

C# Windows Presentation Foundation Graphical User Interface Supported by a VisualAPL Business Rules Component

In this example a C# project which defines a Windows Presentation Foundation (WPF) graphical user interface (GUI) captures user-input data which is validated and processed by a business rules component built with VisualAPL. Both the C# WPF GUI and the VisualAPL components are a fully-managed .Net assemblies.

The WPF GUI is a window containing:

- An TextBox control to receive the user input
- A Button control which, when user-clicked, will run the VisualAPL functions that calculate the values for this application system
- Several TextBlock controls to contain the calculated results
- Several Label controls to identify the other control purposes to the user
- These controls are contained in a vertically-oriented StackPanel 'container' control
- The StackPanel control is contained in a ScrollViewer 'container' control with automatically-visible horizontal and vertical scrollbars
- The ScrollViewer control is contained in a Border 'container' control
- The Border control is contained in the top-most Window 'container' control of the application system
- The Window control employs the 'SizeToContent' attached property so that the entire GUI layout is automatically sized.

The C# WPF GUI is specified using an xml-format XAML file, "Window1.xaml". VisualStudio renders the XAML file specification to visible controls on the design surface of the project.

WPFusingVisualAPL - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Data Tools Test Window Help

Debug Any CPU aplwin

Window1.xaml Class1.apl Window1.xaml.cs Start Page Cielo Explorer

Server Explorer Toolbox

Properties

Design XAML

```
1 <Window
2   x:Class="WPFusingVisualAPL.Window1"
3   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5   Title="C# WPF Using VisualAPL"
6   SizeToContent="WidthAndHeight"
7   WindowStartupLocation="CenterScreen">
8   <Border
9     BorderBrush="DarkBlue"
10    BorderThickness="5"
11    CornerRadius="5">
12     <ScrollViewer
13       ScrollViewer.HorizontalScrollBarVisibility="Auto"
14       ScrollViewer.VerticalScrollBarVisibility="Auto">
15       <StackPanel
16         Orientation="Vertical">
17         <Label
18           Margin="3,3,3,3">
19           Enter a list of numbers separated by spaces:
20         </Label>
21         <TextBox
22           Margin="3,3,3,3"
23           Background="AliceBlue"
24           x:Name="txtbxInput">
25           1 2 7 45.678 7 9.101112 7 2 3
26         </TextBox>
27         <Button
28           Margin="3,3,3,3"
29           x:Name="bnCalculate"
30           Click="bnCalculate_Click">
31           Calculate
32         </Button>
33         <Label
34           Margin="3,3,3,3">
35           Arithmetic mean:
36         </Label>
37         <TextBlock
```

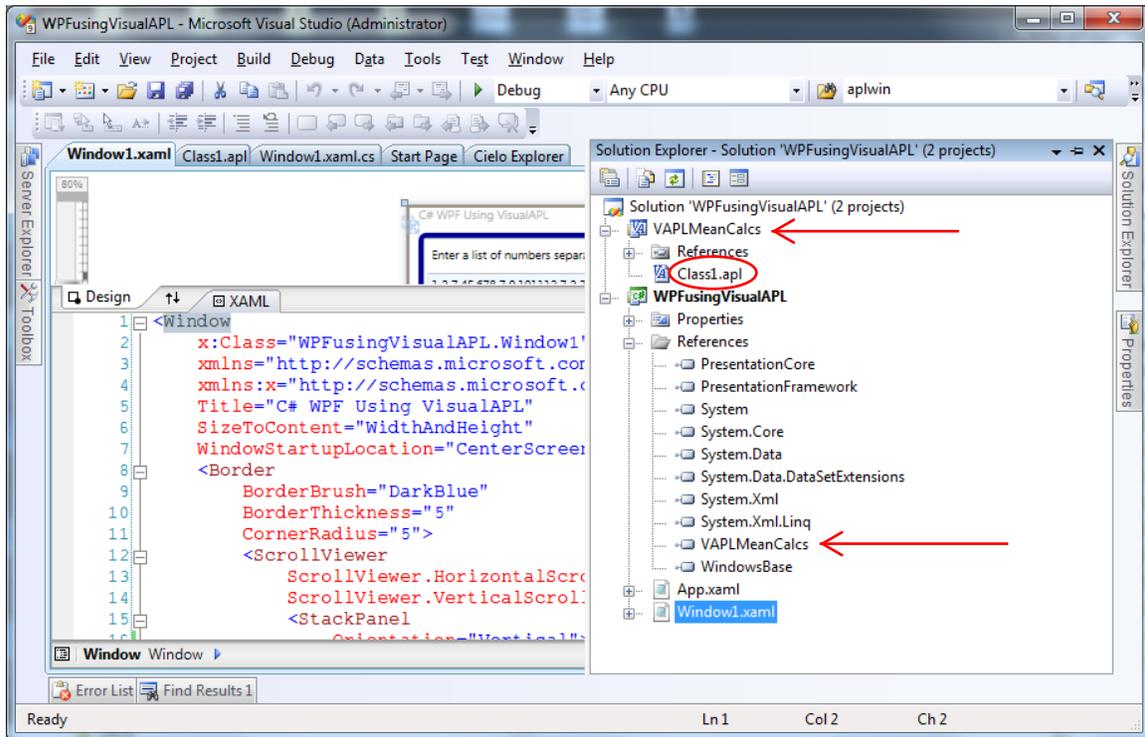
Label Window/Border/ScrollViewer/StackPanel/Label

Error List Find Results 1

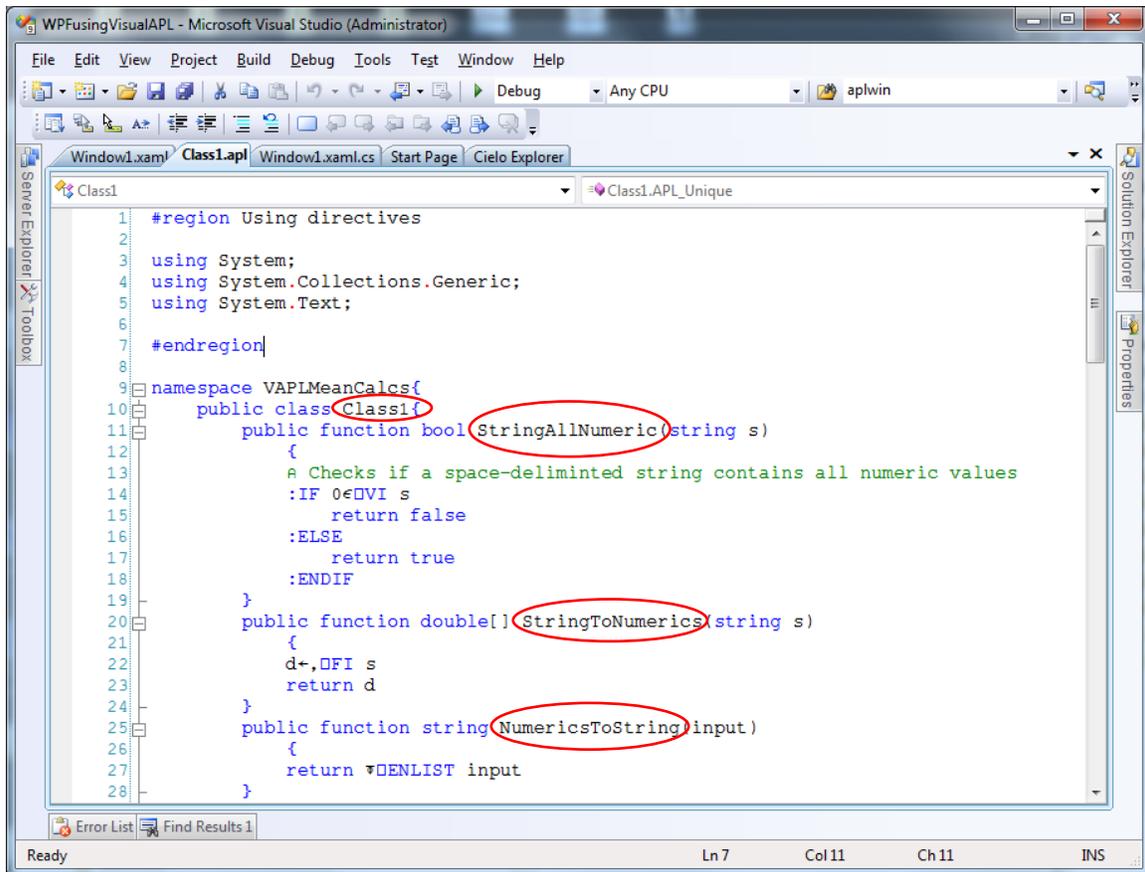
Item(s) Saved Ln 116 Col 35 Ch 35 INS

The VisualAPL project, "VAPLMeanCalcs", in this .Net solution contains a single class, "Class1" which contains all the VisualAPL methods and functions used to support the "business rules" component of the application system.

The C# WPF project in this .Net solution references the VisualAPL project so that an instance of the VisualAPL class can be created and used in the C# WPF project.

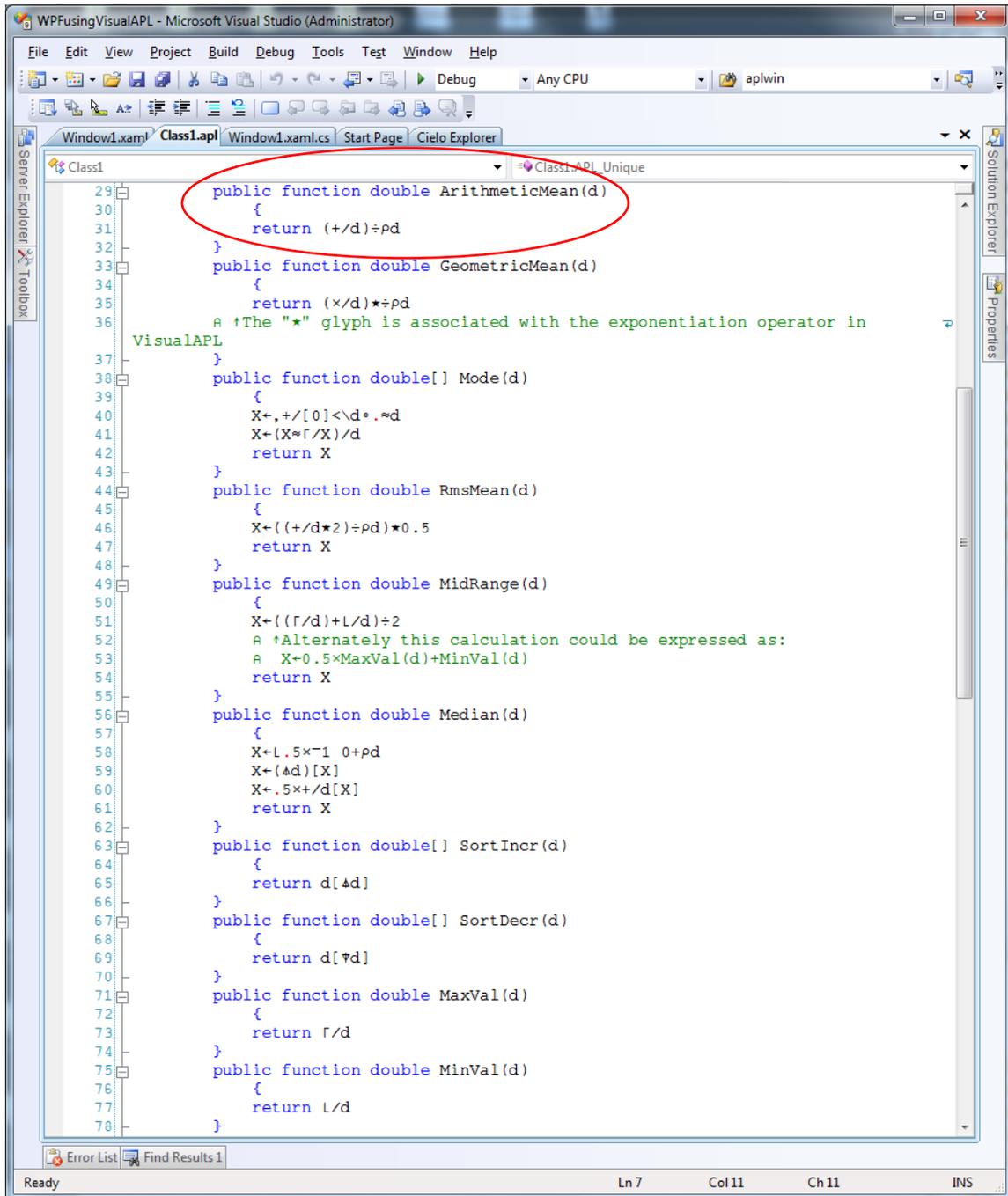


The VisualAPL “Class1” class contains several utility methods that will be used to validate the user-input data, convert the user-input data from string to numeric type and format the calculated values for illustration in the C# WPF GUI.



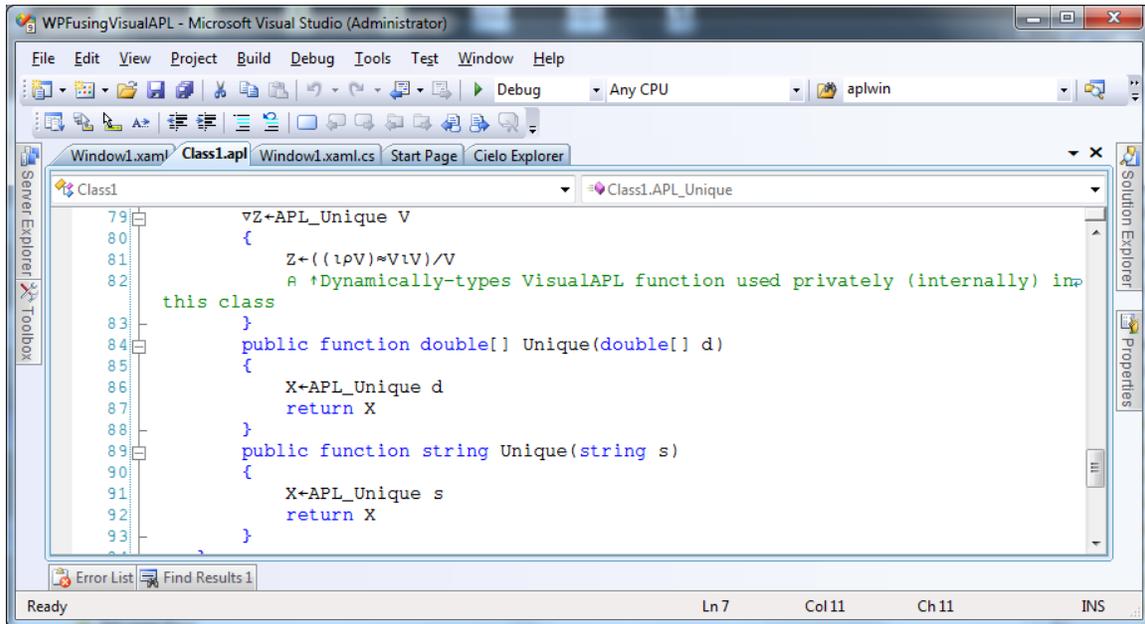
```
1 #region Using directives
2
3 using System;
4 using System.Collections.Generic;
5 using System.Text;
6
7 #endregion
8
9 namespace VAPLMeanCalcs{
10     public class Class1{
11         public function bool StringAllNumeric(string s)
12         {
13             A Checks if a space-delimited string contains all numeric values
14             :IF 0<=DVI s
15                 return false
16             :ELSE
17                 return true
18             :ENDIF
19         }
20         public function double[] StringToNumerics(string s)
21         {
22             d+,DFI s
23             return d
24         }
25         public function string NumericsToString(input)
26         {
27             return ▽DENLIST input
28         }
29     }
30 }
```

The VisualAPL “Class1” class also contains methods to compute the application-specific values from the user input collected by the C# WPF GUI and passed as arguments to the VisualAPL methods. These methods are all ‘public’ so that they will be exposed to the C# WPF GUI. In addition the function header syntax of these methods use the .Net standard format, i.e. “public function result_type function_name(argument_list){...}”, because this format is required for transparent interoperability between .Net assemblies.



```
29 public function double ArithmeticMean(d)
30 {
31     return (+/d)÷pd
32 }
33 public function double GeometricMean(d)
34 {
35     return (×/d)*=pd
36 }
37 A †The "*" glyph is associated with the exponentiation operator in
38 VisualAPL
39 }
40 public function double[] Mode(d)
41 {
42     X+,[0]<\d*.≈d
43     X+(X=Γ/X)/d
44     return X
45 }
46 public function double RmsMean(d)
47 {
48     X+((+/d*2)÷pd)*0.5
49     return X
50 }
51 public function double MidRange(d)
52 {
53     X+((Γ/d)+L/d)÷2
54     A †Alternately this calculation could be expressed as:
55     A X+0.5*MaxVal(d)+MinVal(d)
56     return X
57 }
58 public function double Median(d)
59 {
60     X+L.5×-1 0+pd
61     X+(Δd)[X]
62     X+.5×+/d[X]
63     return X
64 }
65 public function double[] SortIncr(d)
66 {
67     return d[Δd]
68 }
69 public function double[] SortDecr(d)
70 {
71     return d[▽d]
72 }
73 public function double MaxVal(d)
74 {
75     return Γ/d
76 }
77 public function double MinVal(d)
78 {
79     return L/d
80 }
```

In the VisualAPL “Class1” class the implementation of the “Unique” method illustrates how several ‘overloads’ of the same function name can be created differing only in argument composition, e.g. number and data type of the arguments. The action of these overloaded “Unique” methods is supported by a single, dynamically-typed, traditional-format, VisualAPL function, “APL_Unique”.



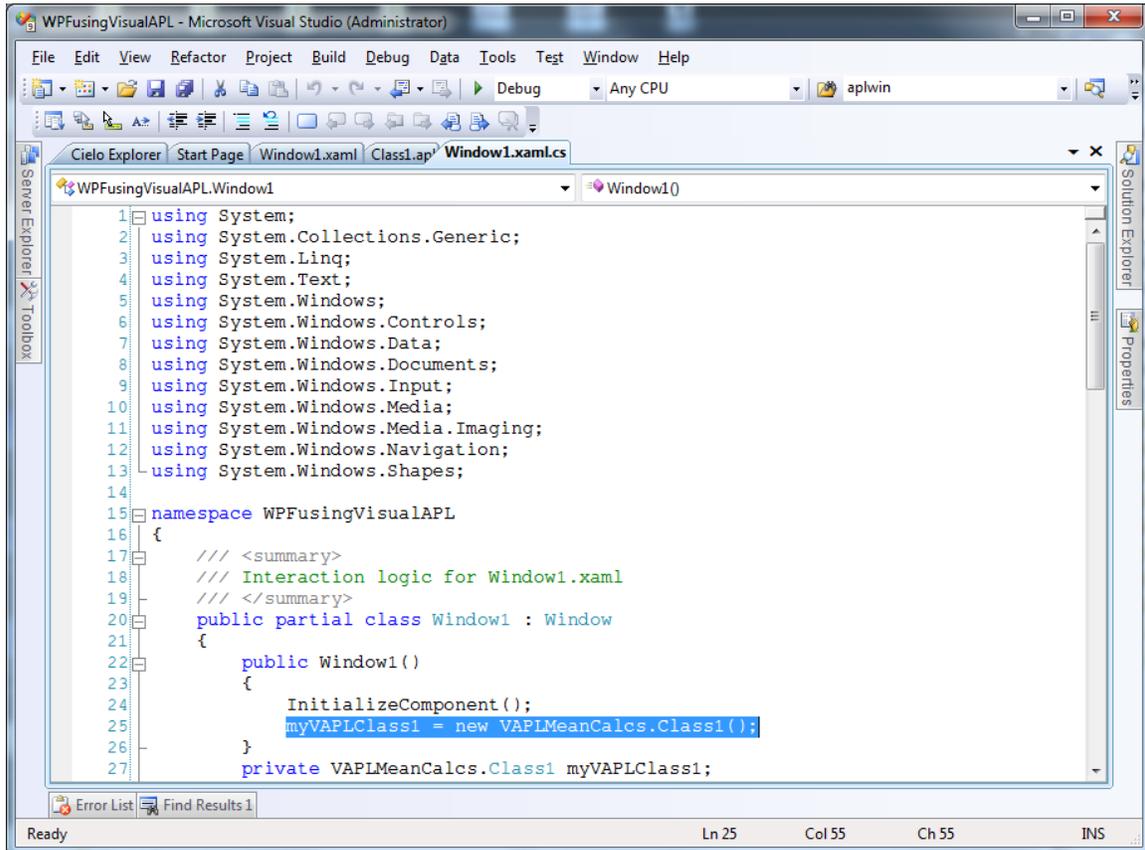
The screenshot shows the Visual Studio IDE with the following details:

- Window Title: WPFusingVisualAPL - Microsoft Visual Studio (Administrator)
- Menu: File, Edit, View, Project, Build, Debug, Tools, Test, Window, Help
- Toolbar: Includes icons for File Explorer, Solution Explorer, and various development actions. A dropdown menu shows 'Debug' and 'Any CPU'. A search box contains 'aplwin'.
- Tab Bar: Shows 'Window1.xaml', 'Class1.apl', 'Window1.xaml.cs', 'Start Page', and 'Cielo Explorer'. 'Class1.apl' is the active tab.
- Solution Explorer: Shows a project named 'Class1' with a file 'Class1.APL_Unique' selected.
- Code Editor: Displays the following VisualAPL code:

```
79      ∇Z+APL_Unique V
80      {
81          Z+((1pV)≈V1V)/V
82          A +Dynamically-types VisualAPL function used privately (internally) im
this class
83      }
84      public function double[] Unique(double[] d)
85      {
86          X+APL_Unique d
87          return X
88      }
89      public function string Unique(string s)
90      {
91          X+APL_Unique s
92          return X
93      }
```
- Server Explorer: Visible on the left side.
- Toolbox: Visible on the left side.
- Properties: Visible on the right side.
- Status Bar: Shows 'Ready', 'Ln 7', 'Col 11', 'Ch 11', and 'INS'.

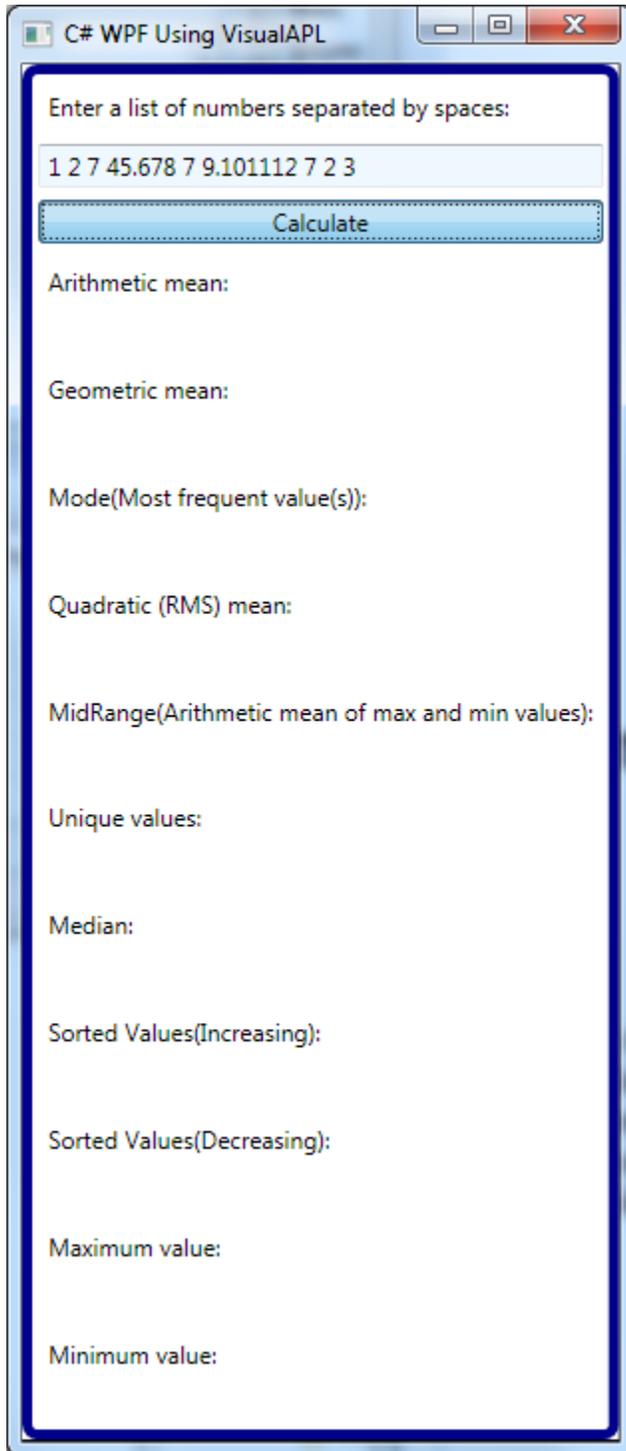
The C# WPF GUI project includes a 'code-behind' file containing the C# source code necessary to control application system interactions between the user and the GUI and between the GUI and the VisualAPL business rules component.

When the C# WPF GUI is initialized, an instance of the VisualAPL "Class1" class is created for use in the application system.

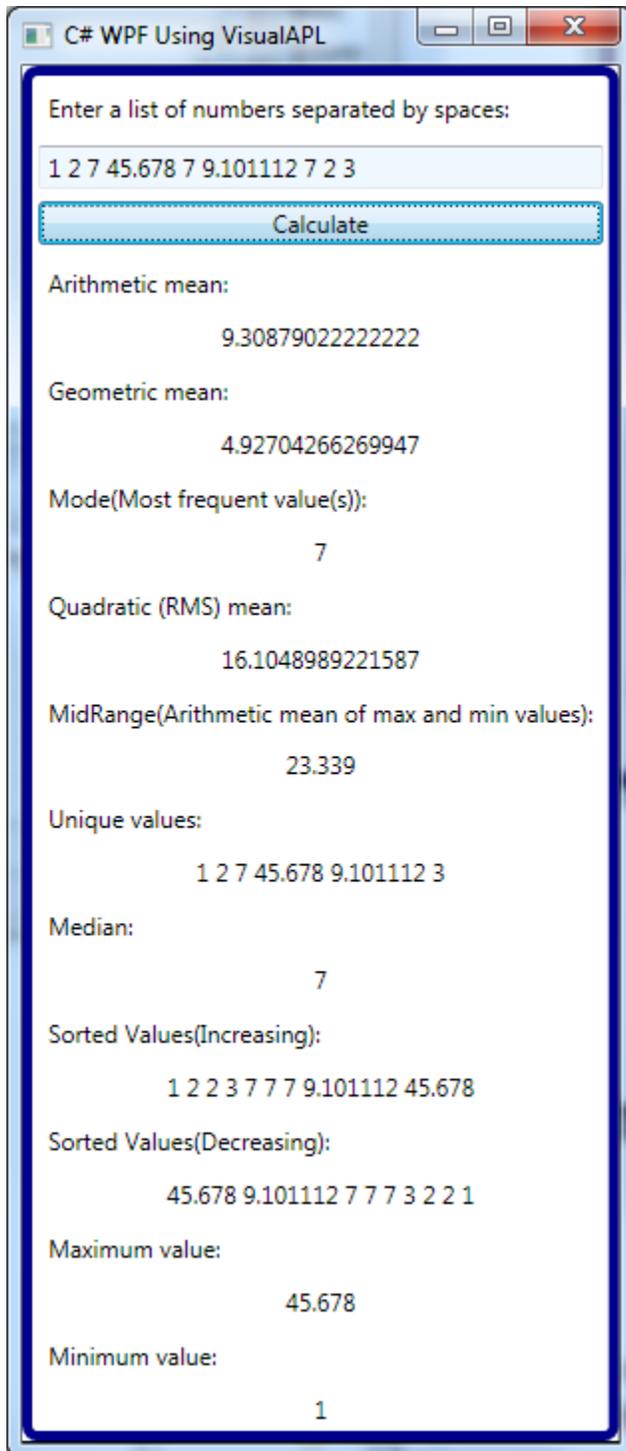


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Windows;
6 using System.Windows.Controls;
7 using System.Windows.Data;
8 using System.Windows.Documents;
9 using System.Windows.Input;
10 using System.Windows.Media;
11 using System.Windows.Media.Imaging;
12 using System.Windows.Navigation;
13 using System.Windows.Shapes;
14
15 namespace WPFusingVisualAPL
16 {
17     /// <summary>
18     /// Interaction logic for Window1.xaml
19     /// </summary>
20     public partial class Window1 : Window
21     {
22         public Window1()
23         {
24             InitializeComponent();
25             myVAPLClass1 = new VAPLMeanCalcs.Class1();
26         }
27         private VAPLMeanCalcs.Class1 myVAPLClass1;
```

In this simple application system the interaction between the user and the GUI is done when the user enters the required input data and clicks the 'Calculate' button to request computation of the application-specific values based on that user input.



In this application system the interaction between the C# WPF GUI component and the VisualAPL business rules component is via the button click event handler method.



The C# WPF GUI code behind file, “Window1.xaml.cs” contains the button click event handler. The processing defined in this event handler occurs when the user clicks the ‘Calculate’ button control in the C# WPF GUI.

The processing involves:

- Disabling the GUI while the event handler processing is being performed
- Validation of the user input for no value using C#
- Validation of the user input for all numeric values using the VisualAPL “StringAllNumeric” method
- Conversion of the user input data from a string to an array of doubles using the VisualAPL “StringToNumerics” method
- Computation of the various application-specific values. C# passes the numeric user input “data” to the applicable VisualAPL method which returns the computed value
- Insertion of the computed values, converted to string, into the C# WPF GUI so it can be seen by the user
- Enabling the GUI after the event handler processing is complete

```
28 private void btnCalculate_Click(object sender, RoutedEventArgs e)
29 {
30     this.IsEnabled = false;
31     string input = txtbxInput.Text;
32     if (String.IsNullOrEmpty(input))
33     {
34         MessageBox.Show("no numbers have been entered!", "Input Error", MessageBoxButton.OK, MessageBoxImage.Error);
35     }
36     else if (!myVAPLClass1.StringAllNumeric(input))
37     {
38         MessageBox.Show("Input values are not all numeric!", "Input Error", MessageBoxButton.OK, MessageBoxImage.Error);
39     }
40     else
41     {
42         double[] data = myVAPLClass1.StringToNumerics(input);
43         txtblkArithmeticMean.Text = myVAPLClass1.ArithmeticMean(data).ToString();
44         txtblkGeometricMean.Text = myVAPLClass1.GeometricMean(data).ToString();
45         txtblkMode.Text = myVAPLClass1.NumericsToString(myVAPLClass1.Mode(data));
46         txtblkQuadraticMean.Text = myVAPLClass1.RmsMean(data).ToString();
47         txtblkMidRange.Text = myVAPLClass1.MidRange(data).ToString();
48         txtblkUniqueVals.Text = myVAPLClass1.NumericsToString(myVAPLClass1.Unique(data));
49         txtblkMedian.Text = myVAPLClass1.Median(data).ToString();
50         txtblkSortIncr.Text = myVAPLClass1.NumericsToString(myVAPLClass1.SortIncr(data));
51         txtblkSortDecr.Text = myVAPLClass1.NumericsToString(myVAPLClass1.SortDecr(data));
52         txtblkMaxVal.Text = myVAPLClass1.MaxVal(data).ToString();
53         txtblkMinVal.Text = myVAPLClass1.MinVal(data).ToString();
54     }
55     this.IsEnabled = true;
56 }
```