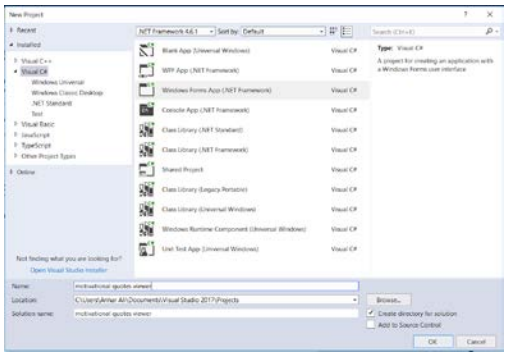
C# Tutorial – Create a Motivational Quotes Viewer Application in Visual Studio

In this tutorial we will create a fun little application for Microsoft Windows using Visual Studio. You can use any version of visual studio to follow along with this tutorial. We are creating a Motivational Quotes viewer. The logic behind this application is that we want to open a TEXT file, load it to the program and populate a list item with the contents of the text file and then when we select an item from the list viewer we want it to be presented nicely on the screen with custom background (the UI) and font.

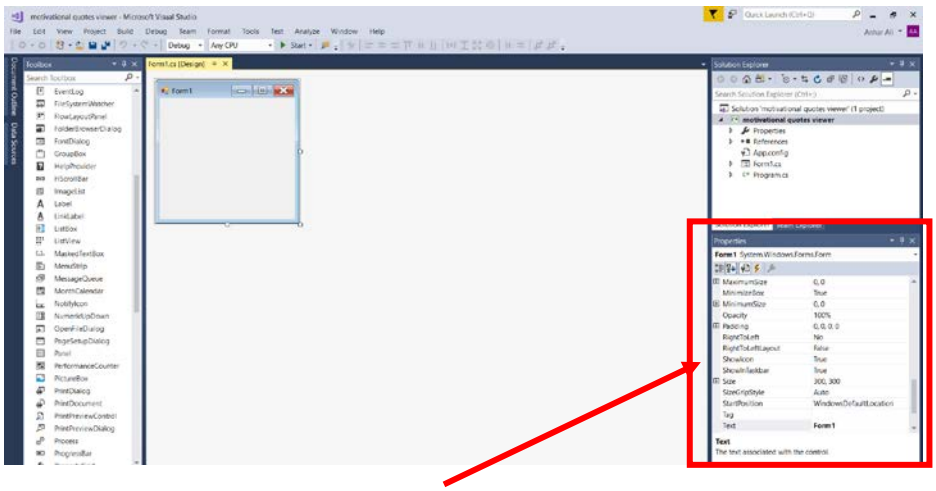
Lesson Objectives –

1. Create a fully working application in Visual Studio
2. Load text files in C# and windows form
3. Use an open dialog option
4. Use custom font for the program
5. Load custom font and background image to application
6. Debug the application to see if it meets the requirement

To start we will be using visual studio 2017. Start a new C# windows form application project and name it motivational quotes viewer



Click OK to get started



This is the empty form. In the **properties** window change the following options

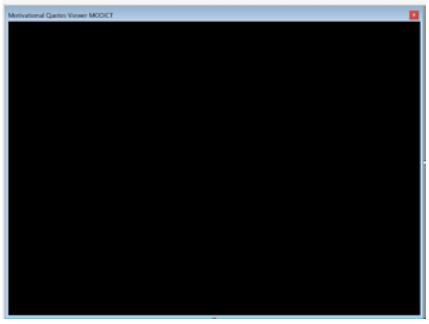
Back Colour – Black

Form Border Style – Fixed Tool Window

Size - 977, 734

Text - Motivational Quotes Viewer MOOICT

Result

 ← this window will not be resized and only can be closed. This is what we wanted.

Now lets go to the downloaded assets for this application.

We have a background image and font file in the assets folder. If you haven't downloaded the asset then click on the download link above in the web page or go to mooict.com and download it from there.

(Here we have the Gabrielle Font file and the UI gif image file)

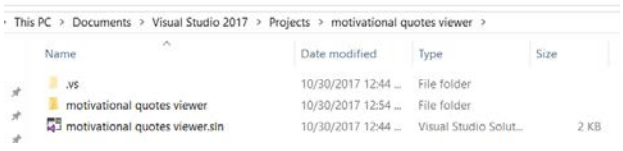
The font file will be moved to the debug folder and the UI image will be loaded in the resources for this application. Let find out how to do these things.

Moving the font file –

First run the application

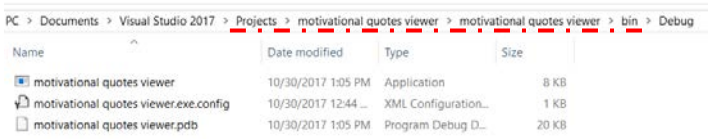
Click on the green image. This will run the empty form in visual studio what this will also do is create the debug location for the files lets take a look.

If you followed the instructions so far then you have created the project called motivational quotes viewer so if you go to the visual studio folder in your my documents then go to projects it should be there as followed



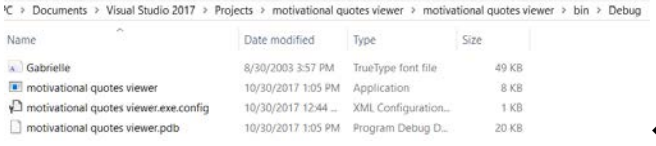
← this is the project folder

Now go in the **Projects\motivational quotes viewer\motivational quotes viewer\bin\Debug**



← this is the debug folder in the project folder

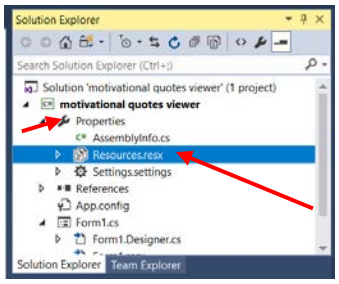
Copy and paste the FONT file in this folder.



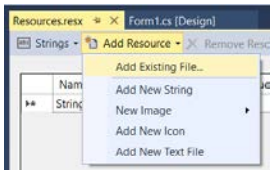
← the font is placed inside the folder.

Go back in visual studio,

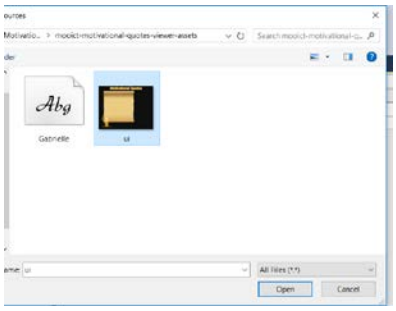
Under the properties – double click on the resources icon in the solutions explorer



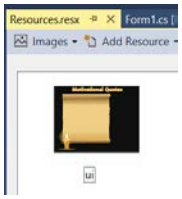
← In the solutions explorer, under the properties menu double click on the Resources.resx file. This will open the resources panel.



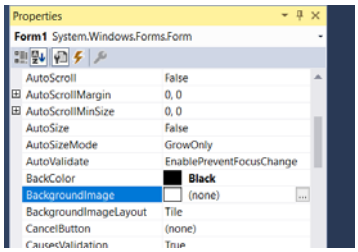
← from the Add Resource drop down menu click on Add Existing File.



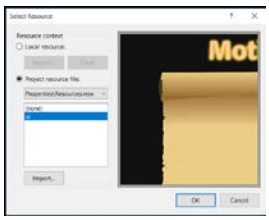
← Find the UI image you saved from MOOICT and click on OPEN.



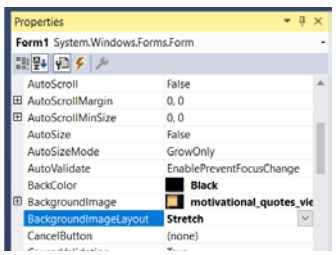
← UI image will be added to the application resources.



← Click on the form and in the properties option look under the background image option, click on the three dotted button …



← you will see the project resource file we added earlier in this window. Select the ui image and click OK.



← back in the form properties options, go to the background image layout option and change it to Stretch.

Adding the components, now we will start adding the necessary components to the application.

First the panel


Drag and drop a panel to the form. While the panel is selected then change the following to its properties window

| | |
|---|---|
| Name | textView |
| Back Colour | Transparent |
| Location | 105, 163 |
| Size | 500, 338 |

Result


← this is what the final panel looks like in the application. We are using a panel because we don't want the writing to get out of the scroll image so it will look like it was written in the scroll.

Label

We need two labels for this application.

Lets add the first one directly into the panel

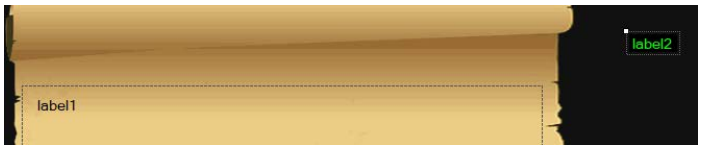

Position label 1 in the panel as seen below


← Place the label on the top left of the panel. If the label is placed on top of the panel then it becomes a CHILD of the panel and it will be grouped together with it. For rest of the components which will be added to the form make sure not to drop on top of the panel because it don't want any other components to be grouped with the panel.

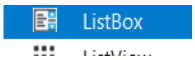
Drag and drop a second label on the screen but make sure you drop it outside the panel from label 1.

Change the following in its properties for label 2, it might be hard to see it on the screen because of the background colour and foreground colour being the same. We need to change them to see it more clearly. Change the following in the label 2 property options

| | |
|---|---|
| Fore Colour | LIME |
| Location | 688, 114 |

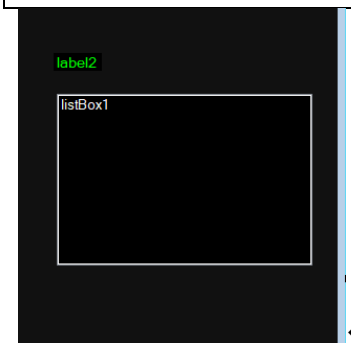



Now drag and drop a list box to the form




Make the following change to it in the properties window

| | |
|---|---|
| Back Colour | Black |
| Fore Colour | White |
| Location | 691, 153 |
| Size | 244, 164 |



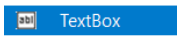
← After the changes this what the list box now looks like.

Now drag and drop a button to the screen


Change the following in its properties

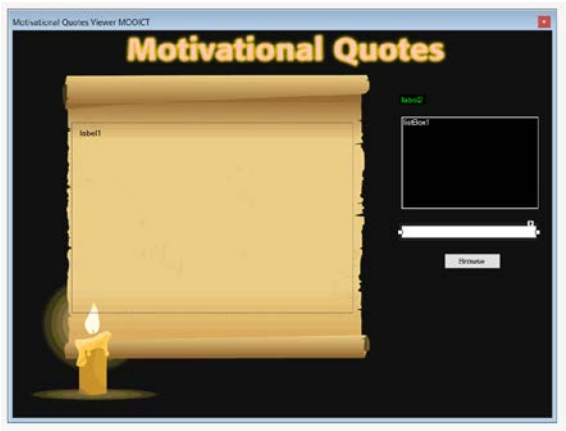
| | |
|---|---|
| Size | 100, 28 |
| Text | Browse |
| Location | 781, 378 |

Now drag and drop a text box to the form and change the following in its properties

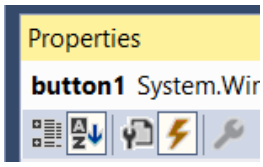



| | |
|---|---|
| Location | 691, 346 |
| Size | 240, 22 |

Final result



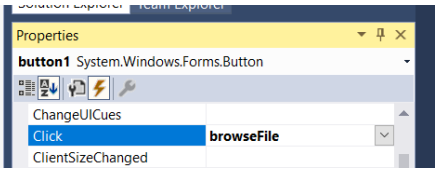
← as you can see we have the text box and the button added to the application. You can run this program now to see the final result on the UI. However we still need to add the events. Events window can be found in the properties window.



← see that little lightning bolt icon, that represents the events for a component in visual studio. Select a component such a button and click on that little icon it will show you many different types of

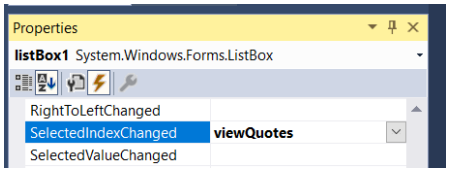
Adding the events

← select the button



← in the events window find the CLICK option and type browseFile and press enter. This will take you to the code view but come back to the design view because have another event to add.

← select the list box from the form



← in the event window find the Selected Index Changed option and type **viewQuotes** and press enter.

Below is the code view so far. - [This is the default view with the events we just added to earlier]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace motivational_quotes_viewer
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void browseFile(object sender, EventArgs e)
        {

        }

        private void viewQuotes(object sender, EventArgs e)
        {

        }
```

```
    }
}
```

We have two events so far in the code but the program doesn't do much at the moment. So we need to add our functionalities in the code. First we highlighted the crucial things we need below. Add them in the appropriate places.

Add the following code (yellow highlighted ones)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.IO; // add this one
using System.Drawing.Text; // add this one

namespace motivational_quotes_viewer
{
    public partial class Form1 : Form
    {

        int totalLines; // this will be used see how many lines have we loaded

        PrivateFontCollection pfc = new PrivateFontCollection(); // to use with the custom font

        public Form1()
        {
            InitializeComponent();

            pfc.AddFontFile("Gabrielle.ttf");// load the font file to the program
        }
        private void browseFile(object sender, EventArgs e)
        {

        }

        private void viewQuotes(object sender, EventArgs e)
        {

        }

        private void readFile()
        {

        }
    }
}
```

So lets discuss what is happening here so far –

```
using System.IO; // add this one
```

```
using System.Drawing.Text; // add this one
```

The two lines above is adding external class which we will need to make this program work. System.IO class will be using read the text file we import into this program.

System.Drawing.Text class will be used to assign the custom font to the label. It's the best idea to comment these lines in the code it makes it easier to identify them.

```
int totalLines; // this will be used see how many lines have we loaded
```

The line above is an integer, this integer is a global variable since we are declaring it outside of the functions it can be amended later on in the program.

```
PrivateFontCollection pfc = new PrivateFontCollection(); // to use with the custom font
```

The line above is a class from the System.Drawing.Text. This is creating a new instance of private font collection class, we can use this to load the custom font to the program. If you didn't import the System.Drawing.Text then we cannot use the private font collection class.

```
pfc.AddFontFile("Gabrielle.ttf");// load the font file to the program
```

The line above needs to be inserted in the Form1() function. This function runs when the form is loaded. So we want the font to be loaded with the form. We are using pfc class we created earlier and loading the Gabrieielle font with it.

```
        private void readFile()
        {


        }
```

Above is a custom function. This function we typed up ourselves, this is not an event. We will write our code to read a text file and load it in the list box in this function.

All of the code will be commented so you won't be lost. If you feel its getting overwhelming then take it a small byte at a time.

**Browse File Function**

Below is the browse file event. This event will trigger when the browse button is click. The main purpose of this function is to open a file open dialog box, user will select a text file and click ok. Once the file is selected and user click ok we will then run the read file function to read and load the text file to the program.

All of the codes are commented so we didn't explain them separately you can check what each line is doing in the comments -

```
        private void browseFile(object sender, EventArgs e)
        {
            // creating a new instance of the OPEN FILE DIALOG
            OpenFileDialog openFileDialog1 = new OpenFileDialog();

            //set the default location for the dialog box to main hard drive
            openFileDialog1.InitialDirectory = @"C:\";
            // set the title of the dialog box
            openFileDialog1.Title = "Browse Text Files";

            //The file and path both need to exist in the system
            openFileDialog1.CheckFileExists = true;
            openFileDialog1.CheckPathExists = true;

            // setting the default file type
            // we only want the program to search for text files
            openFileDialog1.DefaultExt = "txt";
            openFileDialog1.Filter = "Text files (*.txt)|*.txt";
            openFileDialog1.FilterIndex = 2;
            openFileDialog1.RestoreDirectory = true;

            // we want to open the file as read only
            openFileDialog1.ReadOnlyChecked = true;
            openFileDialog1.ShowReadOnly = true;
```

```
            // if the user has selected a file
            // and they pressed the OK button

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                // we pupulate the text box 1 with the file path and name
                textBox1.Text = openFileDialog1.FileName;
                // we clear the list box of any info
                listBox1.Items.Clear();
                // we run the read file function to read the text file
                // and populate the list box with the new info
                readFile();
            }
        }
```

**View Quotes Function**

This function will trigger when the items in the list box is selected. The main purpose of this function is to link the list box to the label 1 which will be used to show the quotes on screen.

All the codes are commented in the codes to help you better understand them.

```
        private void viewQuotes(object sender, EventArgs e)
        {
            // set the label 1 maximum width to the text view panel
            label1.MaximumSize = new Size(textView.Width + 5, 0);

            // auto size will keep the text from over lapping the panel
            // it will automatically re-adjust the width of the label to
            // fit the text view panel
            label1.AutoSize = true;

            // set the font with the new font, size to 18 and style to BOLD for label 1
            label1.Font = new Font(pfc.Families[0], 18, FontStyle.Bold);

            // show the selected item on label 1 from the list box
            label1.Text = listBox1.SelectedItem.ToString();
        }
```

**Read file function –**

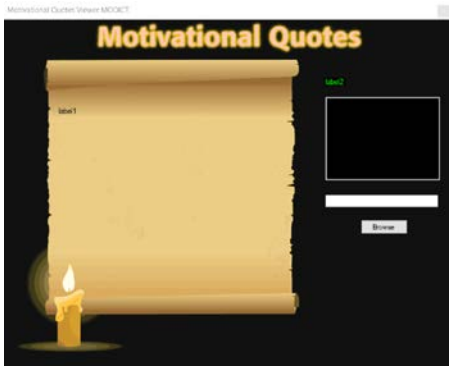
This is the custom function we created in the code. The purpose of this function is load the text file into the program, find the contents of the text file (the texts duuh) and load them into an array by separating the lines, and then it will add all of the text to the list box. This function will also round up the number of lines in a text file and show that result in the label 2.

All of the codes are commented below to help you understand each line better -

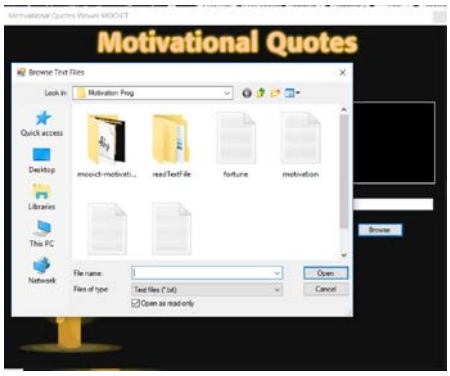
```
        private void readFile()
        {
            string[] lines; // create a string array called lines
            var list = new List<string>(); // creating a list variable

            // below is a file stream variable which will read the text file loaded from
            // the browse button
            // the text box contains the location of the file
            // the file will open in read only mode
            var fileStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);


            // below is the using loop we are creating a stream reader
            // while the conditions match this loop will run till the file has ended
            using (var streamReader = new StreamReader(fileStream, Encoding.UTF8))
            {
                string line; // create a string called line
                // while the line is not empty
                while ((line = streamReader.ReadLine()) != null)
                {
                    // add the line to the list
                    list.Add(line);
                    // add one to the total lines found in the document
                    totalLines++;
                }
```

```
            }
            lines = list.ToArray(); // convert the lines into the array

            // below is the for loop that will add the lines in the list box
            // we run the loop for the length of lines available in the text document

            for (int i = 0; i < lines.Length; i++)
            {
                // below are the items being added to the list box 1
                listBox1.Items.Add(lines[i]);
            }

            // label 2 will show how many quotes have been loaded to the program
            label2.Text = "Total Lines Loaded: " + totalLines;
        }
```

Final Result

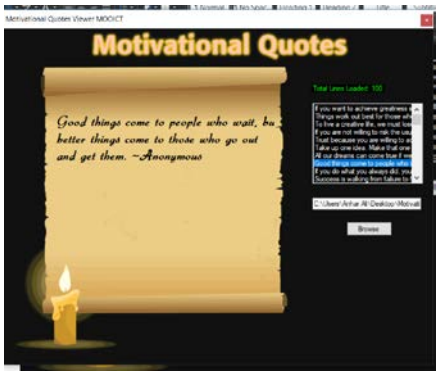



← This is the program when it runs. I clicked on the debug button to run the program is visual studio. Next click on the browse button.




← This open file dialog box comes up, here I am only able to see the text files in the folders. I will select the motivation text file and click OPEN.


← now the list box has been populated by the quotes from the motivation text file and the green lines are showing 100 lines been loaded to the program. That's a whole lot of motivation right there. Notice that we have file path also shown in the text box.



← now by selecting one of the lines from the box is showing the motivational quotes inside the scroll with our custom font. Excellent work everything is working as planned.

If you managed to follow us thus far then very well done and one more done project for the MOO team.

Try out another or make some changes to this to see what YOU can make of it.

**Full Source Code**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.IO; // add this one
using System.Drawing.Text; // add this one

namespace motivational_quotes_viewer
{
    public partial class Form1 : Form
    {
        int totalLines; // this will be used see how many lines have we loaded
        PrivateFontCollection pfc = new PrivateFontCollection(); // to use with the custom font

        public Form1()
        {
            InitializeComponent();
            pfc.AddFontFile("Gabrielle.ttf");// load the font file to the program
        }

        private void browseFile(object sender, EventArgs e)
        {
            // creating a new instance of the OPEN FILE DIALOG
            OpenFileDialog openFileDialog1 = new OpenFileDialog();

            //set the default location for the dialog box to main hard drive
            openFileDialog1.InitialDirectory = @"C:\";
            // set the title of the dialog box
            openFileDialog1.Title = "Browse Text Files";

            //The file and path both need to exist in the system
            openFileDialog1.CheckFileExists = true;
            openFileDialog1.CheckPathExists = true;

            // setting the default file type
            // we only want the program to search for text files
            openFileDialog1.DefaultExt = "txt";
            openFileDialog1.Filter = "Text files (*.txt)|*.txt";
            openFileDialog1.FilterIndex = 2;
            openFileDialog1.RestoreDirectory = true;

            // we want to open the file as read only
            openFileDialog1.ReadOnlyChecked = true;
            openFileDialog1.ShowReadOnly = true;

            // if the user has selected a file
```

```
            // and they pressed the OK button

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                // we pupulate the text box 1 with the file path and name
                textBox1.Text = openFileDialog1.FileName;
                // we clear the list box of any info
                listBox1.Items.Clear();
                // we run the read file function to read the text file
                // and populate the list box with the new info
                readFile();
            }

        }

        private void viewQuotes(object sender, EventArgs e)
        {
            // set the label 1 maximum width to the text view panel
            label1.MaximumSize = new Size(textView.Width + 5, 0);

            // auto size will keep the text from over lapping the panel
            // it will automatically re-adjust the width of the label to
            // fit the text view panel
            label1.AutoSize = true;

            // set the font with the new font, size to 18 and style to BOLD for label 1
            label1.Font = new Font(pfc.Families[0], 18, FontStyle.Bold);

            // show the selected item on label 1 from the list box
            label1.Text = listBox1.SelectedItem.ToString();
        }

        private void readFile()
        {
            string[] lines; // create a string array called lines
            var list = new List<string>(); // creating a list variable

            // below is a file stream variable which will read the text file loaded from
            // the browse button
            // the text box contains the location of the file
            // the file will open in read only mode
            var fileStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);


            // below is the using loop we are creating a stream reader
            // while the conditions match this loop will run till the file has ended
            using (var streamReader = new StreamReader(fileStream, Encoding.UTF8))
            {
                string line; // create a string called line
                // while the line is not empty
                while ((line = streamReader.ReadLine()) != null)
                {
                    // add the line to the list
                    list.Add(line);
                    // add one to the total lines found in the document
                    totalLines++;
                }
            }

            lines = list.ToArray(); // convert the lines into the array

            // below is the for loop that will add the lines in the list box
            // we run the loop for the length of lines available in the text document

            for (int i = 0; i < lines.Length; i++)
            {
                // below are the items being added to the list box 1
                listBox1.Items.Add(lines[i]);
            }
            // label 2 will show how many quotes have been loaded to the program
            label2.Text = "Total Lines Loaded: " + totalLines;
        }
    }
}
```