



Deployment Portfolio Task 1

SWE40006 – SOFTWARE DEVELOPMENT AND EVOLUTION

SUMMER - 2024

Student and Lecturer Details

Name	ID	Lecturer	Class
Vi Luan Dang	103802759	Dr. Thomas Hang Nsam@swin.edu.au	Monday 13:00 PM

Self-Assessment Details

Declaration ò task level attempted (P/C/D/HD)

	Pass	Credit	Distinction	High Distinction
Self-Assessment				✓

	Included & attempted
Task 1.1: Pass	✓
Task 1.2: Credit	✓
Task 1.3: Distinction	✓
Task 1.4: High Distinction	✓

Table of Content

TASK 1.1P: 2

TASK 1.2C: 9

TASK 1.3D: 12

TASK 1.4HD: 16

Assignment Report

Task 1.1P: Follow the Wix walkthrough and deploy a sample desktop

1.1A. Visual Studio and Wix setup

For this task, I have installed **Microsoft Visual Studio 2017**, **Wix Toolset v3.14.1.8722**, and **WiX Extension for Visual Studio 2017 v1.0.0.22**.

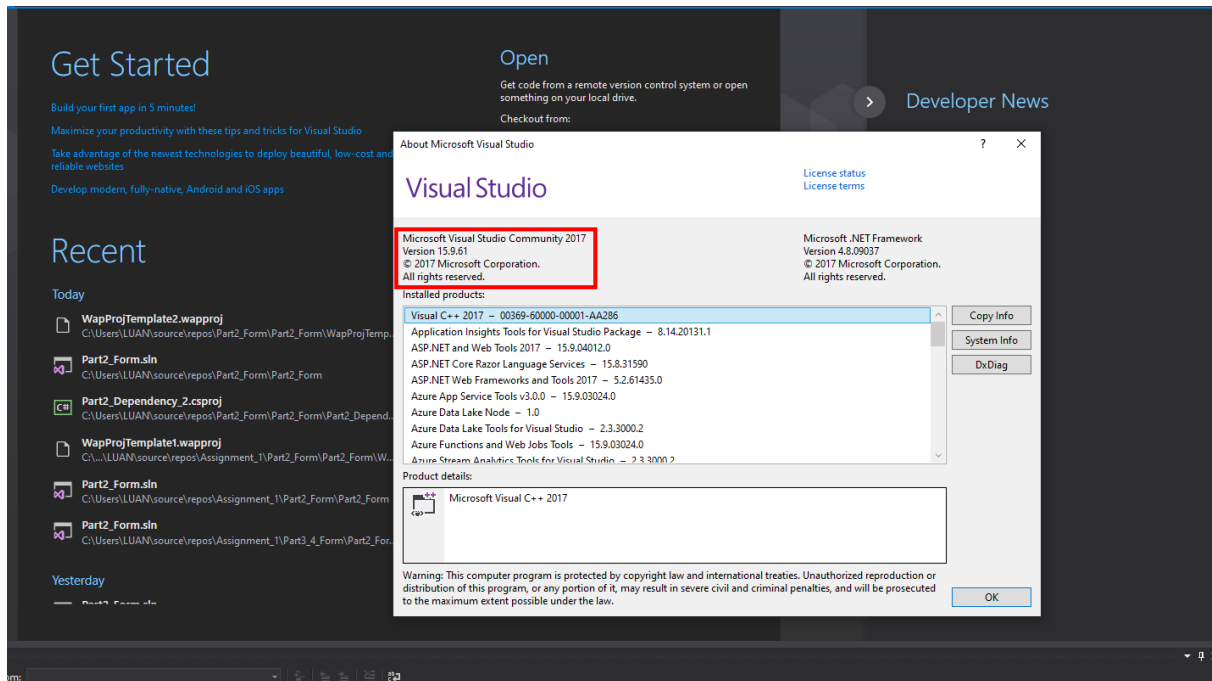


Figure 1: Microsoft Visual Studio 2017



Figure 2: Wix Toolset v3.14.1.8722

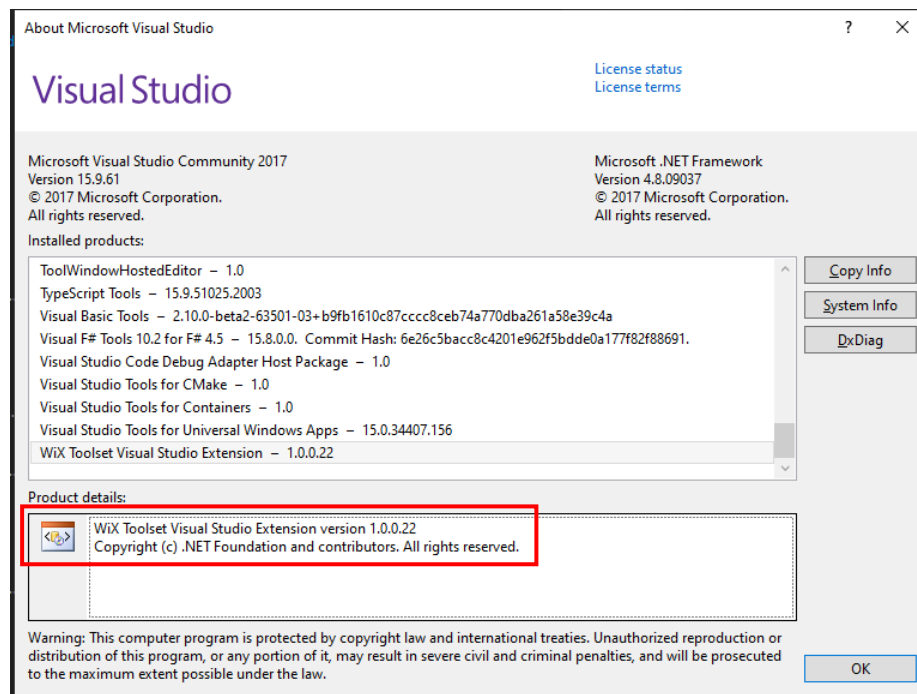


Figure 3: Wix Extension for Visual Studio 2017

1.1B. Simple desktop form setup

For this task, I will set up and test a simple form to prepare for Wix installation.

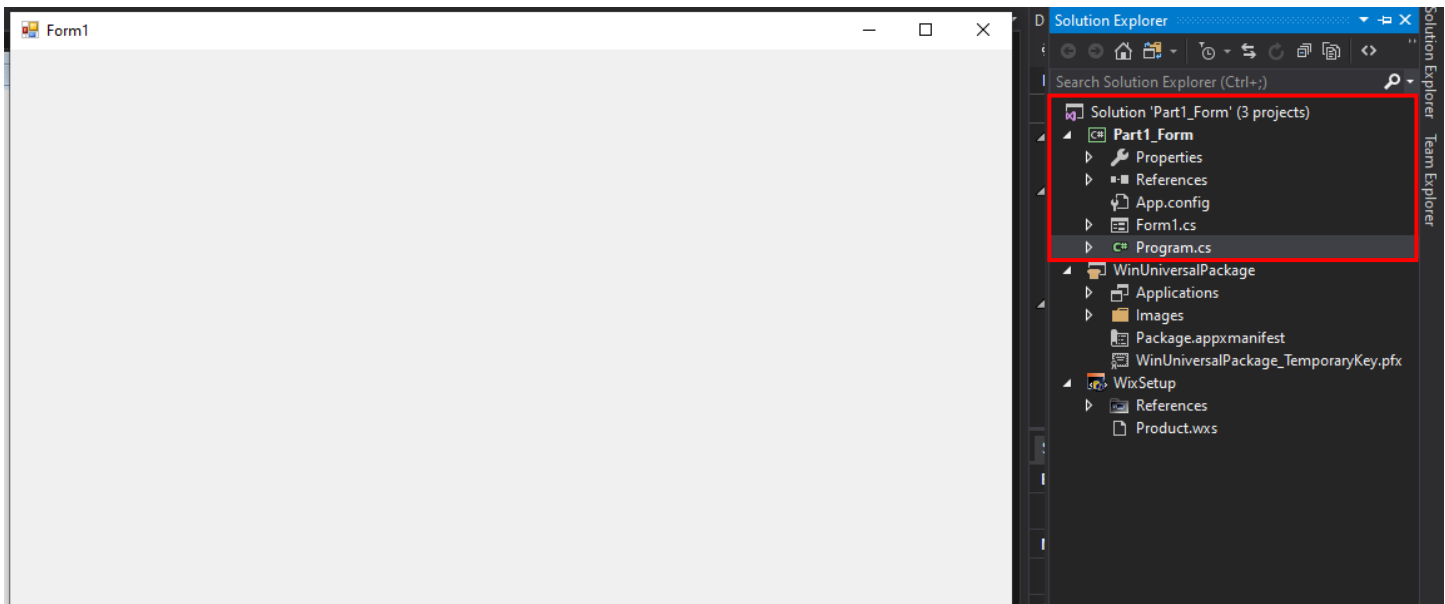


Figure 4: Simple form setup

1.1C. Wix integration for our simple form

For this task, I will add Wix Setup Project into our solution, after that we can add our form onto reference folder of Wix Project.

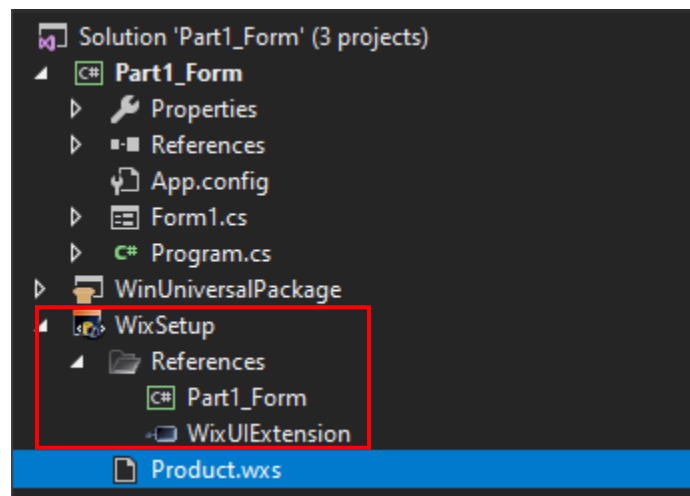


Figure 5: Wix setup project for simple form

I will also make some adjustments to the *Product.wxs* file.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
3   <!-- Product Information Section -->
4   <!-- Product ID: is an ID for a specific version, so new versions will contain different Product IDs -->
5   <!-- UpgradeCode: relates set of products, all version of a product share the same UpgradeCode -->
6   <!-- Window use the ProductID and UpgradeCode to determine whether there is an outdated version of a product is installed -->
7   <!-- For example:
8     Initial Version (v1.0.0):
9       - Product Id: 12345678-1234-1234-1234-123456789012
10      - UpgradeCode: 18b6003f-1521-488f-8bfc-b9feb31778
11
12     Next Version (v1.1.0):
13       - Product Id: 87654321-4321-4321-4321-210987654321 (different from v1.0.0)
14       - UpgradeCode: 18b6003f-1521-488f-8bfc-b9feb31778 (same as v1.0.0)
15
16 ==> When the installer for v1.1.0 runs, it uses the UpgradeCode to check if v1.0.0 (or any other previous version) is installed.
17 If it finds a previous version with the same UpgradeCode, it knows it needs to upgrade the existing installation.-->
18 <Product Id="7480f9b3-8043-4f61-9d07-97356ce0085c"
19   Name="WixSetup"
20   Language="1033"
21   Version="1.0.0.0"
22   Manufacturer="ViluanDang"
23   UpgradeCode="A52E396C-2ADD-4F41-AE24-69D72C4E59F9">
24
25   <!-- Package Information Section -->
26   <Package InstallerVersion="200"
27     Compressed="yes"
28     InstallScope="perMachine" />
29
30   <MajorUpgrade DowngradeErrorMessage="A newer version of [ProductName] is already installed." />
31   <!-- Cab file contains all the files for the application, if this is yes then all the file is contained within MSI file -->
32   <MediaTemplate EmbedCab="yes"/>
33
34   <!-- Feature Information Section -->
35   <Feature Id="ProductFeature" Title="WixSetup" Level="1">
36     <ComponentGroupRef Id="ProductComponents" />
37   </Feature>
38
39   <!-- Set property for customize directory-->
40   <Property Id="WIXUI_INSTALLDIR" Value="INSTALLFOLDER" />
41   <!-- Use WixUI for customize directory-->
42   <UIRef Id="WixUI_InstallDir"/>
43 </Product>
```

Figure 6: Product.wxs file

```
44 </Product>
45 <!-- Installation Directory for our app-->
46 <Fragment>
47   <Directory Id="TARGETDIR" Name="SourceDir">
48     <Directory Id="ProgramFilesFolder">
49       <Directory Id="INSTALLFOLDER" Name="WixSetup" />
50     </Directory>
51   </Directory>
52 </Fragment>
53
54 <!-- Application specification-->
55 <Fragment>
56   <ComponentGroup Id="ProductComponents" Directory="INSTALLFOLDER">
57     <Component Id="Part1_Form.exe" Guid="32730164-0906-4EAB-AB87-4A6F93719257" >
58       <File Id="Part1_Form.exe" Name="Part1_Form.exe" Source="$(var.Part1_Form.TargetDir)Part1_Form.exe"></File>
59     </Component>
60   </ComponentGroup>
61 </Fragment>
62 </Wix>
63
```

Figure 7: Fragment Tag.

In the above code, I have added an “UIRef” and “Property” tags so that we can direct our installation path without it being automatically installed in the “ProgramFile”. I also modified the “Fragment” tags so that our application and its installed directory can be specified. With that in place, we can build our Wix Project and start to install our application.

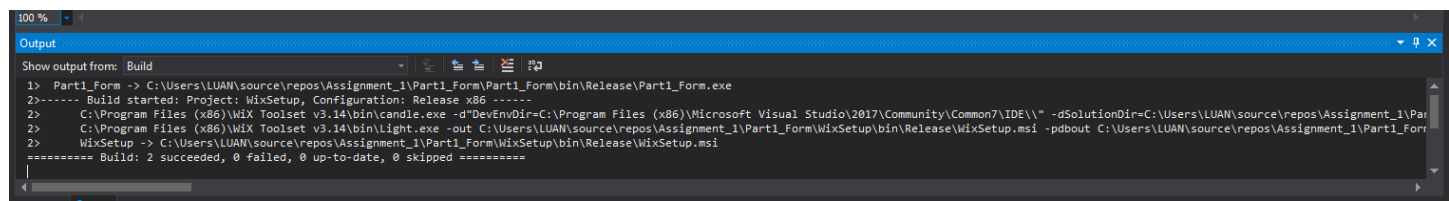


Figure 8: Wix setup project being built.

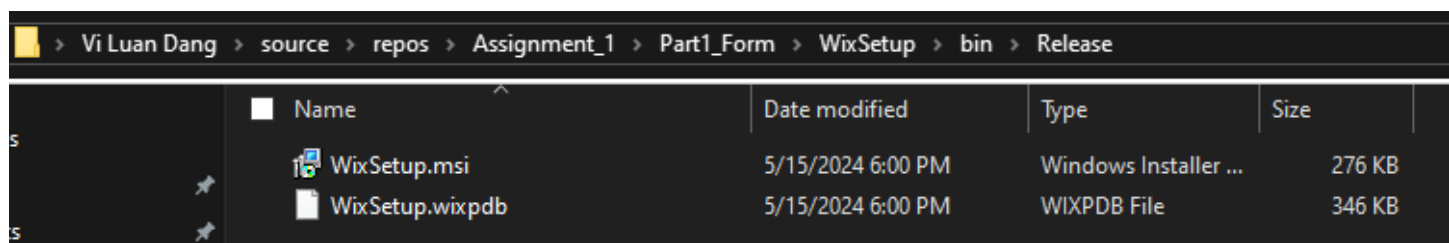


Figure 9: msi file.

At the specified directory, there will only be 2 files as the “cab” file is already bundled into the “WixSetup.msi” file. We can start to install our application.

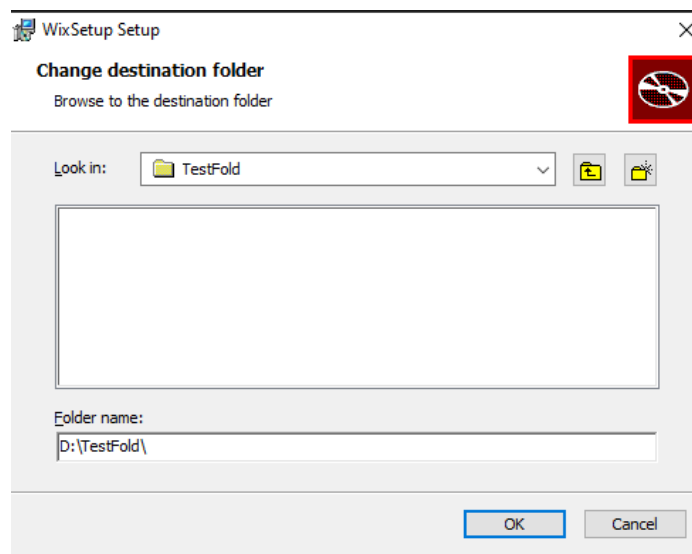


Figure 10: Specification for installed location.

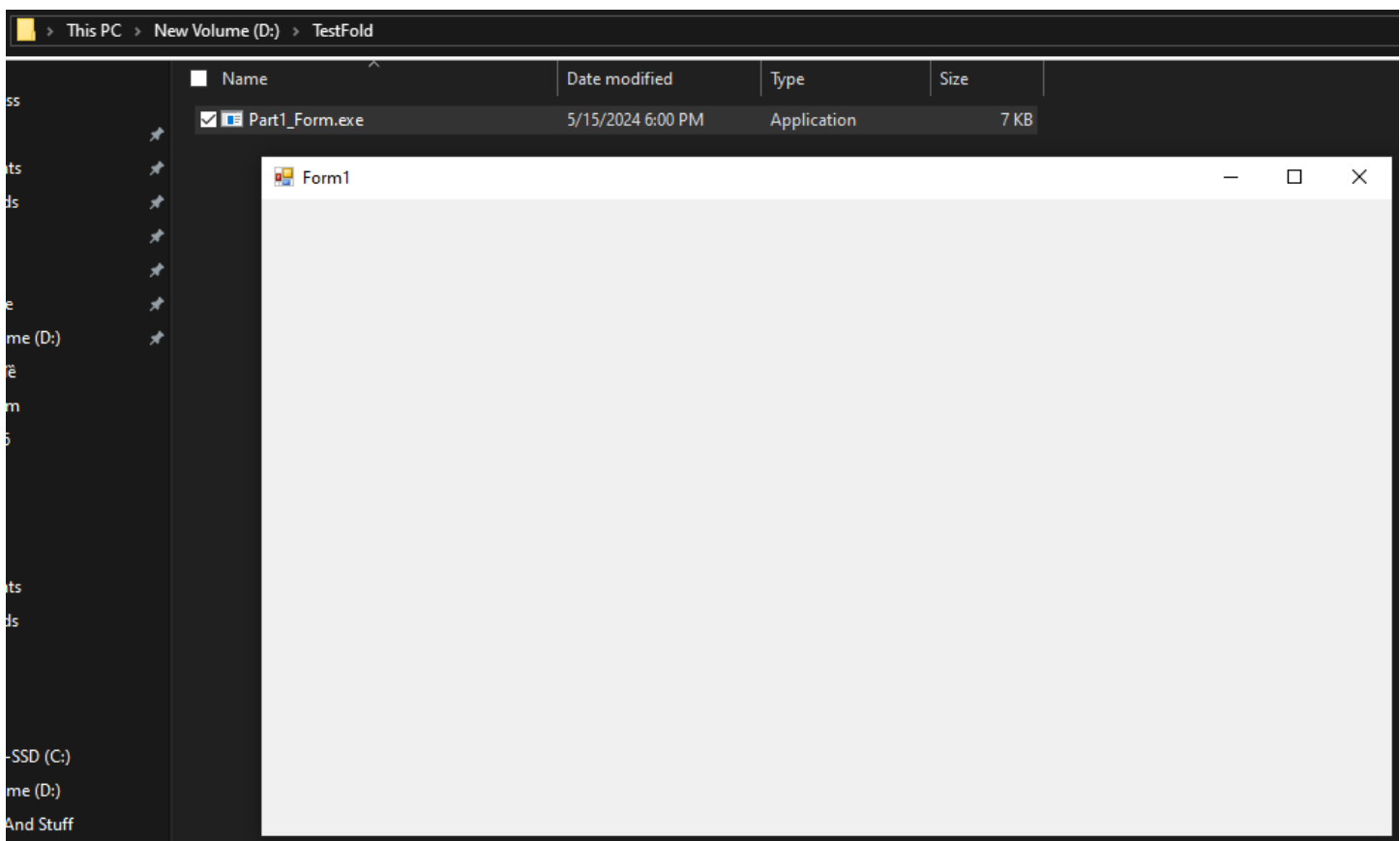


Figure 11: Installed application

After building our Wix Project, we can proceed with our installation process at the “*msi*” files by specifying the location, as shown in *figure 10*. After the installation process, we can check our application at the specified directory and test our application, as shown in *figure 11*. This will also conclude this task.

1.1D. Universal Windows App Packaging

For this task, I will add a Universal Windows App Packaging onto our solution and create a package for our solution.

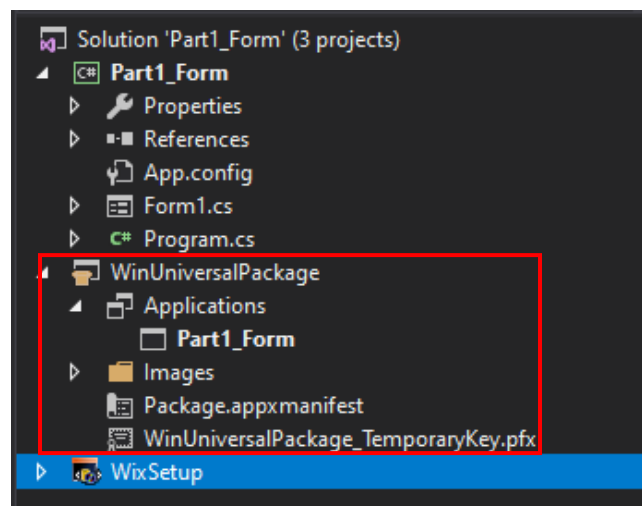


Figure 12: Universal Windows App Packaging for simple form.

With that in place, I will create the package for our form and install the certificate for our package.

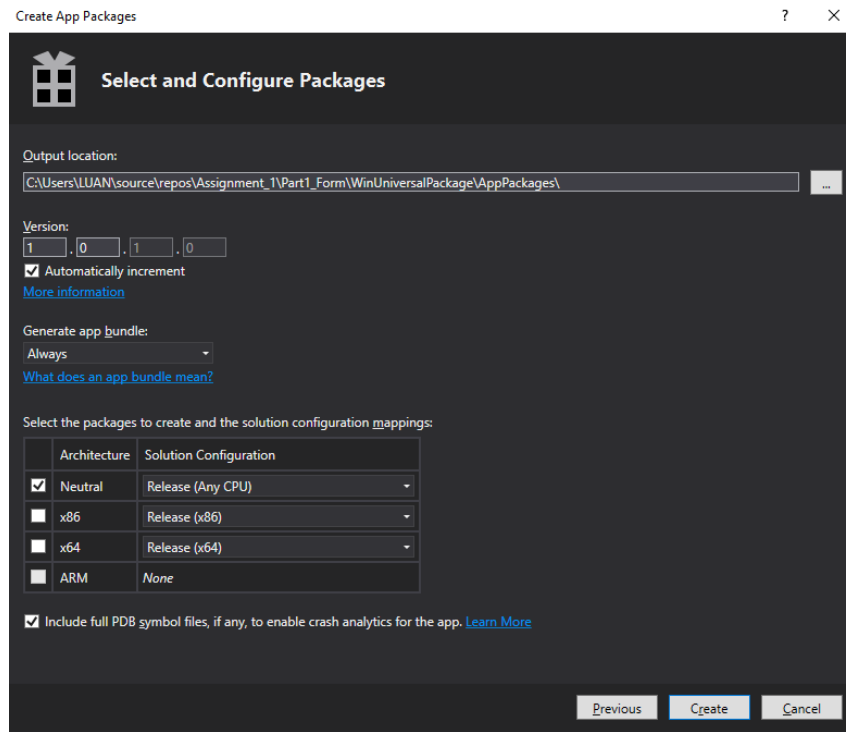


Figure 13: Create package wizard.

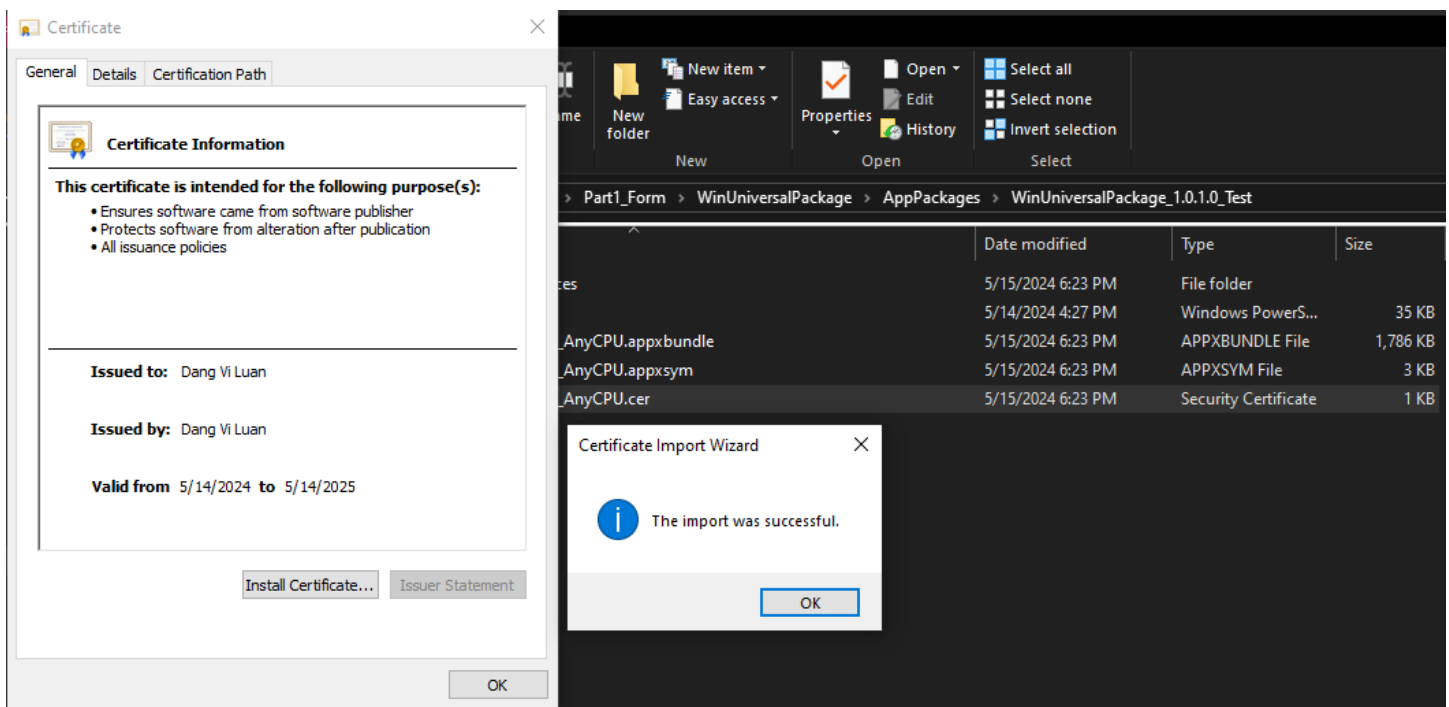


Figure 14: Package certificate

After the preparation process, we can install our package to test our process.

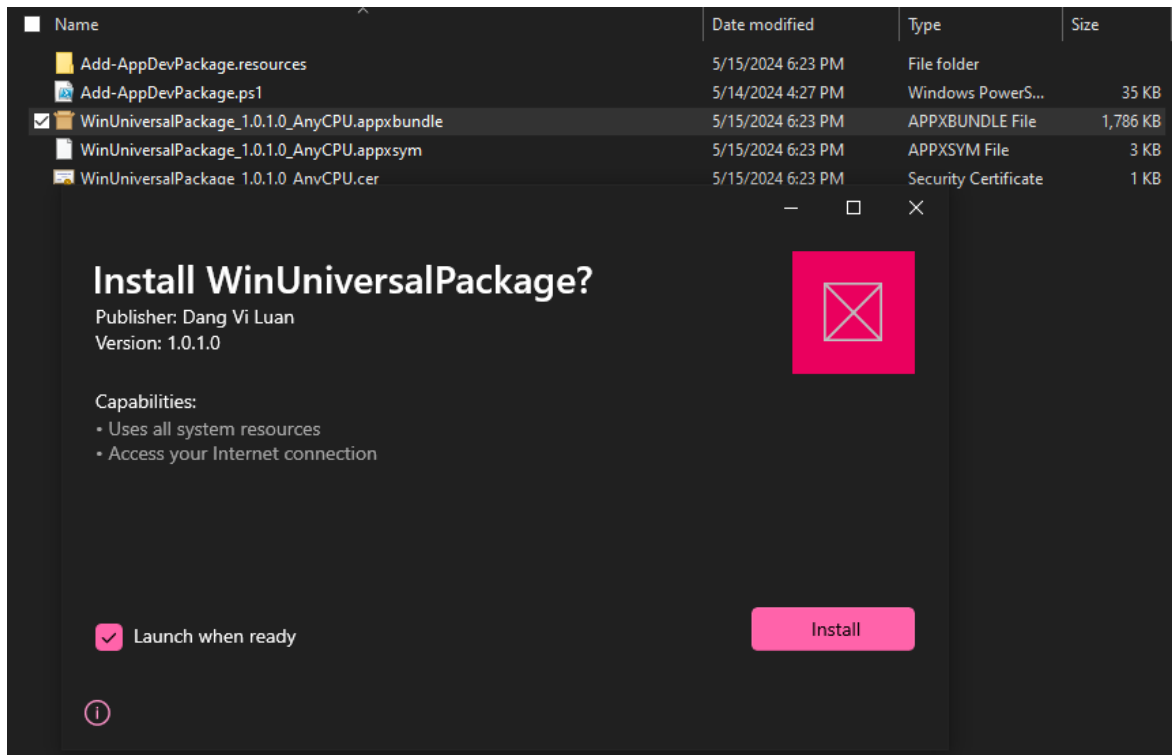


Figure 15: Universal Package installation

We can certify the completion of this task by launching the application after it is installed, as shown in *figure 16* below. This will also conclude task 1.1P.

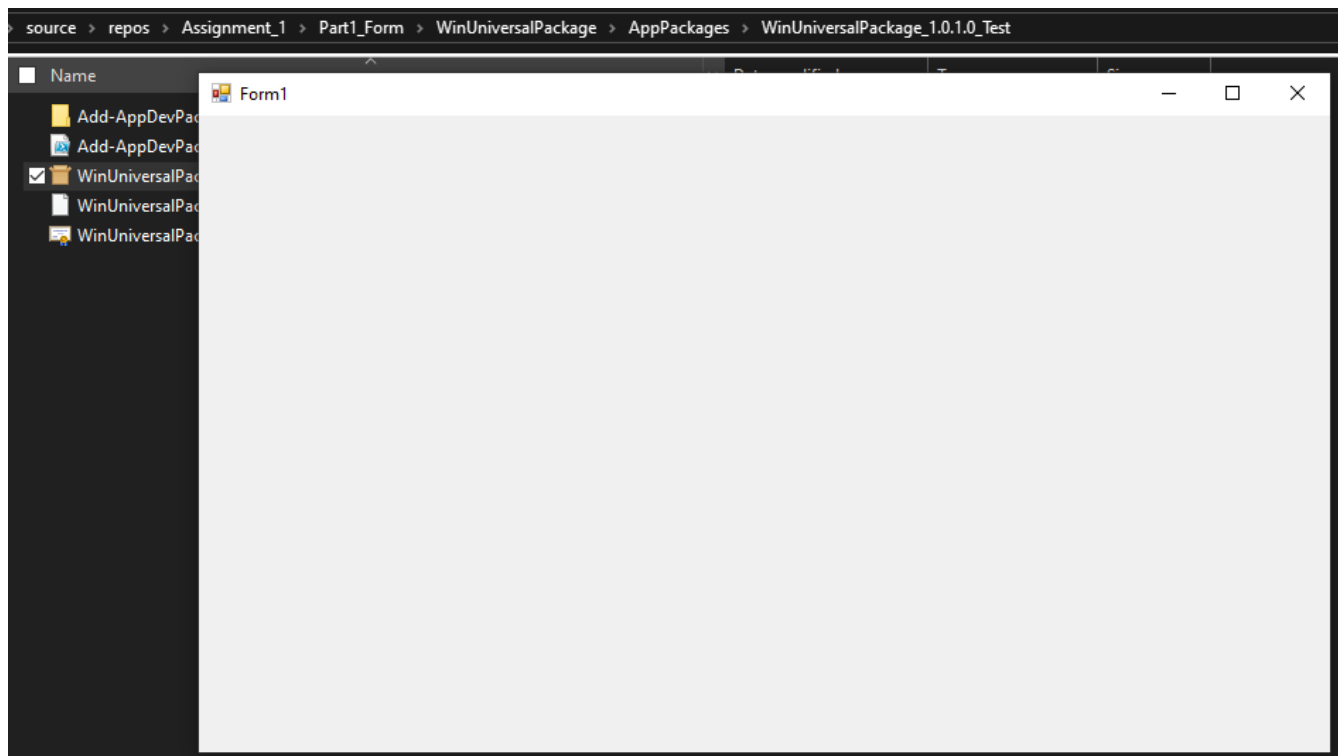


Figure 16: Application successfully installed.

Task 1.2C: Follow the Wix walkthrough and deploy your own C, C++, or C# desktop.

1.2A. Form modification for individual desktop application.

For this task, I will add a button to the form, upon clicking, the button will show a text from a text file. To do this, a button will be added to the form, and I will also modify “*Form1.cs*” file.

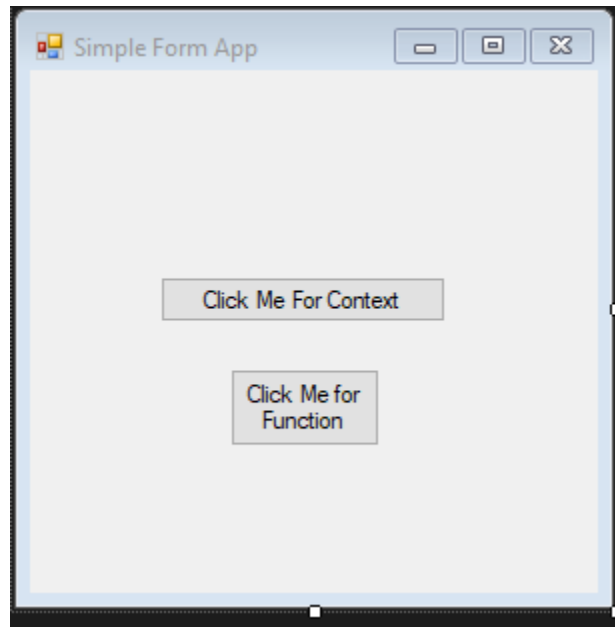


Figure 17: “Context” button for this task

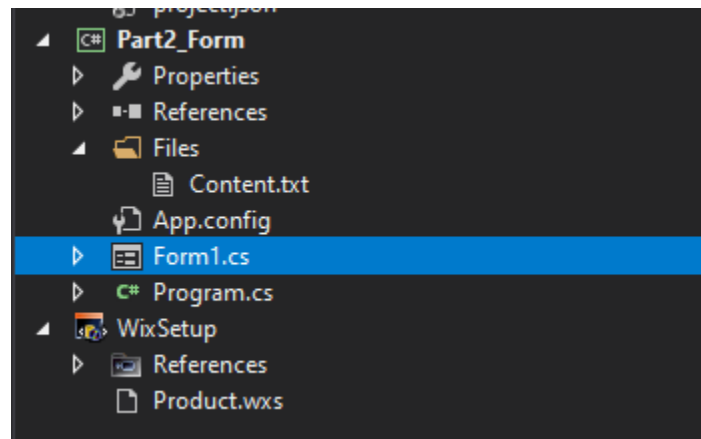


Figure 18: Files folder with content text file

After that, I will use “MessageBox” to show the content in Files Folder, as shown in figure 19 below:

```

namespace Part2_Form
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void buttonShowMessage_Click(object sender, EventArgs e)
        {
            try
            {
                // Specify the path to the text file
                string filePath = "../Files/Content.txt";

                // Read the content of the file
                string fileContent = File.ReadAllText(filePath);

                // Display the content in a message box
                MessageBox.Show(fileContent);
            }
            catch (Exception ex)
            {
                // Display an error message if something goes wrong
                MessageBox.Show("An error occurred: " + ex.Message);
            }
        }
    }
}

```

Figure 19: Function to show text in the text file.

In order for our installed application to work as expected, I will need to modify the “Product.msx” file as well. Specifically, I will add a file component, so that the installed application will come with a File Folder and the content text file.

```

57 <Fragment>
58 <ComponentGroup Id="Files_files" Directory="Files">
59 <!-- File component -->
60 <Component Id="Content.txt" Guid="aea868c1-2d18-400d-88fa-e506b16ef130">
61 <File Id="Content.txt" Name="Content.txt" Source="$(var.Part2_Form.TargetDir)Files\Content.txt" />
62 </Component>
63 </ComponentGroup>
64 </Fragment>

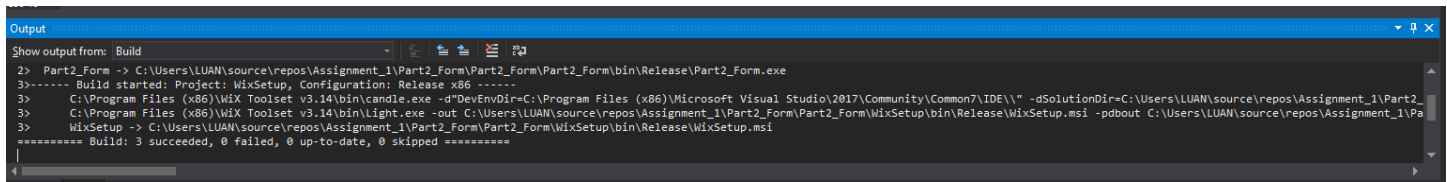
```

Figure 20: File component.

With all of the preparation steps done, we can move to the next task.

1.2B. Build WixSetup to test the newly updated form.

We will build our Wix project and proceed with the installation process to test our form. As the installation process is not so different from Task 1.1P, I will proceed to show the result of it.



```
Output
Show output from: Build
2> Part2_Form -> C:\Users\LUAN\source\repos\Assignment_1\Part2_Form\Part2_Form\bin\Release\Part2_Form.exe
3>----- Build started: Project: WixSetup, Configuration: Release x86 -----
3> C:\Program Files (x86)\WIX Toolset v3.14\bin\candle.exe -d"DevEnvDir=C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\Common7\IDE\" -dSolutionDir=C:\Users\LUAN\source\repos\Assignment_1\Part2_Form\WixSetup -out C:\Users\LUAN\source\repos\Assignment_1\Part2_Form\WixSetup\bin\Release\WixSetup.msi -pdbout C:\Users\LUAN\source\repos\Assignment_1\Part2_Form\WixSetup\bin\Release\WixSetup.msi
3> WixSetup -> C:\Users\LUAN\source\repos\Assignment_1\Part2_Form\Part2_Form\WixSetup\bin\Release\WixSetup.msi
===== Build: 3 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Figure 21: Wix Project successfully built.

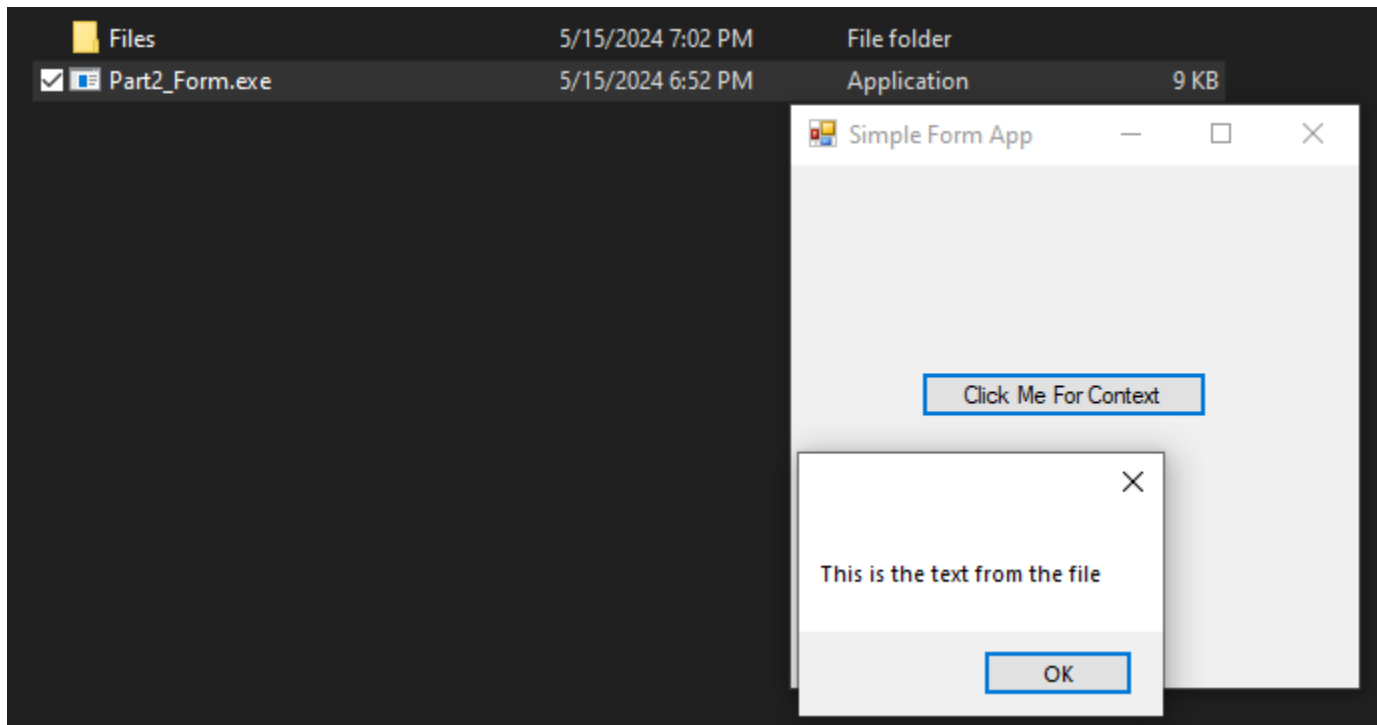


Figure 22: Application successfully installed.

We can see that our application is successfully installed with a File folder and content text file, and the button also works as expected. This will conclude task 1.2C.

Task 1.3D: Complete tasks 1.1 or 1.2 AND deploy an application with multiple DLLs or dependencies.

1.3A. Modify source code to add required functionalities

In this task, we will add two library classes and link it to our main form for additional functionalities. Because of that, I will add two more buttons, each of them will display text according to a class library.

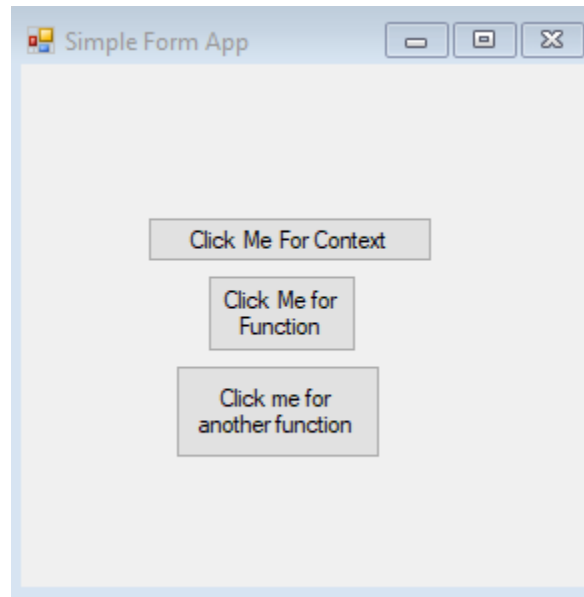


Figure 23: Updated form.

Accordingly, I will also modify *Form1.Designer.cs* and *Form1.cs* to support these additional functionalities.

```
34 //
35 // buttonShowFunction
36 //
37 this.buttonShowFunction.Location = new System.Drawing.Point(93, 105);
38 this.buttonShowFunction.Name = "buttonShowFunction";
39 this.buttonShowFunction.Size = new System.Drawing.Size(75, 39);
40 this.buttonShowFunction.TabIndex = 1;
41 this.buttonShowFunction.Text = "Click Me for Function";
42 this.buttonShowFunction.UseVisualStyleBackColor = true;
43 this.buttonShowFunction.Click += new System.EventHandler(this.buttonShowFunction_Click);
44 //
45 // buttonShowFunction2
46 //
47 this.buttonShowFunction2.Location = new System.Drawing.Point(77, 150);
48 this.buttonShowFunction2.Name = "buttonShowFunction2";
49 this.buttonShowFunction2.Size = new System.Drawing.Size(103, 47);
50 this.buttonShowFunction2.TabIndex = 2;
51 this.buttonShowFunction2.Text = "Click me for another function";
52 this.buttonShowFunction2.UseVisualStyleBackColor = true;
53 this.buttonShowFunction2.Click += new System.EventHandler(this.buttonShowFunction2_Click);
```

Figure 24: *Form1.Designer.cs* file

```

46 private void buttonShowFunction_Click(object sender, EventArgs e)
47 {
48     try
49     {
50         // Create an instance of the class from the class library
51         Class1 class1 = new Class1();
52
53         // Get the message from the class library
54         string message = class1.GetMessage();
55
56         // Display the message in a message box
57         MessageBox.Show(message);
58     }
59     catch (Exception ex)
60     {
61         // Display an error message if something goes wrong
62         MessageBox.Show("An error occurred: " + ex.Message);
63     }
64 }
65
66 private void buttonShowFunction2_Click(object sender, EventArgs e)
67 {
68     try
69     {
70         // Create an instance of the class from the class library
71         Class2 class2 = new Class2();
72
73         // Get the message from the class library
74         string message = class2.GetMessage();
75
76         // Display the message in a message box
77         MessageBox.Show(message);
78     }
79     catch (Exception ex)
80     {
81         // Display an error message if something goes wrong
82         MessageBox.Show("An error occurred: " + ex.Message);
83     }
84 }
85
86 }

```

Figure 25: Form1.cs file

I will also add two class libraries as dependencies to support these buttons, these classes will be added into our application's references, as show in the figure below:

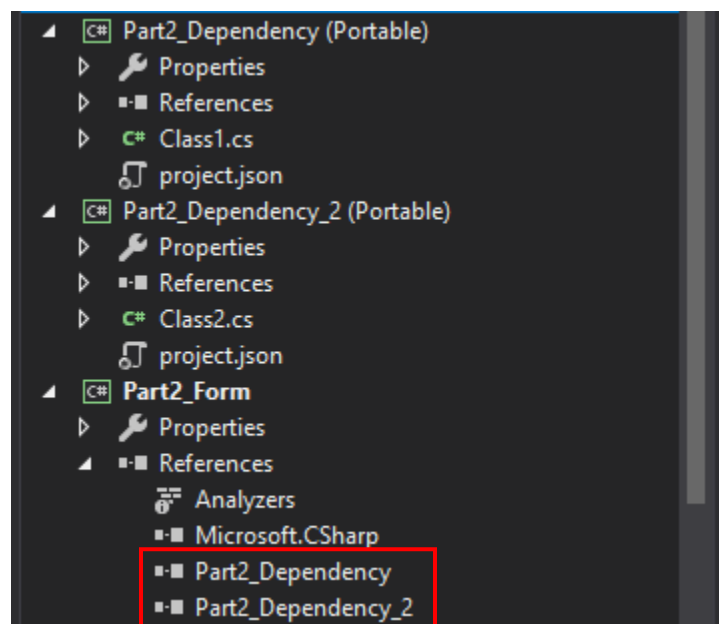
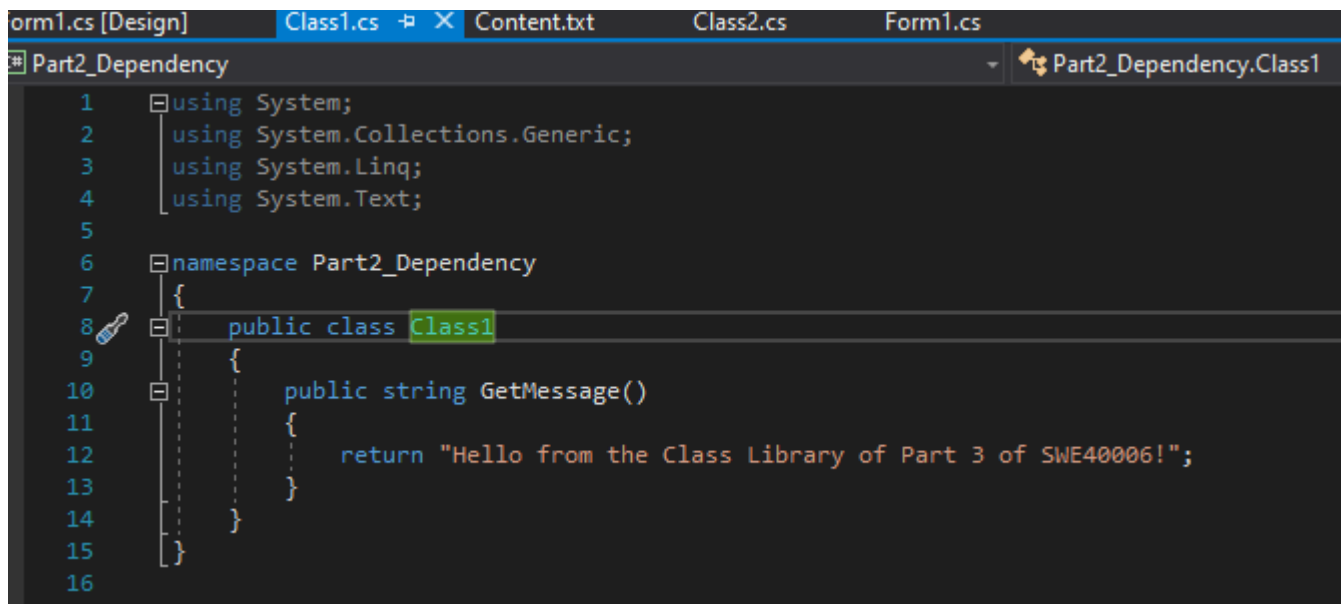


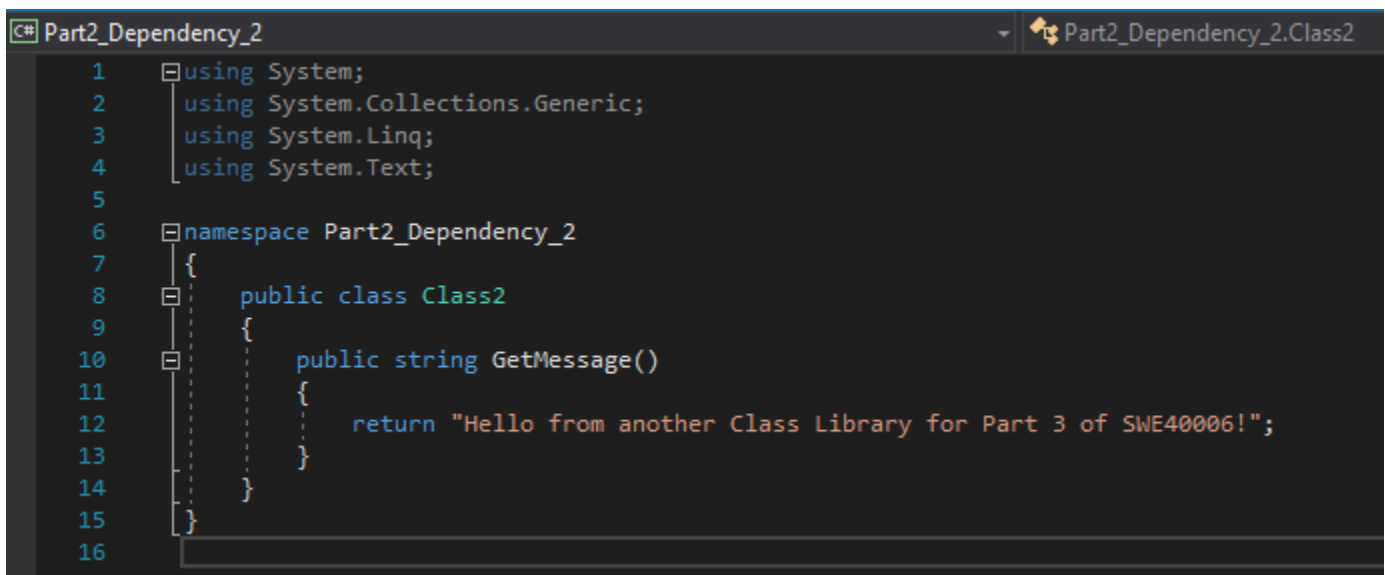
Figure 26: Newly added dependencies

Inside these dependencies, there will be a simple class and a simple method to support the functionalities of the newly added buttons.



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace Part2_Dependency
7  {
8      public class Class1
9      {
10         public string GetMessage()
11         {
12             return "Hello from the Class Library of Part 3 of SWE40006!";
13         }
14     }
15 }
16
```

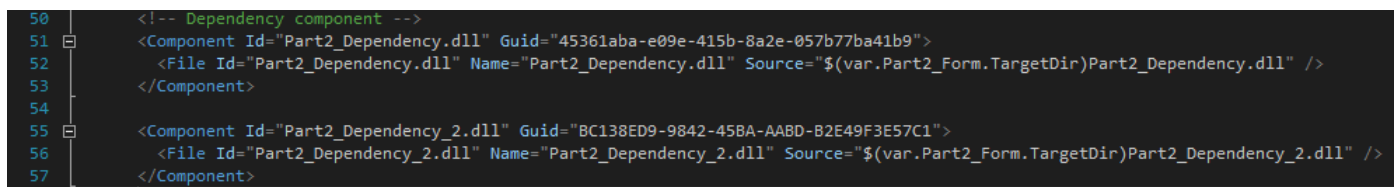
Figure 27: First dependency



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace Part2_Dependency_2
7  {
8      public class Class2
9      {
10         public string GetMessage()
11         {
12             return "Hello from another Class Library for Part 3 of SWE40006!";
13         }
14     }
15 }
16
```

Figure 28: Second dependency

These classes, however, will not be added to the installation file if we don't specifically mention it in our Wix setup project.

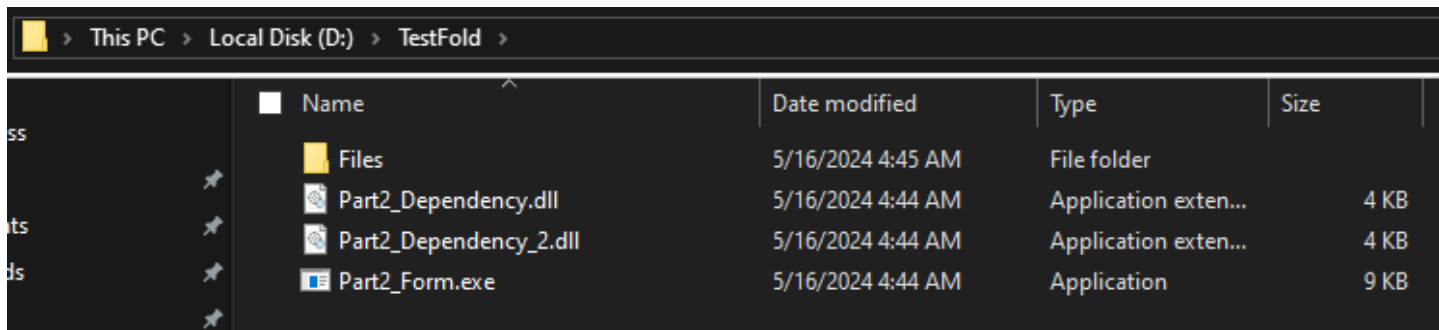


```
50 <!-- Dependency component -->
51 <Component Id="Part2_Dependency.dll" Guid="45361aba-e09e-415b-8a2e-057b77ba41b9">
52     <File Id="Part2_Dependency.dll" Name="Part2_Dependency.dll" Source="$(var.Part2_Form.TargetDir)Part2_Dependency.dll" />
53 </Component>
54
55 <Component Id="Part2_Dependency_2.dll" Guid="BC138ED9-9842-45BA-AABD-B2E49F3E57C1">
56     <File Id="Part2_Dependency_2.dll" Name="Part2_Dependency_2.dll" Source="$(var.Part2_Form.TargetDir)Part2_Dependency_2.dll" />
57 </Component>
```

Figure 29: Dependencies specification in Wix setup project.

1.3B. Installed the application to test additional functionalities

The above steps will conclude the preparation for this task, we can now build the Wix setup and proceed with the installation steps to check if our application works as expected. After the installation steps similar to task 1.1C, we will have the following files.



Name	Date modified	Type	Size
Files	5/16/2024 4:45 AM	File folder	
Part2_Dependency.dll	5/16/2024 4:44 AM	Application exten...	4 KB
Part2_Dependency_2.dll	5/16/2024 4:44 AM	Application exten...	4 KB
Part2_Form.exe	5/16/2024 4:44 AM	Application	9 KB

Figure 30: Installed application

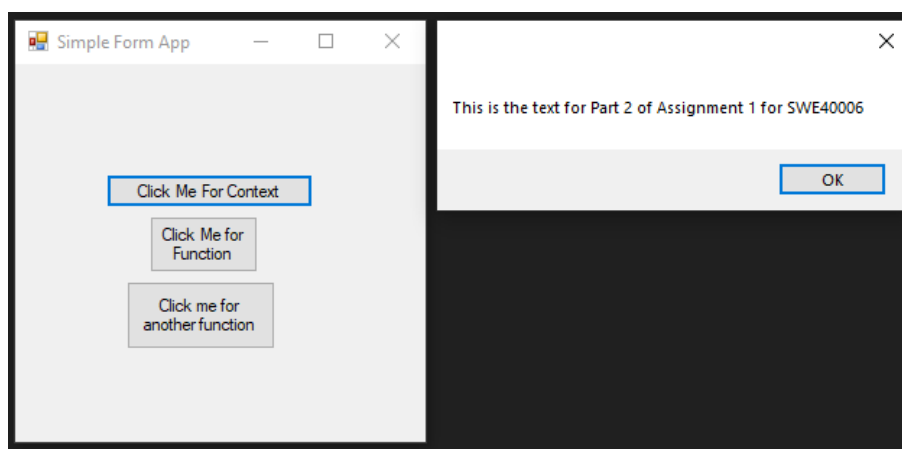


Figure 31: The first button for Part 2.

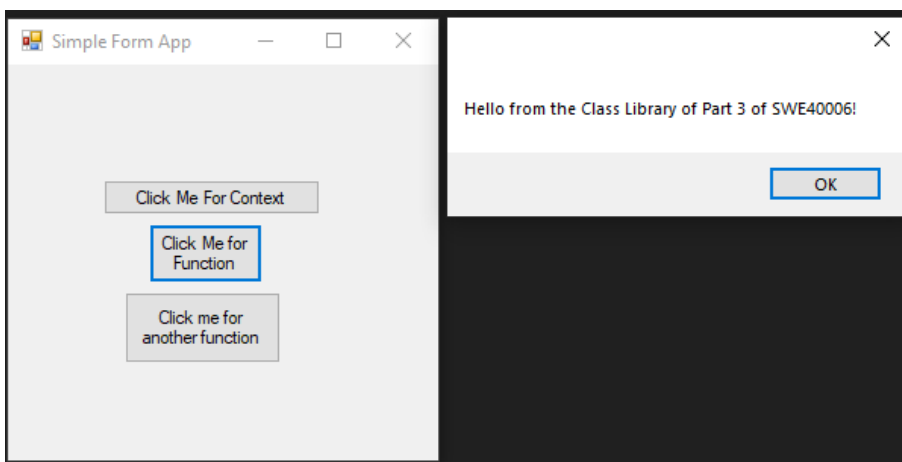


Figure 32: The second button for Part 3.

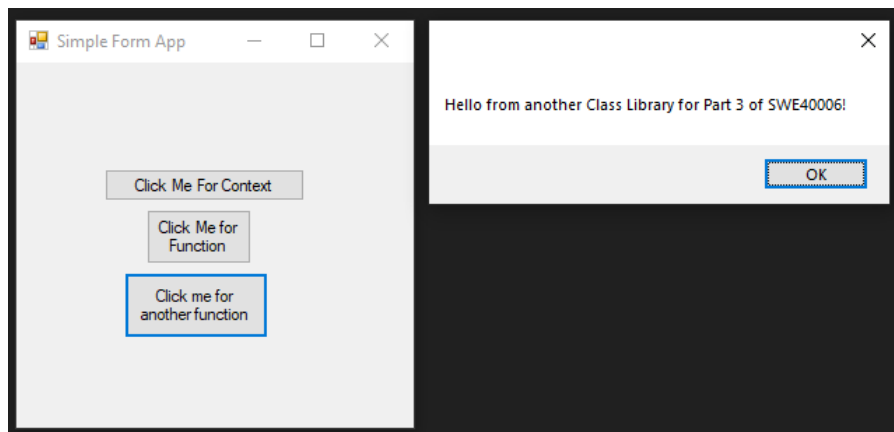


Figure 33: The third button for Part 3.

Our application will read the text file from file folder if we press the first button, the second button will invoke method from “Part2_Dependency.dll”, and the third button will invoke method from “Part2_Dependency_2.dll”. This output will satisfy the requirement for this task and will also conclude the work for this task.

The complete source code for this task can be accessed via [this link](#).

Task 1.4HD: Complete task 1.3 and deploy your application to the Microsoft store for public access and downloads.

1.4A. Rationale for not choosing Microsoft Store as a platform for publishing applications.

To publish an application on the Microsoft Store, a developer account is required. The registration fee is 400,425 VND, or approximately 24 Australian Dollars. Rather than investing this amount for a single-use platform, I opted to use GitHub for releasing my application. This decision allows for a cost-effective and accessible distribution method while leveraging the collaborative features and community support that GitHub provides.

The installation file can be accessed via [this link](#).

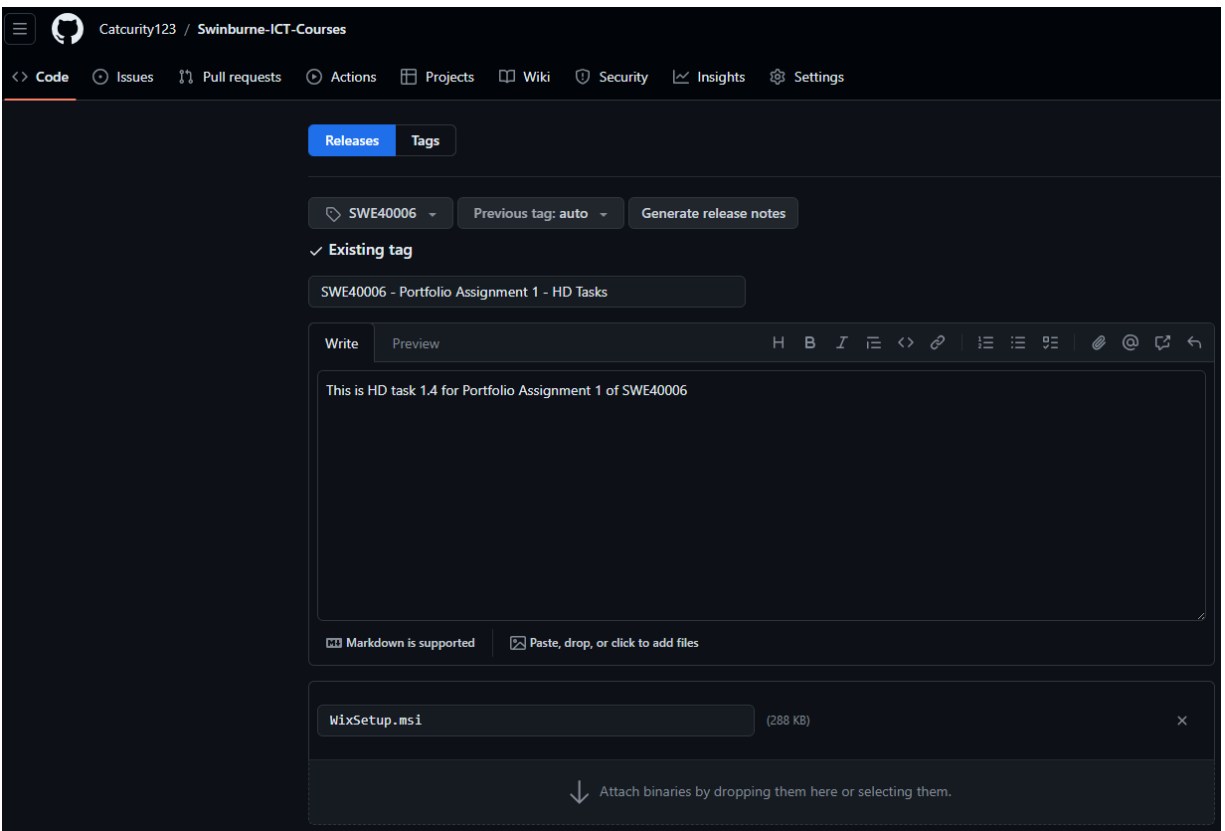


Figure 34: GitHub Release to publish application for public access and download.

This will allow my application to be published for public access and download, similar to Microsoft Store, without paying unnecessary fee.

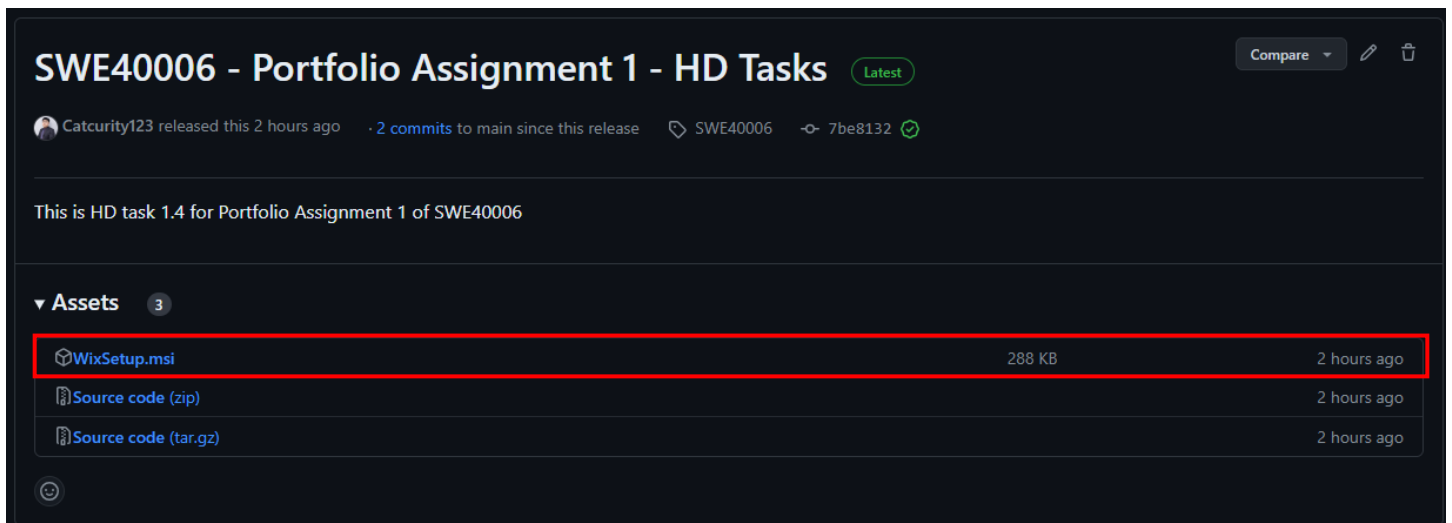


Figure 35: Application ready for public access and download.

To test the published application, I will uninstall the existing application on my machine and install it from the downloaded *WixSetup.msi*.



Figure 36: Warning for unrecognized applications.

Should there be a warning preventing unrecognized applications from running, kindly press “more info”, and proceed with “Run anyway”. We will proceed with the normal installation process after that.

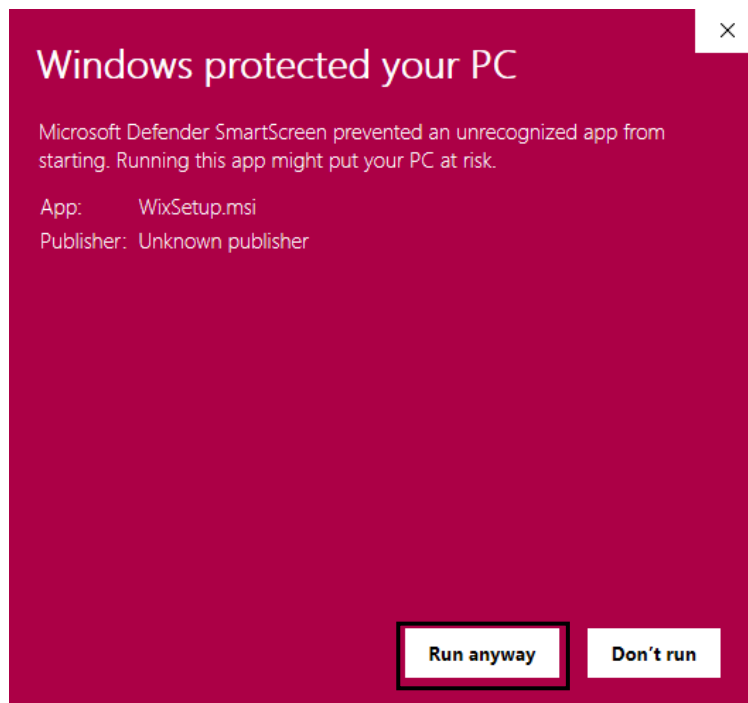


Figure 37: Run the application anyway.

Name	Date modified	Type	Size
Files	5/16/2024 5:10 AM	File folder	
Part2_Dependency.dll	5/15/2024 3:35 PM	Application exten...	4 KB
Part2_Dependency_2.dll	5/15/2024 3:35 PM	Application exten...	5 KB
Part2_Form.exe	5/15/2024 3:35 PM	Application	9 KB

Figure 37: Installed application

After the installation process, we will be presented with 3 files and a folder similar to task 1.3D. This shows that we have satisfied the requirements for this task and will also conclude the Portfolio Assignment 1 to a High Distinction level.

Resources

The complete source code for this task can be accessed via [this link](#).

The installation file can be accessed via [this link](#).