cmd.Connection.Open();
        cmd.ExecuteNonQuery();
        cmd.Connection.Close();
    }

      private void getcontent(string path)
      {
          string[] files;
          files = Directory.GetFiles(path, "*.txt");
          StringBuilder sbData = new StringBuilder();
          StringBuilder sbErrorData = new StringBuilder();
          Testfile df = new Testfile();
          DelegateWriteToDB objDelegate = new DelegateWriteToDB(df.S
          //dt.Columns.Add("Data",Type.GetType("System.String"));


          foreach (string file in files)
          {
              using (StreamReader sr = new StreamReader(file))
              {
                  String line;
                  int linelength;
                  string space = string.Empty;

                  // Read and display lines from the file until the
```

```
                    // the file is reached.
                    while ((line = sr.ReadLine()) != null)
                    {
                        linelength = line.Length;
                        switch (linelength)
                        {
                            case 5:
                                space = "     ";
                                break;

                        }
                        if (linelength == 5)
                        {
                            IAsyncResult ObjAsynch = objDelegate.Begin
null, null);
                        }
                        else if (linelength == 10)
                        {
                            IAsyncResult ObjAsynch = objDelegate.Begin
null);
                        }

                    }
                }
            }
        }
    }
```

edited Jul 10 '13 at 11:32

answered Jul 10 '13 at 11:15

JAY
79 2

I just wondering why you put this piece of code here. -:) – james Mar 22 at 7:43

4

`string text = File.ReadAllText("Path");` you have all text in one string variable. If you need each line individually you can use this:

```
string[] lines = File.ReadAllLines("Path");
```

answered Mar 29 '13 at 6:34

Dilshod
2,004 2 25 53

4

@Cris sorry .This is quote `MSDN Microsoft`

Methodology

In this experiment, two classes will be compared. The `StreamReader` and the `FileStream` class will be directed to read two files of 10K and 200K in their entirety from the application directory.

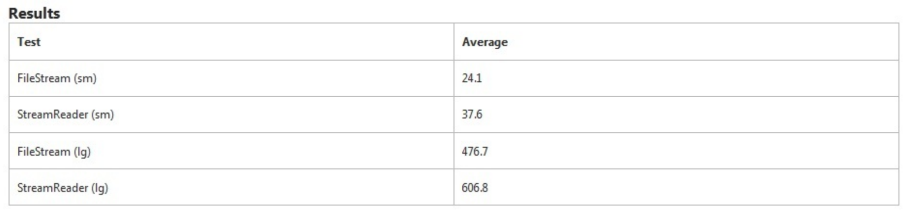
```
StreamReader (VB.NET)

sr = New StreamReader(strFileName)
Do
  line = sr.ReadLine()
Loop Until line Is Nothing
sr.Close()

FileStream (VB.NET)

Dim fs As FileStream
Dim temp As UTF8Encoding = New UTF8Encoding(True)
Dim b(1024) As Byte
fs = File.OpenRead(strFileName)
Do While fs.Read(b, 0, b.Length) > 0
    temp.GetString(b, 0, b.Length)
Loop
fs.Close()
```

Result



Results

Test	Average
FileStream (sm)	24.1
StreamReader (sm)	37.6
FileStream (lg)	476.7
StreamReader (lg)	606.8

`FileStream` is obviously faster in this test. It takes an additional 50% more time for `StreamReader` to read the small file. For the large file, it took an additional 27% of the time.

`StreamReader` is specifically looking for line breaks while `FileStream` does not. This will account for some of the extra time.

Recommendations

Depending on what the application needs to do with a section of data, there may be additional parsing that will require additional processing time. Consider a scenario where a file has columns of data and the rows are `CR/LF` delimited. The `StreamReader` would work down the line of text looking for the `CR/LF`, and then the application would do additional parsing looking for a specific location of data. (Did you think String. SubString comes without a price?)

On the other hand, the `FileStream` reads the data in chunks and a proactive developer could write a little more logic to use the stream to his benefit. If the needed data is in specific positions in the file, this is certainly the way to go as it keeps the memory usage down.

`FileStream` is the better mechanism for speed but will take more logic.

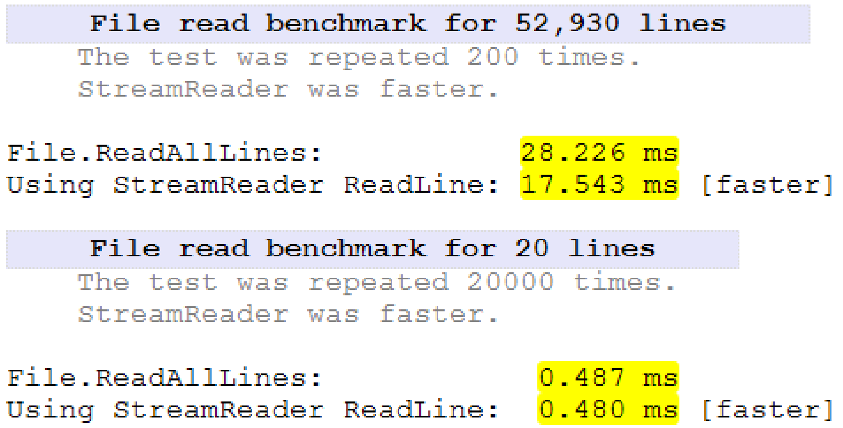
edited Mar 29 '13 at 6:25
Baby Groot
3,979 11 46 66

answered Mar 29 '13 at 5:53
MinhVuong
41 4

But what about `StreamReader.ReadToEnd` ? – Owen Blacker Jun 3 '14 at 11:49

157

A benchmark comparison of `File.ReadAllLines` vs `StreamReader ReadLine` from [C# file handling]



```
     File read benchmark for 52,930 lines
         The test was repeated 200 times.
         StreamReader was faster.

File.ReadAllLines:            28.226 ms
Using StreamReader ReadLine:  17.543 ms  [faster]

     File read benchmark for 20 lines
         The test was repeated 20000 times.
         StreamReader was faster.

File.ReadAllLines:             0.487 ms
Using StreamReader ReadLine:   0.480 ms  [faster]
```

> Results. StreamReader is much faster for large files with 10,000+ lines, but the difference for smaller files is negligible. As always, plan for varying sizes of files, and use File.ReadAllLines only when performance isn't critical.

StreamReader approach

As the `File.ReadAllText` approach has been suggested by others, you can also try the *quicker* (I have not tested quantitatively the performance impact, but it appears to be faster than

`File.ReadAllText` (see **comparison** below)). The difference in performance will be visible only in case of larger files though.

```
string readContents;
using (StreamReader streamReader = new StreamReader(path, Encoding.U
{
     readContents = streamReader.ReadToEnd();
}
```

Comparison of File.Readxxx() vs StreamReader.Readxxx()

Viewing the indicative code through ILSpy I have found the following about `File.ReadAllLines` , `File.ReadAllText` .

- **`File.ReadAllText`** - Uses `StreamReader.ReadToEnd` internally
- **`File.ReadAllLines`** - Also uses `StreamReader.ReadLine` internally with the additionally overhead of creating the `List<string>` to return as the read lines and looping till the end of file.

So both the methods are an *additional layer of convenience* built on top of `StreamReader` . This is evident by the indicative body of the method.

`File.ReadAllText()` implementation as decompiled by ILSpy

```
public static string ReadAllText(string path)
{
    if (path == null)
    {
        throw new ArgumentNullException("path");
    }
    if (path.Length == 0)
    {
        throw new
ArgumentException(Environment.GetResourceString("Argument_EmptyPath"
    }
    return File.InternalReadAllText(path, Encoding.UTF8);
}

private static string InternalReadAllText(string path, Encoding enco
{
    string result;
    using (StreamReader streamReader = new StreamReader(path, encodi
    {
        result = streamReader.ReadToEnd();
    }
    return result;
}
```

edited Sep 12 '11 at 16:25

answered Sep 12 '11 at 11:40

Devendra D. Chavan
7,413 3 26 31

2 Did you compare against `File.ReadAllText` , too ?? – marc_s Sep 12 '11 at 12:13

2 ILSpy suggests that `File.ReadAllText()` is simply a wrapper over `StreamReader.ReadToEnd()` . I am guessing that the additional layer should perform slightly slower than `StreamReader.ReadToEnd()` . – Devendra D. Chavan Sep 12 '11 at 14:46

Great answer. Perhaps a little bit much explanation for those just looking for the fix, but it deserves at least as many votes as the chosen answer. – Sandy Gifford Apr 25 '14 at 15:28

@Devendra D. Chavan: Offtopic, but where can I find reference or documentation for ILSpy? – Viral Jain Jul 22 '14 at 5:25

There are resources on ilspy.net that you might find helpful – Devendra D. Chavan Jul 23 '14 at 18:22

16

```
string contents = System.IO.File.ReadAllText(path)
```

Here's the MSDN documentation

edited Sep 12 '11 at 15:22
Mouna Cheikhna
30.4k 9 42 64

answered Sep 12 '11 at 11:25
Neil Barnwell
29.3k 24 130 202

0

you can read a text from a text file in to string as follows also

```
string str = "";
StreamReader sr = new StreamReader(Application.StartupPath + "\\Samp
while(sr.Peek() != -1)
{
  str = str + sr.ReadLine();
}
```

answered Sep 12 '11 at 11:29
Sai Kalyan Kumar Akshinthala
9,890 7 31 60

6

Take a look at the File.ReadAllText() method

Some important remarks:

> This method opens a file, reads each line of the file, and then adds each line as an element of a string. It then closes the file. A line is defined as a sequence of characters followed by a carriage return ('\r'), a line feed ('\n'), or a carriage return immediately followed by a line feed. The resulting string does not contain the terminating carriage return and/or line feed.
>
> This method attempts to automatically detect the encoding of a file based on the presence of byte order marks. Encoding formats UTF-8 and UTF-32 (both big-endian and little-endian) can be detected.
>
> Use the ReadAllText(String, Encoding) method overload when reading files that might contain imported text, because unrecognized characters may not be read correctly.
>
> The file handle is guaranteed to be closed by this method, even if exceptions are raised

answered Sep 12 '11 at 11:25
sll
49.5k 14 83 133

2

```
string content = System.IO.File.ReadAllText( @"C:\file.txt" );
```

answered Sep 12 '11 at 11:25
Paul Mitchell
2,815 14 20

---

well the quickest way meaning with the least possible C# code is probably this one:

3

```
string readText = System.IO.File.ReadAllText(path);
```

answered Sep 12 '11 at 11:25

Davide Piras
38.4k 8 73 128

3

```
System.IO.StreamReader myFile =
   new System.IO.StreamReader("c:\\test.txt");
string myString = myFile.ReadToEnd();
```

answered Sep 12 '11 at 11:24

Maxim V. Pavlov
5,853 14 51 147
