

Forum:[Hack Forums](#)

Thread: [C# reading a text file](#)

DeadShot32 - September 9, 2019, 9:33 pm

I am looking how to read from a file search for the word then from that word put it in a textbox to a end word
like from date end paragraph

ty

Bunnyslippers69 - September 10, 2019, 12:06 am

You could do something similar to this:

```
Code using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace hfHelp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines("E:/Desktop/passwords.txt");
            foreach (var line in lines)
            {
                if (line.Contains("b"))
                {
                    Console.WriteLine(line);
                }
            }
        }
    }
}
```

vaiperboy - September 10, 2019, 9:42 am

(Sep 10, 2019 12:06 AM)Bunnyslippers69 Wrote: [\[CITE\]](#) You could do something similar to this:

```
Code using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace hfHelp1
```

```

{
    class Program
    {
        static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines("E:/Desktop/passwords.txt");
            foreach (var line in lines)
            {
                if (line.Contains("b"))
                {
                    Console.WriteLine(line);
                }
            }
        }
    }
}

```

well thats not really a good way since it loads all of the lines into the memory, which sometimes can be useless (imagine if the word was found in the first line?) ,slow and being able to crash

also if I understood correctly you want to find where line a contains a certain word and return that line? if so,

@OP, you could load one line at a memory and checking that. This can be done by

```

Code foreach (var line in File.ReadLines("path")) {
    if (line.Contains("word"))
        break;
}

```

or if you feeling linqy

```

Code var lines = File.ReadLines("path")
    .Where(line => line.Contains("word"));

```

but thats just real basic. All depends on how big your file is, if you need faster stuff to handle bigger stuff look at <https://github.com/pdonald/aho-corasick>

FYI, the linq version searches for all occurrences

MavenGradle - September 10, 2019, 12:46 pm

(Sep 10, 2019 12:06 AM)Bunnyslippers69 Wrote: [\[CITE\]](#) You could do something similar to this:

```

Code using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

```

```

namespace hfHelp1
{

```

```

    class Program
    {
        static void Main(string[] args)
        {
            string[] lines = File.ReadAllLines("E:/Desktop/passwords.txt");
            foreach (var line in lines)
            {
                if (line.Contains("b"))

```

```

    {
        Console.WriteLine(line);
    }
}
}
}
}
}

```

No error handling and like vaiperboy pointed out this is very inefficient when dealing with very large files. Same goes for the LINQ method vaiperboy posted. It's just a prettier way to write all that code.

(Sep 10, 2019 09:42 AM)vaiperboy Wrote: [\[CITE\]](#) Code foreach (var line in File.ReadLines("path")) {
 if (line.Contains("word"))
 break;
}

If it's important that these files remain accessible while you're doing your thing, you might not want to use the File.ReadLines method because it will place a full on write block handle on the file. You might be more inclined to use a StreamReader like so:

```

Code var fs = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.ReadWrite);
using (var sr = new StreamReader(fs))
{
    // etc...
}

```

riu41776 - September 11, 2019, 11:48 am

I suggest you to skip the comments above me as they're suggesting you primitive ways to search for a substring.

If you want to go professional, use regular expressions.

DeadShot32 - September 14, 2019, 8:42 pm

Thank you much :) for your help

riu41776 - September 15, 2019, 2:37 am

I'm LMFAO'ing so hard from the usage of ".contains", you serious?

@OP use Regular expressions.

n0s1leep - September 20, 2019, 2:43 pm

string.Contains is perfectly fine for this use case. Use of regexp is overkill, a professional programmer knows when to use the right tool for the right job.

0xDEAD10CC - September 21, 2019, 6:43 pm

(Sep 20, 2019 02:43 PM)n0s1leep Wrote: [\[CITE\]](#) string.Contains is perfectly fine for this use case. Use of regexp is overkill, a professional programmer knows when to use the right tool for the right job.

The above user has no clue what he's talking about so just ignore his idea to use regex. The guy can't even build a regex pattern to match duplicate words.

Use IndexOf and specify the appropriate string comparison enum value.

max_int - October 3, 2019, 3:02 pm

```

Code using System;
using System.IO;

```

```

namespace MyNullSpace
{
    class MyTextFinder
    {
        public static void IsStringInText(string fileName, string someText)
        {
            bool found = false;
            //the StreamReader class does not lock the file
            //-- so other processes can still access file-
            using (StreamReader sr = new StreamReader(fileName))
            {
                while (!sr.EndOfStream)
                {
                    string line = sr.ReadLine();
                    //for more complex match
                    //change the lines below to use regex
                    var comparer = StringComparer.InvariantCultureIgnoreCase;
                    if(line.Contains(someText, comparer))
                    {
                        found=true;
                        break;
                    }
                }
            }
            return found;
        }
    }
}

```

This may help

vaiperboy - October 4, 2019, 6:53 am

(Oct 3, 2019 03:02 PM)max_int Wrote: [\[CITE\]](#) Code using System;
using System.IO;

```

namespace MyNullSpace
{
    class MyTextFinder
    {
        public static void IsStringInText(string fileName, string someText)
        {
            bool found = false;
            //the StreamReader class does not lock the file
            //-- so other processes can still access file-
            using (StreamReader sr = new StreamReader(fileName))
            {
                while (!sr.EndOfStream)
                {
                    string line = sr.ReadLine();
                    //for more complex match
                    //change the lines below to use regex
                    var comparer = StringComparer.InvariantCultureIgnoreCase;
                    if(line.Contains(someText, comparer))
                    {

```

```
    found=true;
    break;
}
}
}
return found;
}
}
```

}This may help
I stopped reading when i saw found=true

cleaning.

The code i posted is from a bigger class where i face the both cases, that's why i'm not exiting in the loop.

Cheers

MavenGradle - October 4, 2019, 8:58 am

(Oct 4, 2019 07:45 AM)max_int Wrote: [\[CITE\]](#) I agree with the idea of returning immediately when the string is found.

Anyway, in some cases, it may not be the best option

1. you need to do some computations inside this function before returning.
2. In case you use an unmanaged resource inside the loop, it may be very bad to exit without cleaning.

The code i posted is from a bigger class where i face the both cases, that's why i'm not exiting in the loop.

Cheers

You have no reason not to return immediately inside the function that you shared with us, which by the way will not compile due to an incorrect return value and because the `string.Contains` method has [NO OVERLOAD for a StringComparer](#). You're using a [using](#) statement which means the `StreamReader` will be disposed of correctly no matter how you look at it. If an exception is thrown at any point (including evaluating the return value) `Dispose` will still be called too. Even if this is part of a larger code base, what you're doing is still incredibly bad practice because it adds more steps in the chain which are simply unnecessary.

I agree that any resources should be disposed of properly because the .NET GC is not perfect, has been known to let certain things linger in memory etc. In such cases, you'd be better off using a `using` statement inside a `using` statement to make sure that the `dispose` method will be called correctly or dispose of it yourself in the right manner by calling the `IDisposable.Dispose()` method.