Beijer ELECTRONICS

iX Developer

Third Party Controls

English

iX Developer Addition, 2010-06

Third Party Controls in iX Developer

Foreword

iX Developer allows using and creating third party controls in order to enhance application functionality and additional customization. This document describes different technologies and includes configuration examples.

To understand and use all the information in this document, .Net development skills are required.

Contents

1 Target Platform

Different technologies are used for third party controls depending on the target platform for the iX Developer application. The target can be either PC or Windows CE.

Windows CE has no support for vector graphic (WPF) and only uses .Net Compact Framework which is a subset or the .Net Framework used on a PC. Windows CE does not natively support GDI+, so GDI+ related functionality was removed from .Net Compact Framework.

1.1 PC Target

Two different technologies can be used for a PC target:

- Standard Windows forms and GDI+
- WPF (Windows Presentation Foundation)

WPF uses vector graphics, and the appearance of the control is described in XAML. Since iX Developer is a WPF application, it is recommended to use WPF when developing customized controls or user controls for a PC target. Controls developed in WPF can bind to a tag value in iX Developer, in opposite to Windows forms controls, that cannot be bound to tag values.

1.2 Windows CE Target

Windows CE only uses the .Net Compact Framework (a subset or the .Net Framework used on a PC), and does not support vector graphics (WPF). Windows CE does not natively support GDI+, so GDI+ related functionality was removed from the .Net Compact Framework.

1.3 Limitations

Some of the limitations regarding third party controls are listed below:

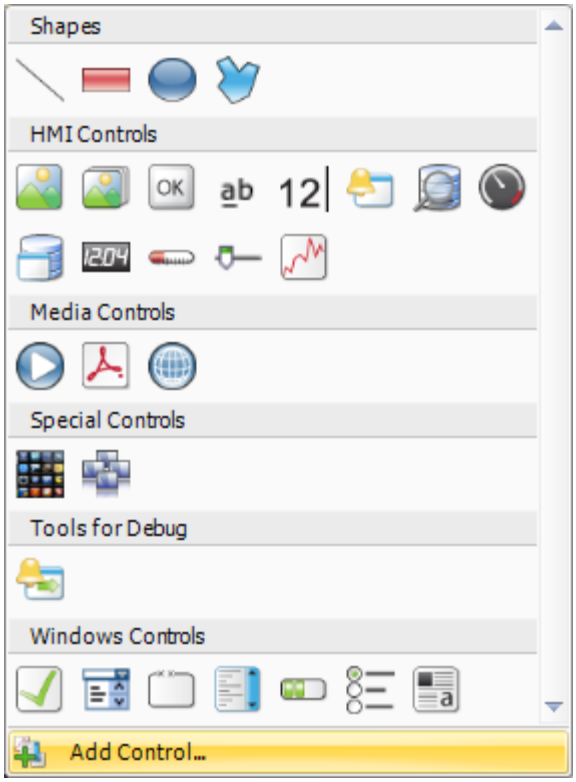
- Control Designers (a designer class that can extend design time support) are currently not supported.
- TypeConverters in a separate design dll are not supported.
- Complex property editing in the property grid is not supported. All complex properties have to be set up in script.
- .Net Compact Framework controls can include design dll and so called AssmetaData dll to handle attributes that are not supported in Windows CE. Currently this is not supported by iX Developer. Because of this, it is important to always test the code on the target platform.
- The Script Editor allows scripting against properties and methods that are not supported in Windows CE. Because of this, it is important to always test the code on the target platform.

2 Adding Controls to the iX Developer Toolbox

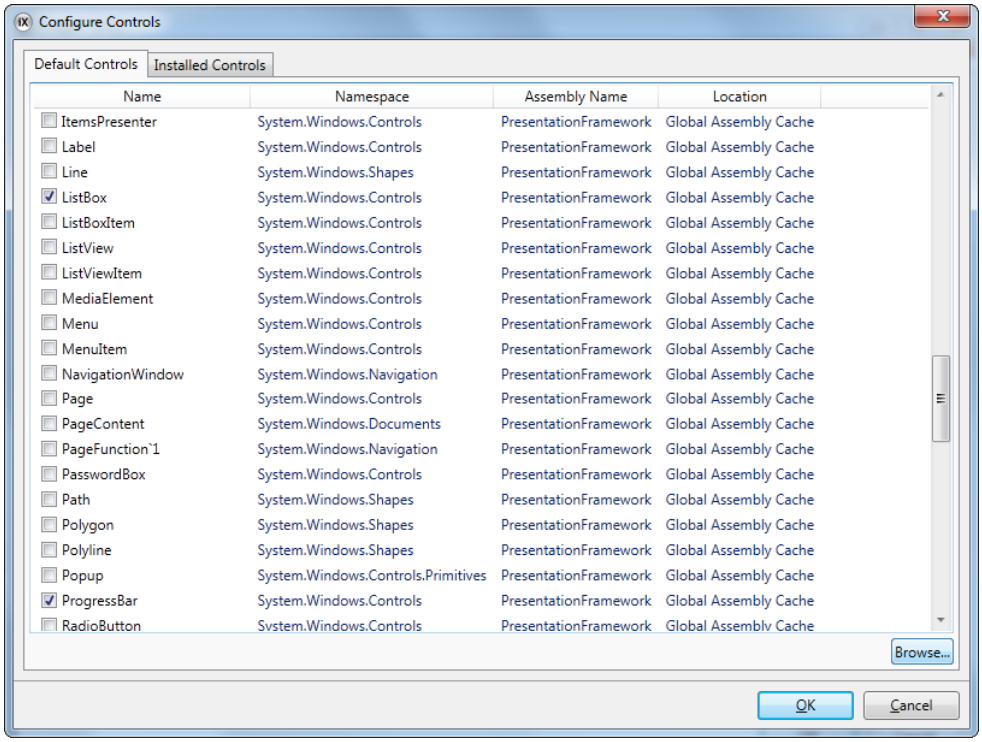
Third party controls can be added to the Objects toolbox in iX Developer, following the steps below:

1. Select the **Objects** group on the **Home** ribbon tab, and fully expand the Objects toolbox by clicking the lower right arrow.

2. Click **Add Control**.

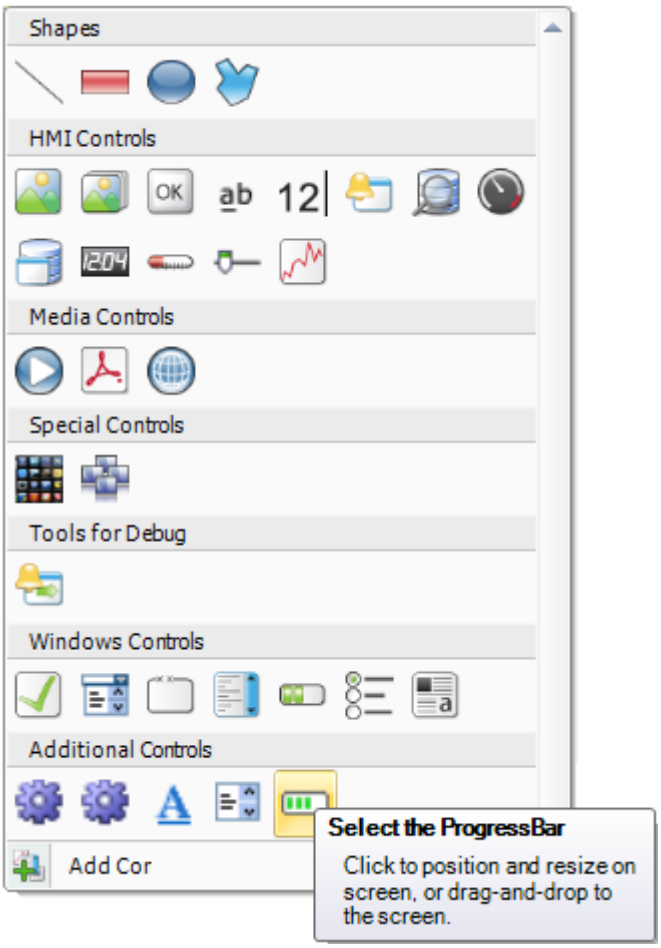


3. Select controls to add among the default controls, or click **Browse** to add customized controls.



4. Click **OK**.

The added controls are now available under **Additional Controls** in the Objects toolbox.



2.1 Default Controls and Installed Controls

Default controls include controls added by the user and the .Net 4 controls installed with the .Net Framework.

Installed controls include all controls that are installed in the GAC (Global Assembly Cache) on your computer.

Note:
Third party controls that are used in a project are not copied to the project folder. This means that it is not possible to open a project with third party controls on another PC without installing the controls. But the application will work in runtime on another target, since references are copied to the output folder when building the project.

---

3 WPF Controls

WPF (Windows Presentation Foundation) uses vector graphics, and the appearance of the control is described in XAML. Since iX Developer is a WPF application, it is recommended to use WPF when developing customized controls or user controls for a PC target. Controls developed in WPF can bind to a tag value in iX Developer.

User controls and custom controls are supported in WPF.

3.1 WPF User Controls

A WPF user control can be described as a composition of different user interface controls. Creating a WPF user control is similar to creating a window:

- You have a XAML file and C# class file for a user control.
- The class file extends the user control class, adding additional behavior and properties.
- The XAML file encapsulates the composing controls; styles, templates, animations and whatever necessary for "Look & Feel".

Since the WPF user control is a just composition, it is really easy to create. It does not require a lot of WPF UI model knowledge.

3.2 WPF Custom Controls

WPF custom controls are more flexible, but are more complicated than a user control, and require a profound understanding of the WPF user interface model.

- A number of certain user interface controls, such as button, progress bar or speedometer has to be extended.
- The appearance of the custom control has to be defined in XAML, as the custom control itself has no look.

Most of the controls in iX Developer are custom controls, which makes it possible to restyle them to various different layouts without changing the code files; just the XAML.

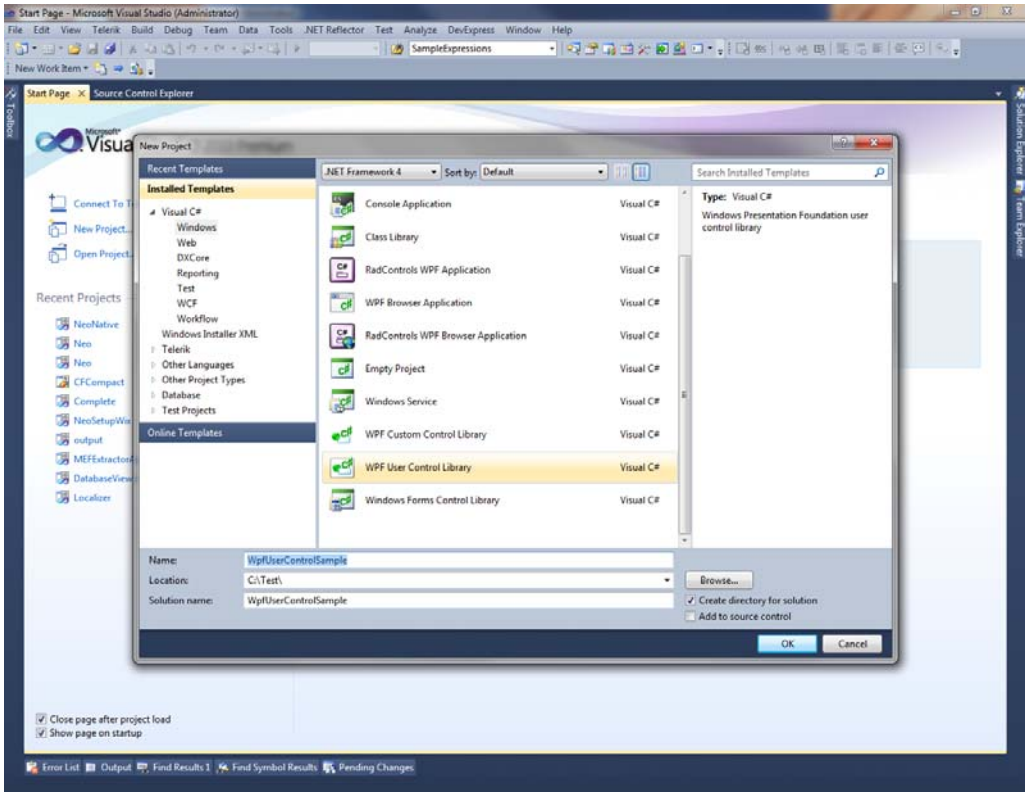
A rounded meter in different styles

3.3 Creating a WPF User Control with Tag Connection

The following example describes how to create a WPF user control that can be connected to a tag.

The complete code is included at the end of the example.

1. Start Visual Studio to create a new project, and select **WPF User Control Library**.



2. Add [DefaultProperty("Value")] to the class, to define which property the tag should set when then value is set.
3. Add a dependency property with same name as the attribute above: static readonly DependencyProperty ValueProperty;
4. Add a static constructor and register the dependency property.
5. Create a Value property of type object.
6. Add a TextBox to the the user control.
7. Add a binding to the TextProperty and bind to the Value Property.

```
<TextBox Text="{Binding Value, ElementName=userControl, FallbackValue=0}"
Name="textBlock1" Background="#FFF7EFEF" TextAlignment="Center" />
```

8. Remember to change ElementName to the name of your control.
9. Compile and test by adding the control to the iX Developer toolbox.

Note:
When an update is made, the existing control must be updated under C:\Users\Public\Documents\Beijer Electronics AB\iX Developer\Thirdparty\.

Example Code

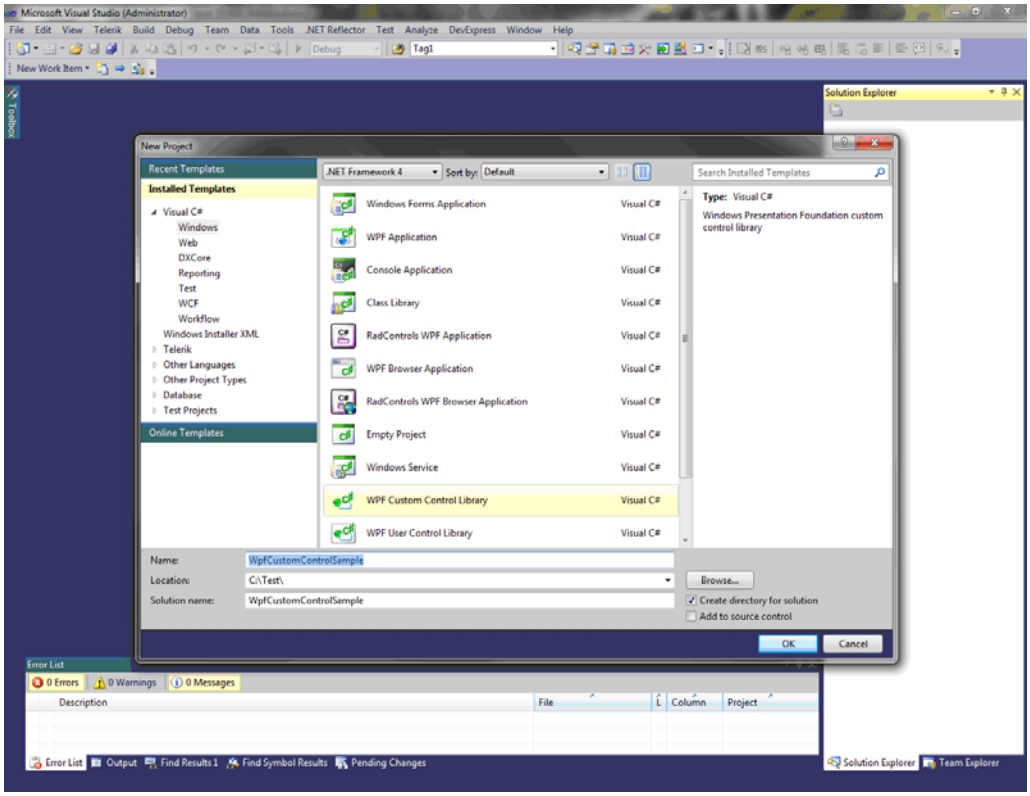
```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;

namespace WpfUserControlSample
{
    /// <summary>
    /// Interaction logic for UserControl1.xaml
    /// </summary>
    [DefaultProperty("Value")]
    public partial class SampleUserControl : UserControl
    {
        public static readonly DependencyProperty ValueProperty;
        static SampleUserControl()
        {
            FrameworkPropertyMetadata frameworkPropertyMetadata = new
            FrameworkPropertyMetadata("0", FrameworkPropertyMetadataOptions.Journal |
            FrameworkPropertyMetadataOptions.BindsTwoWayByDefault);

            ValueProperty = DependencyProperty.Register("Value", typeof(object),
            typeof(SampleUserControl), frameworkPropertyMetadata);
        }

        public SampleUserControl()
        {
            InitializeComponent();
        }

        public object Value
        {
            get { return GetValue(ValueProperty); }
            set { SetValue(ValueProperty, value); }
        }
    }
}
```

3.4 Creating a WPF Custom Control with Tag Connection

The following example describes how to create a WPF custom control that can be connected to a tag.

The complete code is included at the end of the example.

1. Start Visual Studio to create a new project, and select **WPF Cutsom Control Library**.



2. Add [DefaultProperty("Value")] to the class, to define which property the tag should set when then value is set.
3. Add a dependency property with same name as the attribute above: static readonly DependencyProperty ValueProperty;
4. Add a static constructor and register the dependency property.
5. Create a Value property of type string.

6. Replace the code inside the ControlTemplate tag with the following code in the generic.xaml file located in the Themes folder.

```
<Border Background="{TemplateBinding Background}"
  BorderThickness="{TemplateBinding BorderThickness}"
   BorderBrush="{TemplateBinding BorderBrush}">
        <TextBox Text="{Binding Value, RelativeSource={RelativeSource
        Mode=TemplatedParent},Mode=TwoWay,UpdateSourceTrigger=LostFocus}"
        Background="{x:Null}" Margin="1,1,1,1" TextAlignment="Center"
        VerticalAlignment="{TemplateBinding VerticalAlignment}"
        HorizontalAlignment="{TemplateBinding HorizontalAlignment}"
        BorderThickness="0" Foreground="{TemplateBinding Foreground}" />
</Border>
```

   Note the binding on the TextBox text property. The text is bound to the value property that is of type string. If value property is of a different type, a Value converter may be needed.

7. Compile and test by adding the control to the iX Developer toolbox.

Note:
When an update is made, the existing control must be updated under C:\Users\Public\Documents\Beijer Electronics AB\iX Developer\Thirdparty\.

Example Code

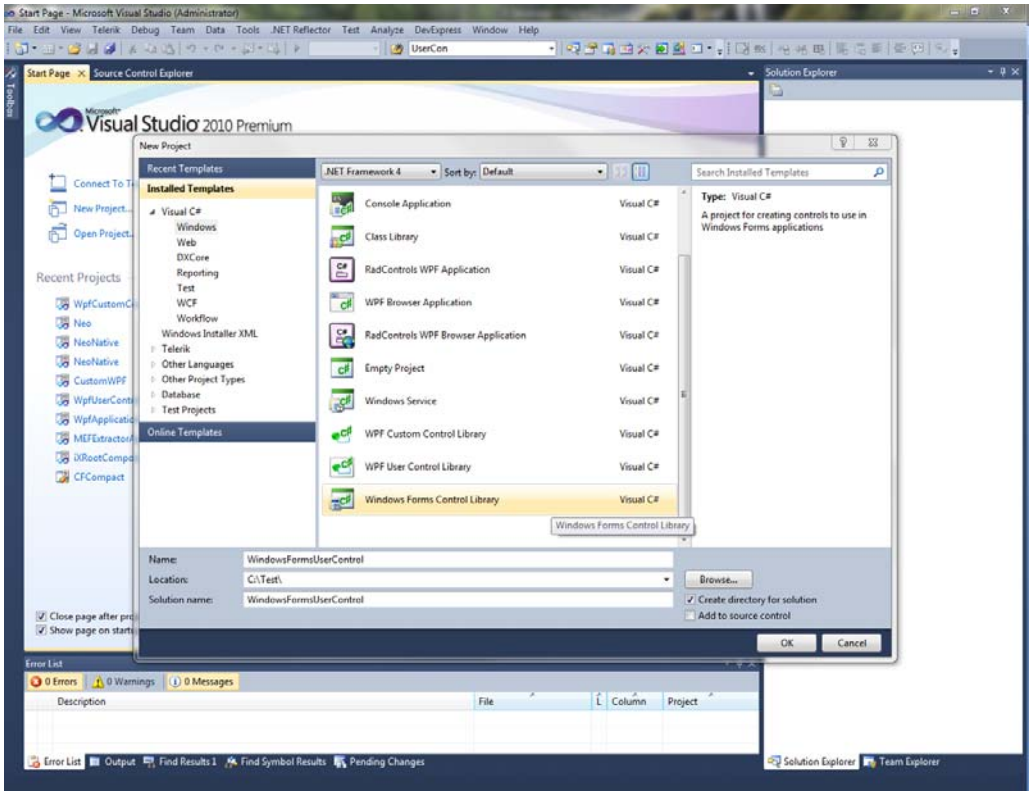
```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.ComponentModel;

namespace WpfCustomControlSample
{
  [DefaultProperty("Value")]
    public class SampleCustomControl : Control
    {
        public static readonly DependencyProperty ValueProperty;

        static SampleCustomControl()
        {

            DefaultStyleKeyProperty.OverrideMetadata(typeof(SampleCustomControl), new
            FrameworkPropertyMetadata(typeof(SampleCustomControl)));

            FrameworkPropertyMetadata frameworkPropertyMetadata = new
            FrameworkPropertyMetadata("0",
            FrameworkPropertyMetadataOptions.AffectsRender |
            FrameworkPropertyMetadataOptions.BindsTwoWayByDefault);
            ValueProperty = DependencyProperty.Register("Value", typeof(string),
            typeof(SampleCustomControl), frameworkPropertyMetadata);
        }

        public string Value
        {
            get { return (string)GetValue(ValueProperty); }
            set { SetValue(ValueProperty, value); }
        }
    }
}
```

4 Windows Forms Controls

4.1 Creating a Windows Forms User Control for a PC Target

The following example describes how to create a Windows Forms user control designated for a PC target.

1. Start Visual Studio to create a new project, and select **Windows Forms Control Library**.



2. Add a TextBox and a Button to the design surface.
3. Add Event Handler for Button click.
4. Add Event Handler for TextBox lost focus.

5. Add a Value Property and INotifyPropertyChanged implementation:

```
public partial class SampleUserControl : UserControl,
INotifyPropertyChanged
{
    public SampleUserControl()
    {
        InitializeComponent();
    }

    public object Value
    {
        get { return textBox1.Text; }
        set
        {
            if (value != null)
            {
                textBox1.Text = value.ToString();
            }
            FirePropertyChanged("Value");
        }
    }

    private void OnButtonClick(object sender, EventArgs e)
    {
        Value = "0";
    }

    private void OnLostFocus(object sender, EventArgs e)
    {
        Value = textBox1.Text;
    }

    public event PropertyChangedEventHandler PropertyChanged;
    public virtual void FirePropertyChanged(string propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
                handler(null/*this*/, new
                PropertyChangedEventArgs(propertyName));
        }
    }
}
```

6. Use the following code to connect the control to a tag value in iX Developer:

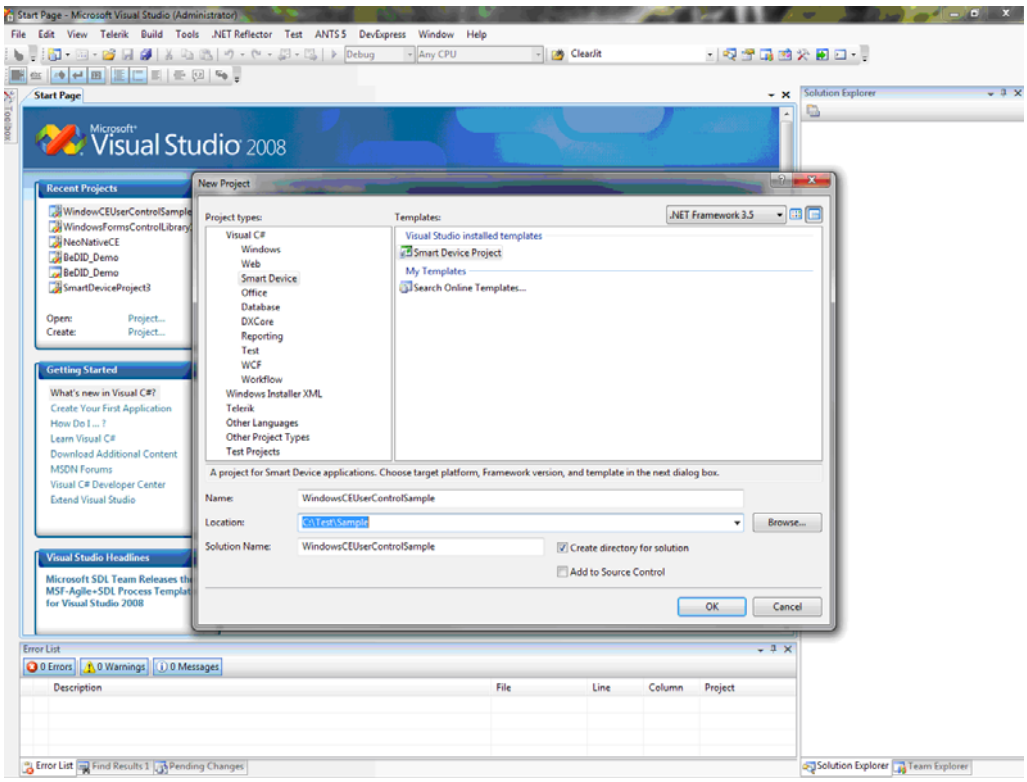
```
public partial class Screen1
{
     void Screen1_Opened(System.Object sender, System.EventArgs e)
     {
         // Hook up value change for a tag
         Globals.Tags.Tag1.ValueChange += OnTagValueChanged;
         // Hook up Property Change on the User Control
        SampleUserControl1.PropertyChanged +=
        OnUserControlValueChanged;
         // Set initial value
         SampleUserControl1.Value = Globals.Tags.Tag1.Value;
    }
        private void OnTagValueChanged(object sender,
        Neo.ApplicationFramework.Interfaces.Events.ValueChangedEventArgs
        e)
     {
         SampleCEUserControl1.Value = e.Value;
     }

     private void OnUserControlValueChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
     {
                 Globals.Tags.Tag1.Value = new
                 VariantValue(SampleCEUserControl1.Value);
     }


 void Screen1_Closing(System.Object sender,
System.ComponentModel.CancelEventArgs e)
     {
        // Always remember to unhook the event handlers, otherwise a
        //memory leak is generated
        Globals.Tags.Tag1.ValueChange -= OnTagValueChanged;
        SampleUserControl1.PropertyChanged -=
        OnUserControlValueChanged;
     }
    }
```

The code shows how the value is set on the user control when the tag changes its value, and how the tag value is changed when the user control changes its value.

4.2 Creating a Windows Forms User Control for a CE Target

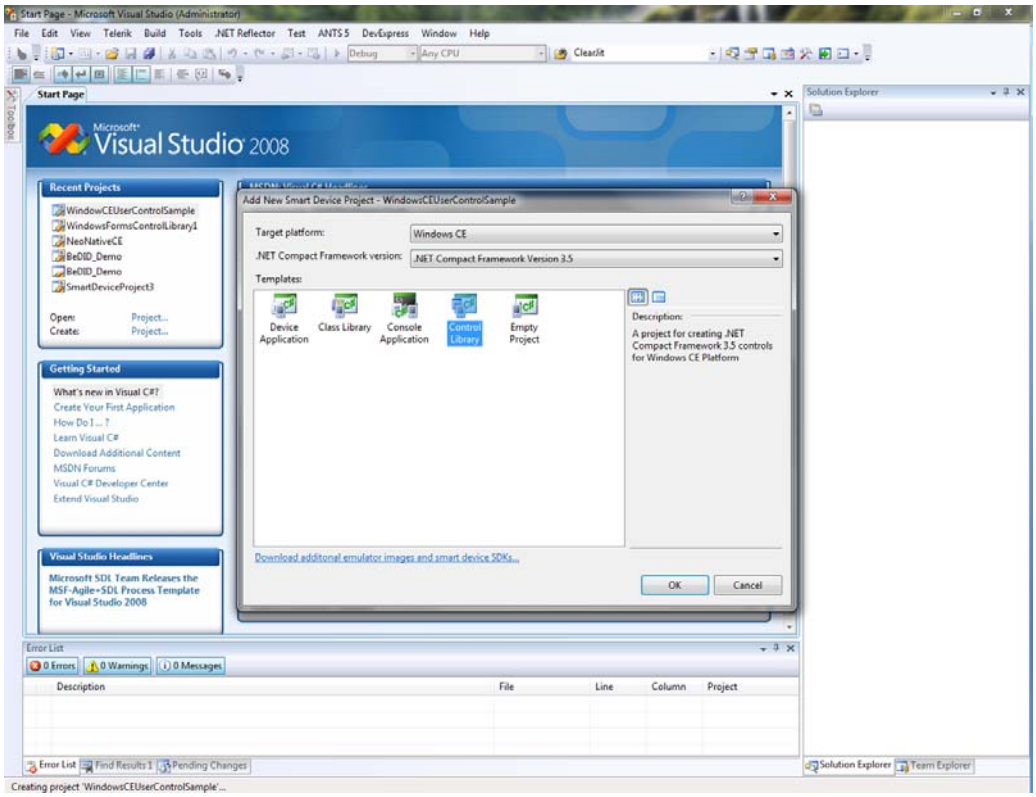
The following example describes how to create a Windows Forms user control designated for a CE target (an operator panel).

1. Start Visual Studio 2005 or 2008 to create a new Smart Device Project.



2. Select **Windows CE** for Target platform.

3. Select **Control Library**.



4. Use the same code as in the *Creating a Windows Forms User Control for a PC Target* example.

Note:
Always test your code on the target platform, as properties/methods currently not supported may be included in the code. See *Limitations* for details.

---

5 Trouble Shooting

Sometimes when using third party controls a build error indicating that a reference is missing may occur when building the project. Try to add that dll to Project\ReferenceAssemblies to solve the problem.

Beijer
ELECTRONICS

HEAD OFFICE

SWEDEN

Beijer Electronics Products AB
Box 426
SE-201 24 Malmö, Sweden
Tel: +46 40 35 86 00
Fax: +46 40 93 23 01
info@beijerelectronics.com

SUBSIDIARIES

GERMANY

Elektronik-Systeme Lauer GmbH & Co. KG
Kelterstraße 59
72669 Unterensingen, GERMANY
Tel: +49 7022 9660 0
Fax: +49 7022 9660 103
info@lauer-hmi.com

TAIWAN

Hitech Electronics Corp.
7 & 8 F, No. 108 Min-Quan Road
Shin-Tien, Taipei Shien, TAIWAN, R.O.C. 231
Tel: +886-2-2218-3600
Fax: +886-2-2218-9547
info.hmi@hitech-lcd.com.tw

USA

Beijer Electronics Inc.
939 N. Plum Grove Road, Suite F
Schaumburg, IL 601 73, USA
Tel: +1 847 619 6068
Fax: +1 847 619 6674
info.usa@beijerelectronics.com

CHINA

Beijer Electronics Co. Ltd
Room 201, Buildning B, No. 1618,
Yishan Road, Shanghai 201103, CHINA
Tel: +86 21 6145 0400
Fax: +86 21 6145 0499
info@beijerelectronics.cn