Kepware Technologies

ClientAce: Creating a Simple Windows Form Application

July, 2013
Ref. 1.03

Table of Contents

1. Overview

This document intends to demonstrate the basic steps needed to create a Simple Windows Form application in Visual Basic .NET (VB .NET) and C-Sharp (C#).

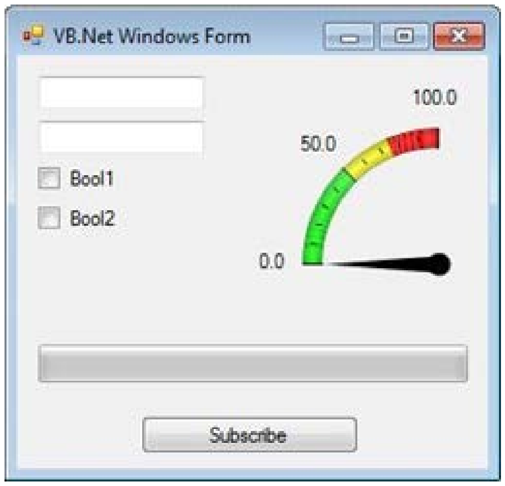
Note: The coding practices used in the following examples are the preference of the developer. The project code was developed in Visual Studio 2010 using ClientAce 4.0. Unless noted otherwise, the same steps and procedures can also be used with ClientAce 3.5.

1.1 Requirements

- Visual Studio 2008 SP1 or higher.
- ClientAce 3.5 or 4.0.
- An OPC Server (like KEPServerEX version 5.x).

2. Creating a Windows Form Application

1. To start, create a new VB .NET or C# project by clicking **File | New Project**.
2. In the list of project templates, locate and select **Windows Forms Application**.
3. Name the project "MyFormApp" and then click **OK**.
4. Next, use the **Toolbox** to drag several controls to the form. In the VB .NET example below, two checkboxes, two textboxes, one progress bar, and a Third-Party gauge control were added.



5. Once finished, add a button that will be used to subscribe to the server's data items. To do so, drag the control from the toolbox to the form.

   Note: Code must be added in order to perform the OPC processes.

2.1 Adding Controls to the Form

1. In **Solution Explorer**, right-click on the **Main.vb** or **Main.cs** form. Then, select **View Designer**.

   Note: The new form will be named "Form1" by default, but can be renamed if desired.

2. Next, add the controls to the form that will be used to display the OPC data (if they haven't been added already). Then, right-click the **Main.vb** or **Main.cs** form and select **View Code**.

2.2 Initializing Global Variables

Global variables can be used by all methods and functions in the project. The following objects should be added and initialized:

- A Server Management Object. This establishes the OPC DA connection to the server.
- A Connection Information Collection Object. This specifies the connection parameters for the Server Management Object connection.
- A Boolean variable. This tests whether the connection to the server succeeded.
- A Server Subscription Handle. This modifies or cancels an active data subscription. More than one handle will be needed when more than one subscription is being performed in the same server connection. The handle will be returned to the application from the server.
- A Client Subscription Handle. This identifies the data change callbacks sent by the server to the application for each subscription. It is specified by the user, and must be unique to each subscription.
- An array of item identifier collection objects. These specify the properties for each item that is added to a particular subscription.

2.2.1 Examples

The VB .NET code is displayed below.

```
Public Class Main
    Dim WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
    Dim connectInfo As New Kepware.ClientAce.OpcDaClient.ConnectInfo
    Dim connectFailed As Boolean
    Dim activeServerSubscriptionHandle As Integer
    Dim itemIdentifiers(4) As Kepware.ClientAce.OpcDaClient.ItemIdentifier
    Dim clientSubscriptionHandle As Integer
```

The C# code is displayed below.

```
public partial class Main : Form
{
    Kepware.ClientAce.OpcDaClient.DaServerMgt DAserver = new
Kepware.ClientAce.OpcDaClient.DaServerMgt();
    Kepware.ClientAce.OpcDaClient.ConnectInfo connectInfo = new
Kepware.ClientAce.OpcDaClient.ConnectInfo();
    bool connectFailed;
    int activeServerSubscriptionHandle;
    int clientSubscriptionHandle;
    ItemIdentifier[] itemIdentifiers = new ItemIdentifier[5];
```

2.3 Adding the Form Load Actions

In this example, a connection to the OPC Server will be established when the form loads.

1. In the project, double-click on the **Windows Form**. The shell code for Form Load Events will be added automatically by Visual Studio Intellisense.
2. Next, add the code needed to initialize the objects for connection to the server.
   - **URL**. This string object specifies the type of OPC Connection, the location of the server, and the Server ID. OPC Connection options include DA, XML-DA, and UA.
     - **Example OPC DA URL:** The OPC DA URL for connection is *opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-6F3B160F4397}*. The Class ID included in the URL is optional.
     - **Example OPD UA URL:** The OPC UA Endpoint URL (with security) for connection is *opc.tcp://127.0.0.1:49320*. The UA server that is being connected must allow connections that have not been secured; however, users can create secure connections through additional coding.
   - **ConnectionInfo**. This object specifies the following parameters:
     - **LocaleID:** This parameter specifies the client application's language, number, and date format preferences.
     - **KeepAliveTime:** This parameter specifies the interval at which the server status will be checked.
     - **RetryAfterConnectionError:** This parameter specifies whether the server object will attempt to reconnect to the server if the connection fails after the initial connection.
     - **RetryInitialConnection:** This parameter specifies whether the server object will retry a connection attempt if the initial attempt fails.
     - **ClientName:** This parameter allows the client application to provide additional identity information for the client connection in the form of a plain language identifier. This parameter is new in ClientAce version 4.0.
   - **ServerHandle**. This unique identifier is assigned by the client application for each server object with which it initializes and establishes a connection to the server.
3. Once finished, set the Connection Failed variable to False.

Note: The VB .NET code is displayed below.

```
' Define the server connection URL
Dim url As String = "opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-
6F3B160F4397}"

' Initialize the connect info object data
connectInfo.LocalId = "en"
connectInfo.KeepAliveTime = 1000
connectInfo.RetryAfterConnectionError = True
connectInfo.RetryInitialConnection = False
connectInfo.ClientName = "Simple VB Form Example"
connectFailed = False

'define a client handle for the connection
Dim ServerHandle As Integer = 1
```

Note: The C# code is displayed below.

```
// Define the server connection URL
string url = "opcda://localhost/Kepware.KEPServerEX.V5/{B3AF0BF6-4C0C-4804-A122-
6F3B160F4397}";

// Initialize the connect info object data
connectInfo.LocalId = "en";
connectInfo.KeepAliveTime = 1000;
connectInfo.RetryAfterConnectionError = true;
connectinfo.RetryInitialConnection = false;
connectInfo.ClientName = "CS Simple Client";
connectFailed = false;

///define a client handle for the connection
/int clientHandle = 1;
```

4. Next, add the code needed to manage the OPC DA connection to the server.
5. Then, set a **Try…Catch Block** to catch and handle any exceptions that are encountered during the connection attempt.

   Note: The VB .NET code is displayed below.

```
'Call the API connect method:
Try
    'Try to connect
    daServerMgt.Connect(url, ServerHandle, connectInfo, connectFailed)

    ' Handle result:
    If connectFailed Then
        ' Tell user connection attempt failed:
        MsgBox("Connect failed")
    End If
Catch ex As Exception
    MsgBox("Handled Connect exception. Reason: " & ex.Message)

    ' Make sure following code knows connection failed:
    connectFailed = True
    'Stop processing
    Exit Sub
End Try
```

Note: The C# code is displayed below.

```
//Try to connect with the API connect method:
try
    {
    DAserver.Connect (url, clientHandle, ref connectInfo, out connectFailed);
    }
catch (Exception ex)
    {
    MessageBox.Show ("Handled Connect exception. Reason: " + ex.Message);

    // Make sure following code knows connection failed:
    connectFailed = true;
    }

// Handle result:
  if (connectFailed)
    {
    // Tell user connection attempt failed:
    MessageBox.Show ("Connect failed");
    }
//Subscribe to events
SubscribeToOPCDAServerEvents();
```

2.3.1 Advising the Server Management Object for Callbacks

The C# code has an additional process that it calls to advise the server or object for callbacks. The OPC DA specification allows users to acquire data through three methods: Synchronous, Asynchronous, and Unsolicited Asynchronous. The Asynchronous methods require that the Server Management Object advise or subscribe for change events.

Applications created using C# do this separately from the Server Management Object declaration. A separate function is used to subscribe to events.

```
//Subscribe to server events
  private void SubscribeToOPCDAServerEvents()
  {
  //DAserver.ReadCompleted += new
DaServerMgt.ReadCompletedEventHandler(DAserver_ReadCompleted);

  //DAServer.WriteCompleted += new
DaServerMgt.WriteCompletedEventHandler(DAserver_WriteCompleted);

  DAserver.DataChanged += new DaServerMgt.DataChangedEventHandler(DAserver_DataChanged);

  DAserver.ServerStateChanged += new
DaServerMgt.ServerStateChangedEventHandler(DAserver_ServerStateChanged);
  }
```

Note: Applications created with VB .NET do this when declaring the object. The "WithEvents" keyword specifies that the named object instance can raise events. The example below was done in the form declarations.

```
Dim WithEvents daServerMgt As New Kepware.ClientAce.OpcDaClient.DaServerMgt
```

2.4 Subscribing for Data

Items must be added through a subscription in order to get data in an Unsolicited Asynchronous callback event (or a DataChanged Event) in ClientAce. The examples below use a button to initiate the subscription. The button makes two calls. The first call is to the Subscribe2Data function, where items are subscribed but inactive. The second call is to the ModifySubscription function, which sets the subscription active.

Note: The VB .NET code is displayed below.

```
Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
Button1.Click
   'Subscribe to the opc data items
    Subscibe2Data()
    SubscriptionModify(True)
End Sub
```

Note: The C# code is displayed below.

```
private void button1_Click(object sender, EventArgs e)
{
    //Call the method for subscribing to data
    Subscribe2Data();
    ModifySubscription(true);
}
```

The Subscribe2Data function does the following:

- Initializes the ClientSubscription handle variable. This identifies data in the DataChanged Event that is specific to this subscription. Each subscription should have a unique handle.
- Initializes the Active variable to False. This specifies whether the subscribed items will be actively polled from the server.
- Initializes the UpdateRate variable. This specifies the rate at which the subscribed items will be polled.
- Initializes the Deadband variable. This specifies the amount of change that is required before a DataChange Event is sent to the client. To disable Deadband, enter zero.
- Initializes the RevisedUpdateRate variable. This value is returned from the server to indicate the update rate that the server will support. For example, although a client might request a rate of 1024 milliseconds, the server might revise it to 1030 milliseconds.
- Initializes each element of the ItemIdentifiers array as the ClientAce ItemIdentifier and its attributes are initialized. The following four items will be added to the subscription (indexes 0-4):
    - o **ItemName:** This attribute is the fully-qualified OPC Item Name for the desired item in the server.
    - o **ClientHandle:** This attribute is a unique number that identifies the item in callbacks from the server.
    - o **DataType:** This attribute specifies the item’s data type. Users can add items with or without specifying the data type. In both cases, the server will return a **RevisedDataType** for each item: this attribute is also part of the ItemIdentifier object.

Once all the variables and array elements have been initialized, users can subscribe to the server for data using a **Try…Catch…** so that any exception errors will be caught and handled.

Once the subscription is complete, the ResultID for the first item will be checked to see whether it subscribed successfully. If it was, the application will move on. In a live production application, every item that is subscribed should be checked for success.

Note: The VB .NET code is displayed below.

```
Sub Subscibe2Data()
    'initialize the client subscription handle
    clientSubscriptionHandle = 1
    'Parameter to specify if the subscription will be added as active or not
    Dim active As Boolean = False

    ' The updateRate parameter is used to tell the server how fast we
    ' would like to see data updates.
    Dim updateRate As Integer = 1000

    ' The deadband parameter specifies the minimum deviation needed
    ' to be considered a change of value. 0 is disabled
    Dim deadBand As Single = 0

    ' The revisedUpdateRate parameter is the actual update rate that the
    ' server will be using.
    Dim revisedUpdateRate As Integer

    'initialize the itemIdentifier values
    itemIdentifiers(0) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
    itemIdentifiers(0).ItemName = "Channel1.Device1.Bool1"
    itemIdentifiers(0).ClientHandle = 0
    itemIdentifiers(0).DataType = Nothing

    itemIdentifiers(1) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
    itemIdentifiers(1).ItemName = "Channel1.Device1.Bool2"
    itemIdentifiers(1).ClientHandle = 1
    itemIdentifiers(1).DataType = Nothing

    itemIdentifiers(2) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
    itemIdentifiers(2).ItemName = "Channel1.Device1.Short1"
    itemIdentifiers(2).ClientHandle = 2
    itemIdentifiers(2).DataType = Nothing

    itemIdentifiers(3) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
    itemIdentifiers(3).ItemName = "Channel1.Device1.Short2"
    itemIdentifiers(3).ClientHandle = 3
    itemIdentifiers(3).DataType = Nothing

    itemIdentifiers(4) = New Kepware.ClientAce.OpcDaClient.ItemIdentifier
    itemIdentifiers(4).ItemName = "Channel1.Device1.Long5"
    itemIdentifiers(4).ClientHandle = 4
    itemIdentifiers(4).DataType = Nothing

    'Call the Subscribe API method:
    Try
        daServerMgt.Subscribe(clientSubscriptionHandle, active, updateRate,
revisedUpdateRate, deadBand, itemIdentifiers, activeServerSubscriptionHandle)

        ' Check item result ID:
        If itemIdentifiers(0).ResultID.Succeeded = False Then

            ' Show a message box if an item could not be added to subscription.
            ' You would probably use some other means to alert
            ' the user to failed item enrollments in an actual application.
            MsgBox("Failed to add item " & itemIdentifiers(0).ItemName & " to
subscription")
        End If
    Catch ex As Exception
        MsgBox("Handled Subscribe exception. Reason: " & ex.Message)
    End Try

End Sub
```

Note: The C# code is displayed below.

```
private void  Subscribe2Data()
{
    // Define parameters for Subscribe method:
    int itemIndex;

    //initialize the client subscription handle
    clientSubscriptionHandle = 1;

    //Parameter to specify if the subscription will be added as active or not
    bool active = false;

    // The updateRate parameter is used to tell the server how fast we
    // would like to see data updates.
    int updateRate = 1000;

    // The deadband parameter specifies the minimum deviation needed
    // to be considered a change of value. 0 is disabled
    Single deadBand = 0;

    // The revisedUpdateRate parameter is the actual update rate that the
    // server will be using.
    int revisedUpdateRate;

    //Initialize the item identifier values
    itemIdentifiers[0] = new ItemIdentifier();
    itemIdentifiers[0].ClientHandle = 0;
    itemIdentifiers[0].DataType = Type.GetType("System.Boolean");
    itemIdentifiers[0].ItemName = "Channel1.Device1.Bool1";

    itemIdentifiers[1] = new ItemIdentifier();
    itemIdentifiers[1].ClientHandle = 1;
    itemIdentifiers[1].DataType = Type.GetType("System.Boolean");
    itemIdentifiers[1].ItemName = "Channel1.Device1.Bool2";

    itemIdentifiers[2] = new ItemIdentifier();
    itemIdentifiers[2].ClientHandle = 2;
    itemIdentifiers[2].DataType = Type.GetType("System.Int16");
    itemIdentifiers[2].ItemName = "Channel1.Device1.Short1";

    itemIdentifiers[3] = new ItemIdentifier();
    itemIdentifiers[3].ClientHandle = 3;
    itemIdentifiers[3].DataType = Type.GetType("System.Int16");
    itemIdentifiers[3].ItemName = "Channel1.Device1.Short2";

    itemIdentifiers[4] = new ItemIdentifier();
    itemIdentifiers[4].ClientHandle = 4;
    itemIdentifiers[4].DataType = Type.GetType("System.Int32");
    itemIdentifiers[4].ItemName = "Channel1.Device1.Long5";

    try
        {
        DAserver.Subscribe(clientSubscriptionHandle, active, updateRate, out
revisedUpdateRate, deadBand, ref itemIdentifiers, out activeServerSubscriptionHandle);

        for (itemIndex = 0; itemIndex <= 4; itemIndex++)
          {
            if (itemIdentifiers[itemIndex].ResultID.Succeeded == false)
            {
                MessageBox.Show("Failed to add item " +
itemIdentifiers[itemIndex].ItemName + " to subscription");
            }
          }
        }
    catch (Exception ex)
        {
            MessageBox.Show("Handled Subscribe exception. Reason: " + ex.Message);
        }
}
```

2.5 Setting Items Active for Polling

Items may be set active for polling after they have been added with a subscription modification function, which is called from the button click after subscribing to the server for items. The function is passed, and the Boolean value that is used is the SubscriptionModify Method of the Server Management Object. This method uses the ActiveServerSubscription Handle (which was returned from the server in the data subscription) and the Action value that was sent when it was called. In this case, the Action value is True.

Note: The VB .NET code is displayed below.

```
Sub SubscriptionModify(ByVal action As Boolean)
    'Set the subscription active or inactive
    daServerMgt.SubscriptionModify(activeServerSubscriptionHandle, action)
End Sub
```

Note: The C# code is displayed below.

```
private void ModifySubscription(bool action)
{
     //Modify the Subscription to set it active
     DAserver.SubscriptionModify(activeServerSubscriptionHandle, action);
}
```

2.6 Handling DataChanged Events

If all the items were successfully subscribed, the server should now be polling the device at the update rate specified in the subscription. Any of the subscribed items that change in value or quality will be sent to the client application in a DataChanged Event. In the code examples below, the events are received and then processed through an event handler on a new thread through the BeginInvoke() Method.

Note: The VB .NET code is displayed below.

```
Private Sub daServerMgt_DataChanged(clientSubscription As Integer, allQualitiesGood As Boolean,
noErrors As Boolean, itemValues() As Kepware.ClientAce.OpcDaClient.ItemValueCallback) Handles
daServerMgt.DataChanged
    BeginInvoke(New Kepware.ClientAce.OpcDaClient.DaServerMgt.DataChangedEventHandler(AddressOf
DataChanged), New Object() {clientSubscription, allQualitiesGood, noErrors, itemValues})
End Sub
```

Note: The C# code is displayed below.

```
public void DAserver_DataChanged(int clientSubscription, bool allQualitiesGood, bool noErrors,
ItemValueCallback[] itemValues)
{
     object[] DCevHndlrArray = new object[4];
     DCevHndlrArray[0] = clientSubscription;
     DCevHndlrArray[1] = allQualitiesGood;
     DCevHndlrArray[2] = noErrors;
     DCevHndlrArray[3] = itemValues;
     BeginInvoke(new DaServerMgt.DataChangedEventHandler(DataChanged), DCevHndlrArray);
}
```

In the invoked DataChange handler, each item returned in the callback will be evaluated. The objects being used to display values on the form will be updated. The unique client handle will also be checked for each item in order to ensure that the proper object is updated.

Note: The VB .NET code is displayed below.

```
Private Sub DataChanged(ByVal clientSubscription As Integer, ByVal allQualitiesGood As Boolean,
ByVal noErrors As Boolean, ByVal itemValues() As
Kepware.ClientAce.OpcDaClient.ItemValueCallback)

    Dim itemValue As Kepware.ClientAce.OpcDaClient.ItemValueCallback

    For Each itemValue In itemValues
            'Update a form control based on the returned client handle. The values could be
            'Null if the quality is bad
        Select Case itemValue.ClientHandle
            Case 0
                If IsNothing(itemValue.Value) Then
                    CheckBox1.Checked = vbNull
                Else
                    CheckBox1.Checked = itemValue.Value
                End If
            Case 1
                If IsNothing(itemValue.Value) Then
                    CheckBox2.Checked = vbNull
                Else
                    CheckBox2.Checked = itemValue.Value
                End If
            Case 2
                If IsNothing(itemValue.Value) Then
                    TextBox1.Text = "Unknown"
                Else
                    TextBox1.Text = itemValue.Value.ToString()
                End If
            Case 3
                If IsNothing(itemValue.Value) Then
                    TextBox2.Text = "Unknown"
                Else
                    TextBox2.Text = itemValue.Value.ToString()
                End If
            Case 4
                If IsNothing(itemValue.Value) Then
                    ProgressBar1.Value = vbNull
                    GaugeAngular1.Value.AsInteger = vbNull
                Else
                    ProgressBar1.Value = itemValue.Value
                    GaugeAngular1.Value.AsInteger = itemValue.Value
                End If
        End Select
    Next
End Sub
```

Note: The C# code is displayed below.

```
public void DataChanged(int clientSubscription, bool allQualitiesGood, bool noErrors,
ItemValueCallback[] itemValues)
{
    try
    {
       For each (ItemValueCallback itemValue in itemValues)
       {
           int itemIndex = (int)itemValue.ClientHandle;
           switch (itemIndex)
           {
               case 0:
                   if (itemValue.Value == null)
                   {
                       checkBox1.Checked = false;
                   }
                   else
                   {
                       checkBox1.Checked = Convert.ToBoolean(itemValue.Value);
                   }
                   break;
               case 1:
                   if (itemValue.Value == null)
                   {
                       checkBox2.Checked = false;
                   }
                   else
                   {
                       checkBox2.Checked = Convert.ToBoolean(itemValue.Value);
                   }
                   break;
               case 2:
                   if (itemValue.Value == null)
                   {
                       textBox1.Text = "Unknown";
                   }
                   else
                   {
                       textBox1.Text = itemValue.Value.ToString();
                   }
                   break;
               case 3:
                   if (itemValue.Value == null)
                   {
                       textBox2.Text = "Unknown";
                   }
                   else
                   {
                       textBox2.Text = itemValue.Value.ToString();
                   }
                   break;
               case 4:
                   if (itemValue.Value == null)
                   {
                       progressBar1.Value = 0;
                       gaugeAngular1.Value.AsInteger = 0;
                   }
                   else
                   {
                       progressBar1.Value = Convert.ToInt32(itemValue.Value);
                       gaugeAngular1.Value.AsInteger = Convert.ToInt32(itemValue.Value);
                   }
                   break;
           }
       }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Handled Data Changed exception. Reason: " + ex.Message);
    }
}
```

2.7 Handling Changes to the Server State

This project also handles changes in the server state. The server state will be checked using the Keep Alive interval specified in the server object's Connect Method. This event is best processed by using a handler in another thread.

Note: The VB .NET code is displayed below.

```
Private Sub daServerMgt_ServerStateChanged(ByVal clientHandle As Integer, ByVal state As
Kepware.ClientAce.OpcDaClient.ServerState) Handles daServerMgt.ServerStateChanged

    BeginInvoke(New
Kepware.ClientAce.OpcDaClient.DaServerMgt.ServerStateChangedEventHandler(AddressOf
ServerStateChanged), New Object() {clientHandle, state})

End Sub
```

Note: The C# code is displayed below.

```
public void DAserver_ServerStateChanged(int clientHandle, ServerState state)
{
    object[] SSCevHndlrArray = new object[2];
    SSCevHndlrArray[0] = clientHandle;
    SSCevHndlrArray[1] = state;
    BeginInvoke(new DaServerMgt.ServerStateChangedEventHandler(ServerStateChanged),
SSCevHndlrArray);
}
```

The server state's significant events are as follows:

- **Disconnected:** This event has an enumerated value of 1.
- **ErrorShutdown:** This event has an enumerated value of 2. It provides the application time to cleanly disconnect from the server.
- **ErrorWatchDog:** This event has an enumerated value of 3.
- **Connected:** This event has an enumerated value of 4.

The server object has additional properties (such as IsConnected and ServerState) that are generally updated after the ServerStateChanged Event. As such, they are useful for status but not for condition warnings.

Note: The VB .NET code is displayed below.

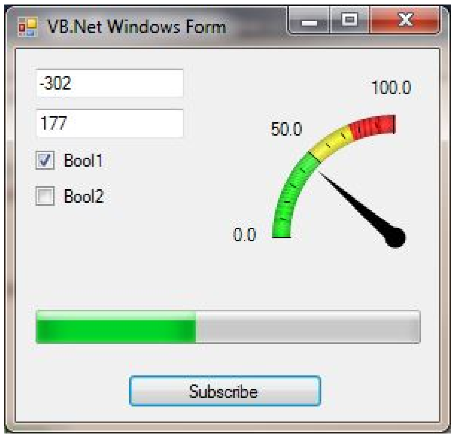
```
Private Sub ServerStateChanged(ByVal clientHandle As Integer, ByVal state As
Kepware.ClientAce.OpcDaClient.ServerState)
    Try
        '~~ Process the callback information here.
        Select Case state
            Case 1
                MsgBox("Server is Disconnected")
            Case 2
                'Unsubscribe()
                'DisconnectOPCServer()
                MsgBox("The server is shutting down. The client has automatically tried to
reconnect.")
            Case 3
                ' Retry Connection error in connection info is checked so the API will continue
                ' trying to reconnect until the project is closed.
                MsgBox("Server connection has been lost. Client will keep attempting to
reconnect.")
            Case 4
                'MsgBox("Connected to server.")
        End Select
    Catch ex As Exception
        '~~ Handle any exception errors here.
    End Try
End Sub
```

Note: The C# code is displayed below.

```
public void ServerStateChanged(int clientHandle, ServerState state)
{
    try
    {
        //~~ Process the callback information here.
        switch (state)
        {
            case ServerState.ERRORSHUTDOWN:
                //Unsubscribe();
                //DisconnectOPCServer();
                MessageBox.Show("The server is shutting down. The client has automatically 
disconnected.");
                break;
            case ServerState.ERRORWATCHDOG:
                MessageBox.Show("Server connection has been lost. Client will keep attempting 
to reconnect.");
                break;
            case ServerState.CONNECTED:
                MessageBox.Show("ServerStateChanged, connected");
                break;
            case ServerState.DISCONNECTED:
                MessageBox.Show("ServerStateChanged, disconnected");
                break;
            default:
                MessageBox.Show("ServerStateChanged, undefined state found.");
                break;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Handled Server State Changed exception. Reason: " + ex.Message);
    }
}
```

3. Testing the Project

1. To test the project, run it in Debug Mode. To do so, press the **F5** key while in the **Visual Studio Development Environment**. The example below displays VB .NET.



2. If the server is running, data will be displayed in the form.

4.Summary

At this time, users should have a better understanding of how to create a simple ClientAce Windows Form application.