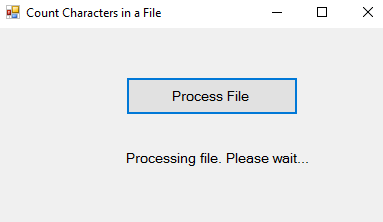
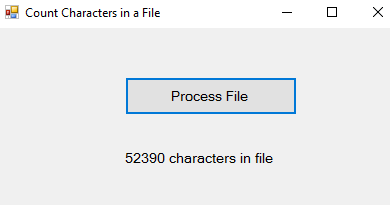
**Async and await in c# example**

Let us create a simple Windows Forms Application that counts the number of characters in a given file. Let us assume the file is very big and it takes around 5 seconds to read and count the number of characters in the file. When the **"Process File Button"** is clicked, the application should display the message **"Processing File. Please wait"**.   
  
  
  
  
  
  
  
As soon as the application finishes processing the file it should display the the number of characters as shown below.  
  
  
  
Another improtant requirement is that the application should remain responsive throughout the entire process, i.e when the application is busy processing the file the application should not hang and we should still be able to interact with the application. We should be able to click with in the other controls on the form, move the form around on the screen, resize it if required etc.  
  
Let us first create the Windows Forms Application without using async and await keywords and see how it behaves. Here are the steps.  
  
1. In your C: drive, create a new folder. Name it **Data**. In the folder create a new Text Document. Name it **Data.txt**. Type some text in the file and save it. The application that we are going to create, counts the number of characters in this file.  
  
2. Create a New **"Windows Forms Application"**. Name it AsyncExample.  
  
3. Drag and Drop a **"Button"** on the Form and set the following properties  
   Name = btnProcessFIle  
   Font - Size = 10  
   Text = Process File  
  
4. Drag and Drop a **"Label"** on the Form and set the following properties  
   Name = lblCount  
   Font - Size = 10  
   Text = ""  
  
5. Double Click on the **"Button"** control to generate the "Click" event handler  
  
6. Copy and paste the following code in Form1.cs

using System;

using System.IO;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace AsyncExample

{

    public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        private int CountCharacters()

        {

            int count = 0;

            // Create a StreamReader and point it to the file to read

            using (StreamReader reader = new StreamReader("C \\Data.txt"))

            {

                string content = reader.ReadToEnd();

                count = content.Length;

                // Make the program look busy for 5 seconds

                Thread.Sleep(5000);

            }

            return count;

        }

        private void btnProcessFIle\_Click(object sender, EventArgs e)

        {

            lblCount.Text = "Processing file. Please wait...";

            int count = CountCharacters();

            lblCount.Text = count.ToString() + " characters in file";

        }

    }

}

7. Run the application and click the **"Process File"** button. You will notice the following problems.

* The application does not display the status message i.e "Processing file. Please wait."
* While the application is busy processing the file, it becomes unresponsive. You cannot move the form window or resize it.

These problems can be very easily fixed by using the **async** and **await**keywords. Notice only the **btnProcessFIle\_Click**() event handler method needs to change.

// Make the method async by using the async keyword

private async void btnProcessFIle\_Click(object sender, EventArgs e)

{

    // Create a task to execute CountCharacters() function

    // CountCharacters() function returns int, so we created Task<int>

    Task<int> task = new Task<int>(CountCharacters);

    task.Start();

    lblCount.Text = "Processing file. Please wait...";

    // Wait until the long running task completes

    int count = await task;

    lblCount.Text = count.ToString() + " characters in file";

}

Now, when we click the **"Process File"** button, notice

* The application displays the status message (**"Processing file. Please wait"**) immediately.
* Even when the application is busy processing the file, it is responsive. You can move the form window around or resize it.

**So what is the use of async and await keywords in C#**  
async and await keywords are used to create asynchronous methods. The async keyword specifies that a method is an asynchronous method and the await keyword specifies a suspension point. The await operator signalls that the async method can't continue past that point until the awaited asynchronous process is complete. In the meantime, control returns to the caller of the async method.  
  
An async method typically contains one or more occurrences of an await operator, but the absence of await expressions doesn’t cause a compiler error.  
  
**You may have a few questions at this point.**  
1. Can't we achieve the same thing using a Thread.   
2. What is the difference between a Thread and a Task  
3. When to use a Task over Thread and vice-versa