



GRAND CANYON
UNIVERSITY™

CST-150 Activity 2 Guide

Contents

Part 1	2
Calculate the Minutes, Hours, and Days from the Number of Seconds	2
Overview	2
Execution	2
1. Start a New Visual Studio Project	2
2. Verify the application is working correctly.	6
3. Configure the Form Layout (FrontEnd Design).	8
4. Configure the Event Handler.	10
5. Start coding the backend.	11
6. Submit the Activity as described in the digital classroom.	16

Part 1

Calculate the Minutes, Hours, and Days from the Number of Seconds

Overview

Write a Windows Forms Application that prompts the user to enter the number of seconds (number of seconds is provided by the instructor and must be used to get full credit for this activity).

The program that is designed will look similar to Figure 1.

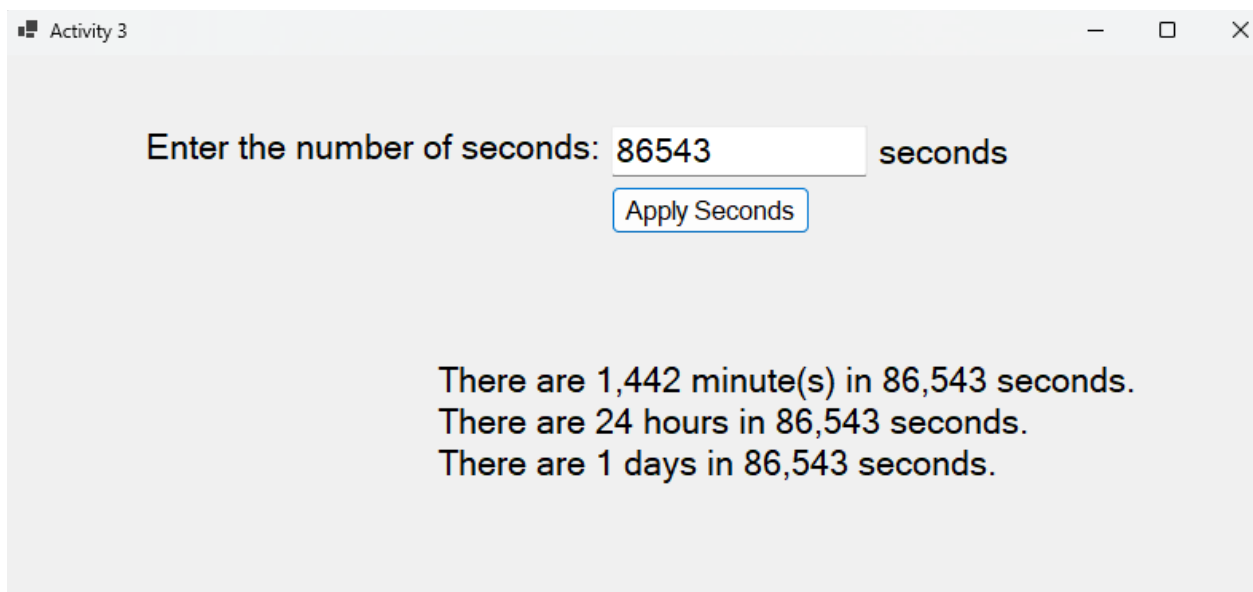


Figure 1: Final state of this activity's program.

Execution

Execute this assignment according to the following guidelines:

1. Start a new Visual Studio Project.
 - a. Draw a flowchart to visually describe the flow of the program. The flowchart is the blueprint to design the program therefore, flowcharting is the first step in designing any program. The flowchart must be created using a flowchart design software as directed by your instructor. Take a screenshot of the flowchart and provide it as one of the deliverables.
 - b. Start Visual Studio and select "Create a new project" as is shown in Figure 2.

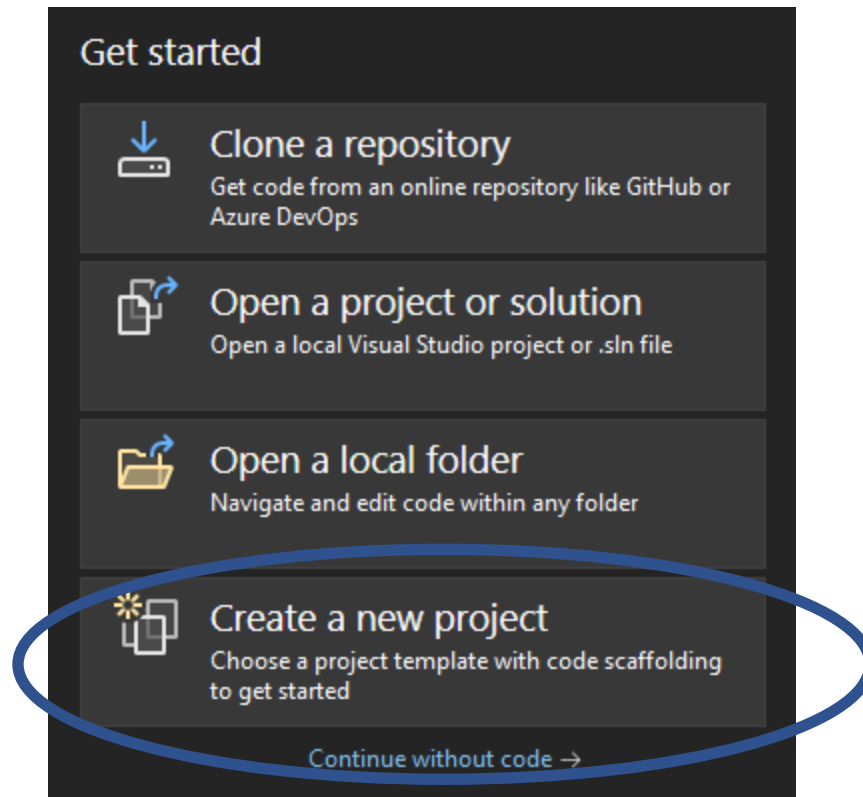


Figure 2: Select "Create a new project."

- 9. Select the Project Template "WindowsFormsApp" as shown in Figure 3. Be sure to select "Windows Forms App." DO NOT select the one with (.NET Framework).

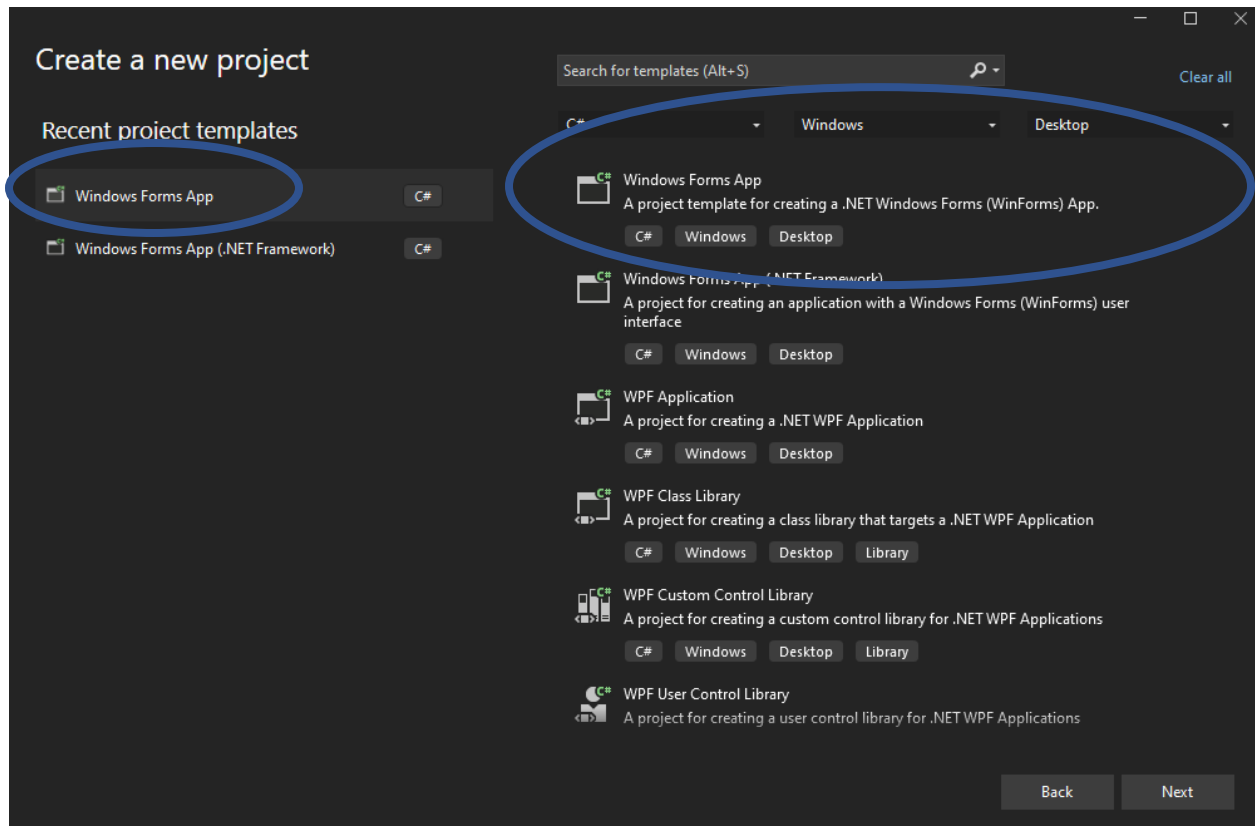
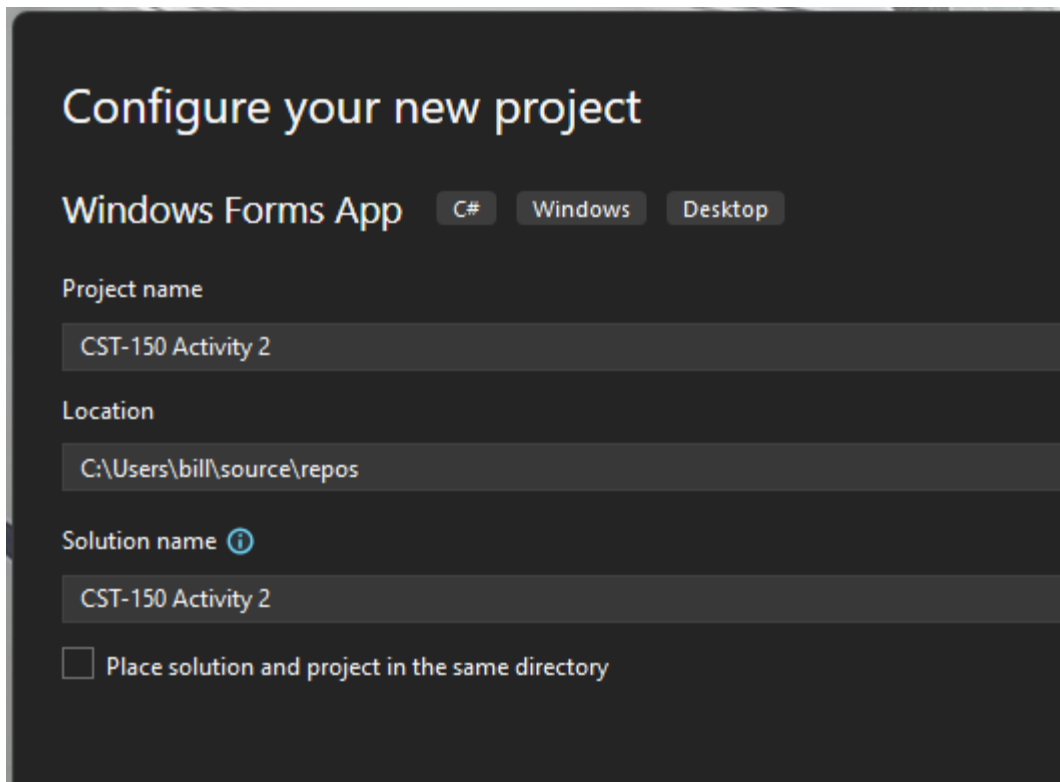


Figure 3: Select the Project Template.

- Enter the Project name: "CST-150 Activity 2" as shown in Figure 4 followed by the "Next" button. Be sure the check box is **not** checked for "Place solution and project in the same directory."



Configure your new project

Windows Forms App C# Windows Desktop

Project name

CST-150 Activity 2

Location

C:\Users\bill\source\repos

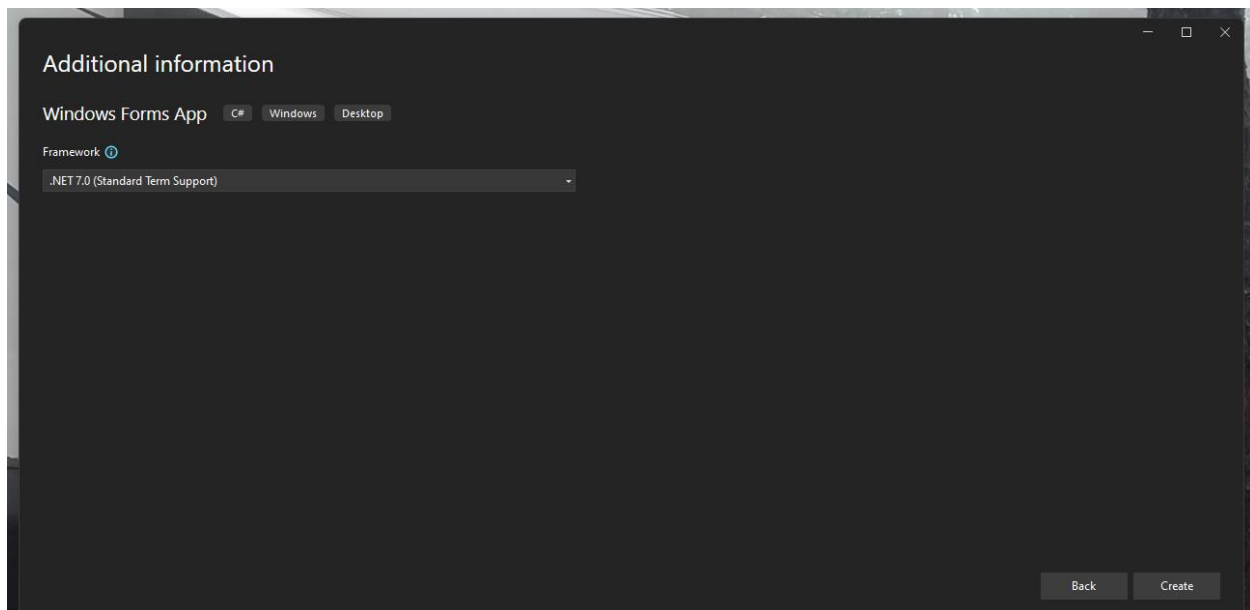
Solution name ⓘ

CST-150 Activity 2

☐ Place solution and project in the same directory

Figure 4: Enter Project Name.

4. Select the Framework as is shown in Figure 5 followed by the "Create" button. These in-class applications have all been tested using .NET 7.0. (The instructor may have the class use a different .NET version.)



Additional information

Windows Forms App C# Windows Desktop

Framework ⓘ

.NET 7.0 (Standard Term Support)

Back Create

Figure 5: Select Framework.

2. Verify the application is working correctly.

In some cases, the designer file will not recognize controls placed on the form. The controls will not be present when the application is run and will require the form tools to be replaced on the form. Follow these steps to confirm the application is functioning correctly.

- Step 1: Change the name of Form1.cs to FrmTest.cs as is shown in Figure 6.

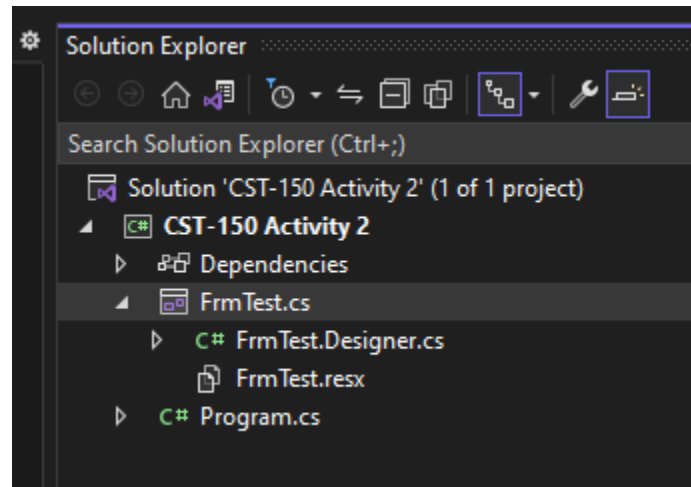


Figure 6: Solution Explorer.

- Step 2: Requires the form to be saved and then close just the FrmTest.cs[Designer] file.
- Step 3: Open the FrmTest.cs[Designer] file back up.
- Step 4: Place a button on the form from the toolbox as is shown in Figure 7.

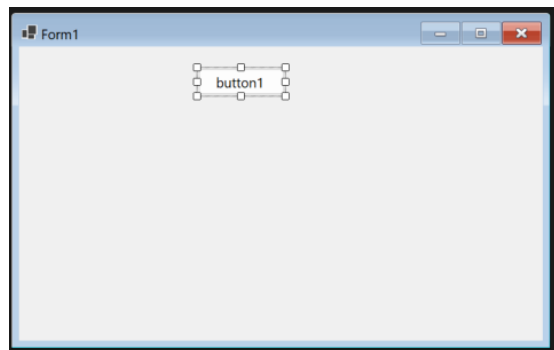


Figure 7: Place a Button control on the Form.

- Step 5: Open FrmTest.Designer.cs file and verify if the button is recognized. Figure 8 shows a Designer file that did not recognize the button, or the form name change as the form name is still the default "Form1." Figure 9 shows the designer file that correctly recognized the control "button1" and form name "FrmTest."

```

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support -
/// the contents of this method with the co
/// </summary>
1 reference
private void InitializeComponent()
{
    this.components = new System.ComponentMo
    this.AutoScaleMode = System.Windows.Form
    this.ClientSize = new System.Drawing.Siz
    this.Text = "Form1";
}

#endregion

```

Figure 8: Designer that did not recognize the button or new form name.

```

private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(229, 91);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    //
    // FrmTest
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(800, 450);
    this.Controls.Add(this.button1);
    this.Name = "FrmTest";
    this.Text = "Form1";
    this.ResumeLayout(false);
}

#endregion

private Button button1;

```

Figure 9: Designer file that recognized the button and FrmTest.

3. If the FrmTest.Designer.cs file is correct, the application is ready. If the button control is not present as shown in Figure , start back at Step 2 and repeat the process.
4. Configure the Form Layout (FrontEnd Design).
 - a. Rename the form name to "FrmSeconds.cs."
 - b. Add a button control, three labels, and one textbox on the form as shown in Figure 10.

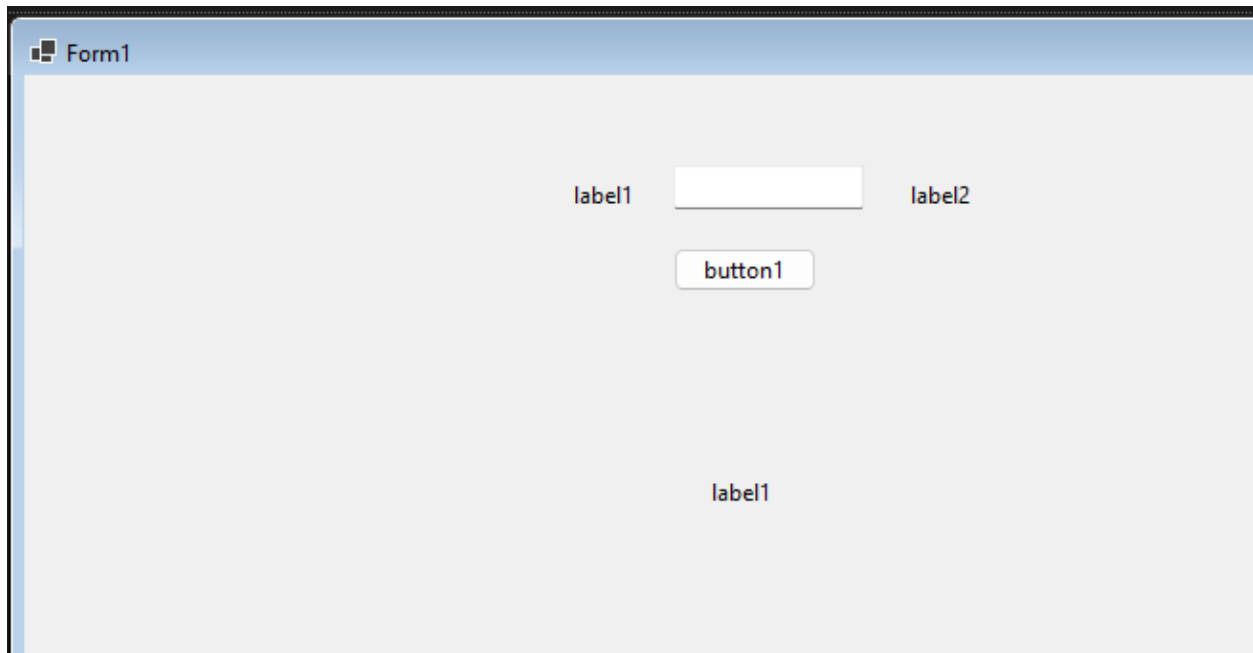


Figure 10: Configure the Form Layout

- e. In the label property text for the first label, "Enter the Number of Seconds:" as shown in Figure 11.
- d. Make the Font size: "16pt" and Font name: "Microsoft Sans Serif."
- e. Configure the textbox font property to 16pt size and use "Microsoft Sans Serif."
- f. Since we will need to have access to this textbox from the code behind, name it "txtUserEntry" using camelCasing.
- g. Change the property width to "161."
- h. Then, place it just to the right of the label we just configured.
- i. Configure units label text property to "seconds."
- j. Update the font size to "16pt" and font name to "Microsoft Sans Serif."
- k. Then, place it just to right of the txtbox we just configured.
- l. Configure the button text property to "Apply Seconds."
- m. Update the font size to "12pt" and font name to "Microsoft Sans Serif."
- n. Set the update property to True.
- o. Name the button "btnRun."
- p. Then, place it just below the txtbox we just configured.
- q. Configure the last label.

- Update the font size to "16pt" and font name to "Microsoft Sans Serif."
- Name the label "lblResults" using camelCasing.
- The last step is to update the form text to "Activity 3."

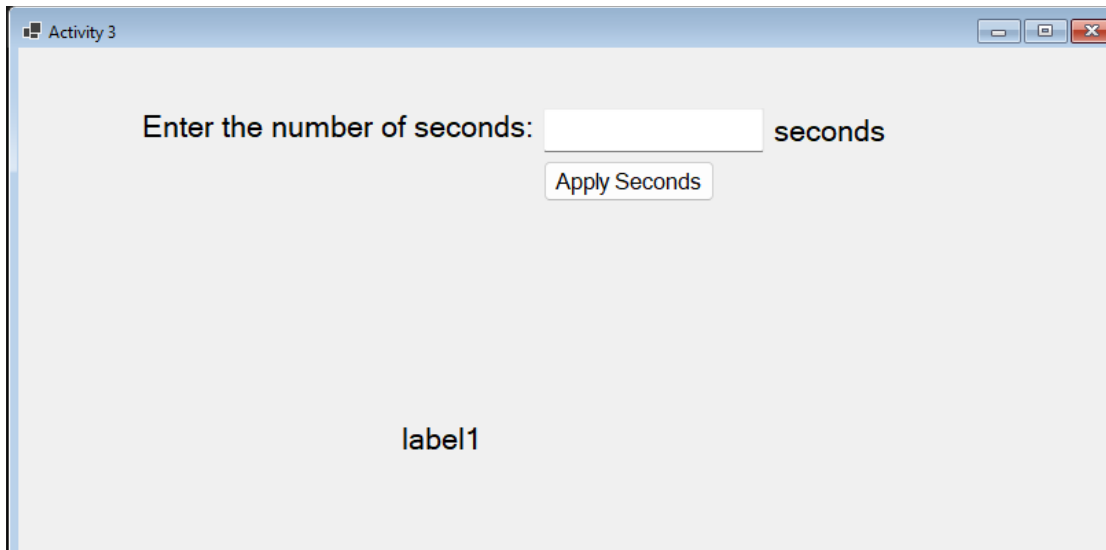


Figure 11: Configure the Form

- Configure the Event Handler.
 - Continue in the form and be sure the button is selected. Create a click event handler for the start button using the Events button and name the Click "ManageSecondsEventHandler" as shown in Figure 12. Be sure we are using PascalCasing for all method names.

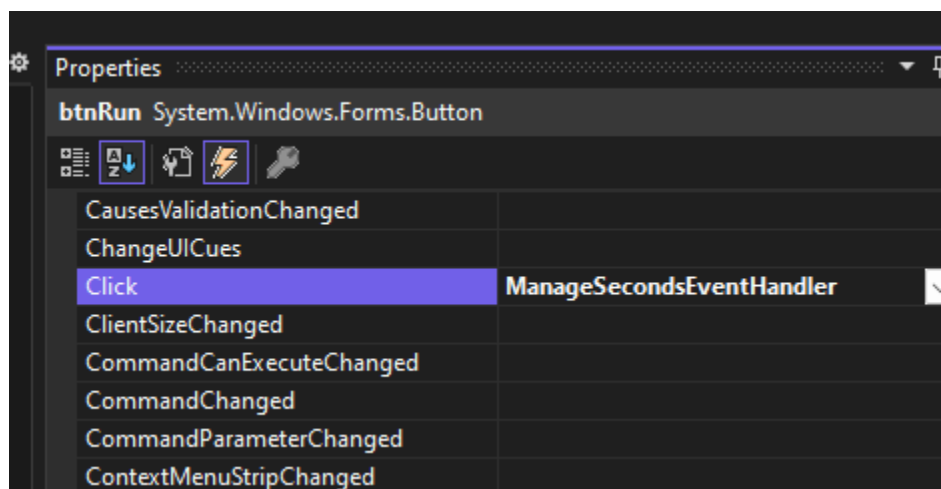
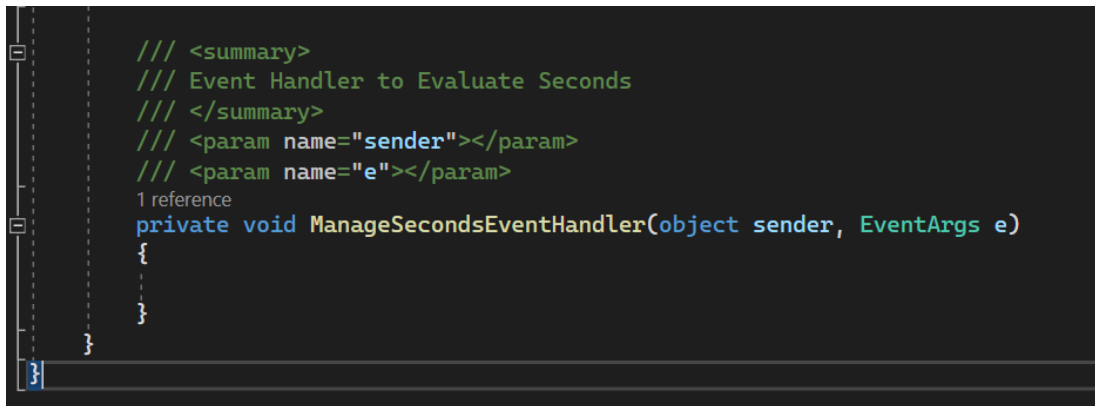


Figure 12: Click Event Handler

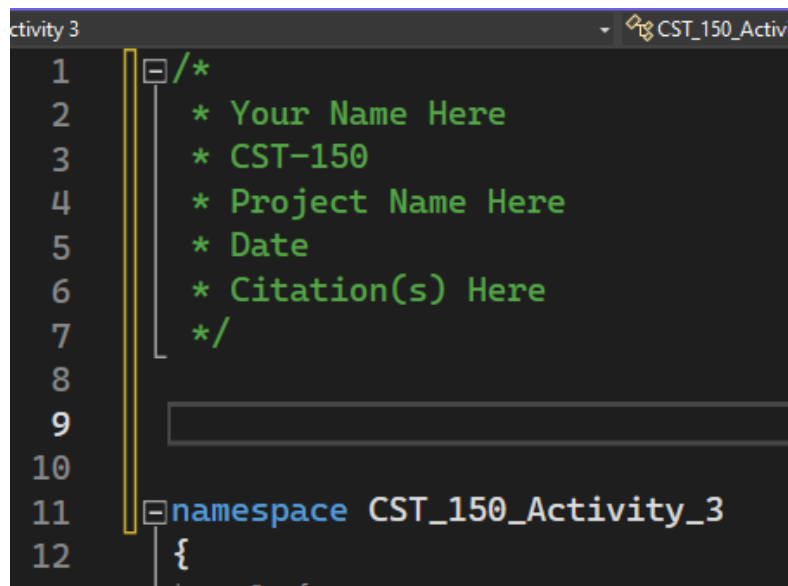
5. Add in the Method comments as shown in Figure 13.



```
/// <summary>
/// Event Handler to Evaluate Seconds
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void ManageSecondsEventHandler(object sender, EventArgs e)
{
    ...
}
```

Figure 13: Comments for Event Handler Method

6. Whenever a new cs page is started, the first item we complete is to add the citations at the very top as shown in Figure 14.



```
activity 3 CST_150_Activity_3
1  /*
2   * Your Name Here
3   * CST-150
4   * Project Name Here
5   * Date
6   * Citation(s) Here
7   */
8
9
10
11 namespace CST_150_Activity_3
12 {
```

Figure 14: Citation at the top.

6. Start coding the back end.

5. Open the FrmSeconds.cs file.
5. In the constructor, be sure the results label is not visible as shown in Figure 15.

```

public partial class FrmSeconds : Form
{
    /// <summary>
    /// Constructor
    /// </summary>
    1 reference
    public FrmSeconds()
    {
        InitializeComponent();
        // Make sure the label is not visible
        lblResults.Visible = false;
    }

    /// <summary>
    /// Event Handler to Evaluate Seconds
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    1 reference
    private void ManageSecondsEventHandler(object sender, EventArgs e)
    {
        // Declare and Initialize
        int seconds = 0; // Declare the value user entered in seconds
        // Working with Constants in C#
    }
}

```

Figure 15: Hide Results Label

- Start with Declare and Initialize section as shown in Figure 16.

```

/// <summary>
/// Event Handler to Evaluate Seconds
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void ManageSecondsEventHandler(object sender, EventArgs e)
{
    // Declare and Initialize
    int seconds = 0; // Declare the value user entered in seconds
    // Working with Constants in C#
}

```

Figure 16: Declare and Initialize

- In the Declare and Initialize, put in the constants as shown in Figure 17 using PascalCasing.
- Constants are exactly like they sound; they will stay constant and not change throughout the program.

```

/// <param name="e"></param>
1 reference
private void ManageSecondsEventHandler(object sender, EventArgs e)
{
    // Declare and Initialize
    int seconds = 0;           // Declare the value
    // Working with Constants in C#
    const int SecondsInMinutes = 60;
    const int SecondsInHours = 3600;
    const int SecondsInDays = 86400;
    // Define flags

```

Figure 17: Constants

- f. Default the results label to not be visible as shown in Figure 18. We do this here since we know the label will be visible after we run the program and first enter seconds. We do not want it visible after we first display results and then run it again.
- g. Also, be sure the font color is black. This avoids any issues from the error messages since they are displayed in red.

```

const int SecondsInHours = 3600;
const int SecondsInDays = 86400;

// Make sure the label is not visible
lblResults.Visible = false;
// Default the color to black
lblResults.ForeColor = Color.Black;

// Test to determine if integer was entered correctly
// if try to parse string to int is successful

```

Figure 18: Manage Label and Color

- h. Now, test the entry from the user using TryParse. If the entry is not correctly displayed, there will be a message to the user in a red font as shown in Figure 19.

```

lblResults.Visible = false;
// Default the color to black
lblResults.ForeColor = Color.Black;

// Test to determine if integer was entered correctly by user
// if try to parse string to int is successful continue else show message to user
if (int.TryParse(txtUserEntry.Text, out seconds) )
{
}
else
{
    // This is the code block that be executed if the user did not put an int in
    lblResults.Text = "Please enter an int to continue...";
    lblResults.ForeColor = Color.Red;
    lblResults.Visible = true;
}
}

```

Figure 19: TryParse

- i. Start nested if statements. Now that we know the data entry is valid, start testing for minutes.
- j. If the number is less than 60, display message as shown in Figure 19. Also, format the strings with commas as shown in Figure 20.

```

// if try to parse string to int is successful continue else show message to user
if (int.TryParse(txtUserEntry.Text, out seconds) )
{
    // This block of code is where all our business logic will be placed.
    // In future classes we will put this code (business logic) in the business layer
    // Practice with nested if statements
    if (seconds >= SecondsInMinutes)
    {
        // if we are here we know the user has entered a value that we can, at a minimum
        // display how many minutes are in the seconds entered.
        lblResults.Text = string.Format("There are {0:#,##} minute(s) in {1:#,##} seconds.\n", seconds / SecondsInMinutes, seconds);
        lblResults.Visible = true;
    }
    else
    {
        // if we are here we know the user did not enter a value that meets the minimum
        // requirements of at least 60
        // Ask the user to enter a valid number
        lblResults.Text = "Please enter an integer larger than 59.";
        lblResults.ForeColor = Color.Red;
        lblResults.Visible = true;
    }
}
else

```

Figure 20: Testing for Minutes

- k. Since we know the user entry is greater than 60, we need to now test if it greater than hours as shown in Figure 21.

```
// Display how many minutes are in the seconds entered.
lblResults.Text = string.Format("There are {0:#,##} minute(s) in {1:#,##} seconds.\n", seconds / SecondsInMinutes, seconds);
lblResults.Visible = true;

//-----
// Now that we know there were minutes - test for hours
// using nested if statements
// Nested statements stops code from being executed if there is
// no need.
//-----
if(seconds >= SecondsInHours)
{
    // Display how many hours are in the seconds entered.
    lblResults.Text += string.Format("There are {0:#,##} hours in {1:#,##} seconds.\n", seconds / SecondsInHours, seconds);

    //-----
    // Now that we know there were hours - test for days
```

Figure 21: Test for Hours

- Since we know the user entry is greater than SecondsInHours, we need to now test if it greater than SecondsInDays as shown in Figure 22.

```
if(seconds >= SecondsInHours)
{
    // Display how many hours are in the seconds entered.
    lblResults.Text += string.Format("There are {0:#,##} hours in {1:#,##} seconds.\n", seconds / SecondsInHours, seconds);

    //-----
    // Now that we know there were hours - test for days
    // using nested if statements
    //-----
    if(seconds >= SecondsInDays)
    {
        // Display how many days are in the seconds entered.
        lblResults.Text += string.Format("There are {0:#,##} days in {1:#,##} seconds.\n", seconds / SecondsInDays, seconds);
    }
}
```

Figure 22: Test for Seconds in Days

- Run the program and test the code. Enter "86543" for a test number and the end result is shown in Figure 23.

Activity 3

Enter the number of seconds: seconds

There are 1,442 minute(s) in 86,543 seconds.
There are 24 hours in 86,543 seconds.
There are 1 days in 86,543 seconds.

Figure 23: End Results

1. Submit the Activity as described in the digital classroom.
 - a. Be sure to use the number of seconds that is provided by your instructor before taking screenshots.