AicStudio 用户手册

时间：2014.12.23
版本：1.0
编辑：董松涛，何沛华
邮箱：dongsongtao@aic-tech.com

目录

1 AicStudio 简介

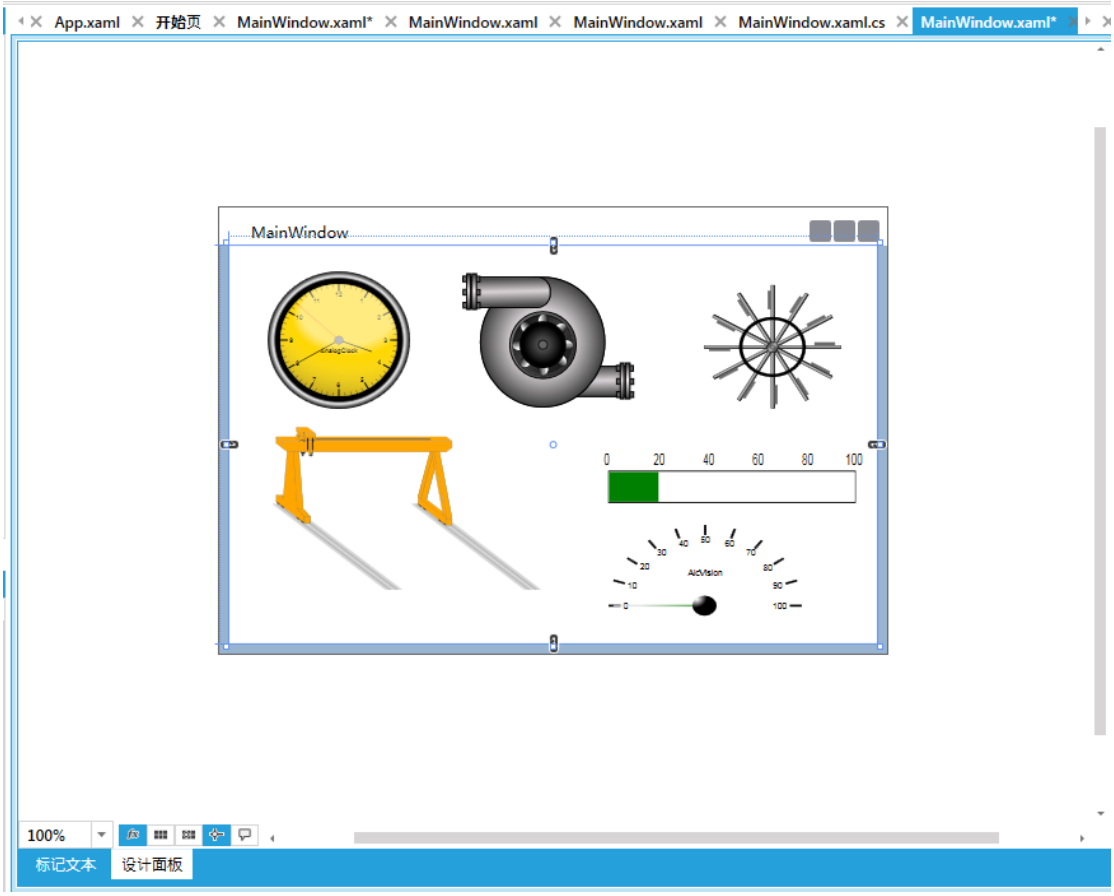
1.1AicStudio 功能介绍

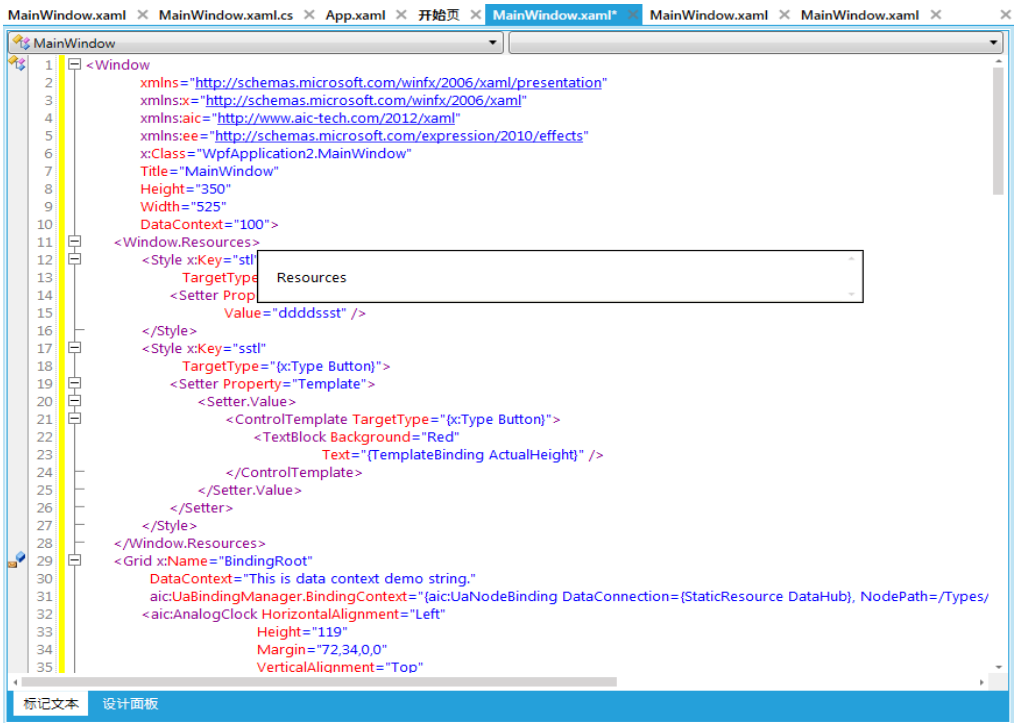
AicStudio 是艾克信控科技有限公司开发的 SCADA 产品套件 AicVision 的客户端监控画面组态及编程开发工具，基于微软最新的.NET Framework 4.5 框架和 OPC UA 国际标准（IEC 62541）开发，编程语言为 C#，界面描述语言为 XAML，目前的最新版本为 2.0。

1.2AicStudio 平台特性

AicStudio 具有如下特点：

1) AicStudio 提供 WYSIWYG（所见即所得）的图形画面组态编辑功能：从“工具箱”中拖拽控件到设计面板，然后在“属性”面板中设置控件属性或关联实时变量，即可完成画面的组态。此外，主编辑区域可以在设计面板和标记文本(XAML)之间进行切换，对于较资深的程序员用户可以通过直接修改 XAML 标记文本来修改界面。



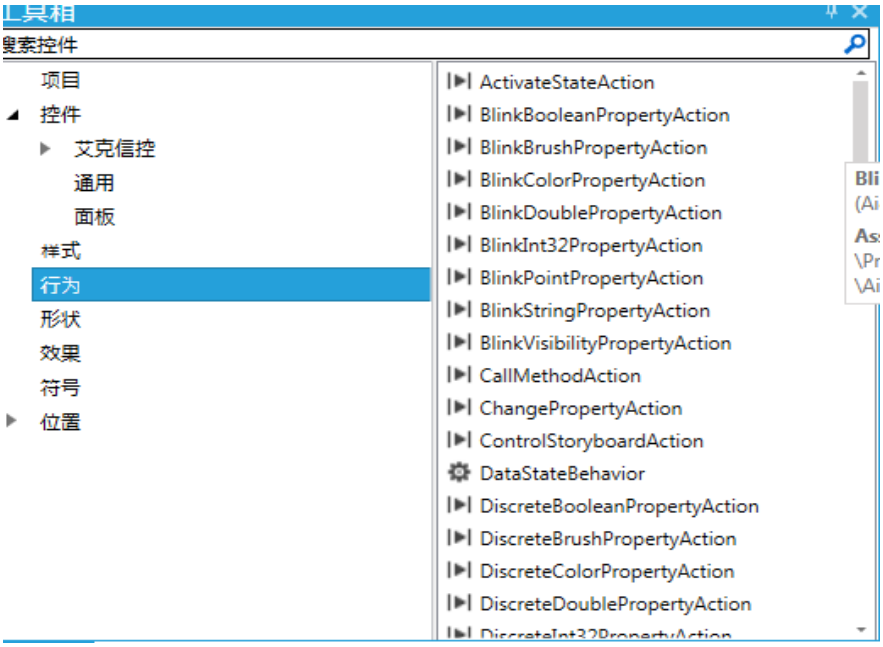


2) AicStudio 同时支持基于 WPF 4.0/4.5 的 C/S 架构工程以及基于 Silverlight 5.0 的跨浏览器、跨平台的 B/S 架构工程。

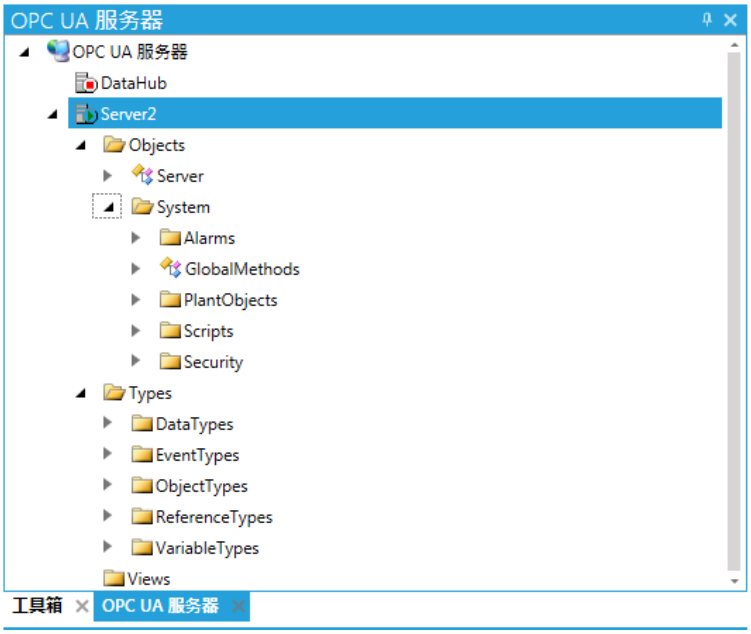
3) AicStudio 提供常用的图素和控件库，包括仪表盘、开关、趋势曲线、实时报警控件、历史回放控件以及各种常用符号等。此外，AicStudio 还支持第三方提供的 WPF/Silverlight 标准控件，例如 Telerik、Syncfusion 等控件开发商提供的 WPF 和 Silverlight 控件。对于有控件开发能力的用户，还可以自行开发符合 WPF/Silverlight 标准的自定义控件。

4) AicStudio 基于微软最新的 WPF/Silverlight 技术的图形系统，所有的画面都是矢量图，可以实现无级的缩放而不会影响显示的效果。并且支持控件的倾斜、旋转、平移、缩放、翻转等多种变换；支持控件的组合和分解操作；支持不受限制的图层。

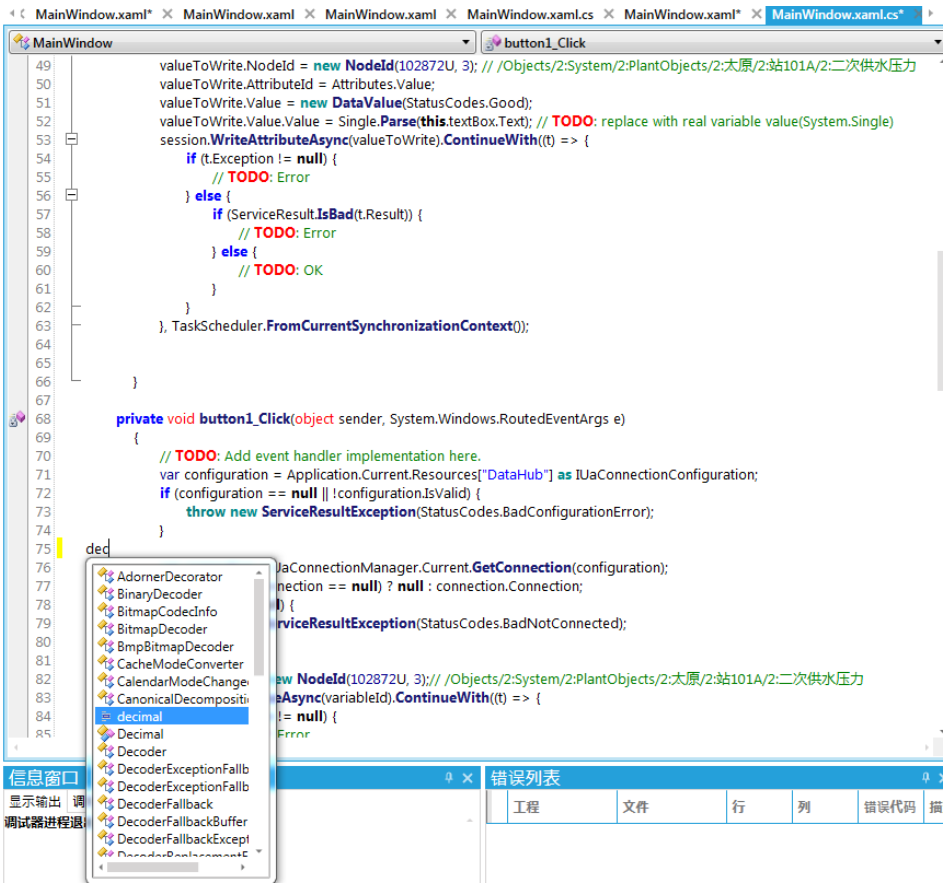
AicStudio 可以很方便的添加多种动画与特效，让你的界面炫酷起来。



5) AicStudio 提供基于 OPC UA 国际标准的开放式数据连接，支持任何标准 OPC UA 服务器（例如 AicVision 套件中的实时数据库产品 AicDataHub 服务器）变量的数据绑定：变量可通过组态直接绑定到控件或图素的属性上，运行时自动向实时数据服务器进行数据订阅，无需用户进行额外的编程。在 AicStudio 提供的 OPC UA 服务器窗口中可以连接 OPC UA 服务器，查看 OPC UA 服务器地址空间，选择关联变量等。



6) AicStudio 支持使用 C#进行编程，功能扩展不受限制。AicStudio 拥有友好的 C#代码编辑环境，智能提示和代码助手功能大大提升了用户的编程体验。



7) AicStudio 的工程是编译执行的，所有的画面和程序在发布之前都会编译成程序集，运行时无需再进行额外的处理，大大提高了系统运行时的效率。

8) AicStudio 支持面向对象/设备的画面组态，这样组态而成的画面是基于对象/设备类型而非对象/设备实例，在运行时可通过脚本实时切换对象/设备实例，画面所显示的数据将会相应动态切换到指定的对象/设备实例。

9) AicStudio 采用了与微软 Visual Studio 开发工具完全兼容的工程文件，各自的 WPF 或者 Silverlight 工程无需修改在 2 个平台上运行都没有障碍，在具体项目开发时，可以结合两个平台各自的优点进行快速便捷的开发操作。例如，AicStudio 只提供了 WPF 工程的简单调试跟踪功能，对于 Silverlight 工程则暂未提供调试跟踪功能，因此在需要的时候，用户完全可以利用 Visual Studio 强大的调试功能对 AicStudio 工程（WPF 或 Silverlight）进行深层次的跟踪和调试。

关于相关具体功能的使用，会在后续的章节一一详细进行介绍。

2 菜单与窗口

2.1 AicStudio 的菜单

AicStudio 这个软件有 5 个菜单，分别为文件菜单、主页（Home）、工程（Project）、视图（View）和工具（Tools），如图 2.1 所示。

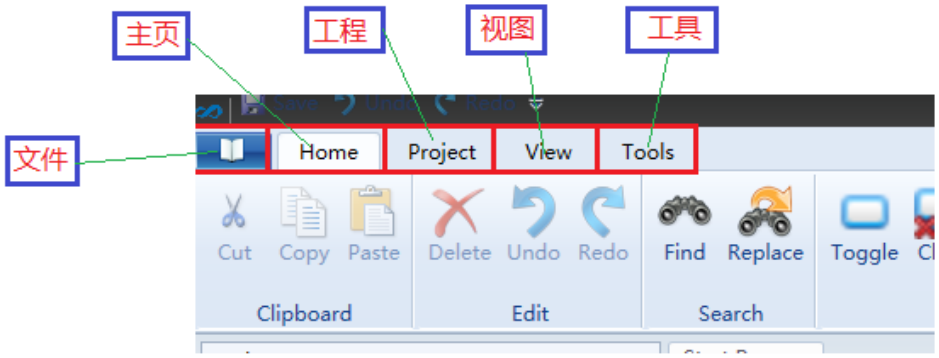


图 2.1

2.1.1 文件菜单

文件菜单如图 2.1.1 所示，可用于新建、打开解决方案，添加新工程和已经存在的工程，右边是最近使用的解决方案（最近的解决方案也可以在窗口中的开始页看到）。

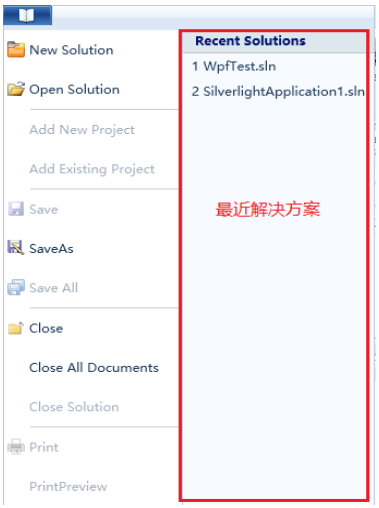


图 2.1.1

2.1.2 主页菜单（Home）

主页菜单如图 2.1.2 所示，包括剪贴板、编辑、查找、书签、编译、语言设置和界面的主题选择等工具项。

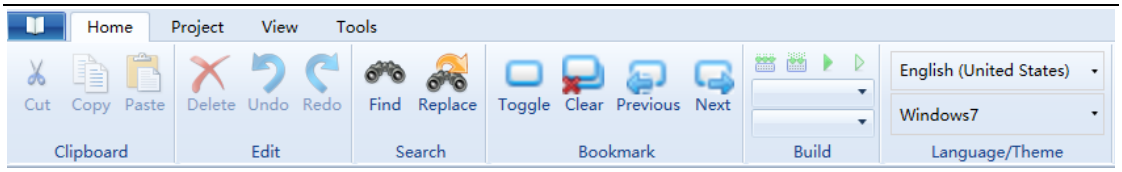


图 2.1.2

2.1.3 工程菜单（Project）

工程菜单如图 2.1.3 所示，包括整体解决方案编译、当前项目编译，调试运行等工具项。

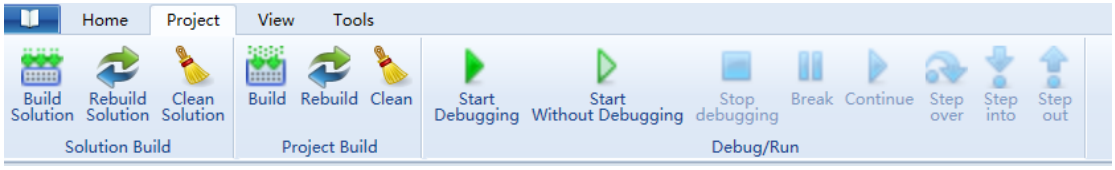


图 2.1.3

2.1.4 视图菜单（View）

视图菜单如图 2.1.4 所示，包括错误列表、输出窗口、属性窗口、解决方案浏览器、搜索结果窗口、工具窗口、大纲窗口、起始页、资源窗口、OPCUA 服务浏览器、本地变量、监视串窗口、调用堆栈窗口、模块窗口、线程窗口、断点窗口。这些开发过程中显示用的窗口的功能和用法都和 Visual Studio 类似。

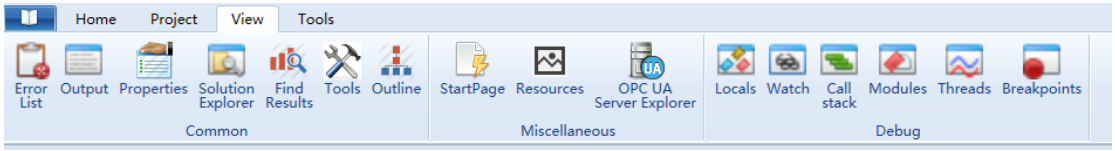


图 2.1.4

2.2 AicStudio 的窗口

打开 AicStudio，窗口如图 2.2 所示，包括工具窗口、OPCUA 服务浏览器窗口、大纲窗口、主窗口、信息输出窗口、解决方案浏览器窗口、属性窗口和资源窗口。这些窗口都可以在视图菜单（View）中打开，如不需要可以关掉。

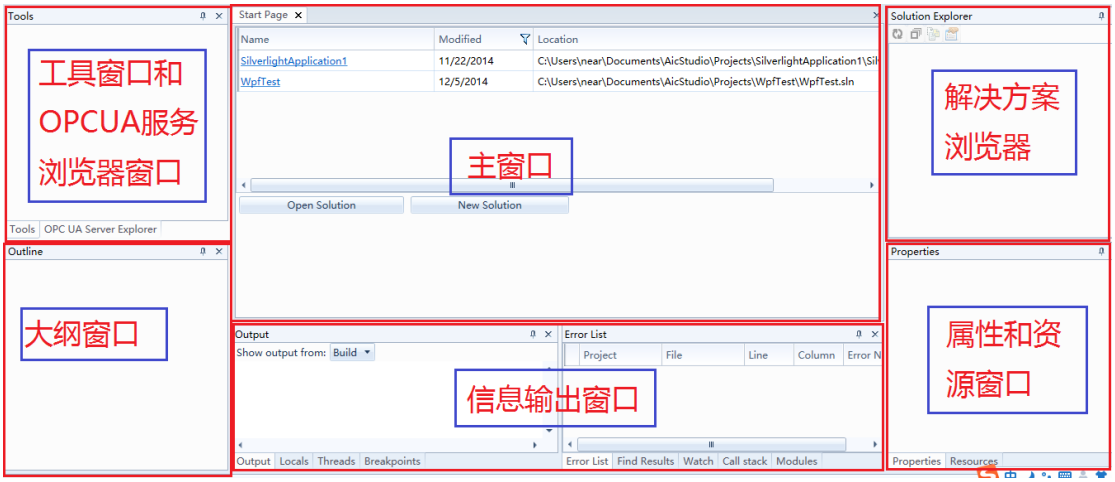


图 2.2

工具窗口如图 2.3 所示，开发中可以拖拽基本的控件、AicVision 的控件、第三方控件（如 Telerik 控件）、样式，控件动画等，在下文中会具体用到。

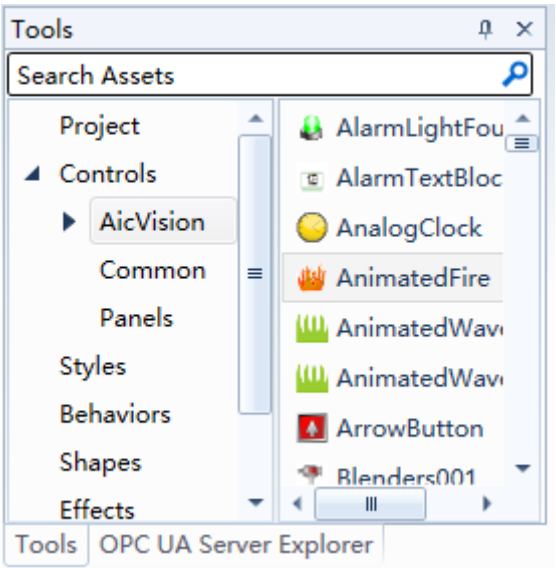


图 2.3

OPC UA 服务浏览器窗口如图 2.4 所示，用于添加、删除 OPC UA 服务器，在下文中会用到。

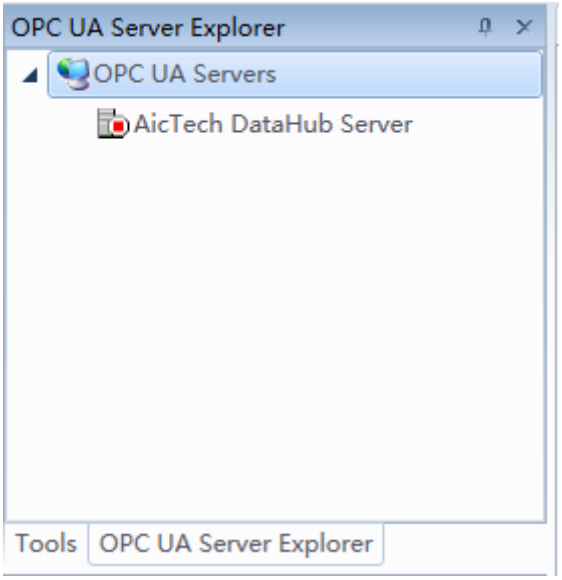


图 2.4

大纲窗口如图 2.5 所示，可以看到当前的编辑界面的所有控件和其层级关系，图中的编辑界面可见按钮可以让对应的控件在当前编辑界面中不可见或可见(不会影响控件的实际属性，只是方便用户开发)，编辑界面 Lock 按钮用于给对应的控件加锁，加锁后此控件将不可编辑（修改控件的属性)。

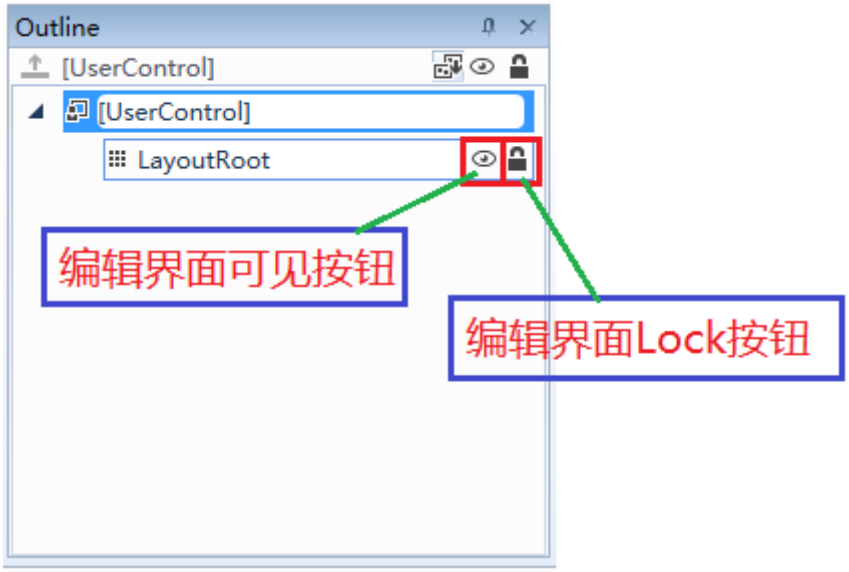


图 2.5

主窗口如图 2.6 所示，用于显示当前的画面的 XAML 标记文本和可视化设计界面。

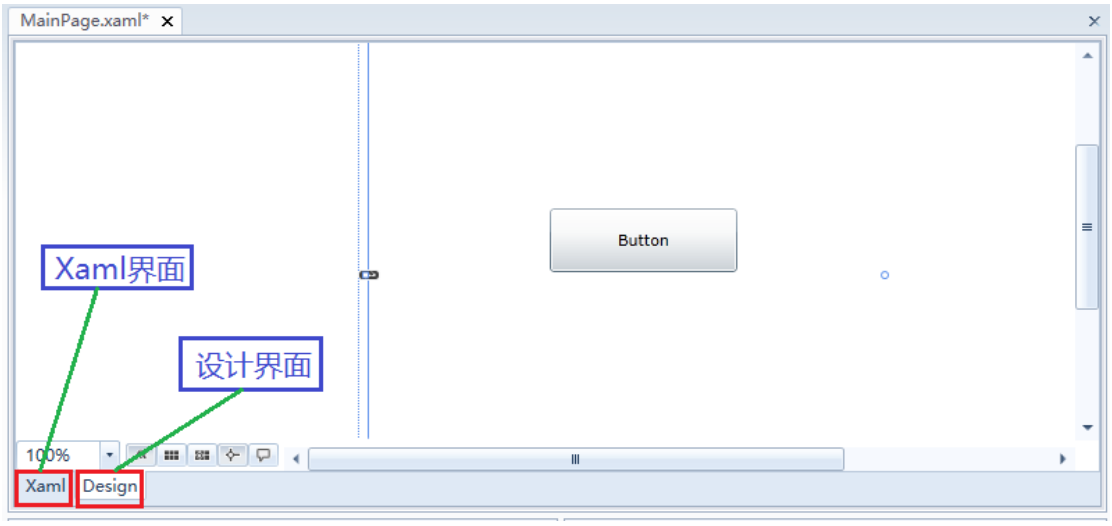


图 2.6

信息输出窗口包含以下几个窗口：输出窗口、本地变量窗口、线程窗口、断点窗口、包括错误列表窗口、搜索结果窗口、监视串窗口、调用堆栈窗口、模块窗口。这些窗口可以用于调试进行数据的输出。

解决方案浏览器窗口如图 2.7 所示，可看到当前的解决方案下的各个项目以及各个项目下的各个文件，也可以在此窗口进行项目和文件的添加、编辑、删除等操作。

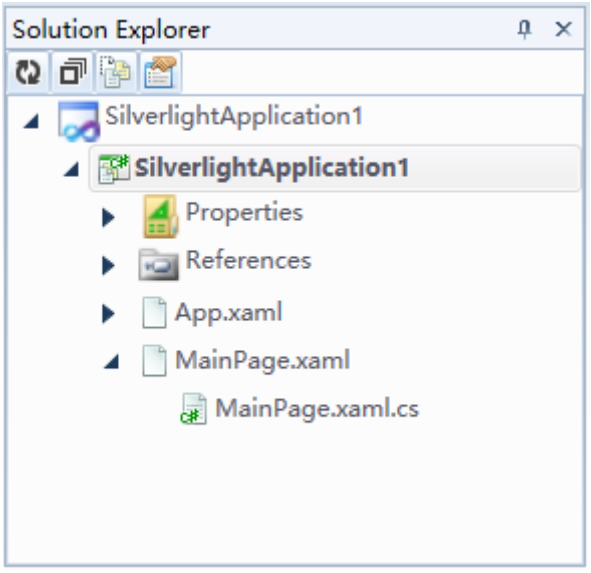


图 2.7

属性窗口如图 2.8 所示，显示当前选中控件的属性，直观地设置控件的属性、事件、数据绑定等。

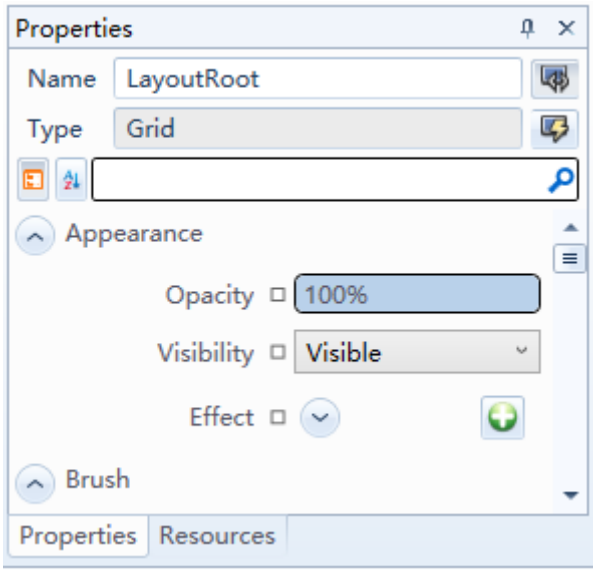


图 2.8

资源窗口如图 2.9 所示，层级显示当前选中的控件的 XAML 标记文本中定义过的资源。

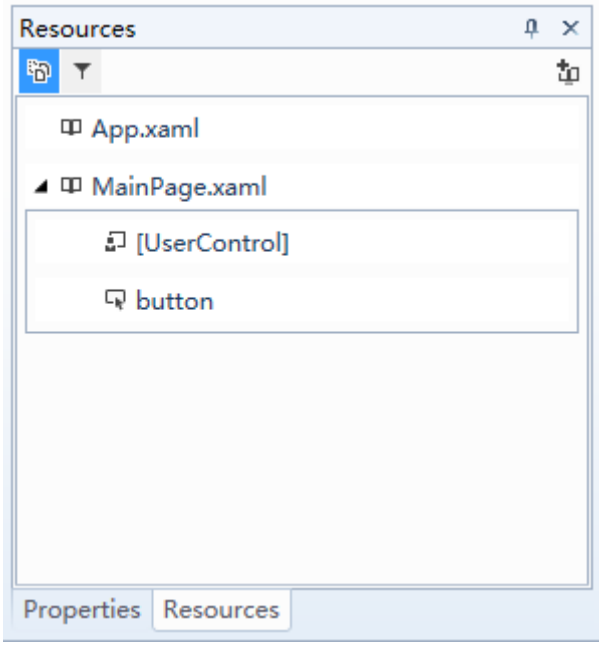
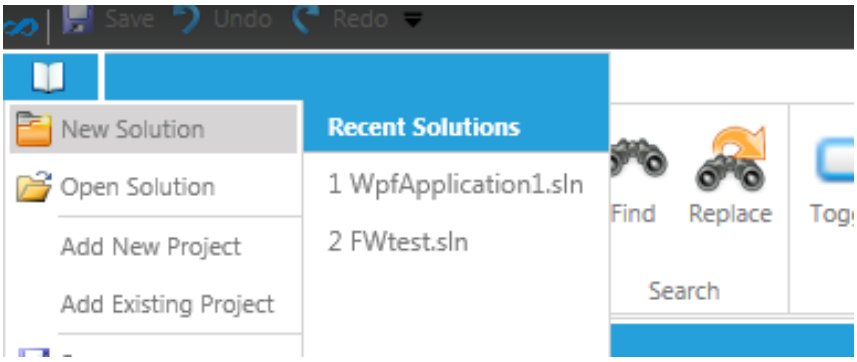


图 2.9

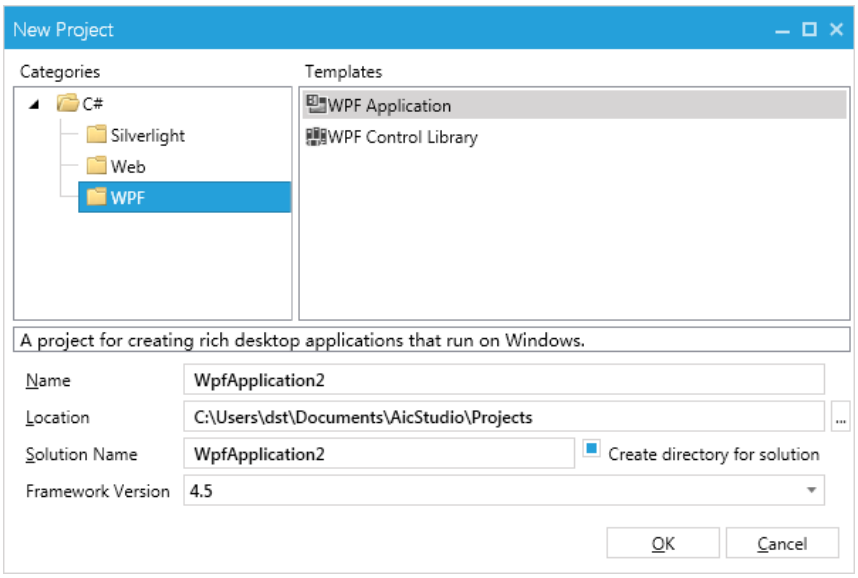
3 工程管理

3.1 工程创建

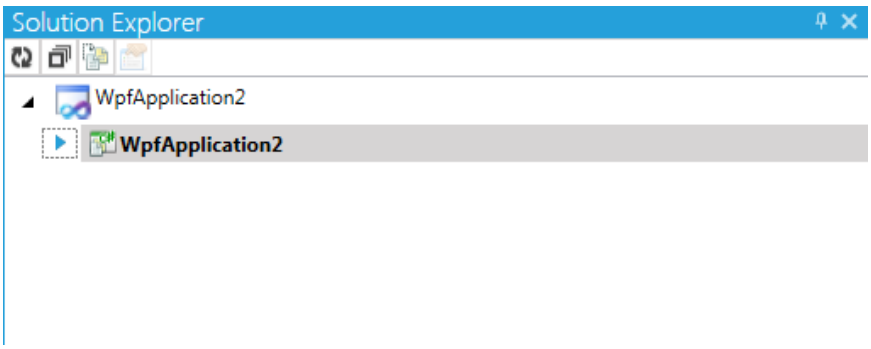
运行 AicStudio 主程序，在主菜单点击 New Solution 新建解决方案。



AicStudio 可以新建的工程类别有三种，WPF、Silverlight 和 Web。可以根据工程需要进行选择，现以 WPF Application 工程为例新建项目。

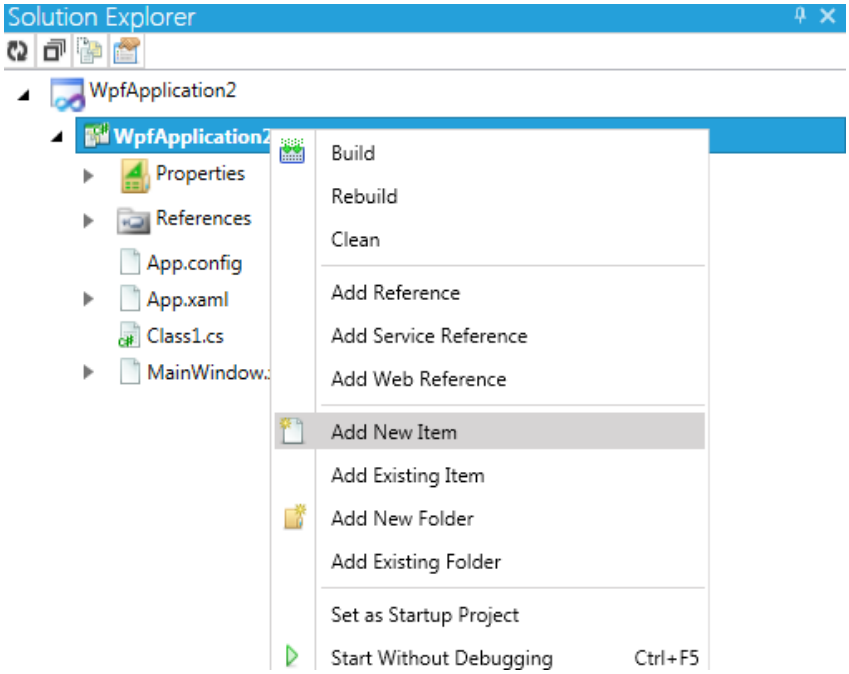


新建工程完成后，在屏幕的右上角可以看到解决方案浏览器(Solution Explorer)。我们在这里进行文件的添加、删除，程序集引用的添加和删除等操作。

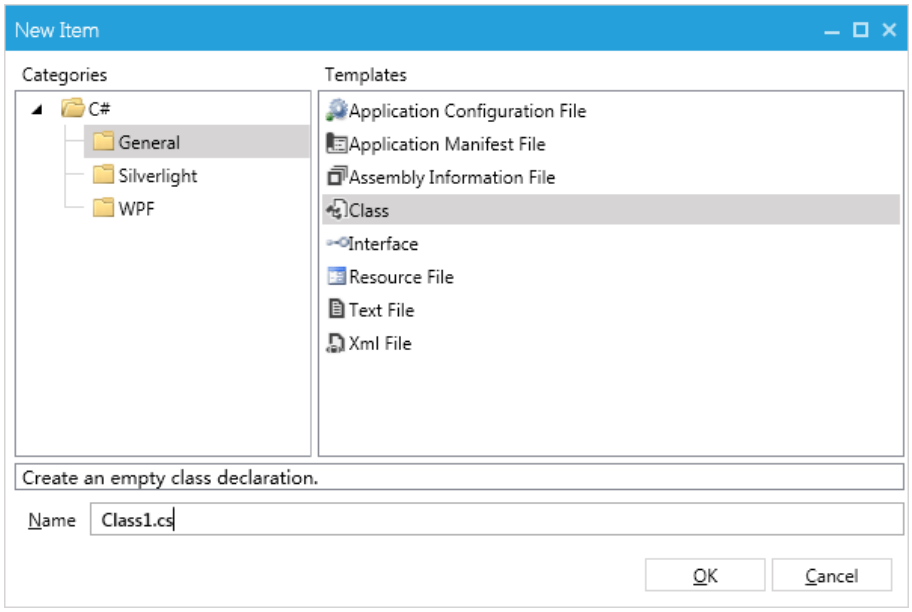


3.2 文件添加

右键点击工程，选择 Add New Item 选项进行添加新文件。

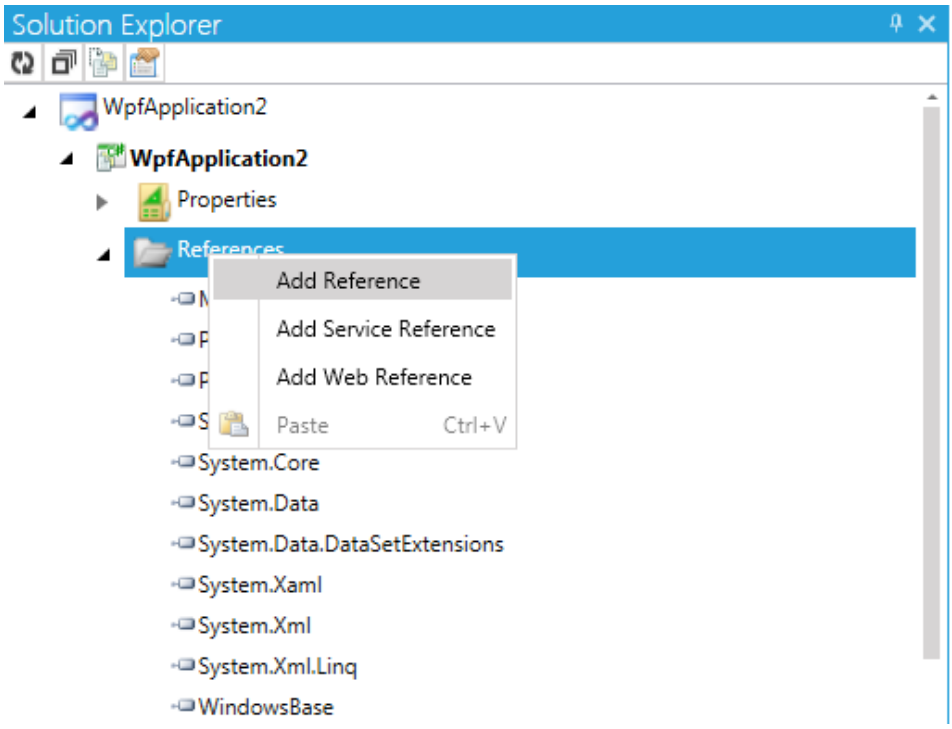


以添加新的类为例，在 General 菜单下选择 Class，输入类名称，点击 OK 进行添加。



3.3 程序集引用

在工程的 References 文件夹下右键点击 Add Reference，添加程序集引用



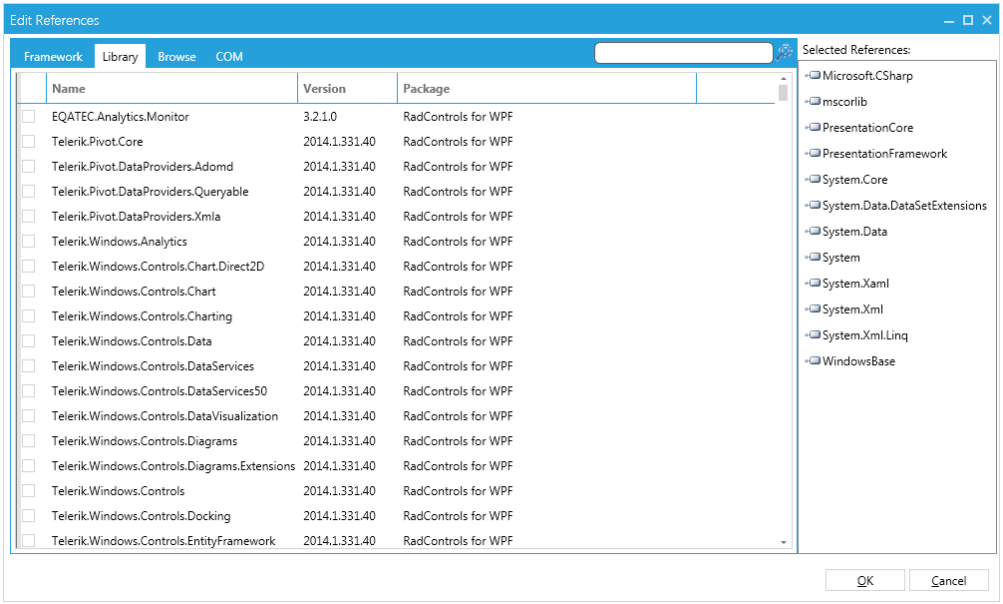
引用的程序集主要分 4 类。

1）Framework，是.NET Framework 框架下的程序集；

2）Liabrary，是扩展或第三方控件库的集合，例如 Telerik，AicVision，ASP.NET，SQLServer 等；

3）Browse，通过浏览文件来添加程序集引用；

4）COM，添加 COM 组件。

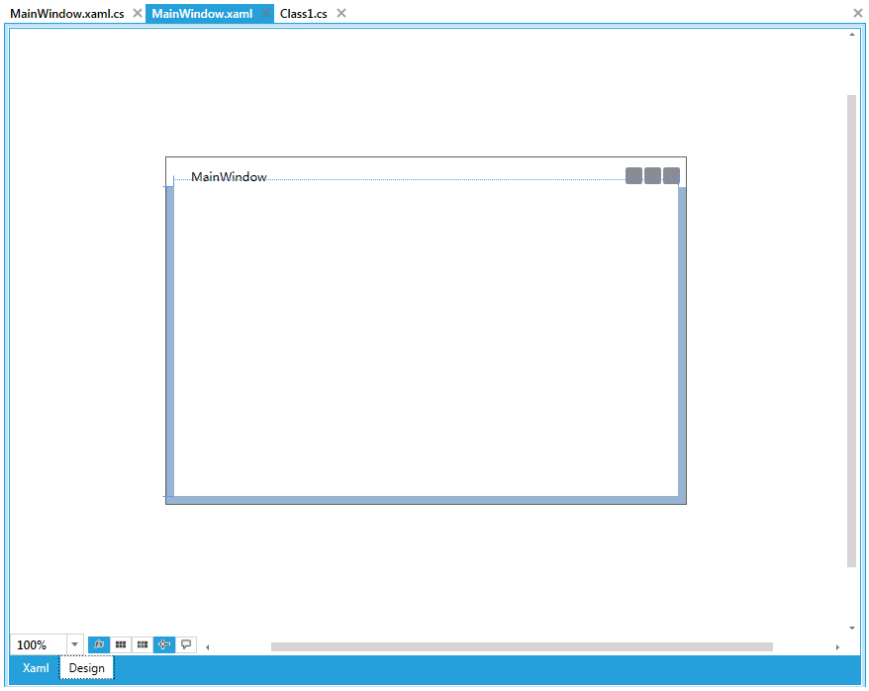


4 界面编辑

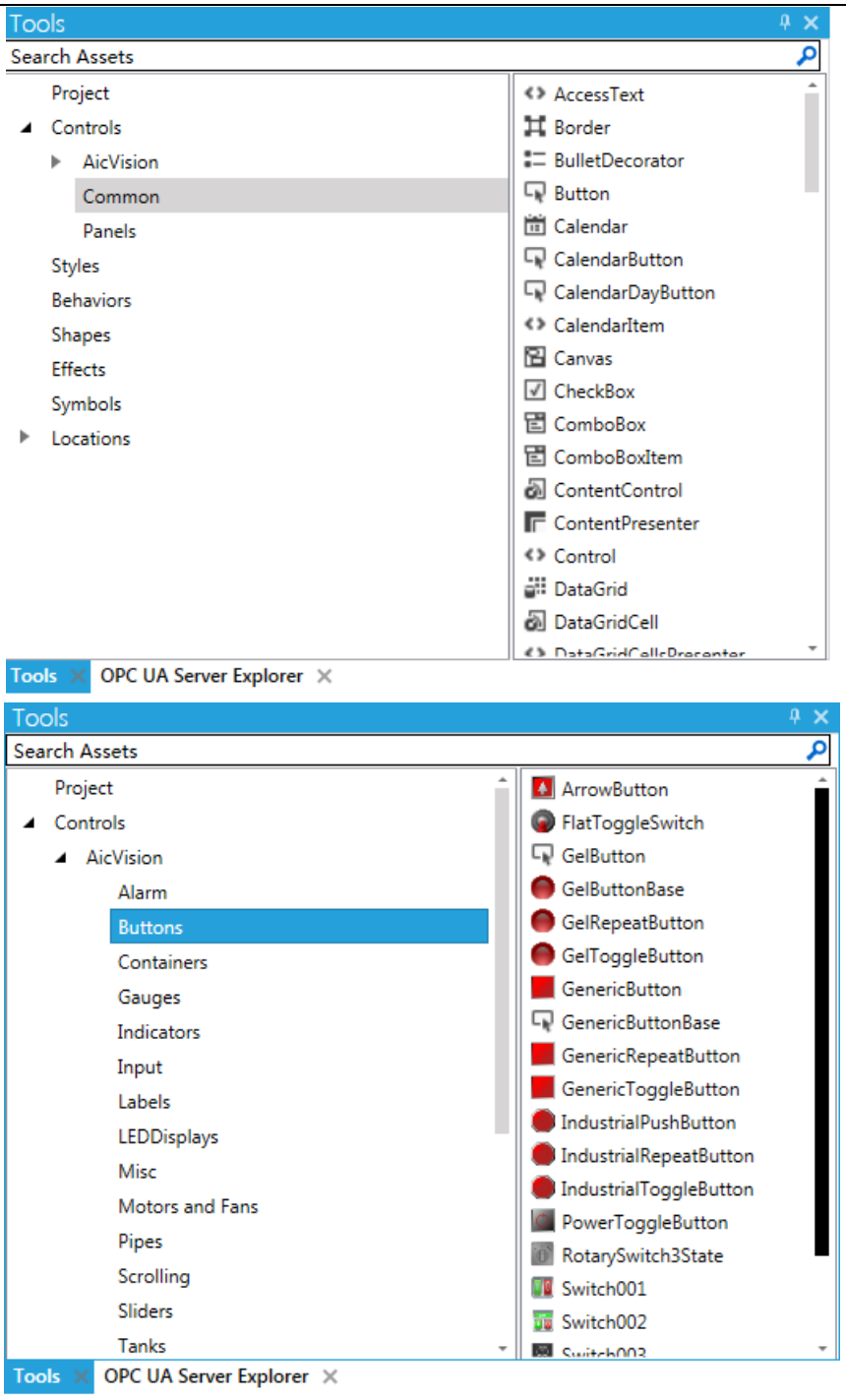
界面编辑是软件的 UI 部分，决定软件界面的显示和与用户交互时的行为。在 AicStudio 中，有非常方便的界面编辑工具，下面的小节将从各类控件，数据绑定选择，动画添加等几个方面进行介绍。

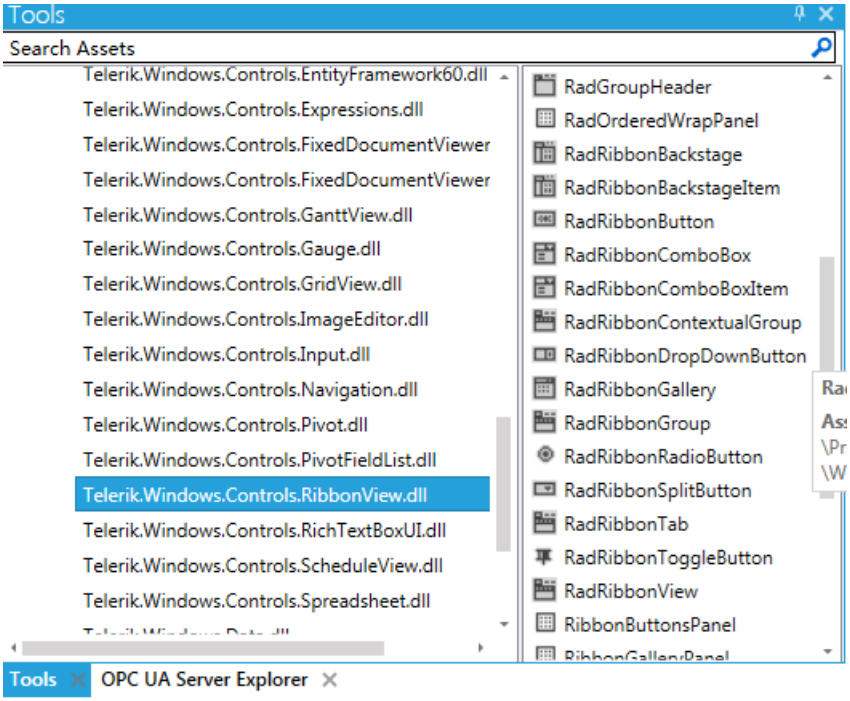
4.1 控件 Controls

进行控件添加之前，要先打开编辑界面，即双击 MainWindow.xaml，显示设计界面。

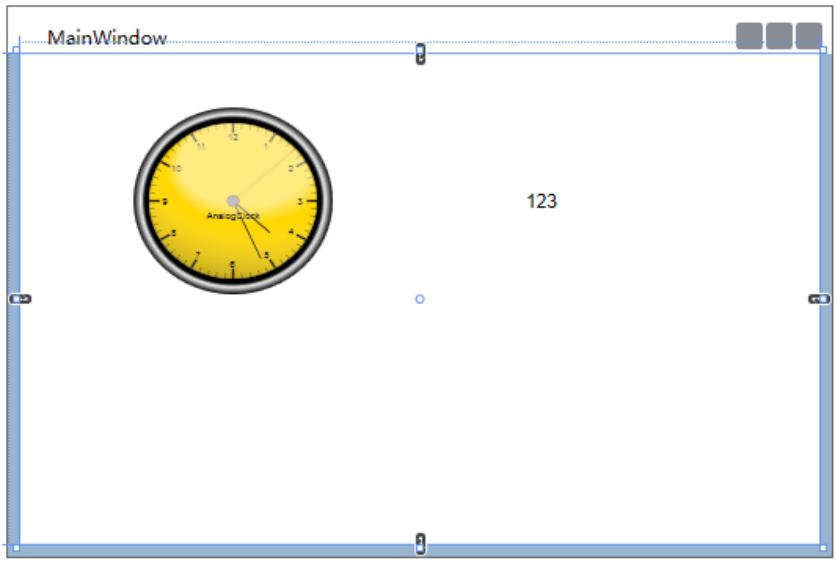


然后可以看到屏幕左上角显示出来的工具箱 Tools，工具箱中包含了可以添加进界面的所有控件，除了 Common 菜单下的基本控件 Button、Border、Canvas、CheckBox 之外，还有 AicVision 菜单下，艾克信控公司开发的系列控件，在 Locations 菜单下包含所有本地的控件程序集，常用的如 Telerik 等第三方提供的控件库。


Tools
Search Assets
Project
Controls
AicVision
Common
Panels
Styles
Behaviors
Shapes
Effects
Symbols
Locations
AccessText
Border
BulletDecorator
Button
Calendar
CalendarButton
CalendarDayButton
CalendarItem
Canvas
CheckBox
ComboBox
ComboBoxItem
ContentControl
ContentPresenter
Control
DataGrid
DataGridCell
Tools
OPC UA Server Explorer
Tools
Search Assets
Project
Controls
AicVision
Alarm
Buttons
Containers
Gauges
Indicators
Input
Labels
LEDDisplays
Misc
Motors and Fans
Pipes
Scrolling
Sliders
Tanks
ArrowButton
FlatToggleSwitch
GelButton
GelButtonBase
GelRepeatButton
GelToggleButton
GenericButton
GenericButtonBase
GenericRepeatButton
GenericToggleButton
IndustrialPushButton
IndustrialRepeatButton
IndustrialToggleButton
PowerToggleButton
RotarySwitch3State
Switch001
Switch002
Tools
OPC UA Server Explorer



现在做一个简单的示例，添加几个控件到界面，添加 AicVision/Misc 菜单下的 AlarmTextBlock 和 AnalogClock 控件。



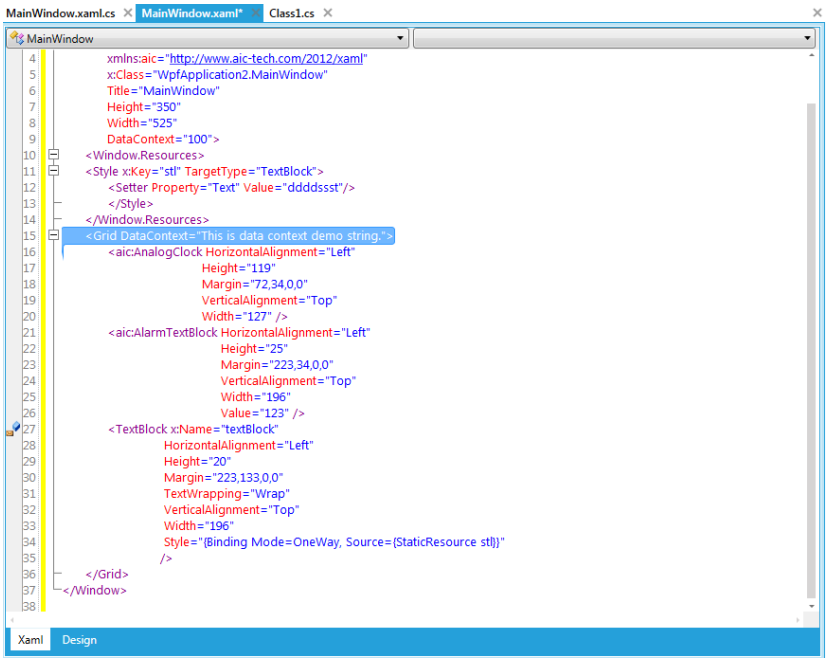
4.2 数据绑定 DataBinding

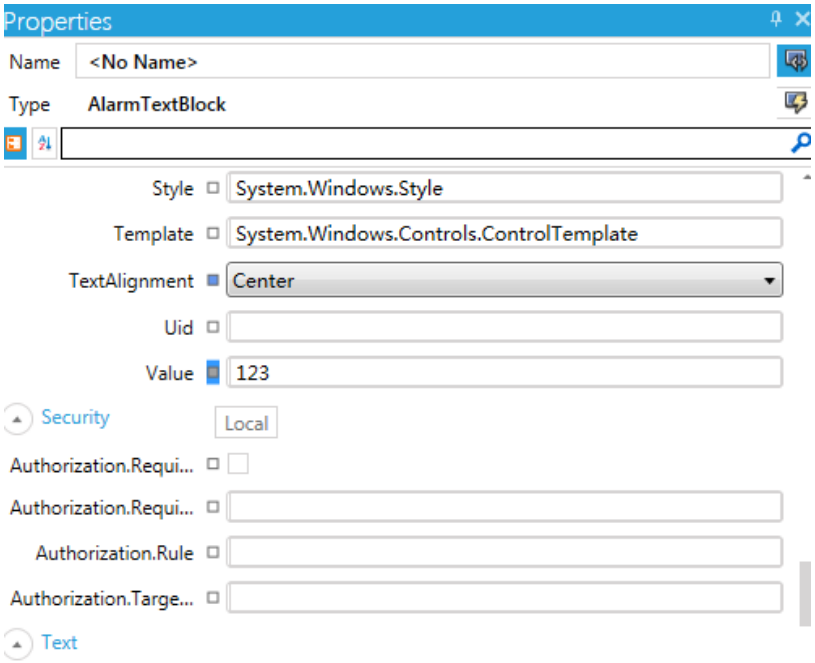
数据绑定(Data Binding)是数据源与控件的关联，绑定的目标是控件中的某依赖项属性，绑定的数据源是对象的公开属性。根据数据源和绑定方式的不同，数据绑定分为很多种。

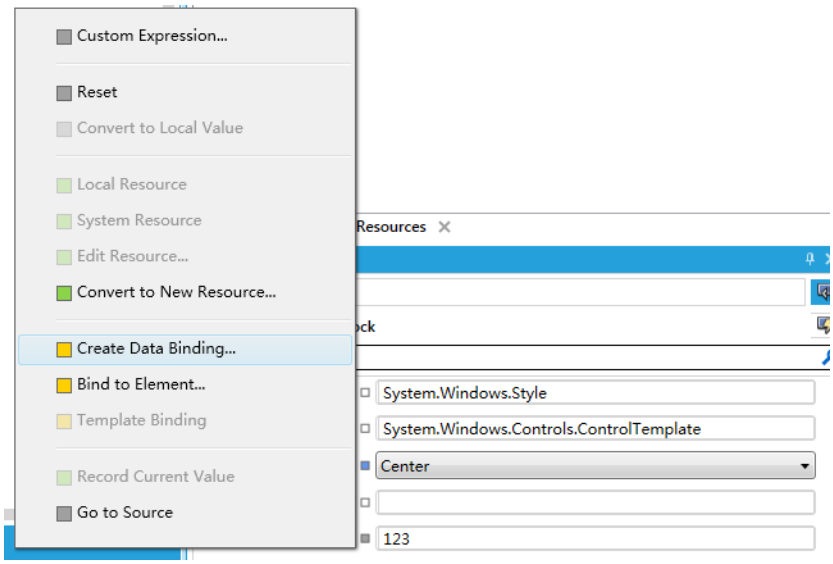
4.2.1DataContext 绑定

在标记文件界面的 Grid 定义一个 DataContext，然后切回设计界面，点击 AlarmTextBlock

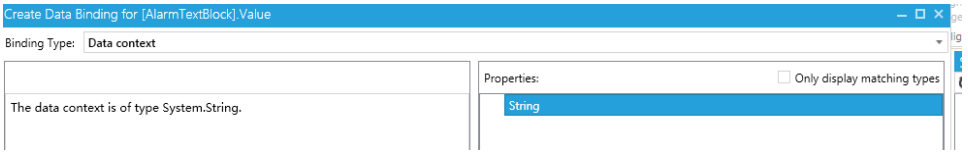
控件，在屏幕右下角的属性(Properties)窗口中找到此控件的 Value 属性，点击旁边的小方块，然后选择 Create Data Binding 建立数据绑定。

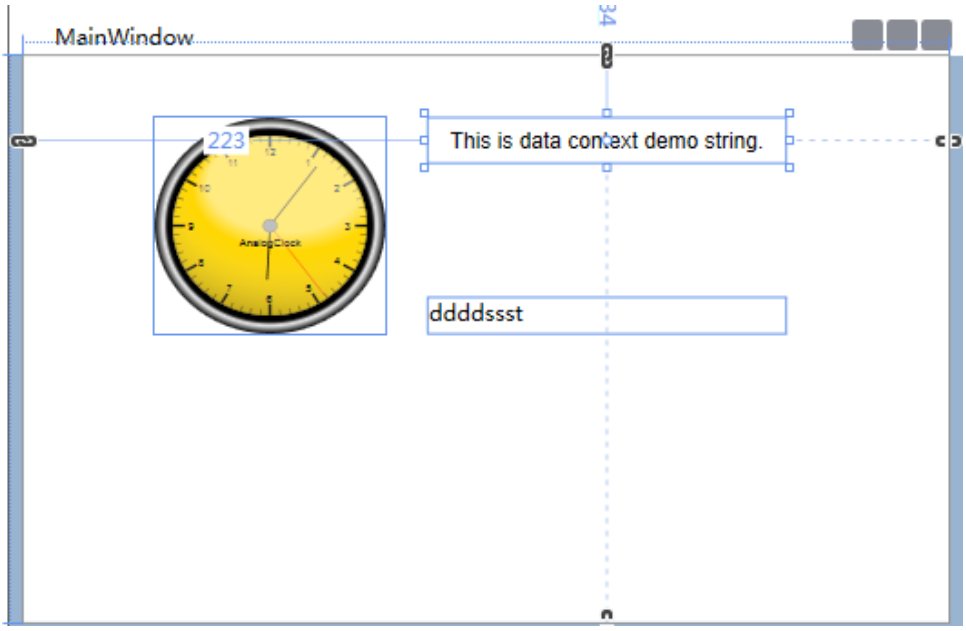






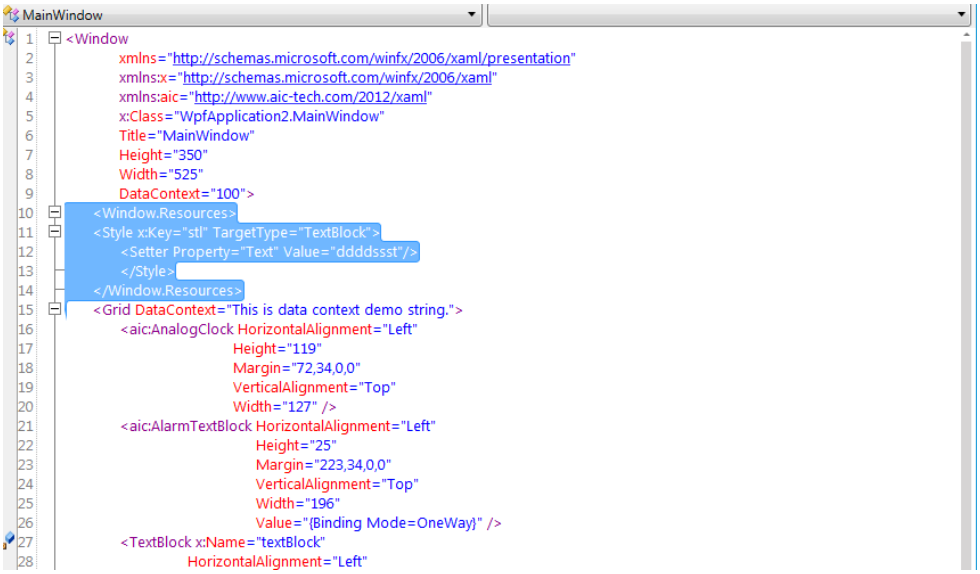
在 Binding Type 中选择 Data Context 项，选择在 Xaml 中定义好的 string 类型的 DataContext，然后点击确定，将此 DataContext 字符串绑定到 AlarmTextblock 的 Value 属性上，在 MainWindow 的 Design 界面可以看到 AlarmTextblock 显示了 Xaml 界面 Grid 中定义的 DataContext 的字符串值。



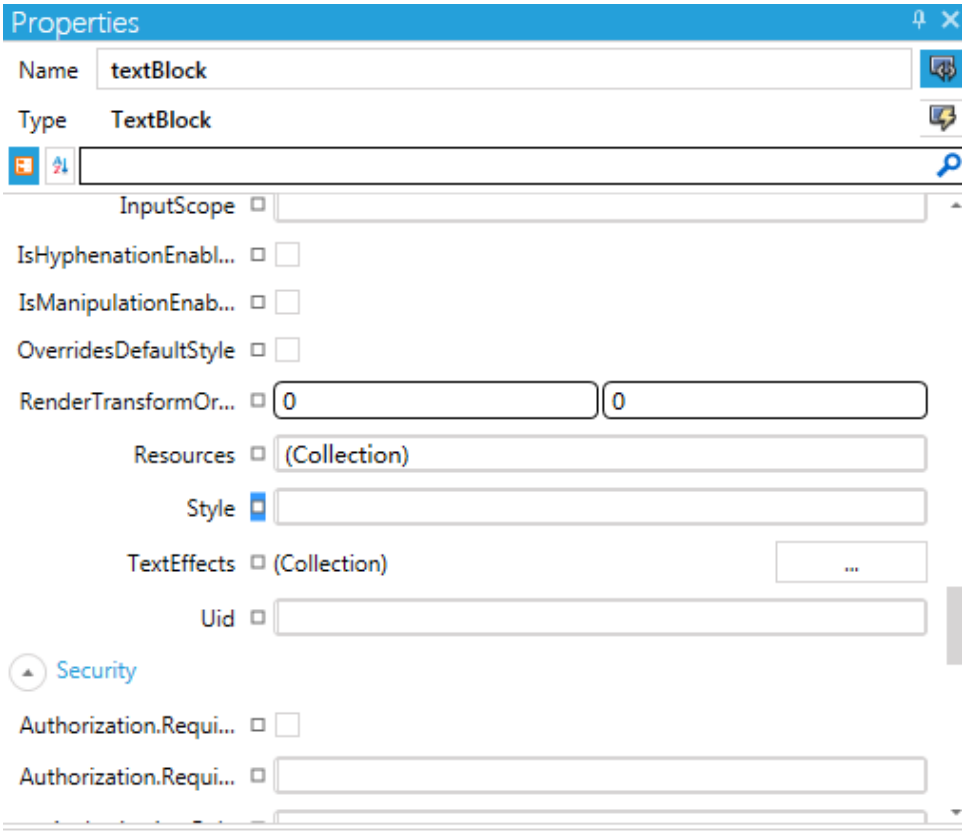


4.2.2 StaticResource 绑定

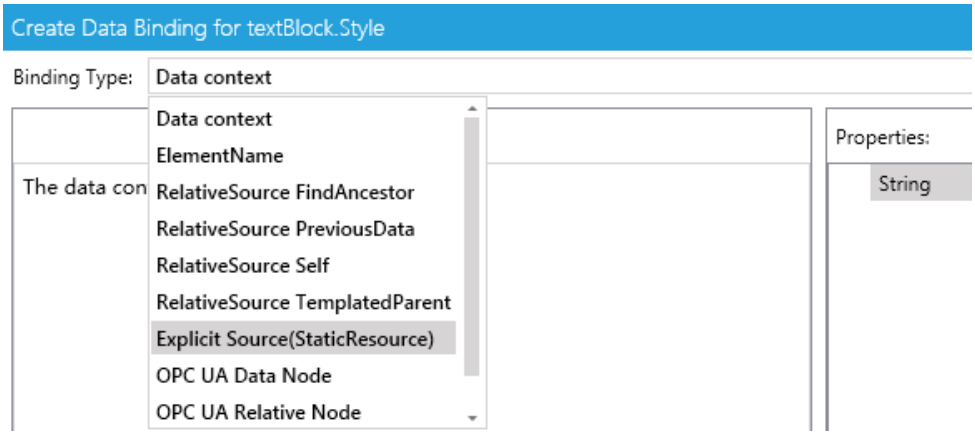
我们做一个示例，在 Xaml 界面定义一个静态资源 Style，键为“stl”，目标类型为 TextBlock.

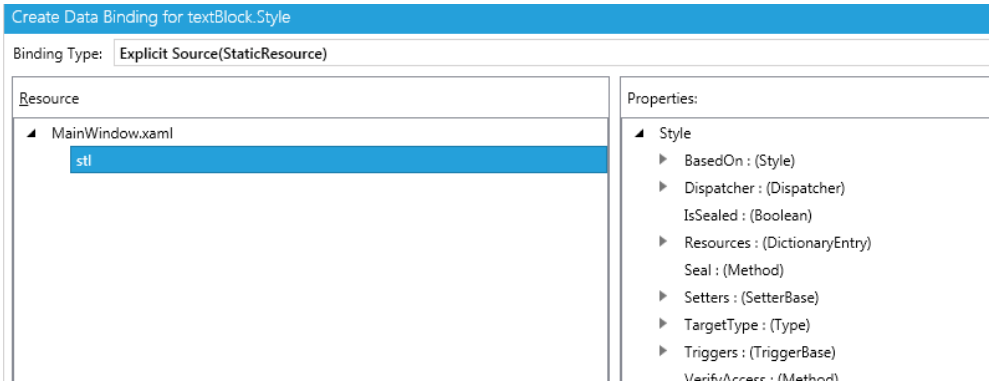


为将此定义好的 StaticResource 绑定到我们的 TextBlock 控件中，我们在 TextBlock 控件的 Properties 窗口找到 Style 属性，点击旁边的小方块，选择 Creat DataBinding 建立数据绑定。

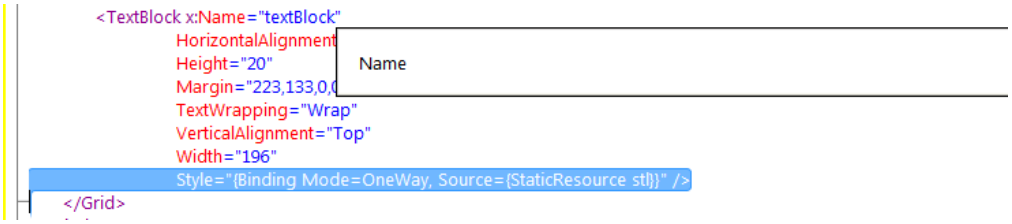


在 Binding Type 中选择 Explicit Source（StaticResource）,选择已经定义的 stl，点击 OK，使 stl 这个 Style 绑定到 TextBlock 中。



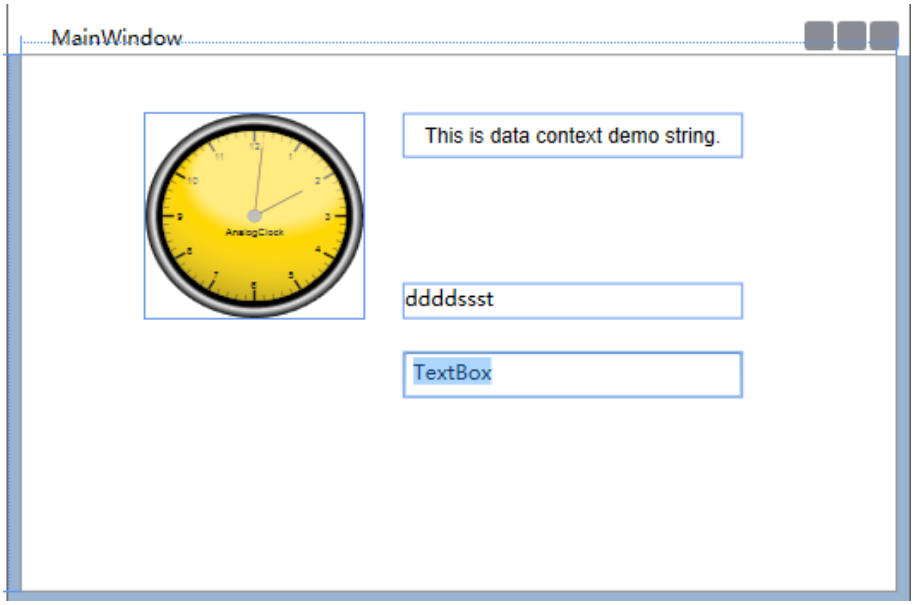


绑定完成后 Xaml 中的 TextBlock 代码，TextBlock 中如果设置了 Text 属性，记得要删掉，不然会因为优先级比 Style 高，而不显示静态资源 Style 中定义的 Text 值。

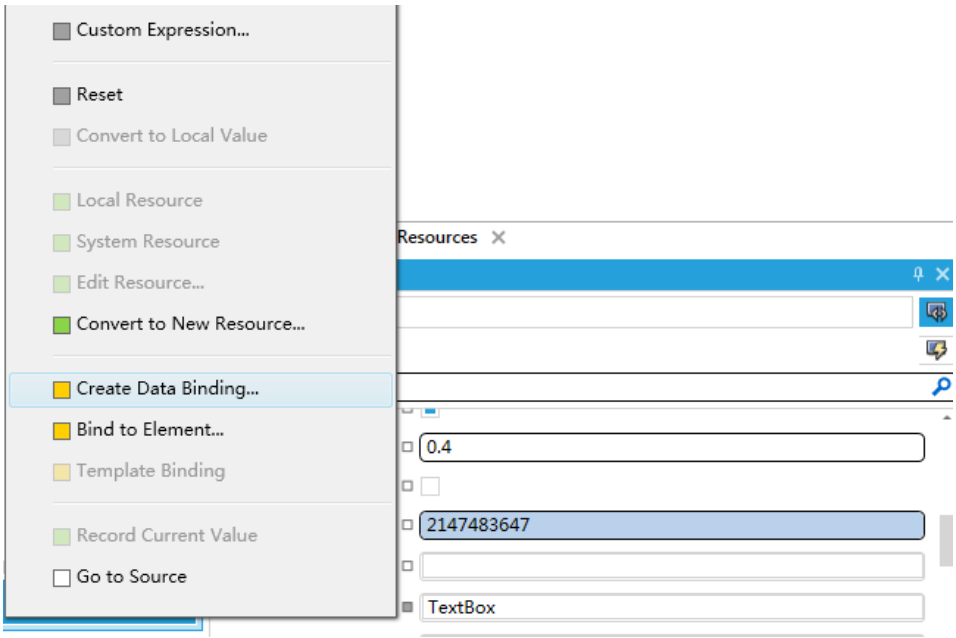


4.2.3 Element Name

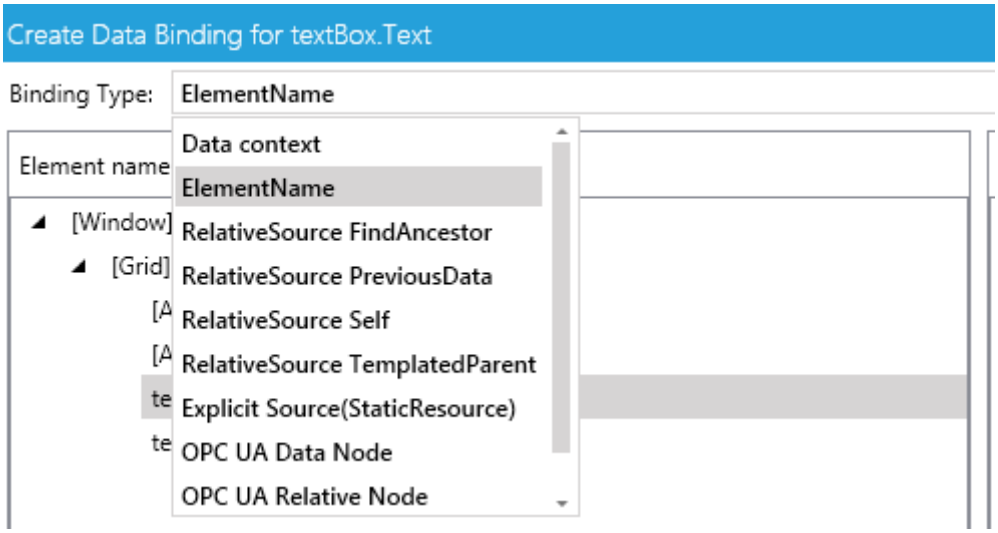
Elemen Name 是对于绑定的 Source 来说的，指明绑定的源控件的名称。下面做一个示例，增加一个 TextBox 到界面



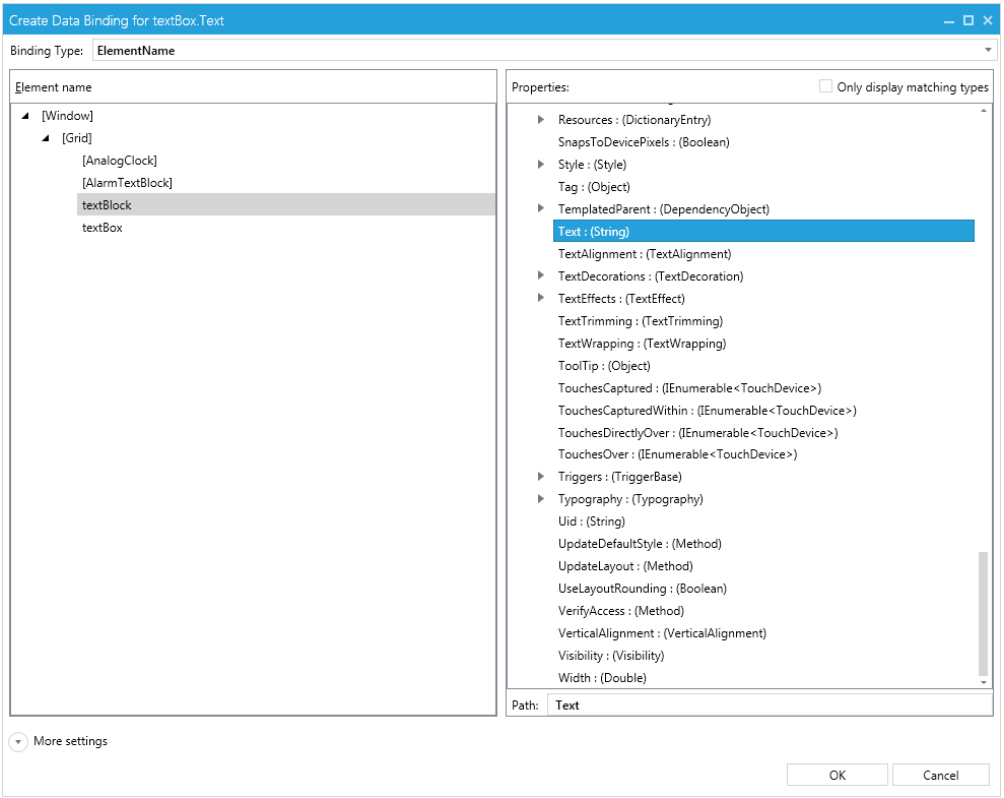
在 TextBox 的 Properties 属性框中找到 Text 属性，点击小方块，选择 Create Data Binding



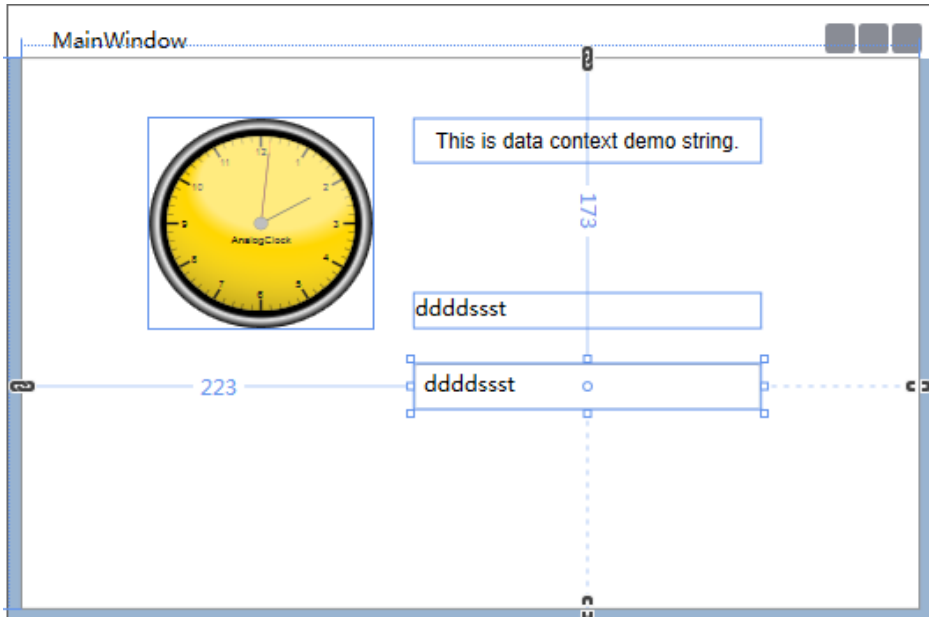
在 BindingType 中选择 ElementName.



然后选择 TextBlock 的 Text 属性作为绑定源。



点击 OK，后可以看到 TextBox 显示出了绑定的字符串，在 Xaml 中可以看到 TextBox 进行绑定的代码。

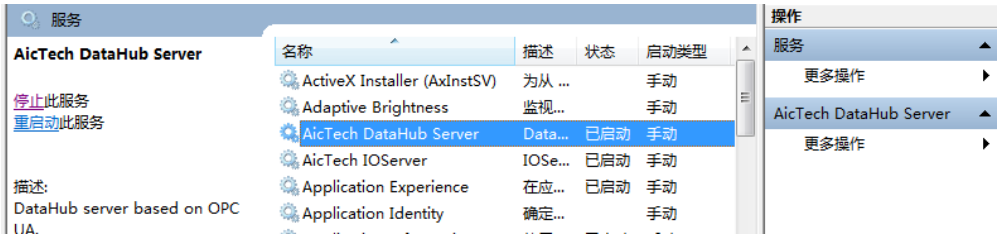



```
<TextBox x:Name="textBox"
         HorizontalAlignment="Left"
         Height="26"
         Margin="223,173,0,0"
         TextWrapping="Wrap"
         Text="{Binding Text, ElementName=textBlock}"
         VerticalAlignment="Top"
         Width="196" />
    </Grid>
</Window>
```

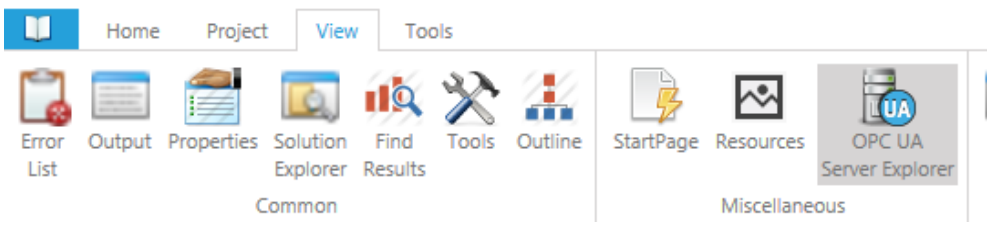
4.2.4 OPC UA Data Node

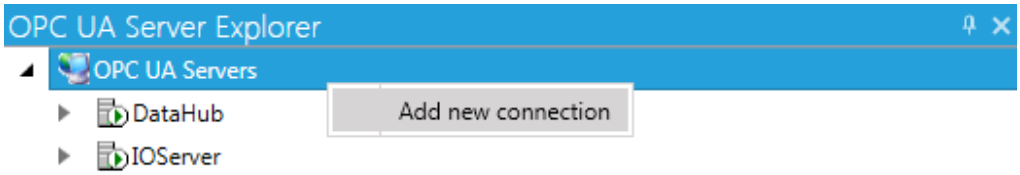
OPC UA 数据节点绑定，是特殊类型的绑定，它的绑定的对象是标准的 OPC UA 服务器（例如 AicDataHub）中已经定义好的变量。

首先我们连接 DataHub，以本机的 DataHub 为例，先打开 DataHub 服务。

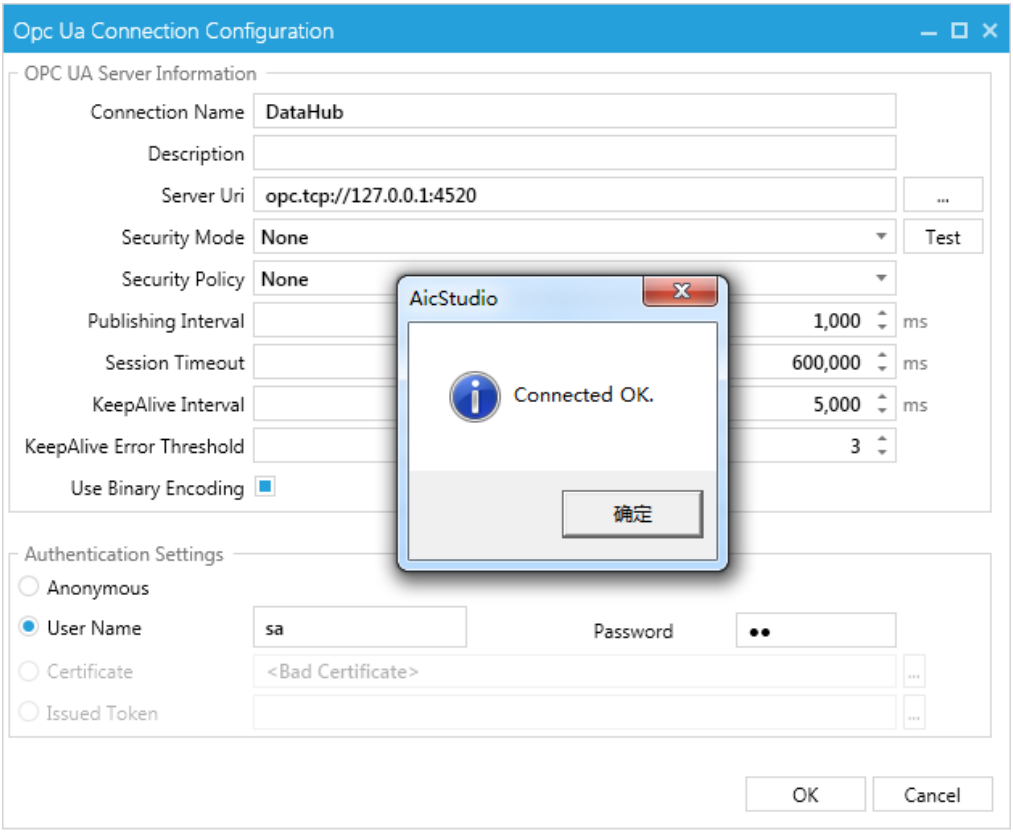


然后在 AicStudio 中的 View 菜单下打开 OPC UA Server Explorer，右键添加 Add New Connection



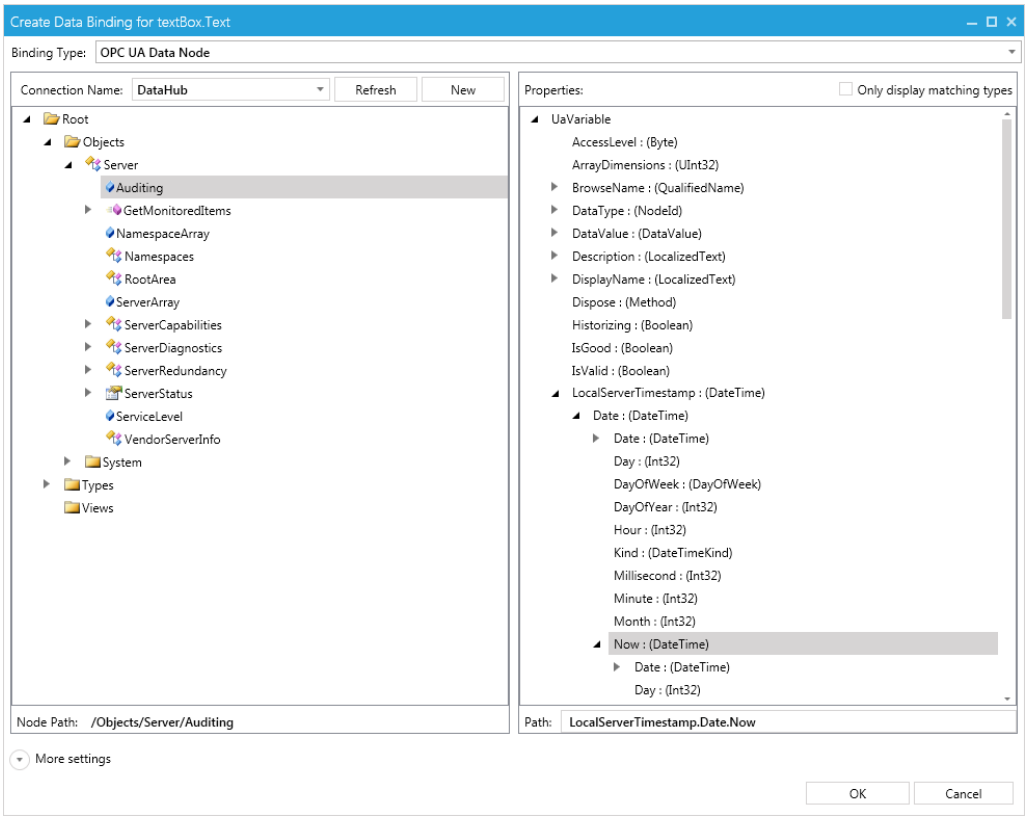


填写连接名称 Connection Name，Server Uri 地址，填写用户名 sa 和密码 sa（AicDataHub 默认的用户名和密码，可以通过 AicDataHub Configurator 工具进行修改）。点击 Test 测试连接，出现 Connected OK 表示连接成功，点击 OK 即已经添加 OPC UA Connection。双击连接的名称，即可进行连接。

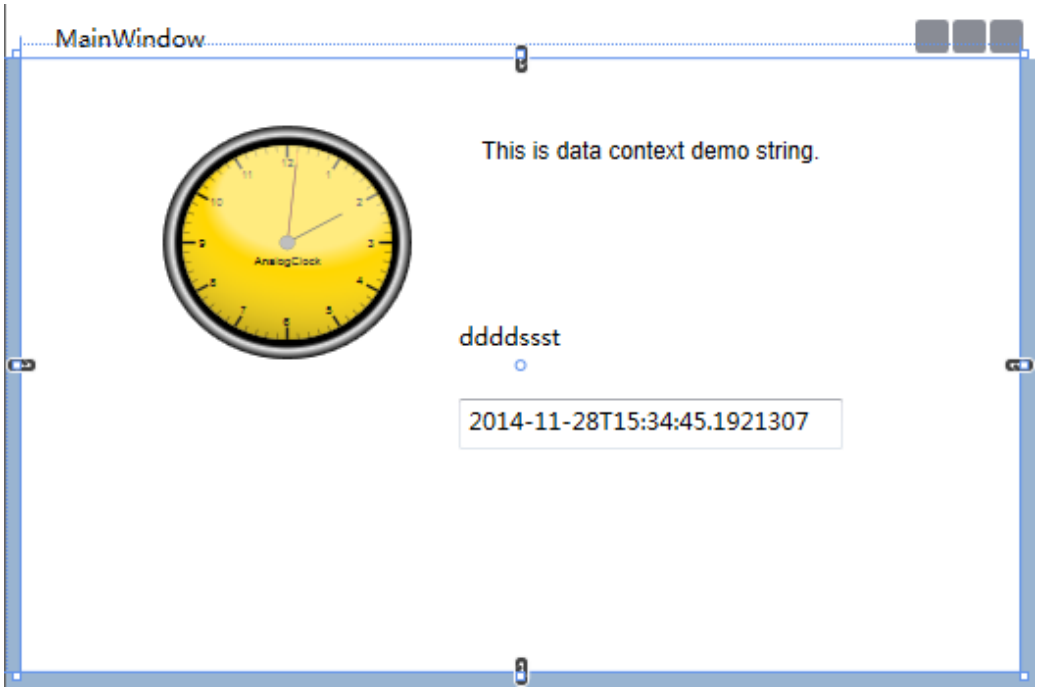


连接完成后，我们可以进行 OPC UA Data Node 的绑定添加。

在上个示例的 TextBox 的 Properties 属性界面中找到 Text 属性，点击小方块，选择 Create Data Binding，在弹出的对话框中，Binding Type 选框中选择 OPC UA Data Node，Connection Name 选择 DataHub，点击 Refresh，获取此 OPC UA 的连接，若连接没问题的话，会出现此连接下的菜单项，Root/Objects 等。我们选择将 Text 属性绑定到 Root/Objects/Server/Auditing 下 LocalServerTimestamp.Data.Now，即当前本地服务时间。点击 OK



在 MainWindow 界面，TextBox 显示了当前时间的字符串形式。

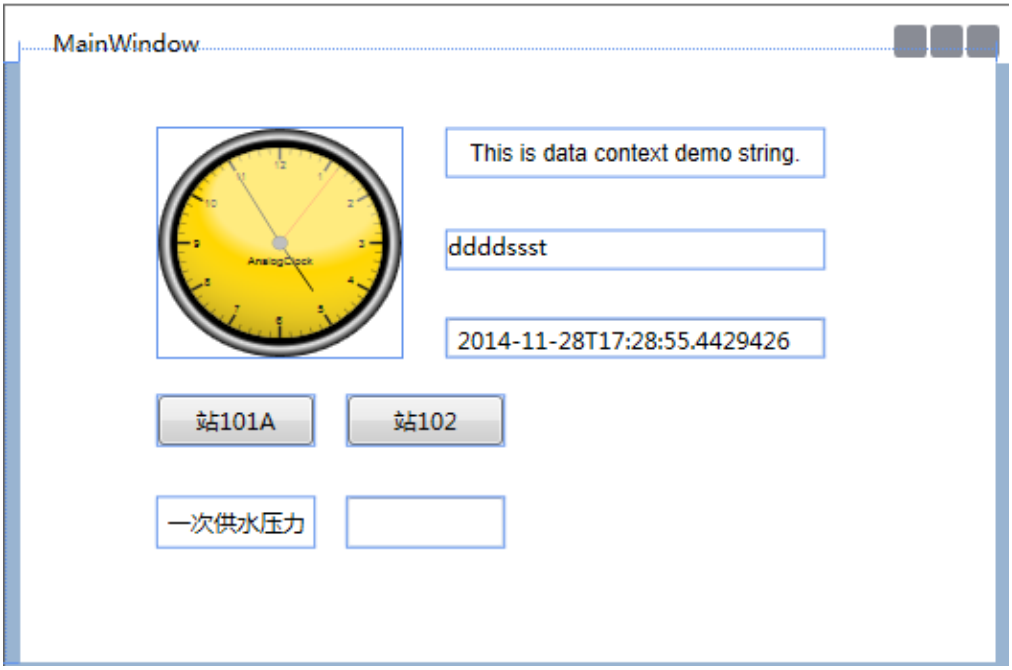


4.2.5 OPC UA Relative Node

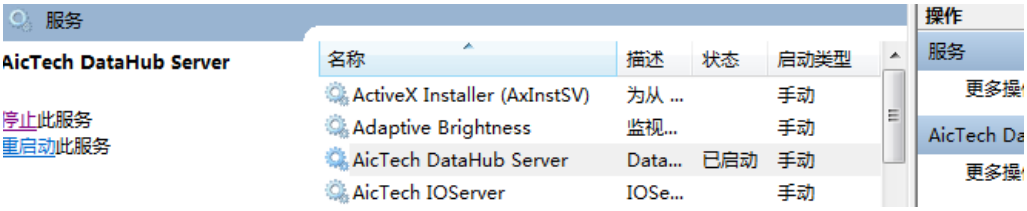
使用 OPC UA 节点相对绑定，可以动态切换绑定的对象，达到用一个显示界面可以切换显示多个对象的效果，这是 AicVision 套件（AicStudio+AicDataHub+AicIOServer）提供的独一

无二的面向对象/设备组态的功能。

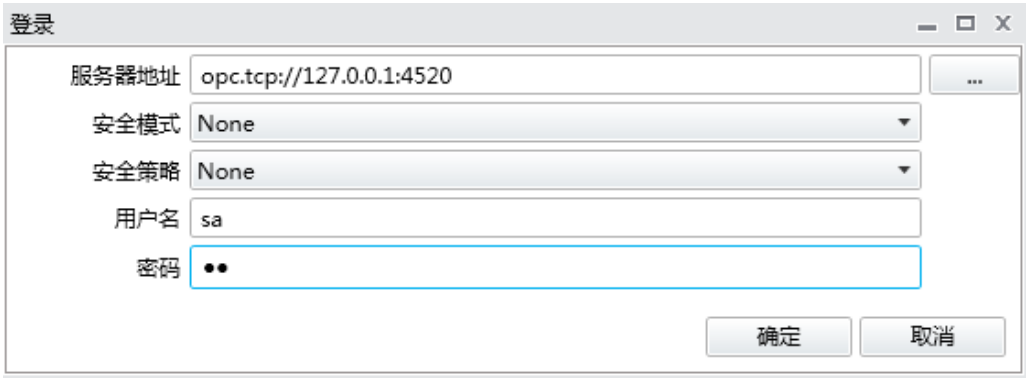
下面是一个简单的示例。首先添加显示的控件和按钮，两个按钮“站 101A”和“站 102”用于在点击时切换绑定对象；标签（一次供水压力）旁边的 TextBox 表示“站 101A”或者“站 102”的一次供水压力的值。



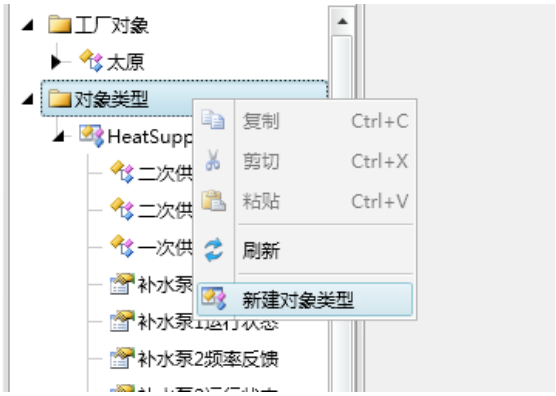
然后在系统的管理工具，服务中，打开 AicTech DataHub Server 服务。



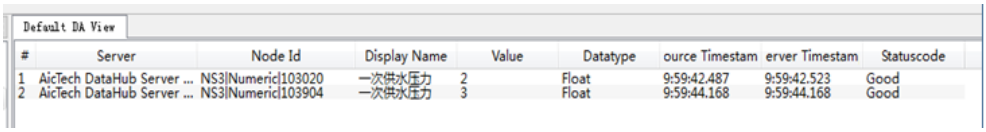
打开 AicTech DataHub Configurator 进行连接（密码 sa）。



以下是以 HeatSupplyStation 为示例，用户可新建对象类型及其变量，并在工厂对象中添加新建类型的对象。更多详细操作请参见 AicDataHub 文档资料。



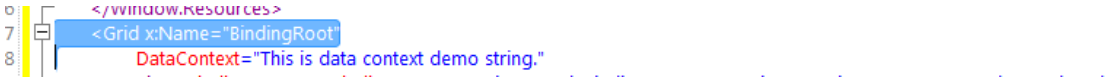
然后打开 UaExpert,为 DataHub 中相关变量赋值。



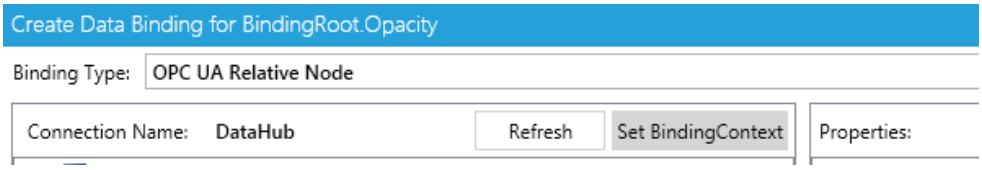
开始进行相对绑定：

步骤一、父类容器控件中指定 BindingContext 节点的对象类型

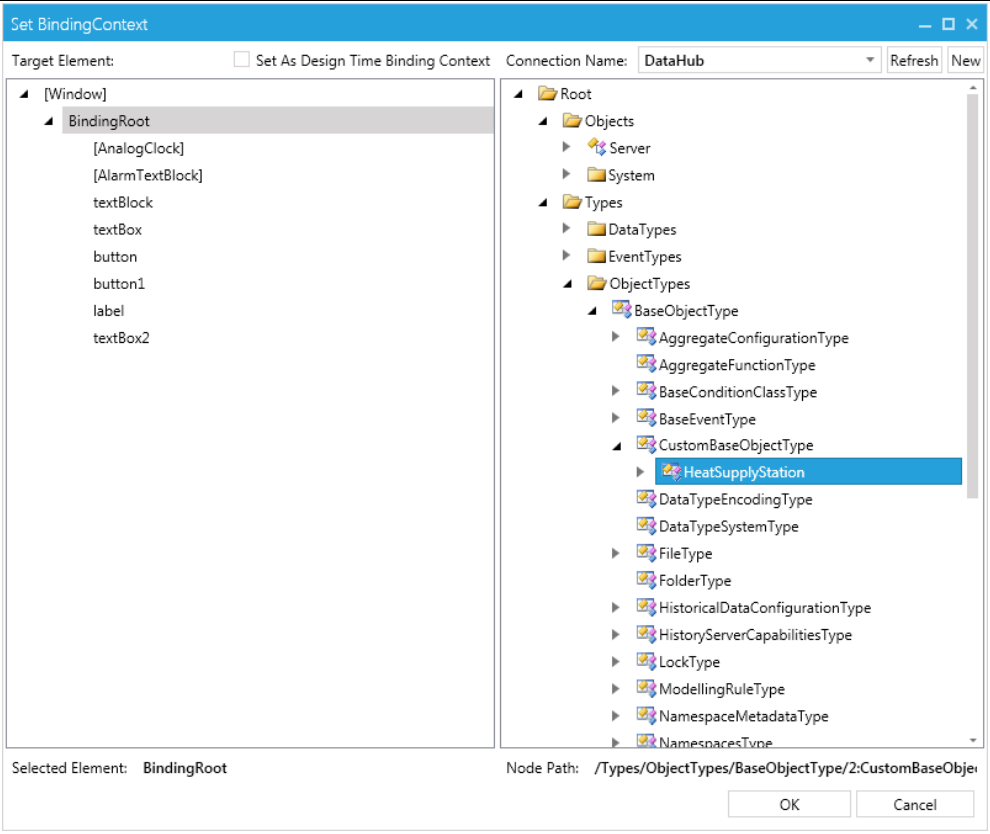
在 Grid 中设置好 Name 属性，将 Grid 命名。



随意点击一个属性旁边的小方块，选择 Create DataBinding 进入数据绑定界面，在 BindingType 栏选择 OPC UA Relative Node 项，然后点击 Set BindingContext。



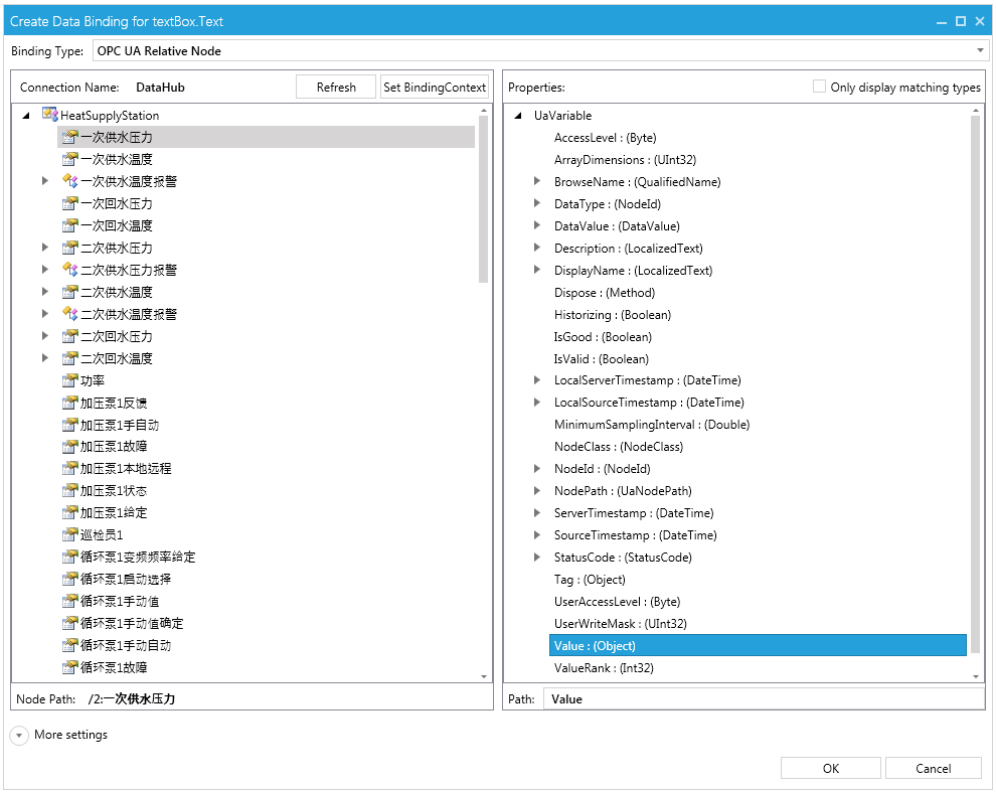
在 Set BindingContext 界面将命名 BindingRoot 的 Grid 与 DataHub 中用户定义的 HeatSupplyStation 类型进行绑定。注意：HeatSupplyStation 类型和此类型的对象“站 101”等还有此类型下的“一次供水压力”，都是 DataHub 工程中用户自定义的类型，如果本机 DataHub 没有此数据类型，可以登陆 DataHub 进行自定义，详细操作请参见 AicDataHub 文档。



点击 OK，步骤一进行完毕。

步骤二、确定要显示的属性值

在要显示一次供水压力值的 TextBox 的 Properties 窗口中找到 Text 属性，点击旁边的小方块，选择 Create Data Binding 建立数据绑定，选择一次供水压力的 Value 属性。



步骤三、建立 DataHub 中实际对象与 BindingContext 的关联

分别在站 101A 和站 102 两个按钮添加 Click 事件。

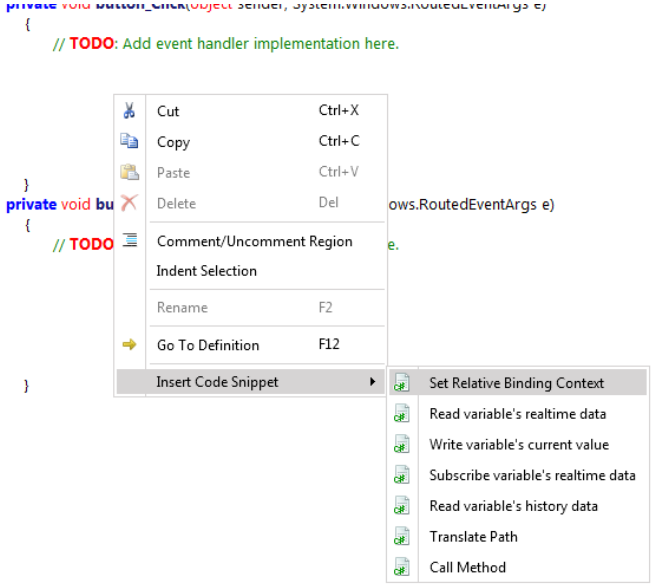
```
<Button x:Name="button"
        Content="站101A"
        HorizontalAlignment="Left"
        Height="26"
        Margin="72,173,0,0"
        VerticalAlignment="
        Width="81"
        Click="button_Click"
<Button x:Name="button1"
        Content="站102"
        HorizontalAlignment="Left"
        Height="26"
        Margin="171,173,0,0"
        VerticalAlignment="Top"
        Width="81"
        Click="button_Click1" />
```

Margin

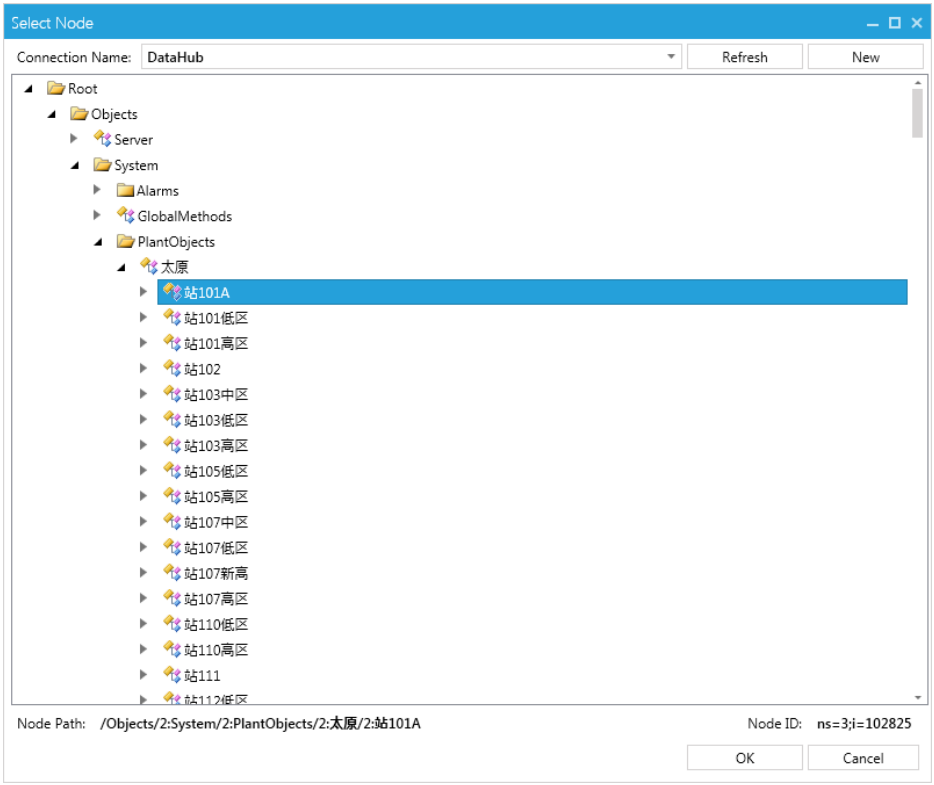
在.xaml.cs 中实现 Click 方法

```
private void button_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.



    }
private void button_Click1(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.



    }
```

在站 101A 按钮的方法中的空白处点击右键，选择 Set Relative Binding Context,用脚本编辑器自动添加 BindingContext 与实际对象关联的代码。



点击 Refresh,按图中路径选择站 101A，点击 OK



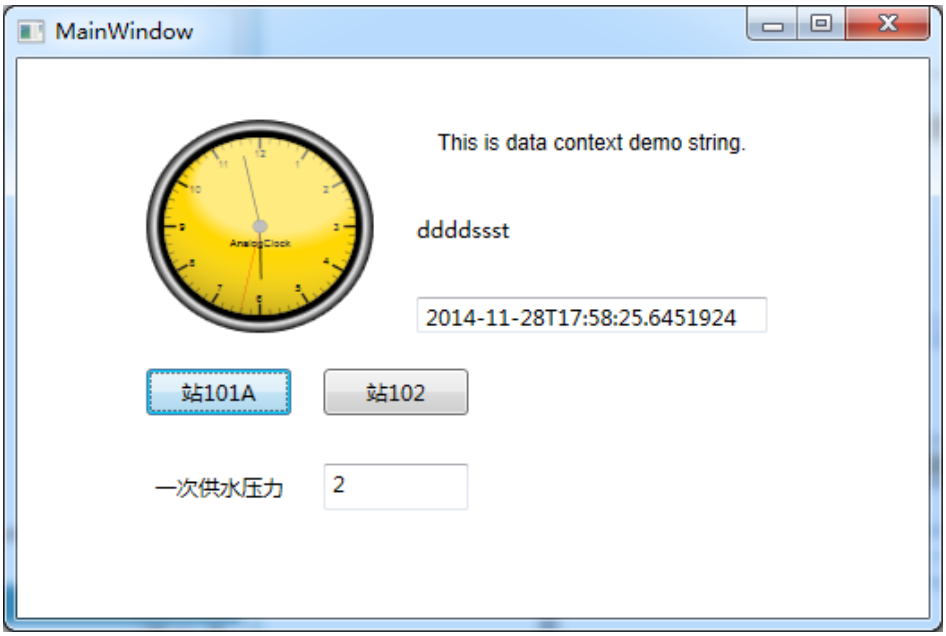
可以看到在 Click 方法中添加了如下的代码

```
private void button_Click(object sender, System.Windows.RoutedEventArgs e)
{
    // TODO: Add event handler implementation here.

    var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
    if (configuration == null || !configuration.IsValid) {
        throw new ServiceResultException(StatusCodes.BadConfigurationError);
    }

    UaBindingManager.SetBindingContext(
        this, // TODO: replace with binding target element(normally use root UI element)
        new UaNodeReference(configuration, new UaNodePath("/Objects/2:System/2:PlantObjects/2:太原/2:站101A")));
```

将 SetBindingContext 方法中的 this 参数改为 **this**.BindingRoot。
另外一个按钮的 Click 方法也按此步骤实现。最终代码如下图

```
private void button_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.

        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        UaBindingManager.SetBindingContext(
            this.BindingRoot, // TODO: replace with binding target element(normally use root UI element)
            new UaNodeReference(configuration, new UaNodePath("/Objects/2:System/2:PlantObjects/2:太原/2:站101A")));
    }
private void button_Click1(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.


        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        UaBindingManager.SetBindingContext(
            this.BindingRoot // TODO: replace with binding target element(normally use root UI element)
            new UaNodeReference(configuration, new UaNodePath("/Objects/2:System/2:PlantObjects/2:太原/2:站102")));
    }
```

然后运行 Start Debugging，点击按钮控件进行测试



4.2.6 相对源绑定 RelativeSource

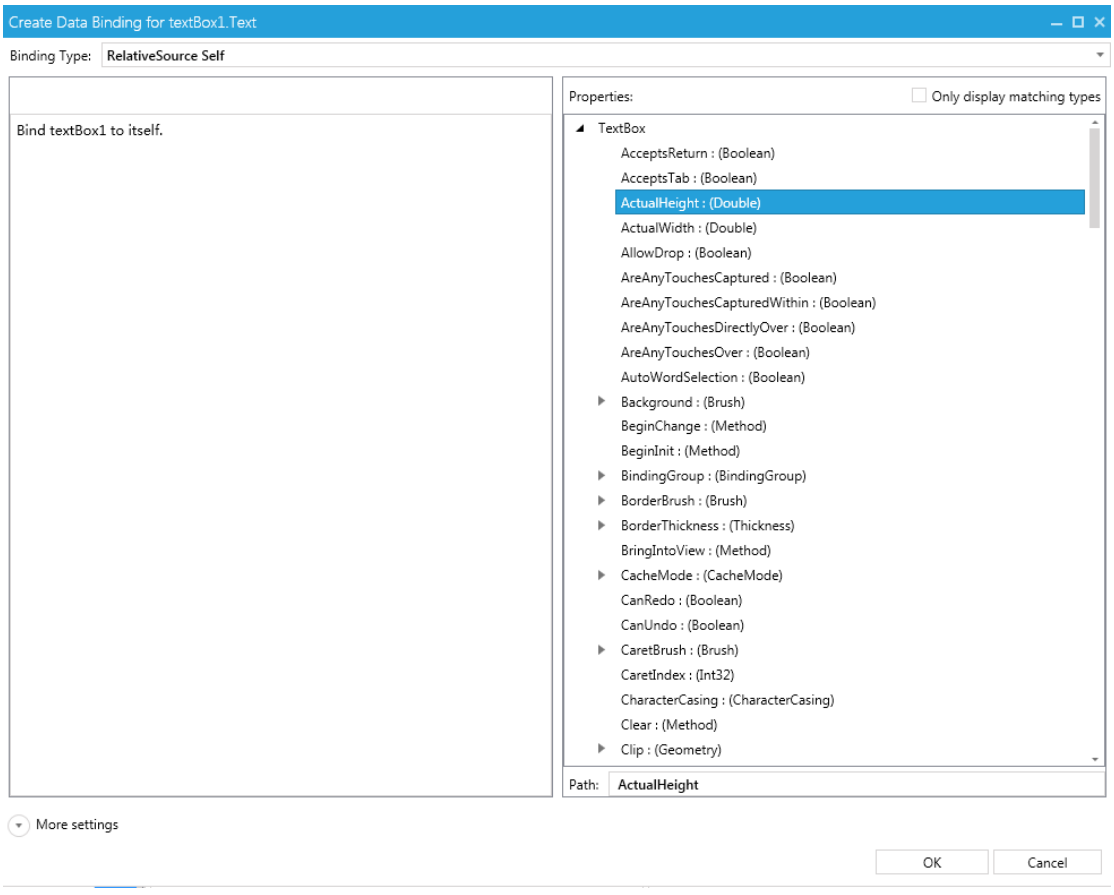
相对源绑定有 4 种模式，PreviousData, TemplatedParent, Self, FindAncestor
下图截至 System.Windows.Data.RelativeSourceMode

```
namespace System.Windows.Data
{
    // 摘要:
    //     描述相对于绑定目标位置的绑定源的位置。
    public enum RelativeSourceMode
    {
        // 摘要:
        //     允许您绑定所显示数据项列表中以前的数据项（不是包含数据项的控件）。
        PreviousData = 0,
        //
        // 摘要:
        //     引用应用了模板（其中有数据绑定元素）的元素。 这类似于设置 System.Windows.TemplateBindingExtension，并仅当
        //     System.Windows.Data.Binding 在模板中时适用。
        TemplatedParent = 1,
        //
        // 摘要:
        //     引用您对其设置绑定的元素，并允许您将该元素的一个属性绑定到同一元素中的其他属性。
        Self = 2,
        //
        // 摘要:
        //     引用数据绑定元素父链中的上级。 您可以使用它绑定到特定类型或其子类的上级。 如果您要指定 System.Windows.Data.RelativeSource.AncestorType
        //     和/或 System.Windows.Data.RelativeSource.AncestorLevel，可以使用此模式。
        FindAncestor = 3,
    }
}
```

Self 模式：

Self 模式将元素自身的属性作为绑定的数据源。

示例：将 TextBox 的 ActualHeight 属性作为数据源绑定到 Text 属性上，进行显示。在 Properties 窗口找到 Text 属性，点击小方块，选择 CrateDataBinding 进入数据绑定界面。



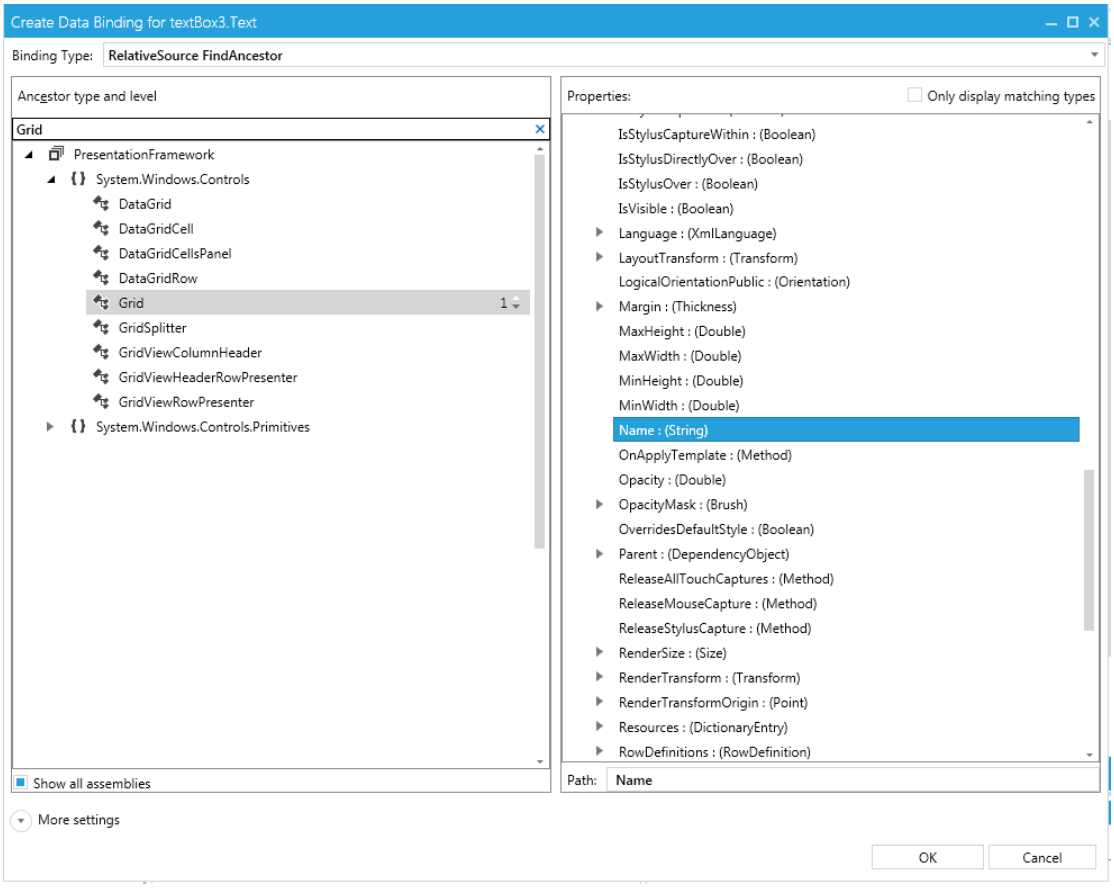
选择 ActualHeight,点击 OK 确定绑定，绑定后的 Xaml 代码如图。

```
<TextBox x:Name="textBox1"
         HorizontalAlignment="Left"
         Height="26"
         Margin="289,173,0,0"
         TextWrapping="Wrap"
         Text="{Binding ActualHeight, Mode=OneWay, RelativeSource={RelativeSource Self}}"
         VerticalAlignment="Top"
         Width="35" />
```

FindAncestor 模式：

此模式的标记扩展可以实现访问此元素的 n 级父类的所有公开属性，作为绑定的数据源。示例：选择一个 TextBox，将此控件的容器控件 Grid 的 Name 作为数据源，绑定到 Text 属性上。

在 Text 属性上选择 CreateDataBinding,进入绑定窗口，BindingType 选择 RelativeSource FindAncestor,在 Ancestor type and level 框中输入 Grid 搜索出 Grid 控件，在右边 Properties 选框中选择 Name 属性，作为数据源进行绑定。



点击 OK，绑定成功后的 Xaml 代码如下

```
<TextBox x:Name="textBox3"
         HorizontalAlignment="Left"
         Height="26"
         Margin="289,226,0,0"
         TextWrapping="Wrap"
         Text="{Binding Name, RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Grid}}}"
         VerticalAlignment="Top"
         Width="83" />
```

TemplatedParent 模式：

RelativeSource 使用 TemplatedParent 模式时，仅在控件模板（ControlTemplate）或者数据模板（DataTemplate）下有效。不同的模板，将返回不同类型的绑定结果。例如，在一个 ListBox 数据模板（DataTemplate）中应用 RelativeSource 的 TemplatedParent 模式，则会返回 ContentPresenter 模板内容到对应数据模板中。TemplatedParent 模式可以帮助开发人员轻松绑定模板中的属性值到目标对象属性。例如：

使用 TemplatedParent 定义 Button 样式

```
<Style x:Key="sstl"
       TargetType="{x:Type Button}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <TextBlock Background="Red"
                           Text="{Binding ActualHeight, Mode=OneWay, RelativeSource={RelativeSource  TemplatedParent}}"
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

使用此样式的按钮 XAML 代码

```
<Button x:Name="button2"
        Style="{StaticResource sstl}"
        HorizontalAlignment="Left"
        Height="34"
        Margin="72,257,0,0"
        VerticalAlignment="Top"
        Width="81" />
```

按钮效果图,显示按钮高度。

值得注意的是，在控件模板（ControlTemplate）中使用 RelativeSource 的 TemplatedParent 模式，” Binding RelativeSource={RelativeSource TemplatedParent}} “等价于” {TemplateBinding}" 标记扩展。

两者不同在于，TemplateBinding 仅支持单向（One-Way）绑定，而 RelativeSource 标记扩展支持双向（Two-Way）绑定，这个功能在创建自定义控件模板时特别有用。

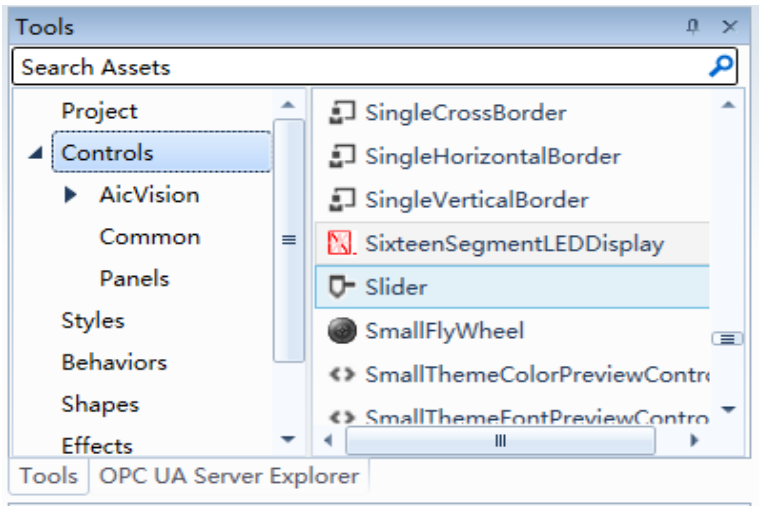
PreviousData 模式

该模式很少使用，需要了解的可以参考微软 MSDN 中的相关内容：
https://msdn.microsoft.com/zh-cn/library/system.windows.data.relativesource.previousdata%28v=vs.110%29.aspx

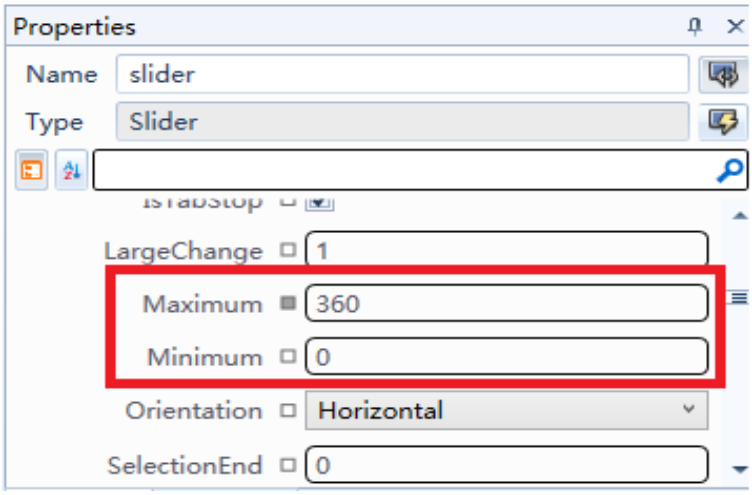
4.3 动画 Animation(行为)

4.3.1 旋转（RotateAction）

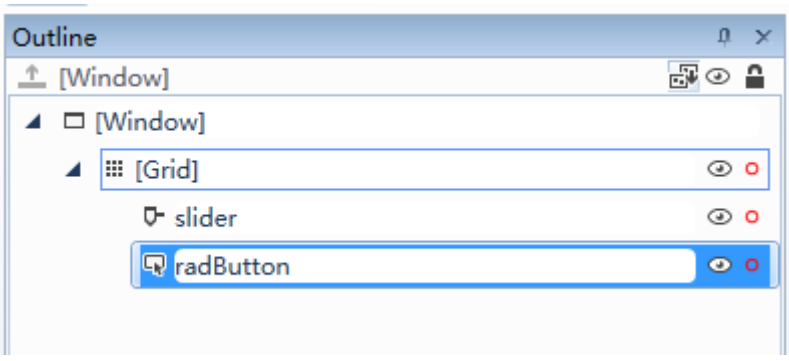
在 AicStudio 中新建一个 wpf 界面，从控件区拖入一个 slider 控件如下图：



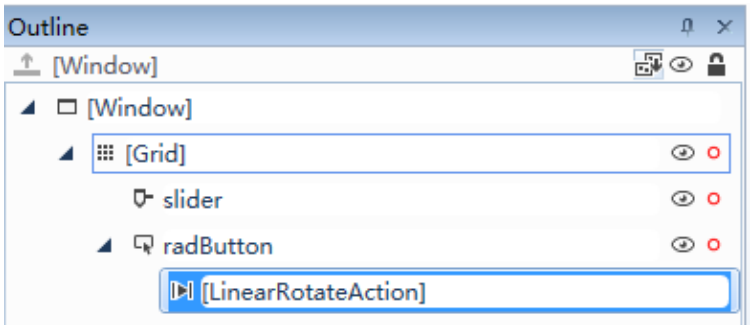
选中后适当调整控件长宽高，并在属性窗口设置其最大值（Maximum）和最小值（minimum）为 360 和 0，如下图：



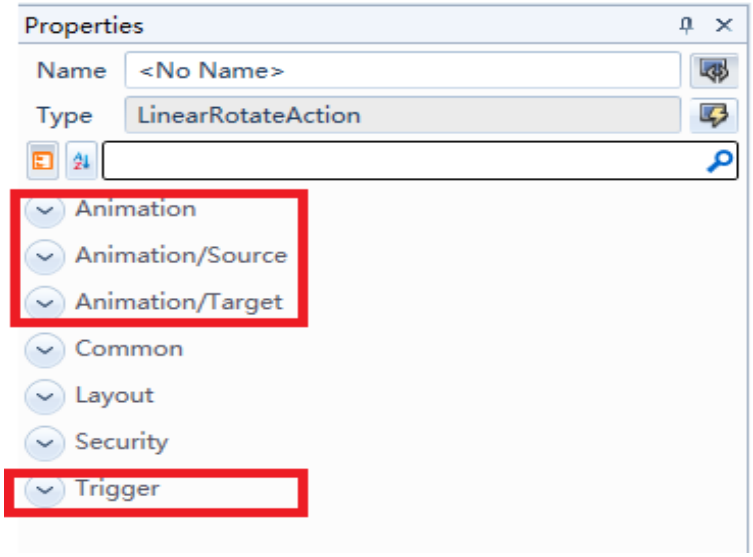
再从控件区拖入一个 RadButton 控件，适当调整大小，将其 Content 属性改为 Rotate，在大纲处选中，如下图：



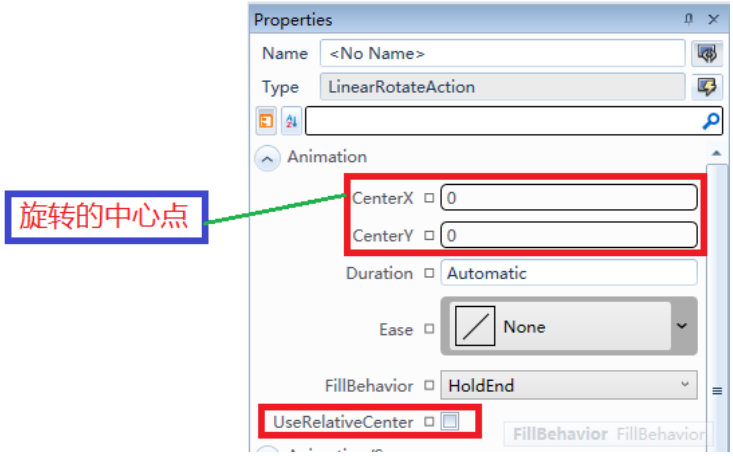
选中后在工具窗口的左窗口选择行为（Behaviors），在右窗口双击 LinearRotateAction 为此按钮控件添加旋转动画。添加进来后在大纲中选中 radButton 下的 LinearRotateAction，如图：



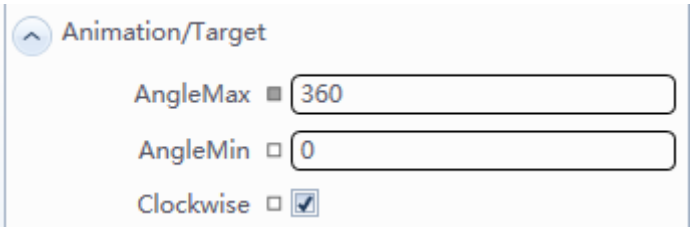
选中后看属性窗口这个动画的属性如下图：



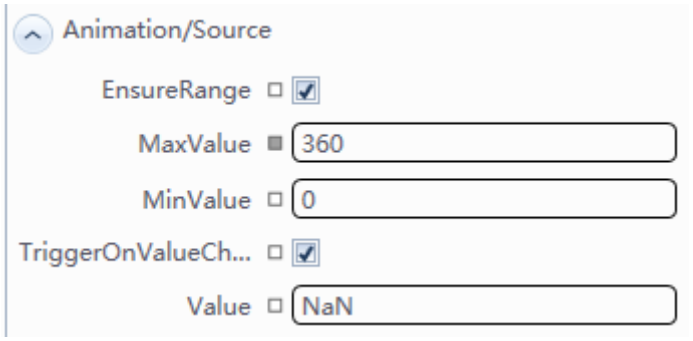
这里讲几个红线框中常用的几个属性，在 Animation 中 CenterX 和 CenterY 用于确定旋转的中心点，举个例子 CenterX 和 CenterY 都填 0.5，并且在 UseRelativeCenter 打上勾，则旋转的中心点就是这个按钮控件的中心点。注意：若要自己设置中心点，须在 UseRelativeCenter 打上勾，不打勾则是默认的中心点，为 0 和 0，即按钮控件的左上角的顶点。在此设置 CenterX 和 CenterY 都为 0.5，在 UseRelativeCenter 打上勾。



在 Animation/Target 中（如下图）AngleMax 和 AngleMin 用于设置旋转角度的范围，在此设置最大角度为 360，最小角度为 0，ClockWise 为动画时旋转方向（在下文会详细介绍）。

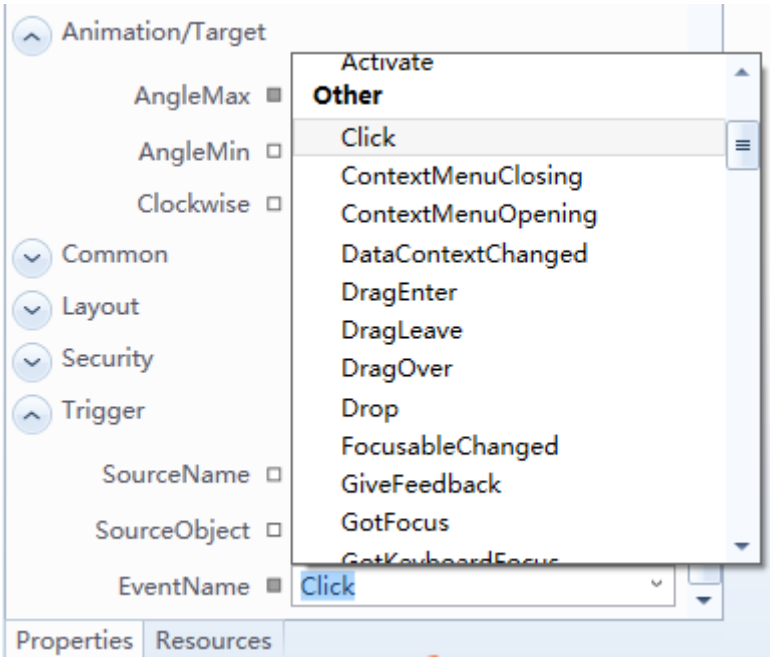


在 Animation/Target 中（如下图）设置最大值为 360，最小值为 0。说明：这里的最大值、最小值和之后要绑定的 Value 值最终反应目标控件的旋转动画上时是按照比例进行运算的，举个例子，如果最大值是 100，最小值是 50，Value 值是 75，则按钮控件实际的角度为(75-50)/(100-50)*(360-0)+0=180，这里的 360 和 0 是 Animation/Target 中 AngleMax 和 AngleMin 的属性值。如图中的 Value 值是 NaN，表示 Not a Number，这里需要为 Value 值进行绑定，具体绑定的方法见本文档 4.2 数据绑定章节中的第 3 部分，这里将 Value 值绑定到 slider 的 Value 属性。TriggerOnValueChanged 默认是打勾的。（在下文会详细介绍）

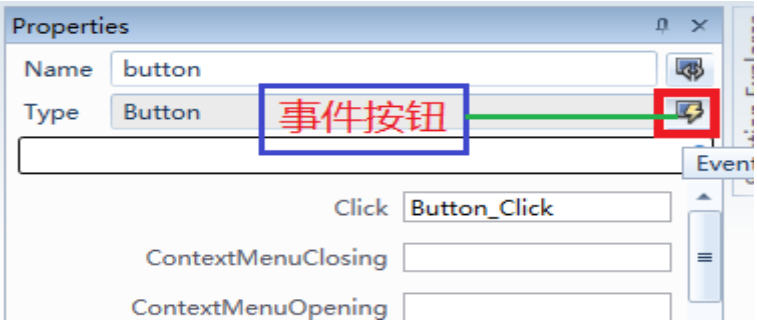


至此，可以运行当前的程序可以看到效果如下图，滑动 slider 控件滑块，则按钮控件会绕其中心店旋转：

在 Trigger（如下图）中，有 EventName 属性，在这里，这个属性是就是 radButton 所拥有的事件，我们现在已经可以实现 radButton 的角度随 slider 拉动滑块进行变动，若将 Animation/Target 中的 TriggerOnValueChanged 的勾去掉，并选择 EventName 为 Click，则我们可以看到滑动 slider 的滑块，radButton 并不旋转，需要点击 radButton（出触发 Click 事件）之后 radButton 转到对应的角度。



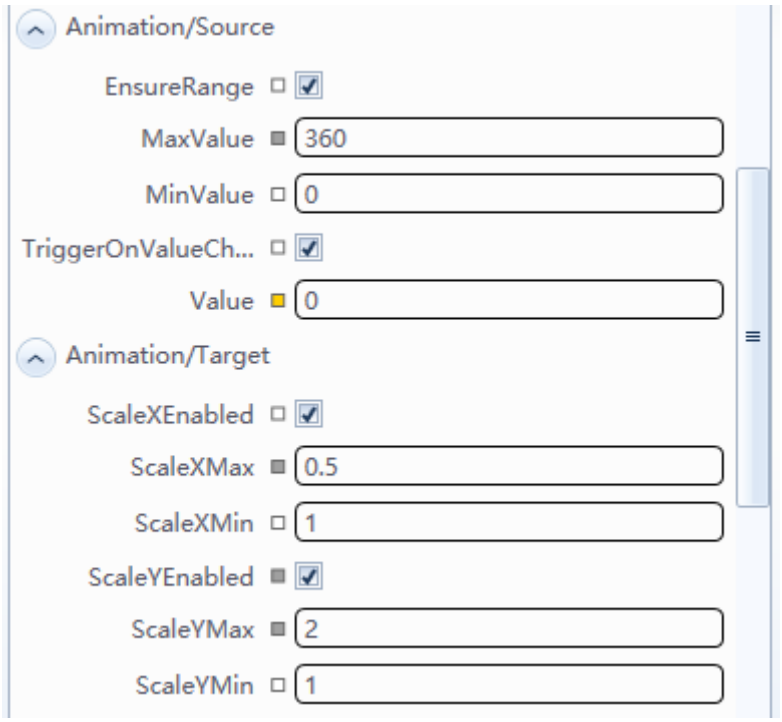
在 Animation/Target 中，有个 Clickwise 属性：这个属性是动画过程中目标旋转的方向，默认为顺时针，即 Clickwise 打上勾。说明：我们现在已经实现的功能主要的应用情况是：实时地监测某一个物体的二位角度姿势（绑定实时数据库中相应的数据），这只能说是实时监控的动画表现，对于这种动画，Clickwise 是不起任何作用的。其实，和 Clickwise 属性关系很密切的一个属性是 Animation 中的 Duration 属性，这个属性的赋值举例为：00:00:03，这个就是动画持续的时间，此为 3 秒，这里所说的动画其含义为：从一个角度到另一个角度所用的时间需要 3 秒（如果 Duration 属性是默认值 Automatic 的话，从一个角度到另一个角度是跳变的，我们现在已经实现的效果就是这种跳变式的）。我们可以做个试验：将 linearRotateAction 属性中的 Value 属性的数据绑定去掉（重新对 Value 赋值按 Tab 即可），从工具窗口的控件区拖入一个 Button，将其 Content 属性设置为 RotateTest，在属性窗口点击其事件按钮（如下图），在 Click 右边输入 Button_Click，双击添加单击路由事件处理方法，在方法中输入图示代码，设置 TriggerOnValueChanged 属性打上勾，设置 Animation 中的 Duration 属性为 00:00:01，设置 Clickwise 属性打上勾，运行程序后点击后添加的按钮，动画效果点击一次是顺时针转到 90 的位置，若把 Clickwise 属性的勾去掉，则是逆时针旋转到 90 度的位置。

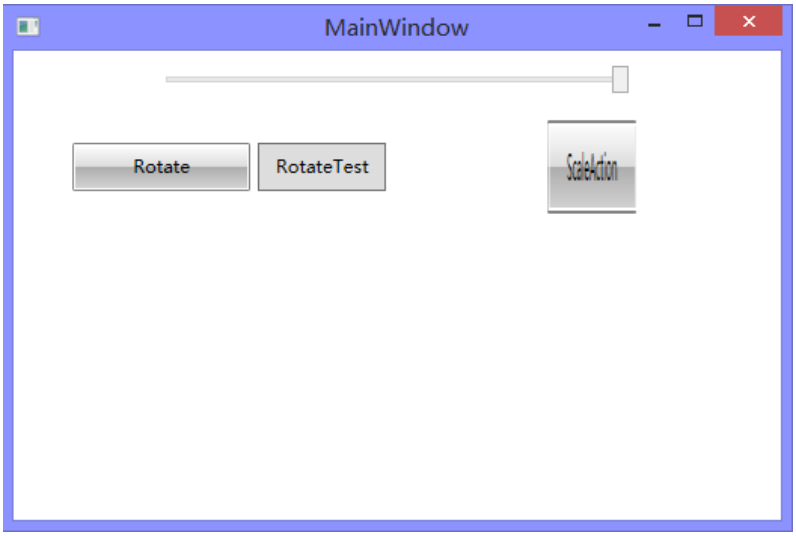


```
        private void Button_Click(object sender, System.Windows.RoutedEventArgs e)
        {
            this.linearRotateAction.Value+=90;
        }
    }
}
```

4.3.2 缩放（ScaleAction）

从工具窗口的控件区再拖入一个 RadButton，将其 Content 属性设置为 Scale，按 4.3.1 中添加动画的方法，添加 LinearScaleAction 动画，此动画的属性中的 TriggerOnValueChanged 属性、EventName 属性和上文一样，不再赘述；Animation 中的 CenterX 和 CenterY 属性和旋转中的定义相同，(0,0)表示按钮的左上角的顶点，设置 CenterX 和 CenterY 为 0.5，在 UseRelativeCenter 属性打上勾；Animation/Source 中的 MinValue 和 MaxValue 属性设置为 0 和 360，同 4.3.1 旋转中的 Value 属性，将 Value 属性绑定到 slider 控件的 Value 属性；在 Animation/Target 中（如下图），缩放可以是只缩放 X 方向（横向）、只缩放 Y 方向（纵向），也可以是两个方向同时缩放，将 ScaleXEnabled 和 ScaleYEnabled 属性都打上勾，设置 ScaleXMax 为 0.5，ScaleXMin 为 1，ScaleYMax 为 2，ScaleMin 为 1，这里说明一点，不一定 Max 值一定要大于 Min，这几个值只是表示 X 方向和 Y 方向两个极端（最大和最小）的两种状态，联合 Animation/Source 中的 MaxValue、MinValue 和 Value，根据比例算的具体的缩放状态，运行滑动 slider 滑块效果如图。

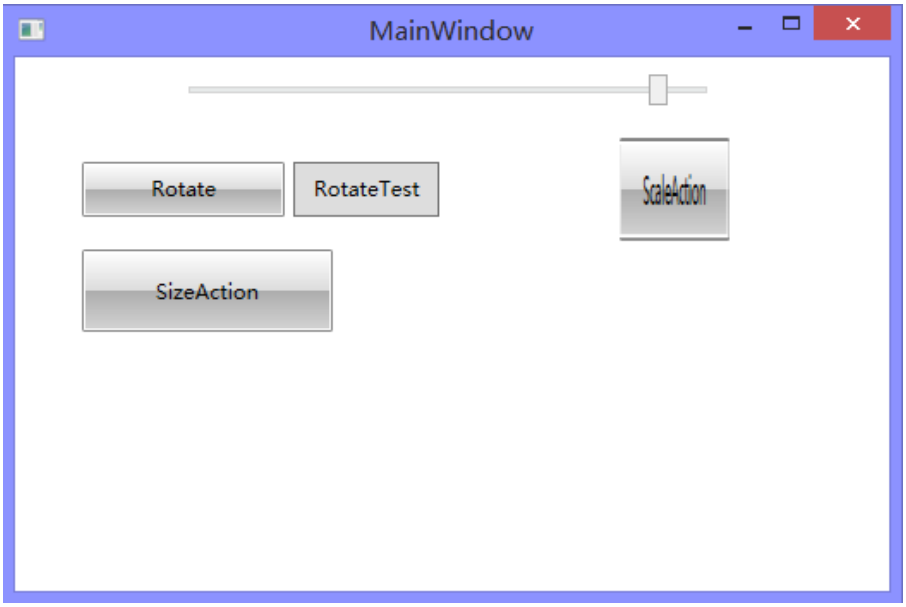




4.3.3 尺寸变化（SizeAction）

从工具窗口的控件区再拖入一个 RadButton，将其 Content 属性设置为 SizeAction，按 4.3.1 中添加动画的方法，添加 LinearSizeAction 动画，此动画的属性中的 TriggerOnValueChanged 属性、EventName 属性、Duration 属性和上文一样，不再赘述。

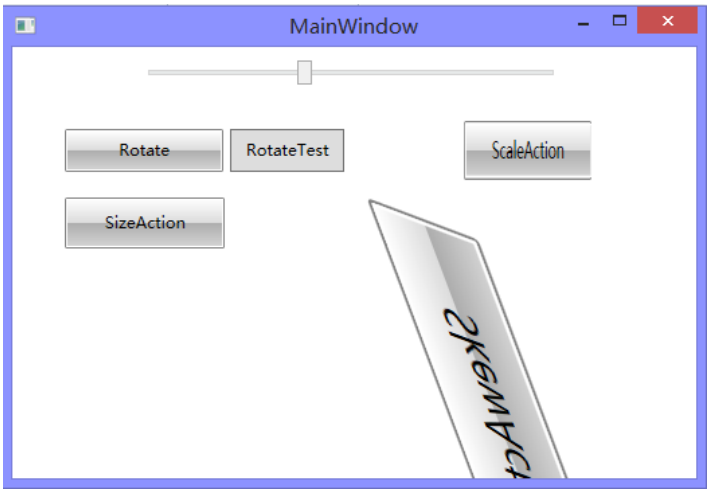
此动画和 4.3.2 缩放较相似：都可以控制控件外形尺寸上的变化；都可以单独控制横向或纵向的尺寸变化，也可以同时控制。区别在于：此动画没有 CnterX，CenerY 属性，默认为左上角的顶点；缩放的尺寸变化是相对于原来控件的宽和高来说的（即成比例的放大或缩小），但此动画为绝对的尺寸变化（和控件初始状态的宽和高无关），设置 Animation/Target 中的 HeightEnabled 和 WidthEnabled 都打上勾，设置 HeightMax 为 30，HeightMin 为 50，WidthMax 为 150，WidthMin 为 100，Animation 中的 MaxValue 为 360，MinValue 为 0，Value 属性绑定到 slider 控件的 Value 属性，运行滑动 slider 滑块效果如下图。从图中我们还可以发现：此动画值变化尺寸，并不变化控件中显示的文字的大小，但 4.3.2 缩放动画不但改变控件尺寸，也改变控件中的文字的尺寸。



4.3.4 倾斜扭曲（SkewAction）

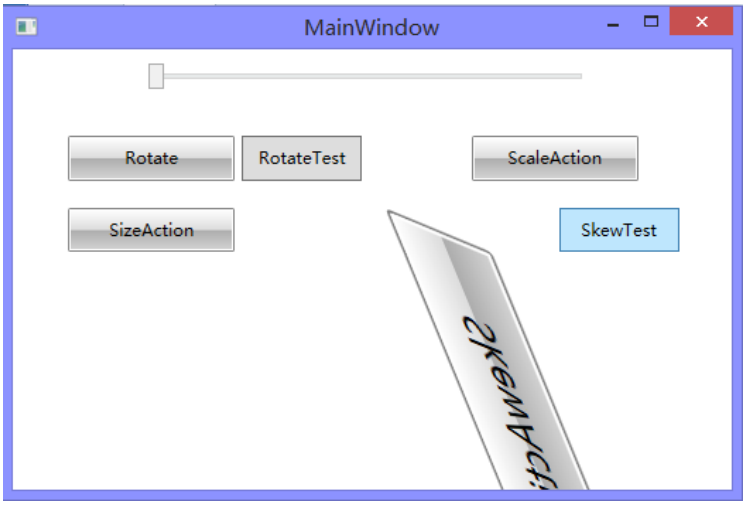
从工具窗口的控件区再拖入一个 RadButton，将其 Content 属性设置为 SkewAction，按 4.3.1 中添加动画的方法，添加 LinearSkewAction 动画，此动画的属性中的 TriggerOnValueChanged 属性、EventName 属性、Duration 属性和上文一样，不再赘述。

此动画的 CenterX 属性和 CenterY 属性和 4.3.1 旋转的对应属性相同，(0,0)表示控件左上角的顶点。设置 Animation/Source 的 MaxValue 属性为 360，MinValue 为 0，Value 属性绑定到 slider 控件的 Value 属性；Animation/Target 中，设置 SkewXEnabled 和 SkewYEnabled 打上勾，设置 SkewXMax 为 180，SkewXMin 为 0，SkwYMax 为 180，SkewYMin 为 0，运行，滑动 slider 控件的滑块，效果如下图。



插入一个 Button 控件将 SkewAction 控件的 LinearSkewAction 的 Name 属性设置为 linearSkewAction1，将其 Content 属性设置为 SkewTest，删除 SkewAction 控件的 Value 的数据绑定，按 4.3.1 添加 SkewTest 控件的单击事件，在添加的路由事件处理方法中添加如下代码，设置 Duration 为 00:00:01，运行，点击 SkewTest 按钮，运行效果如下图。

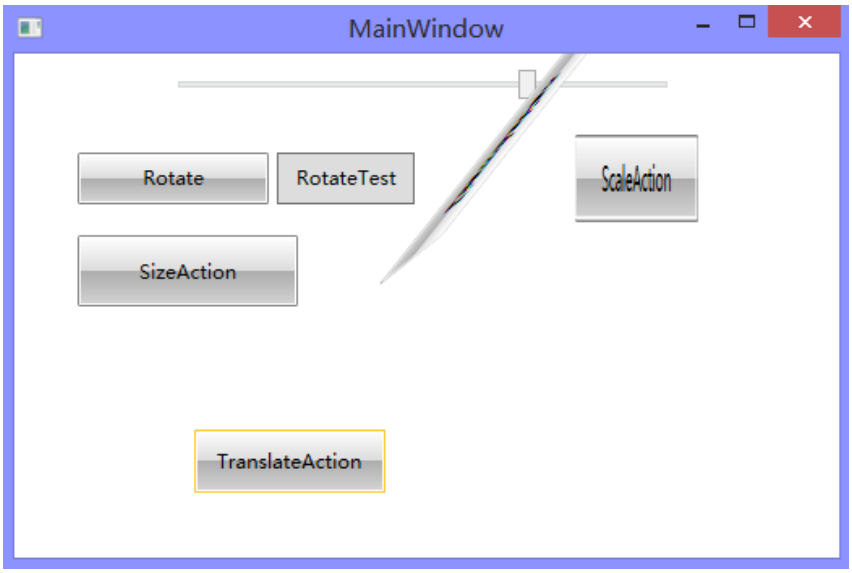
```
private void SkewTest_Click(object sender, System.Windows.RoutedEventArgs e)
{
    // TODO: Add event handler implementation here.
    this.linearSkewAction1.Value+=45;
}
```



4.3.5 平移（TranslateAction）

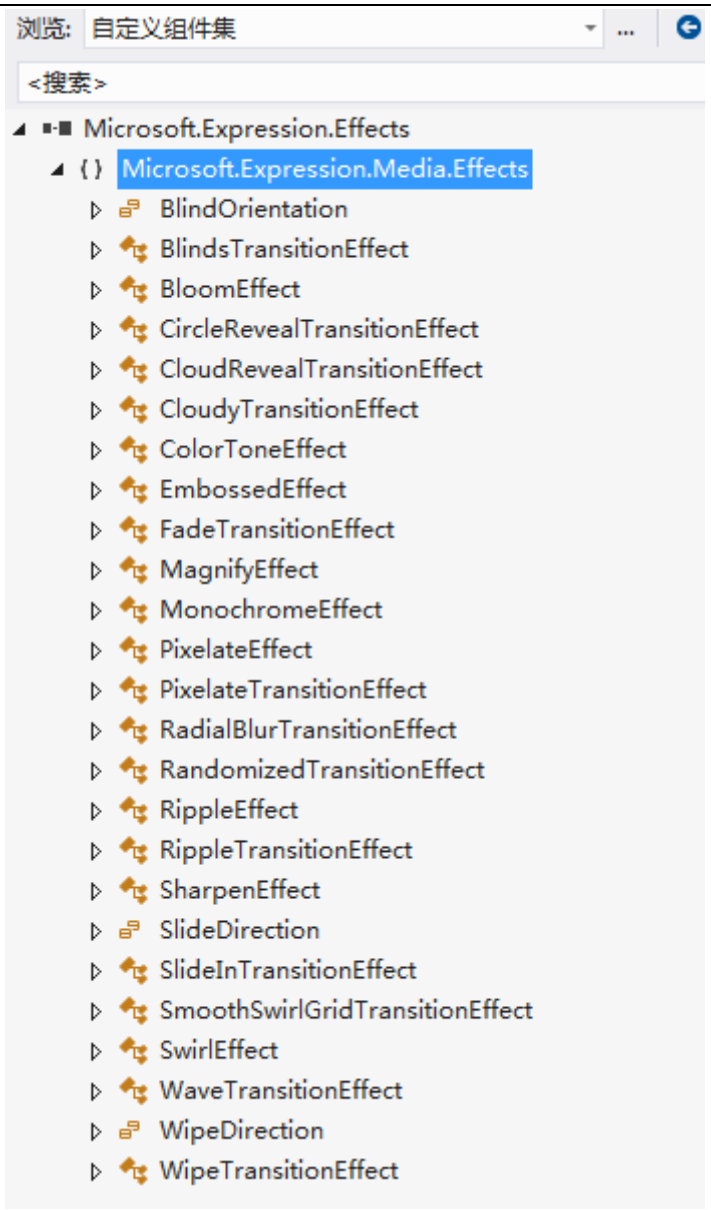
从工具窗口的控件区再拖入一个 RadButton，将其 Content 属性设置为 TranslateAction，按 4.3.1 中添加动画的方法，添加 LinearTranslateAction 动画，此动画的属性中的 TriggerOnValueChanged 属性、EventName 属性、Duration 属性和上文一样，不再赘述。

设置 Animation/Source 的 MaxValue 属性为 360，MinValue 为 0，Value 属性绑定到 slider 控件的 Value 属性；Animation/Target 中，Animation/Target 中，设置 TranslateXEnabled 和 TranslateYEnabled 打上勾，设置 TranslateXMax 为 180，TranslateXMin 为 0，TranslateYMax 为 180，TranslateYMin 为 0，运行，滑动 slider 控件的滑块，效果如下图。



4.4 特效 Effects(效果)

AicStudio 一共集成了 17 种特效，它们都来自于 Microsoft.Expression.Media.Effects 命名空间。所有的这些特效（包括模糊、锐化、滤镜、浮雕等）处理之后和效果和一些图像处理软件（Photoshop）类似。在程序中使用特效与直接使用 PS 等软件进行图像处理相比，优势在于这些特效的属性可以根据程序的需要非常方便的进行调整和改善，应用于动画中甚至可以动态改变做出更炫目的效果。

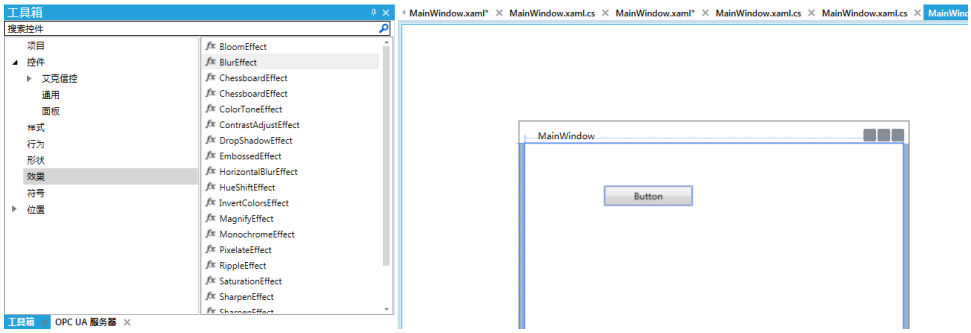


以 WPF 为例，这些特效都是 BitmapEffect（位图特效）,使用 BitmapEffect 对所有 Visual 对象进行位图特效处理（比如文本，按钮，图像，甚至矩形、画布等），它是基于像素级别的，而且是基于软件处理模式而非硬件加速的处理模式。

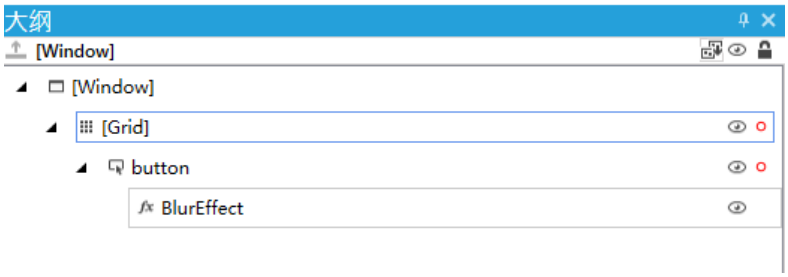
BitmapEffect 在控件渲染时起作用，当控件渲染时，该 Visual 对象转化成相应的 BitmapSource 并作为 BitmapEffect 的输入，显示为经过 BitmapEffect 处理后的效果。

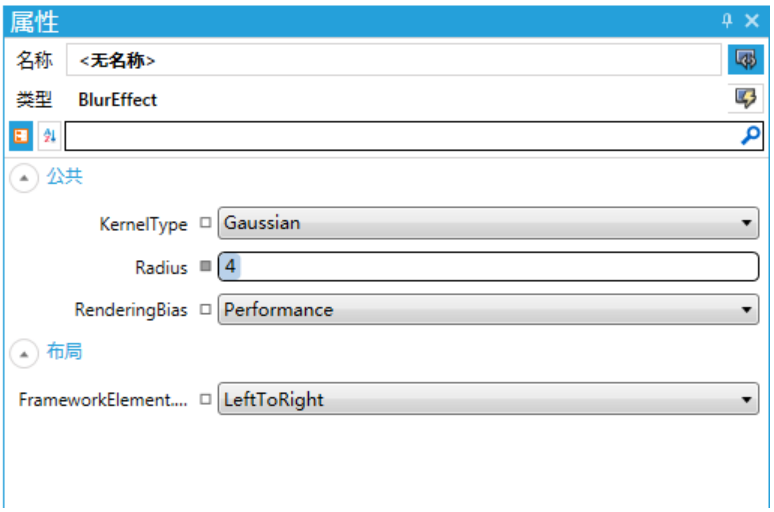
由于软件处理模式的效率相对较低，在有大量 Visual 对象或在有大量动画时慎用，此时你必须在效果与效率之间做一个相对的平衡。

在 AicStudio 中，添加特效的操作非常简单，将选中的特效左键按住拖放到控件上即可。每个控件仅允许存在一个特效。



修改特效参数步骤为，在大纲窗口左键选中已添加的特效 BlurEffect，然后在特效的属性窗口中进行相关参数的修改；也可以在 Xaml 代码中直接进行修改。





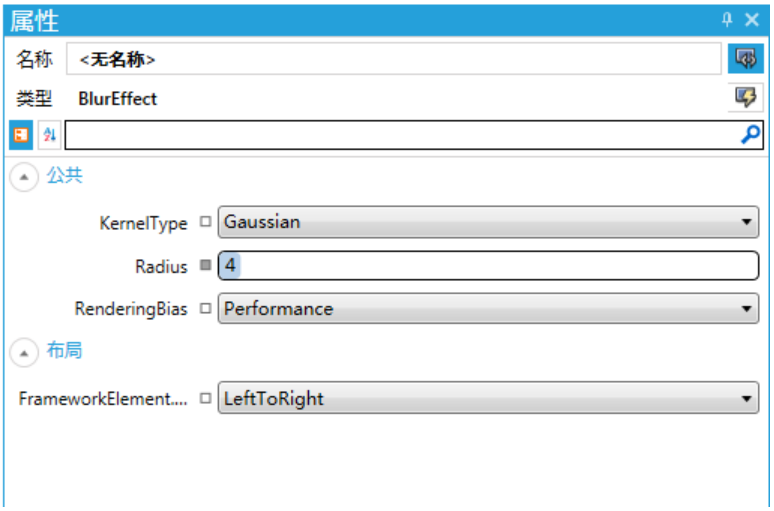
在 AicStudio 中，我们可以对控件进行以下几种常见的位图特效处理：

（1）虚化效果（BlurEffect）

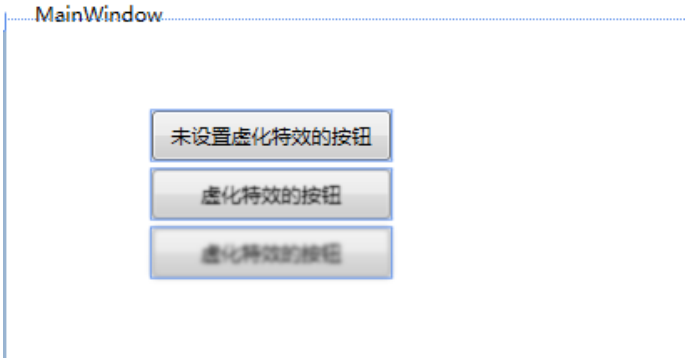
我们来看下虚化效果的简单示例：

添加 3 个按钮，第一个按钮无虚化效果，通过修改虚化效果的属性 Radius,第二个和第三个的虚化效果逐渐增强。

虚化效果的属性窗口为



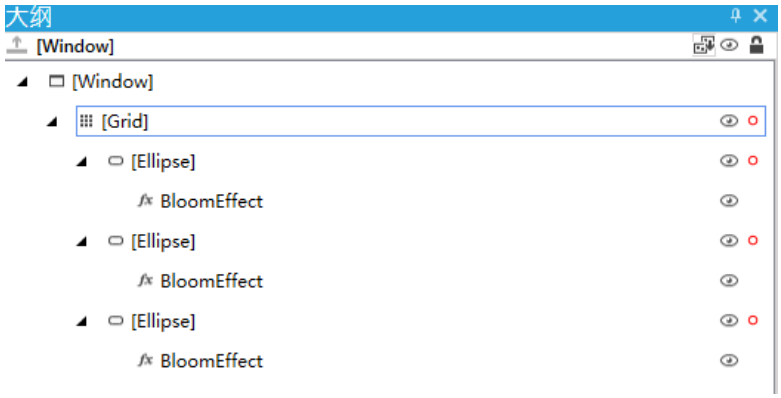
最终的虚化显示效果如下

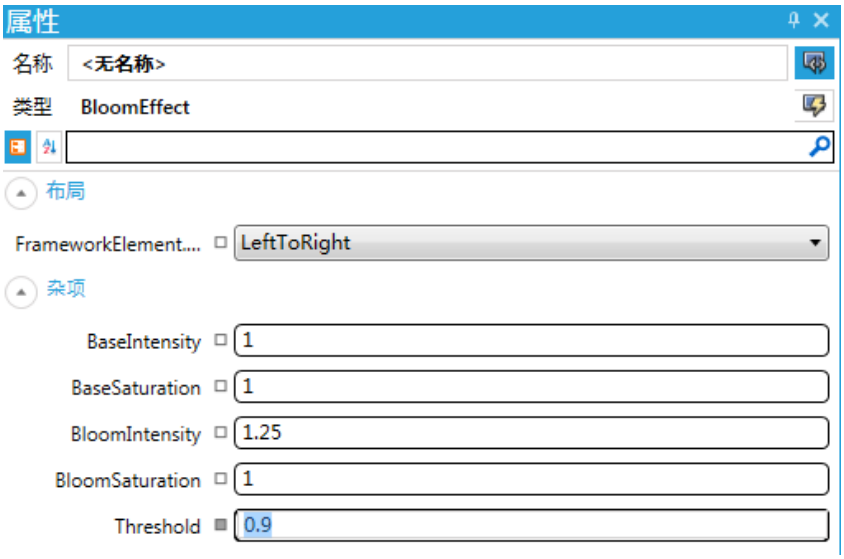


（2）泛光效果（BloomEffect）

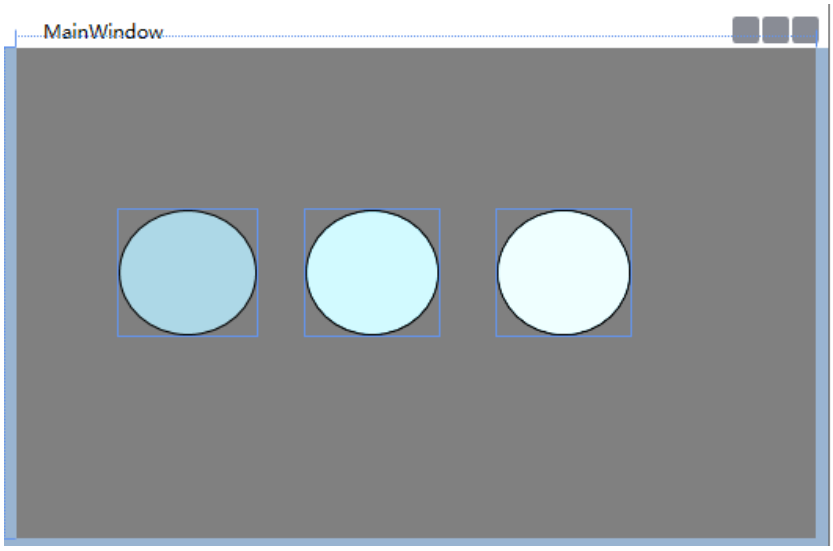
泛光效果可以使相对明亮的物体进行发光，下面我们做一个简单的示例。

添加 3 个椭圆 Ellipse，将它们的 Fill 属性设置为淡蓝色 LightBlue,然后将 BloomEffect 拖放到 3 个椭圆上，在 BloomEffect 的属性窗口上将 Threshold(阈值)分别设置为 0.9,0.5,0.1





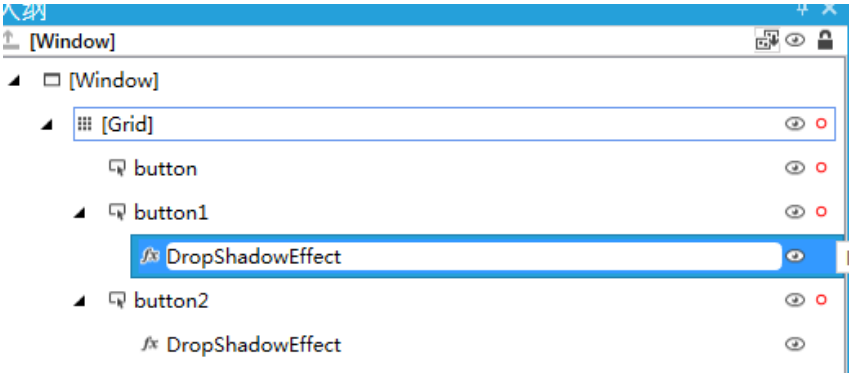
最终的效果如图所示，从左到右，由于阈值减小，泛光效果的亮度明显增加。



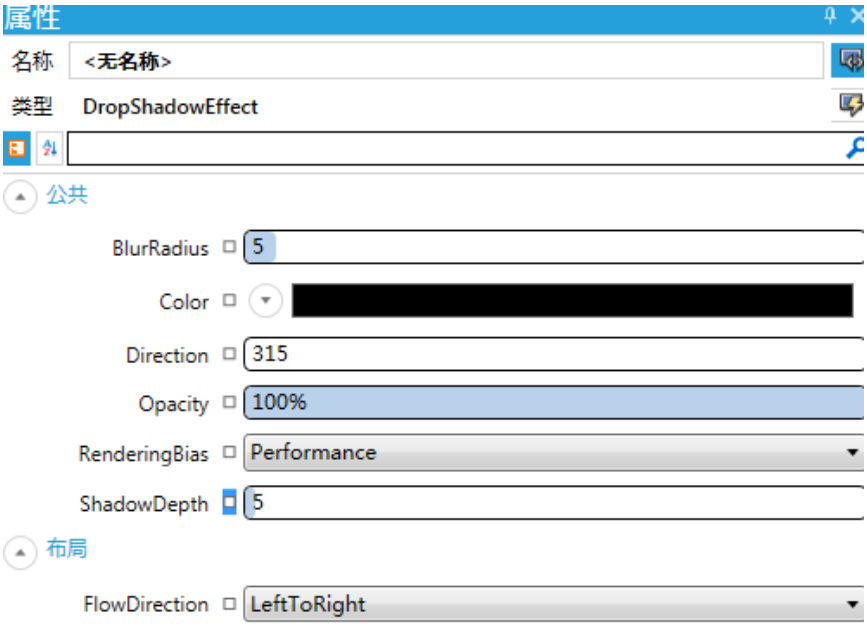
（3）阴影效果（DropShadowEffect）

阴影效果的示例也通过 3 个按钮来展示，通过改变各自的阴影效果的属性，达到合适的阴影效果的目的。

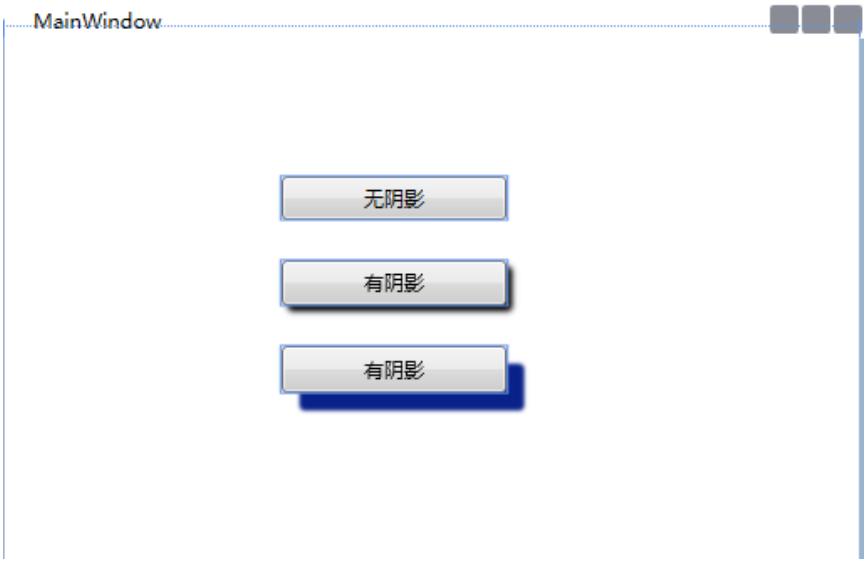
大纲窗口，显示了在 button1 和 button2 下，各自拖放进去了一个 DropShadowEffect。



属性窗口显示了 DropShadowEffect 可以配置的属性。



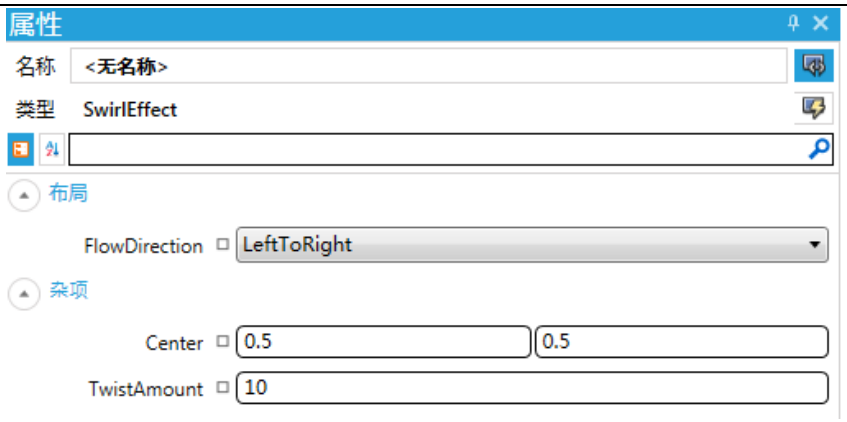
最终设置的阴影效果如图，其中第三个按钮的阴影效果将 Color（阴影颜色）属性设置为 Blue，ShadowDepth(阴影深度)设置为 10。



（4）涡轮效果(SwirlEffect)

涡轮效果可以把平整的位图变为螺旋状的涡轮样式。

在 Xaml 界面放入 2 个按钮控件，第二个按钮控件添加涡轮效果（SwirlEffect）。Center 为涡轮效果的中心点，TwistAmount 属性为涡轮效果的扭曲程度。

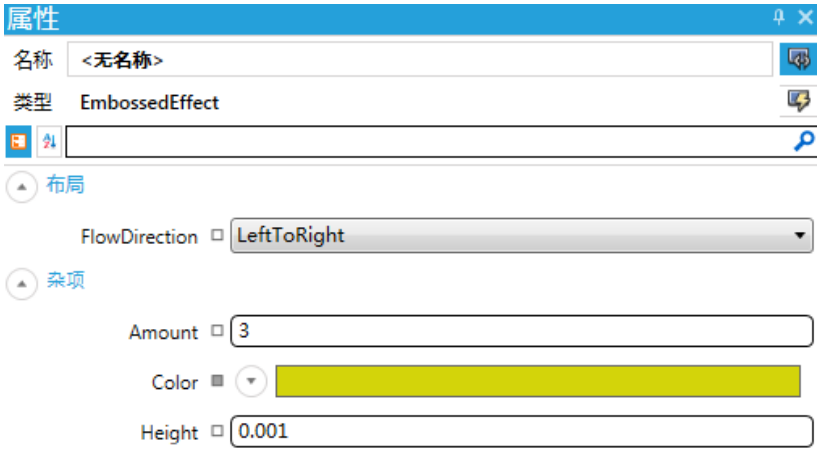


得到涡轮效果的对比图

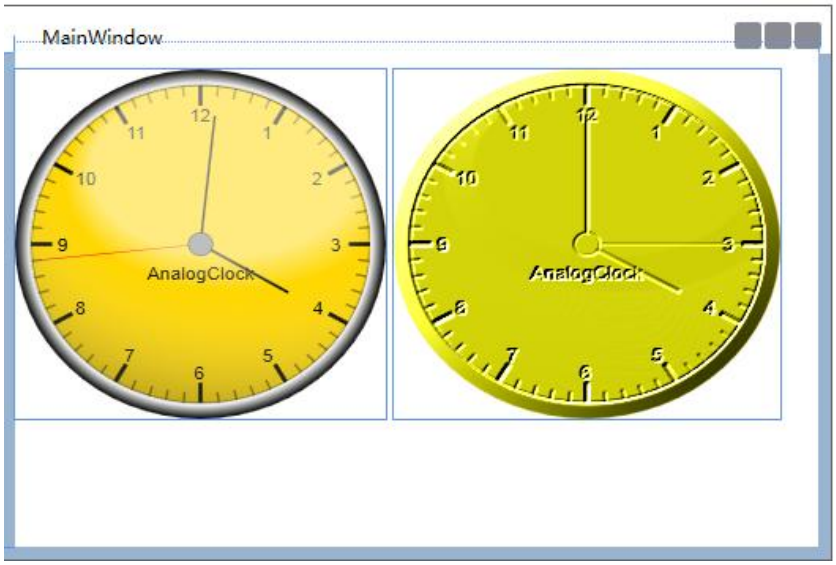
（5）浮雕特效（EmbossedEffect）

浮雕特效可以展现出雕刻的 3D 效果，现在我们做一个简单的示例。

在 Xaml 文本标记界面添加 2 个艾克信控的 AnalogClock 时钟控件，在第二个时钟控件上拖放 EmbossedEffect 效果。在 EmBossedEffect 属性窗口配置其属性。

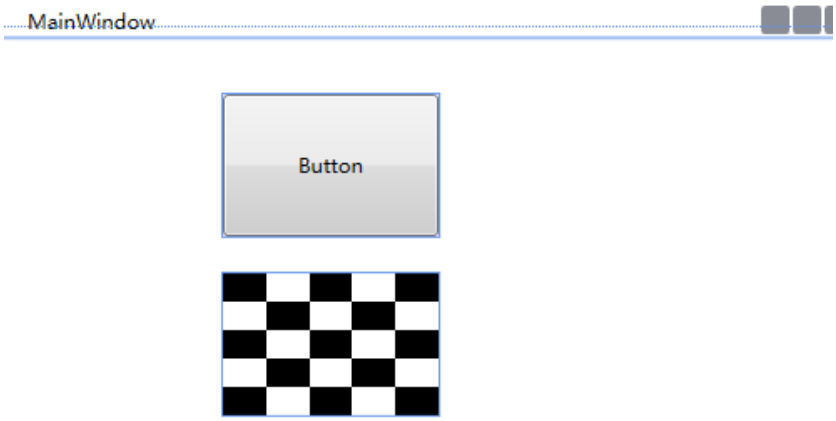


将 Height(雕刻深度)设置为 0.001，Amount 设置为 3，Color 选为黄色，效果显示如下

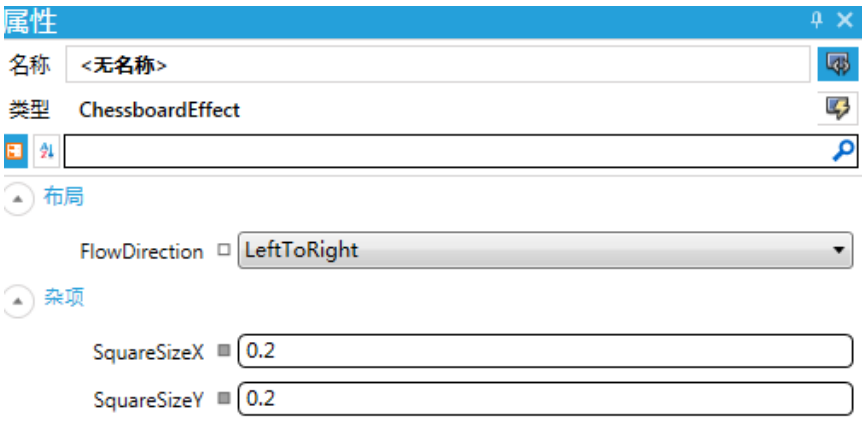


（6）棋盘效果（ChessboardEffect）

棋盘特效的示例通过 2 个按钮来进行展示。



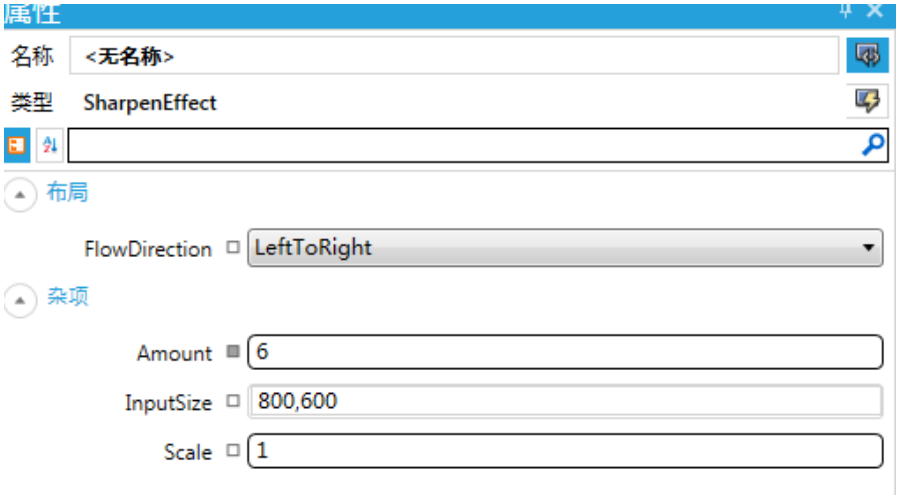
第二个按钮的棋盘效果参数窗口设置如下



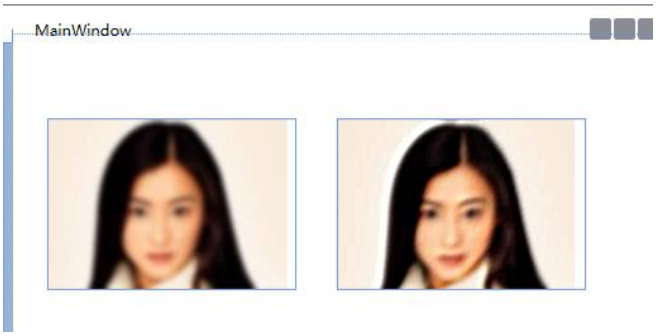
SquareSizeX 和 SquareSizeY 两个参数范围为 0-1，表示分割的小块的单位，两个参数都为 0.2 表示整个按钮被分割成 5*5 的小块。

（7）锐化效果(SharpenEffect)

锐化效果与模糊效果相反，可以提高位图的清晰度，使图像特定区域的色彩更加鲜明。我们来做一个简单的示例，添加 2 个 Image 控件，Source 属性都为同一个模糊图片的路径，然后在其中的一个 Image 控件上添加锐化效果。



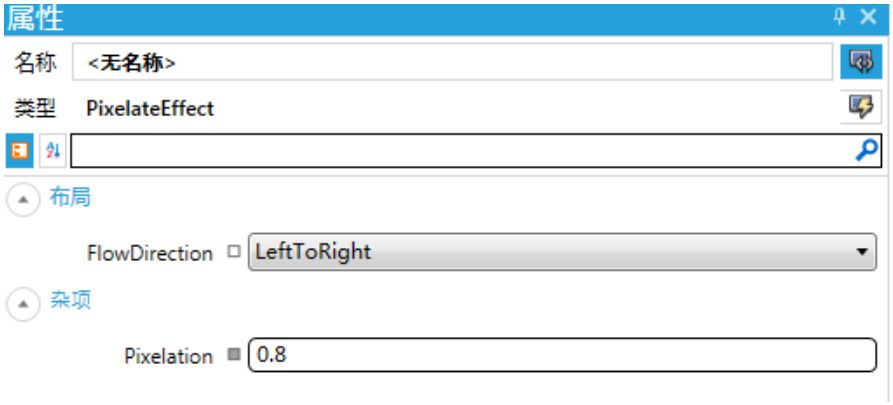
通过调整锐化效果属性参数，获取更加清晰的图像显示效果。



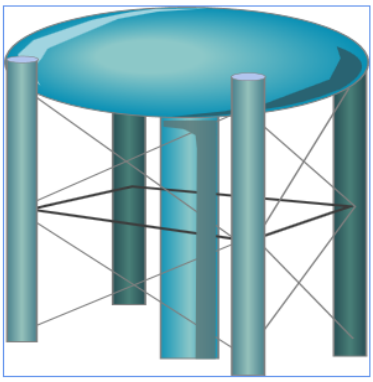
（8）像素化滤镜（PixelateEffect）

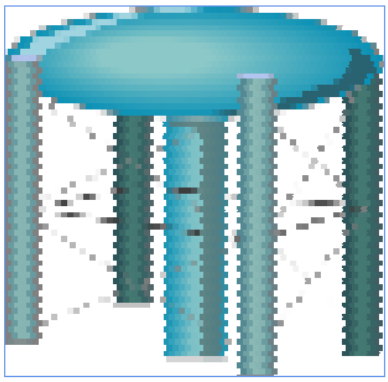
像素化滤镜将位图进行像素化的分割，看起来就像由一个个的小块组成。

我们放入 2 个艾克信控的 Towers004 控件，在第二个控件上拖放 PixelateEffect 效果，设置参数如下图所示。



得到的效果如图

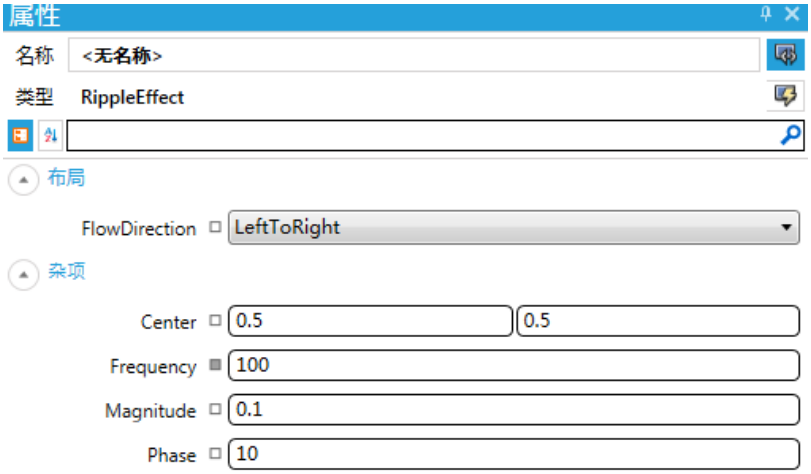




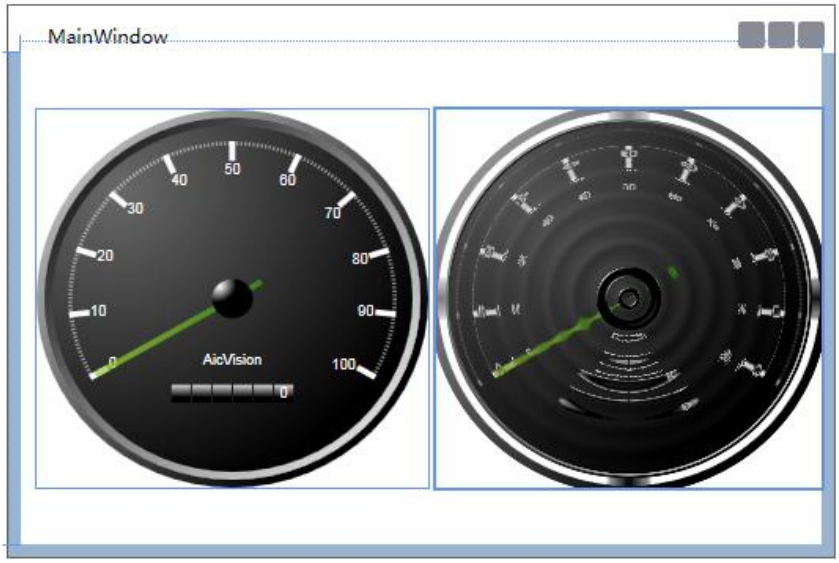
（9）波纹效果（RippleEffect）

波纹效果将 Visual 位图变换为波纹的图像，以下是简单的示例。

在 Xaml 界面放入 2 个艾克信控的 RacingGague(速率仪表)控件，第二个控件拖放 RippleEffect，然后配置 RippleEffect 参数，将 Frequency（波纹频率）设置为 100.



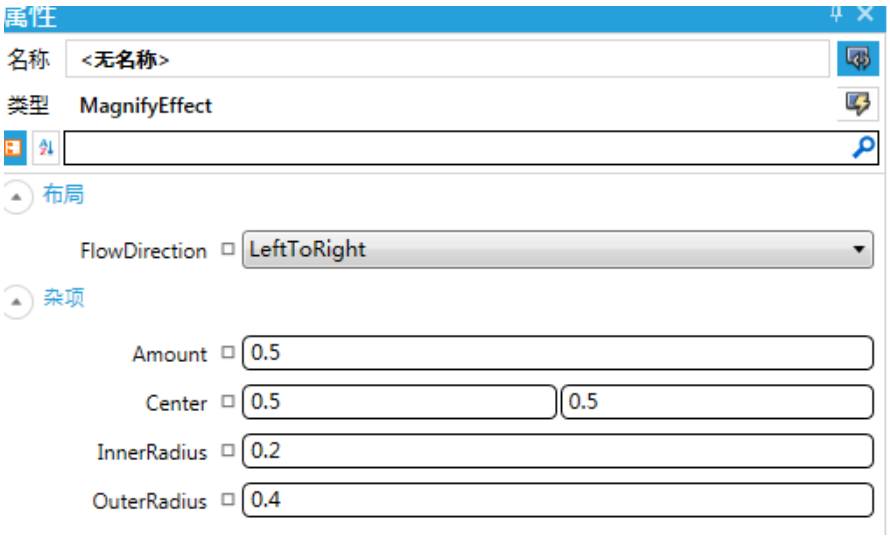
下面是应用波纹效果后，控件的效果对比图



（10）放大镜效果（MagnifyEffect）

下面来一个简单的示例演示放大镜的效果，放大镜放大的不是控件，而是控件的显示元素。

在 Xaml 界面添加 2 个 Button，其中一个拖放 MagnifyEffect，然后设置其参数。Center 属性为放大的中心点，InnerRadius 和 OuterRadius 分别为放大镜的内角和外角（放大镜内外边缘厚度），Amount 为放大镜离放大物体的距离。

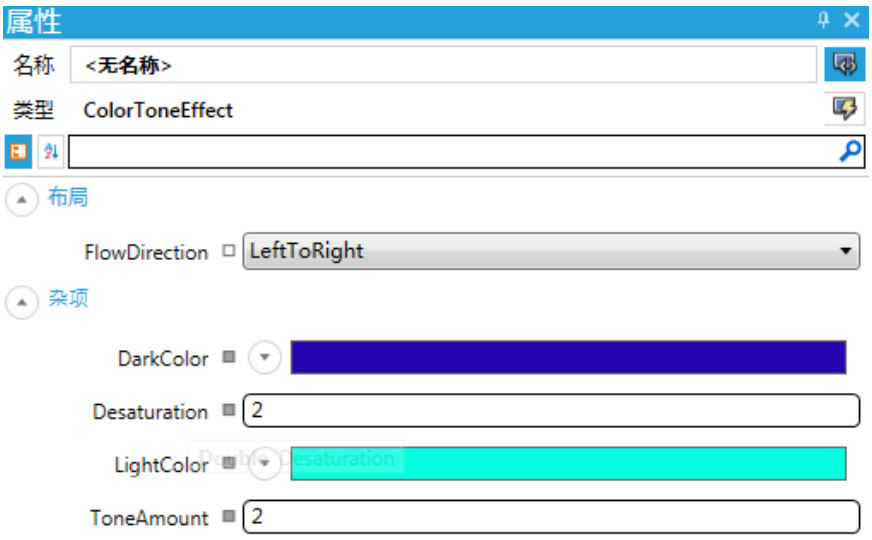


调整完毕后的显示效果图如下。

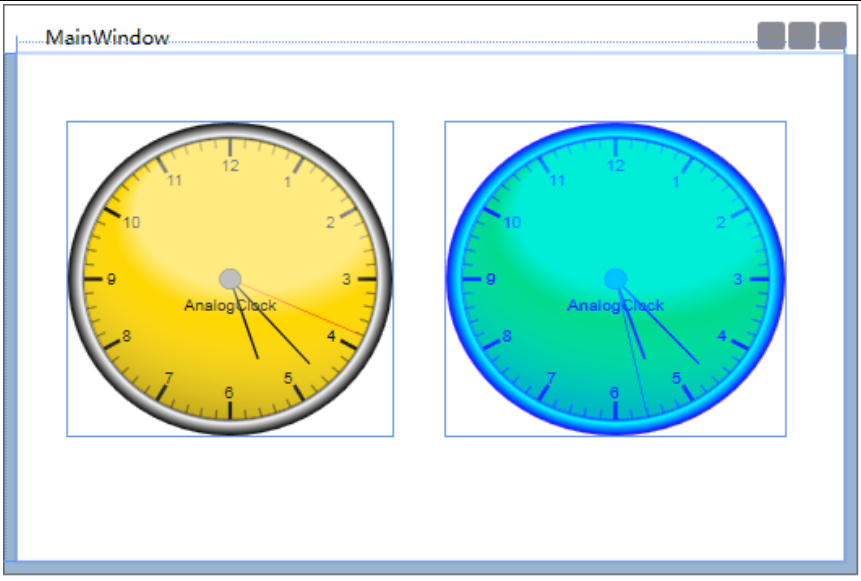
（11）色调（ColorToneEffect）

色调指的是一幅画中画面色彩的总体倾向，是大的色彩效果，我们可以通过色调效果改变控件的显示。

做一个演示示例，在 Xaml 界面中放入 2 个 AnalogClock 控件。其中一个拖放 ColorEffect 效果，并配置其属性，因其默认的为亮黄色的暖色调，我们用 ColorEfeect 将之改为冷色调。



显示效果如图

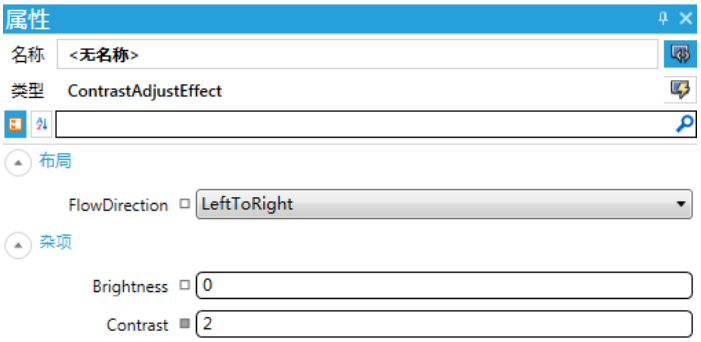


（12）对比度（ContrastAdjustEffect）

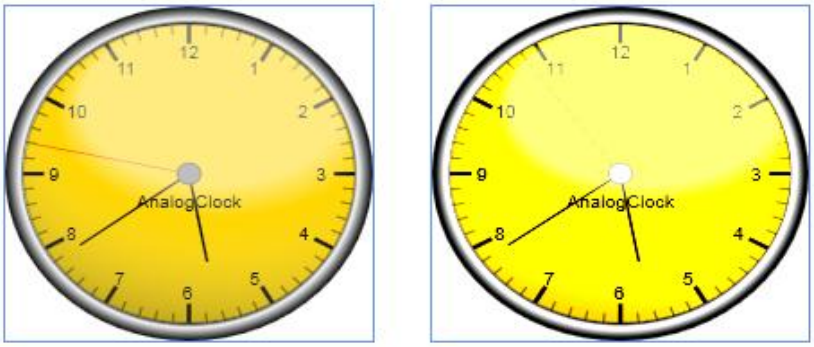
对比度指的是一幅图像中明暗区域最亮的白和最暗的黑之间不同亮度层级的测量，差异范围越大代表对比越大，差异范围越小代表对比越小。

对比度对视觉效果的影响非常关键，一般来说对比度越大，图像越清晰醒目，色彩也越鲜明艳丽；而对比度小，则会让整个画面都灰蒙蒙的。

来做一个简单的示例，在 Xaml 界面中放入 2 个 AnalogClock 控件，其中一个拖放 ContrastAdjustEffect,然后配置其属性，Brightness 属性增加亮度，Contrast 属性配置对比度。

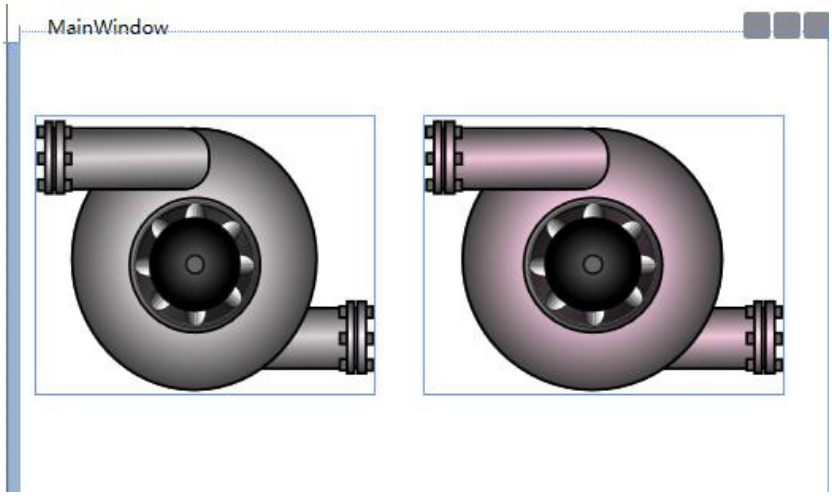


配置完毕的效果如图，可以看出，对比度提高后更加清晰。



（13）饱和度（SaturationEffect）

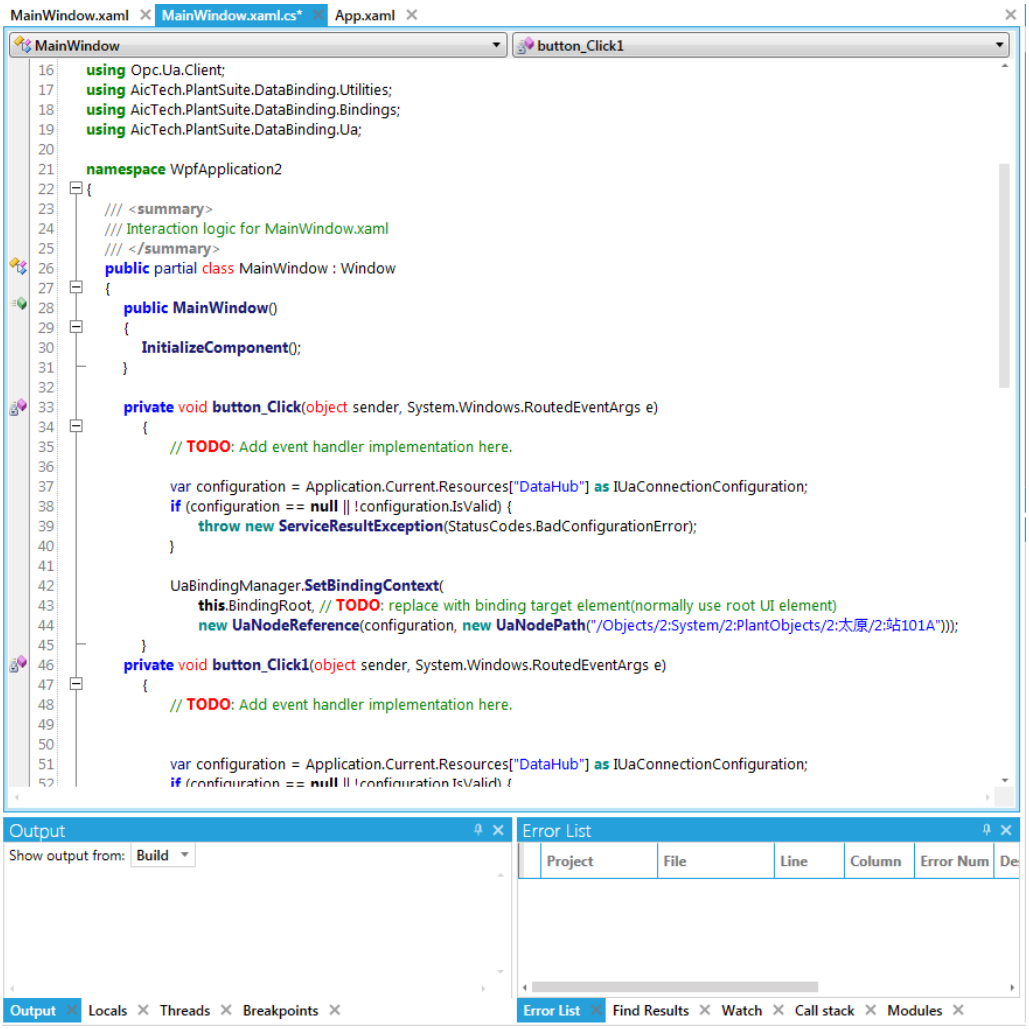
饱和度是指色彩的鲜艳程度，也称色彩的纯度。



5 脚本

5.1 代码编辑器

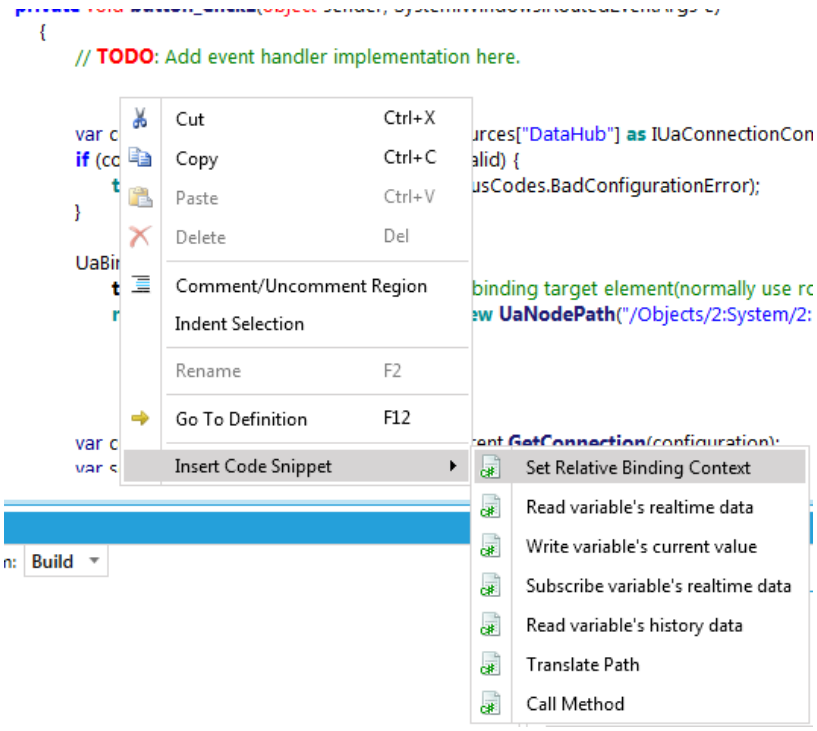
在 AicStudio 打开类文件(.cs 文件)的同时，我们进入 C#代码编辑器界面，编辑器有强大的代码编辑相关功能，人性化的智能提示工具，自动添加代码的脚本向导等。



5.2 脚本向导

AicStudio 的脚本向导是为代码编辑服务，自动添加代码实现与 OPC UA Server 的交互，省去了添加引用程序集和命名空间引用，寻找适合的方法等步骤，简化了代码编辑操作，提高编辑效率。

在可编辑代码的位置，右键单击，选择 Insert Code Snippet 插入代码段，目前主要有 7 项代码段提供便捷插入。



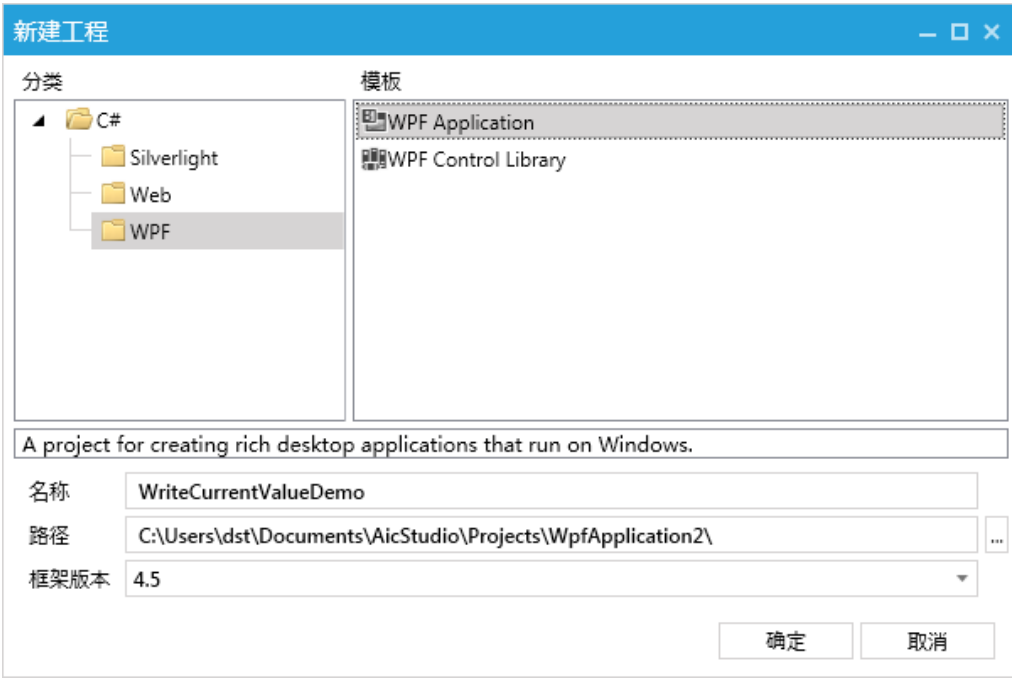
5.2.1 设置相对绑定对象(Set Relative Binding Context)

此项请参见 4.2 节数据绑定的第五部分，在设置 OPC UA 节点相对绑定的时候，就使用过此脚本添加相应的代码段，将根节点绑定数据上下文，子节点的相对路径切换时，相对绑定不同的对象随之切换。

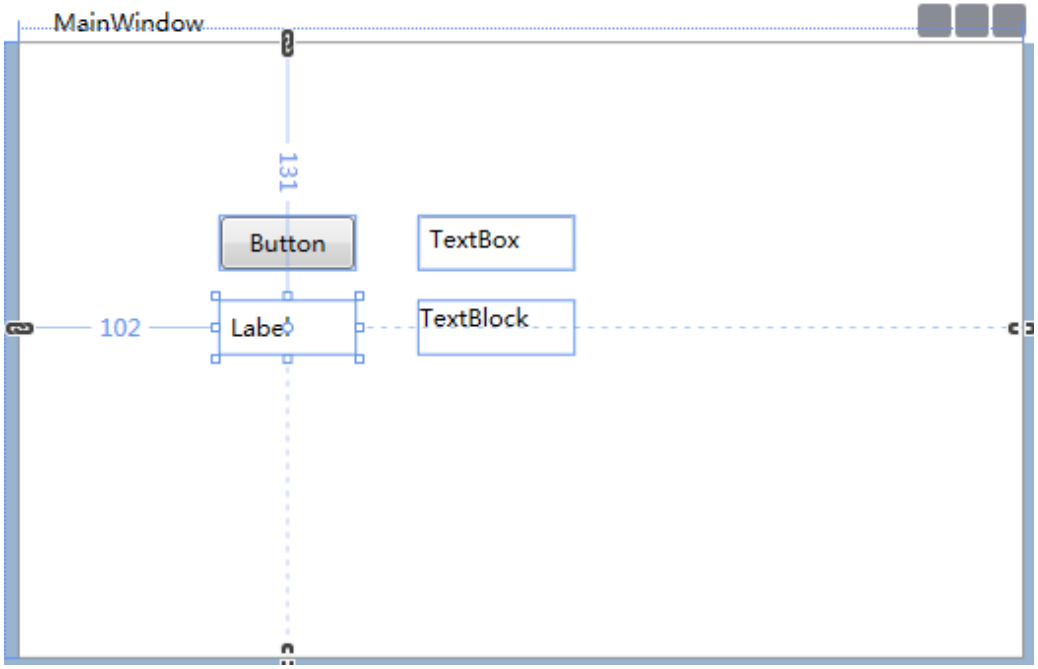
5.2.2 修改变量当前值(Write variable's current value)

我们为此代码段做一个简单的示例，用来修改本机 DataHub 中的某一变量的值，为了验证是否已经修改，我们将修改后的值进行显示。选定的变量可以根据自身本机 DataHub 中已有的进行选择，也可以新建变量（参考 AicDataHub 文档）。

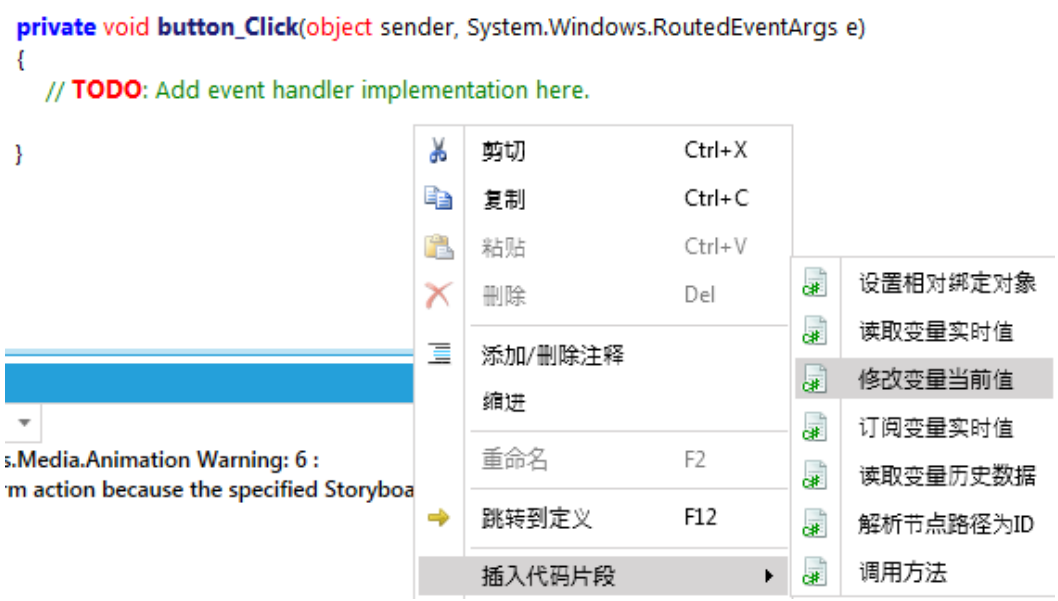
步骤一、新建 WPF 工程 WriteCurrentValueDemo



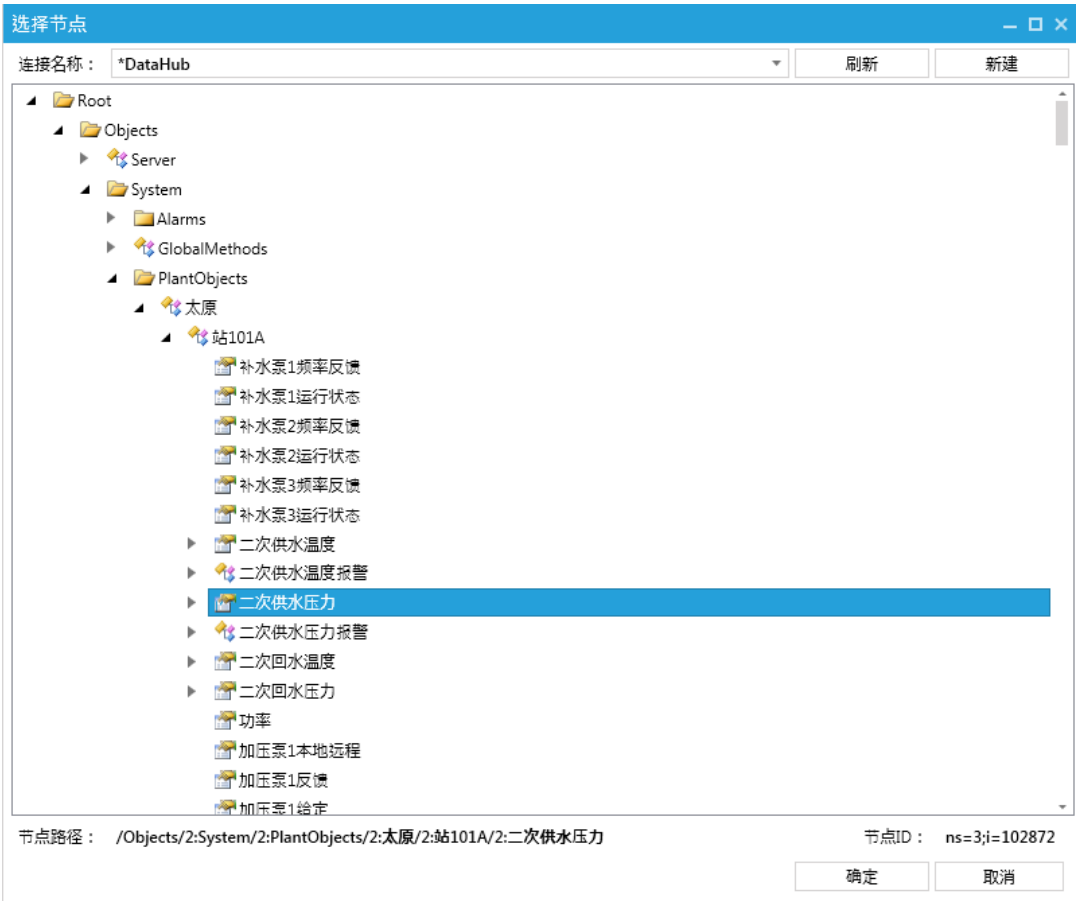
步骤二、添加控件，这里添加了 Button,Label,TextBox,TextBlock。



步骤三、为按钮添加单击事件，添加修改变量值的代码段
在要添加代码段的空白处右键单击选择插入代码段->修改变量当前值。



这里的示例选择节点，选中 Float 类型的变量“二次供水压力”



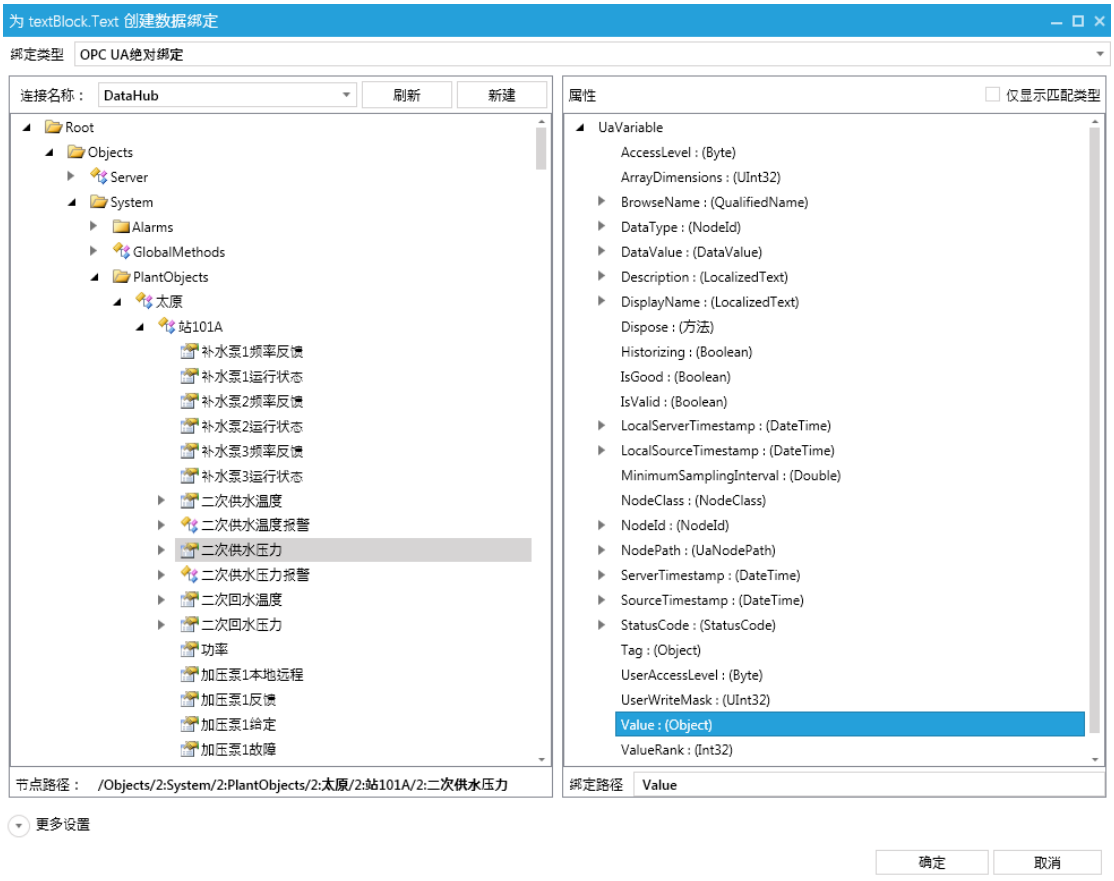
点击确定后，可以看到已经在按钮事件处理方法中添加了修改变量值的代码，图中标示红色字体 TODO 的部分为可以自行修改的部分。

```
    private void button_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.
        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        var connection = UaConnectionManager.Current.GetConnection(configuration);
        var session = (connection == null) ? null : connection.Connection;
        if (session == null) {
            throw new ServiceResultException(StatusCodes.BadNotConnected);
        }

        var valueToWrite = new WriteValue();
        valueToWrite.NodeId = new NodeId(102872U, 3); // /Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力
        valueToWrite.AttributeId = Attributes.Value;
        valueToWrite.Value = new DataValue(StatusCodes.Good);
        valueToWrite.Value.Value = (Single)0.0; // TODO: replace with real variable value(System.Single)
        session.WriteAttributeAsync(valueToWrite).ContinueWith((t) => {
            if (t.Exception != null) {
                // TODO: Error
            } else {
                if (ServiceResult.IsBad(t.Result)) {
                    // TODO: Error
                } else {
                    // TODO: OK
                }
            }
        }, TaskScheduler.FromCurrentSynchronizationContext());


    }
```

我们将修改的新值确定为 TextBox 中输入的值。

```
private void button_Click(object sender, System.Windows.RoutedEventArgs e)
{
    // TODO: Add event handler implementation here.
    var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
    if (configuration == null || !configuration.IsValid) {
        throw new ServiceResultException(StatusCodes.BadConfigurationError);
    }

    var connection = UaConnectionManager.Current.GetConnection(configuration);
    var session = (connection == null) ? null : connection.Connection;
    if (session == null) {
        throw new ServiceResultException(StatusCodes.BadNotConnected);
    }

    var valueToWrite = new WriteValue();
    valueToWrite.NodeId = new NodeId(102872U, 3); // /Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力
    valueToWrite.AttributeId = Attributes.Value;
    valueToWrite.Value = new DataValue(StatusCodes.Good);
    valueToWrite.Value.Value = Single.Parse(this.textBox.Text); // TODO: replace with real variable value(System.Single)
    session.WriteAttributeAsync(valueToWrite).ContinueWith((t) => {
        if (t.Exception != null) {
            // TODO: Error
        } else {
            if (ServiceResult.IsBad(t.Result)) {
                // TODO: Error
            } else {
                // TODO: OK
            }
        }
    }, TaskScheduler.FromCurrentSynchronizationContext());


}
```

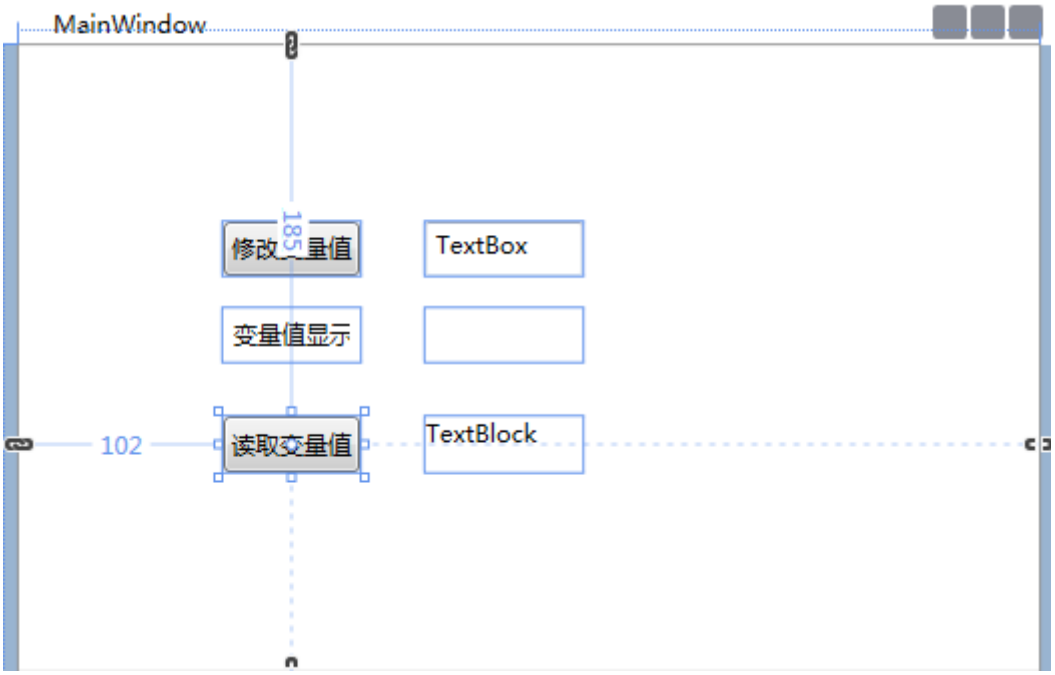
然后将此变量的 Value 属性绑定到 TextBlock.Text 依赖项属性,将此变量的值在 TextBlock 显示。



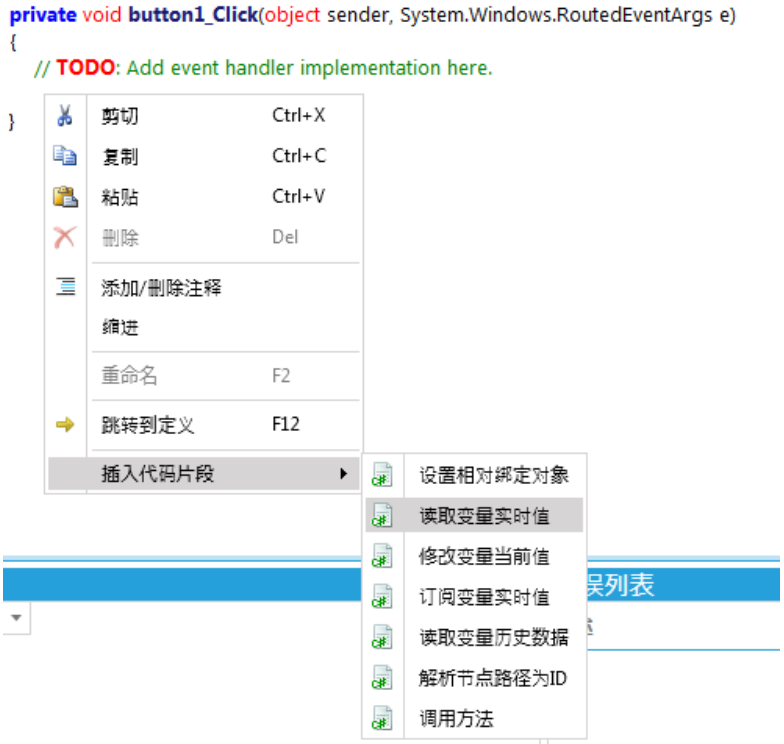
点击确定后，将此工程设置为启动工程，运行测试。

5.2.3 读取变量实时值(Read variable's realtime data)

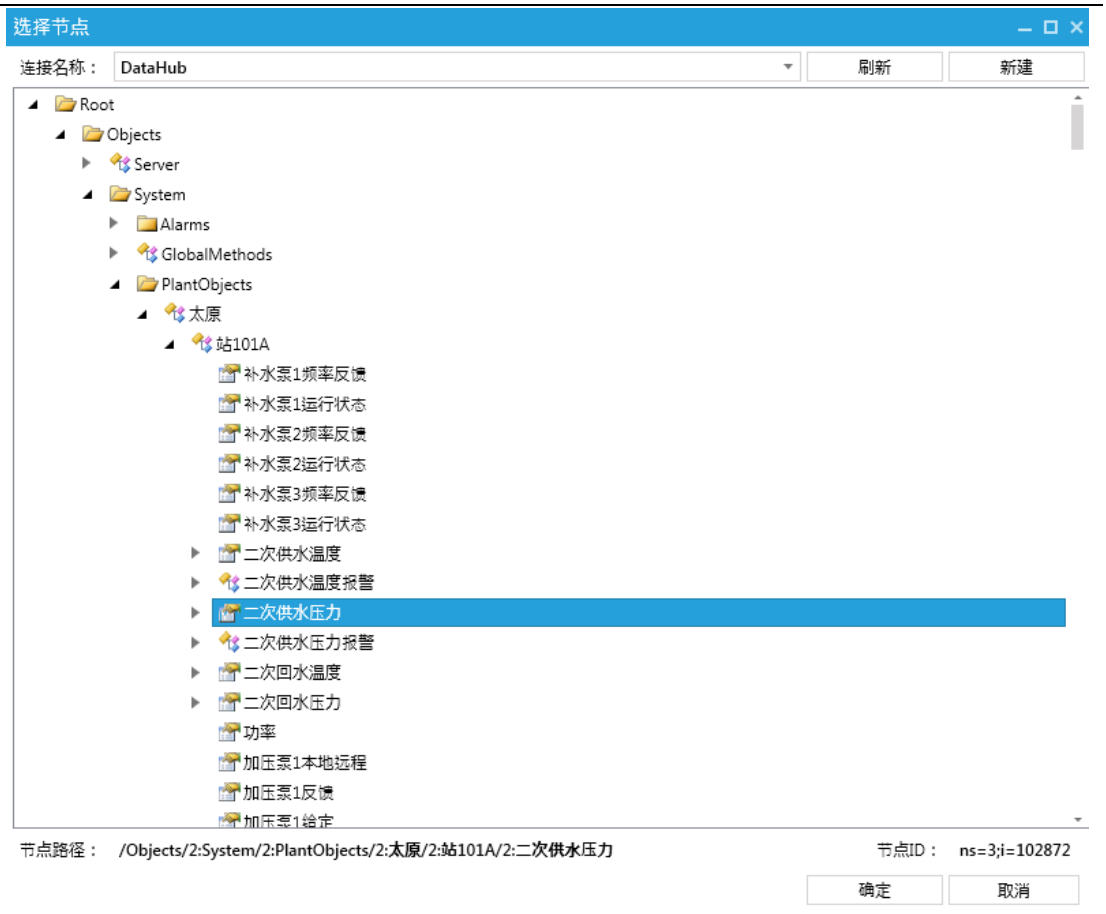
读取变量实时值为一个取值的过程，这里的示例将值取出并进行显示。
此示例沿用了 2）中的全部代码，并进行进一步添加，增加读取变量值的按钮和显示读取结果的 TextBlock.



读取变量值按钮添加单击事件，并右键添加读取变量实时值的代码段。



选择节点



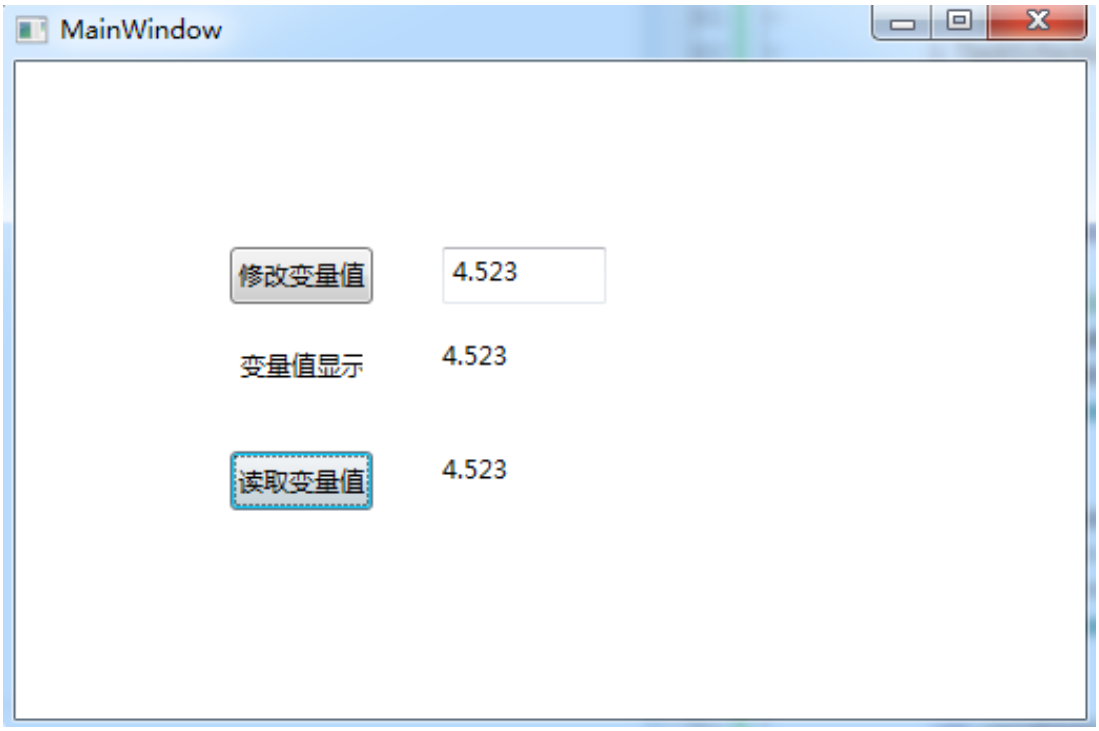
点击确定后添加的代码段如下

```
private void button1_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.
        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        var connection = UaConnectionManager.Current.GetConnection(configuration);
        var session = (connection == null) ? null : connection.Connection;
        if (session == null) {
            throw new ServiceResultException(StatusCodes.BadNotConnected);
        }

        var variableId = new NodeId(102872U, 3);// /Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力
        session.ReadValueAsync(variableId).ContinueWith((t) => {
            if (t.Exception != null) {
                // TODO: Error
            } else {
                // TODO: OK
                // var dataValue = t.Result;

            }
        }, TaskScheduler.FromCurrentSynchronizationContext());


    }
```

修改蓝色背景标示的代码，将读取到的值进行显示

```
private void button1_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.
        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        var connection = UaConnectionManager.Current.GetConnection(configuration);
        var session = (connection == null) ? null : connection.Connection;
        if (session == null) {
            throw new ServiceResultException(StatusCodes.BadNotConnected);
        }

        var variableId = new NodeId(102872U, 3);// /Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力
        session.ReadValueAsync(variableId).ContinueWith((t) => {
            if (t.Exception != null) {
                // TODO: Error
            } else {
                // TODO: OK
                this.textBlock1.Text = t.Result.ToString();

            }
        }, TaskScheduler.FromCurrentSynchronizationContext());


    }
```

点击运行测试。

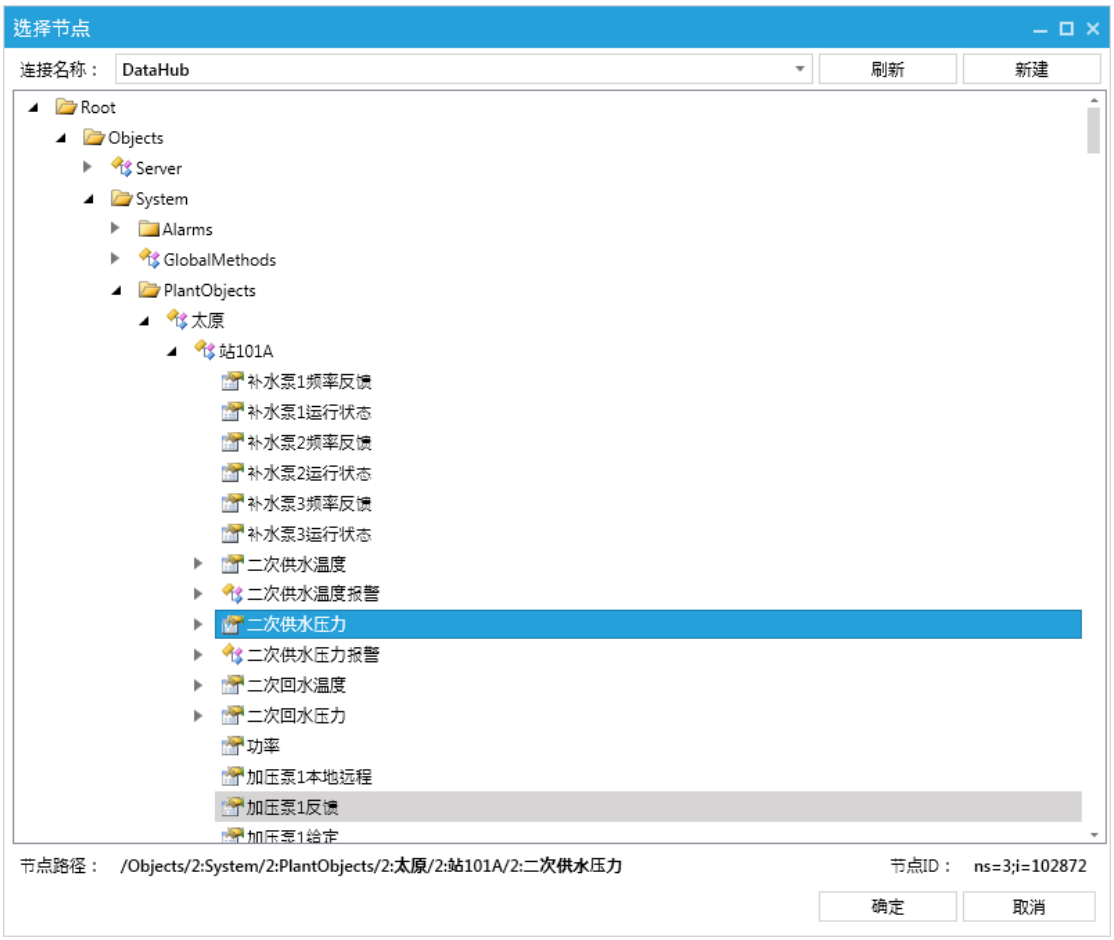


5.2.4 订阅变量实时值(Subscribe variable's realtime data)

订阅变量实时值完成之后，当此变量发生变化时，会触发值变化通知事件，由事件参数获取变量的新值。

在按钮 Click 方法中，右键选择插入代码段->订阅变量实时值。

选择需要订阅的变量的节点；



点击 OK 后，如图，选中的蓝色背景部分为 Notification 事件触发的方法。DataValue 类型的对象 v 为此变量的数据类示例。

```
        // TODO: Add event handler implementation here.
        var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        var connection = UaConnectionManager.Current.GetConnection(configuration);
        var session = (connection == null) ? null : connection.Connection;
        if (session == null) {
            throw new ServiceResultException(StatusCodes.BadNotConnected);
        }

        var subscription = new Subscription();
        subscription.PublishingEnabled = true; // Enable publish function
        subscription.PublishingInterval = 500; // publish interval in milliseconds
        subscription.KeepAliveCount = 100; // if no data changes in continuous KeepAliveCount periods,will send KeepAlive mess
        subscription.LifetimeCount = 300; // if not received publish request in continuous LifetimeCount periods,server will stop a
        subscription.MaxNotificationsPerPublish = 1000; // max notifications in one publish message

        var m = new MonitoredItem();
        m.SamplingInterval = 100;
        m.AttributeId = Attributes.Value;
        m.QueueSize = 1;
        m.DiscardOldest = true;
        m.StartNodeId = new NodeId(102872U, 3); // /Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力
        m.Handle = 0; // custom value to identifier monitoreditem
        m.Notification += (MonitoredItem monitoredItem, MonitoredItemNotificationEventArgs e) => {
            if (e.NotificationValue is MonitoredItemNotification) {
                // MonitoredItemNotification n = (MonitoredItemNotification)e.NotificationValue;
                // DataValue v = n.Value;
            }

        };
        subscription.AddItem(m);

        session.AddSubscription(subscription);
```

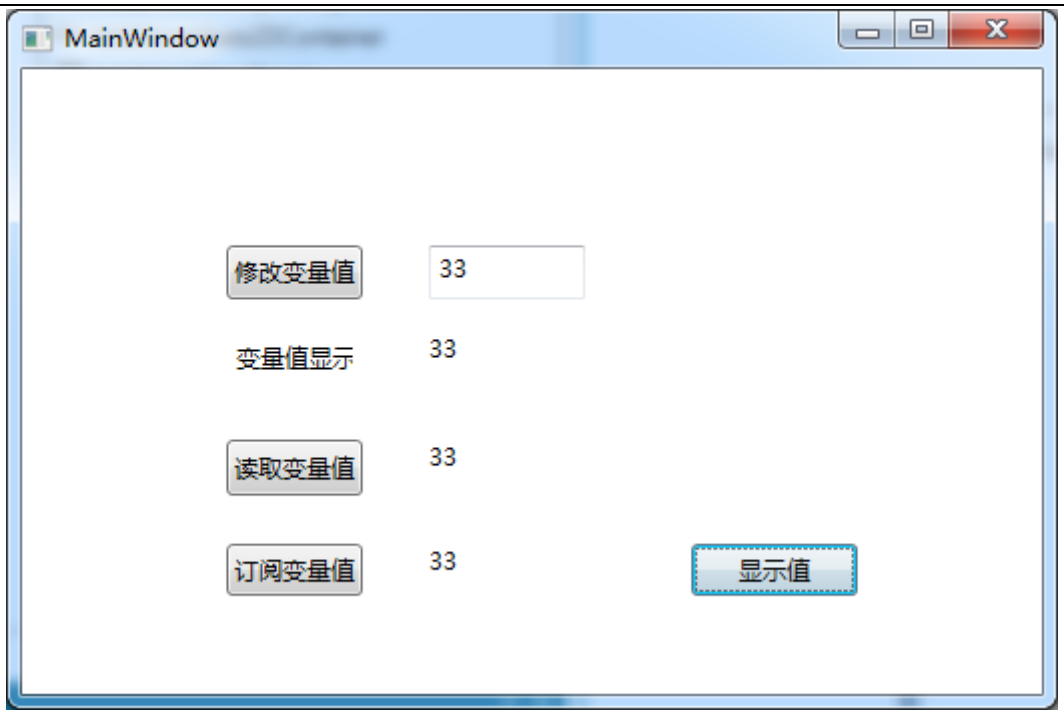
由于事件参数 e 会有重名冲突，我们将此事件进行修改。

```
        //m.Notification += (MonitoredItem monitoredItem, MonitoredItemNotificationEventArgs ee) => {
        //    if (ee.NotificationValue is MonitoredItemNotification) {
        //        MonitoredItemNotification n = (MonitoredItemNotification)ee.NotificationValue;
        //        DataValue v = n.Value;

        //    }

            //};
            m.Notification += m_Notification;
```

单独定义一个通知事件触发执行的方法 m_Notification().并将此变量的值赋值给新定义的 string 类型的 monitoredItemValueString.然后在新添加的按钮中定义显示此变量值的 Click 方法，将值在新添加的 TextBlock 中显示出来。

```
    private void m_Notification(MonitoredItem monitoredItem, MonitoredItemNotificationEventArgs e)
    {
        if (e.NotificationValue is MonitoredItemNotification)
        {
            MonitoredItemNotification n = (MonitoredItemNotification)e.NotificationValue;
            DataValue v = n.Value;
            this.monitoredItemValueString = v.Value.ToString();
        }
    }


    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
        this.textBlock2.Text = monitoredItemValueString;
    }
```

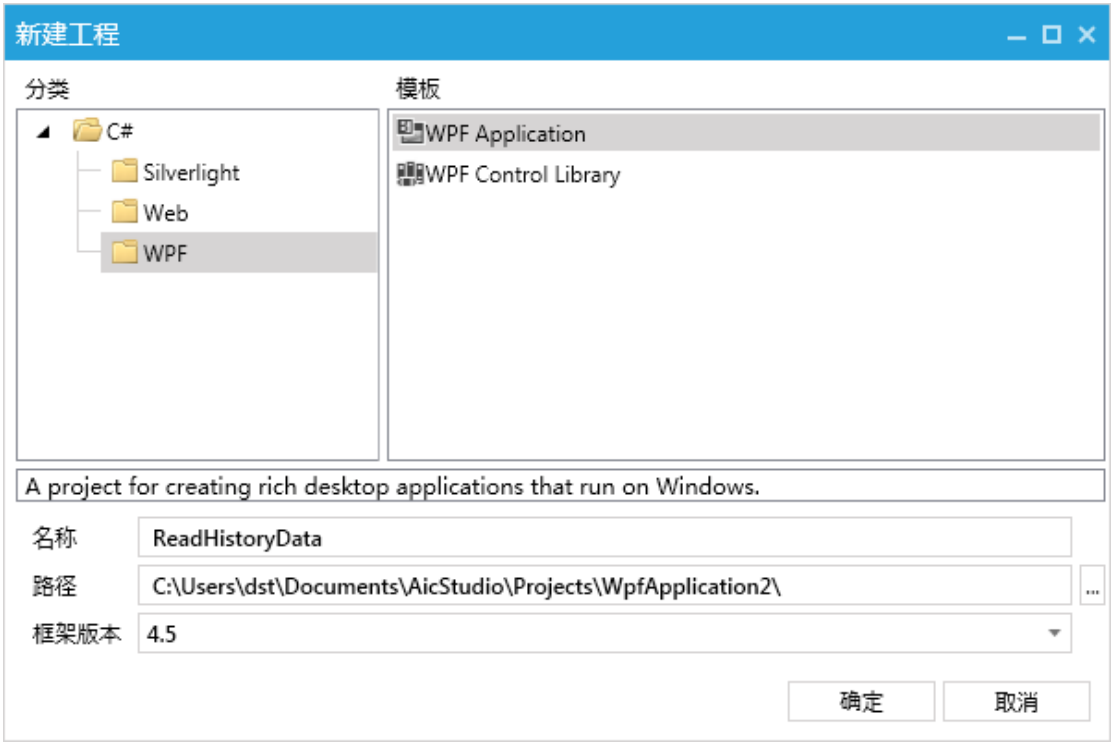
开始调试



5.2.5 读取变量历史数据(Read variable's history data)

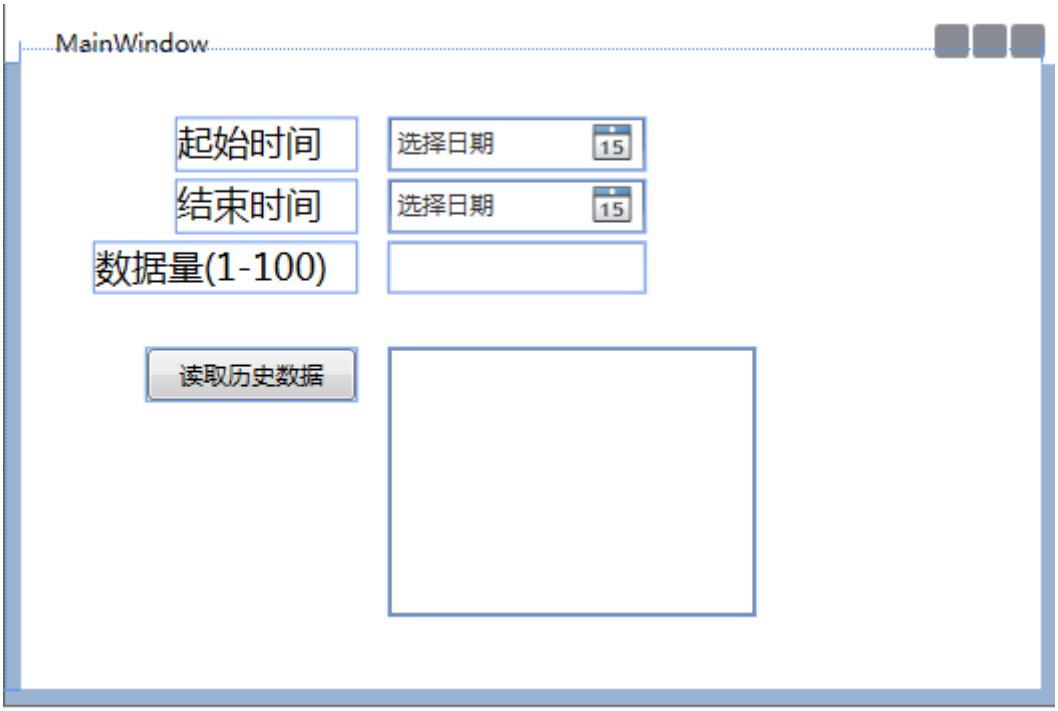
读取历史数据的时候，需要 DataHub 服务器上配备密码锁，给予历史数据查看和取出权限。

以下为读取变量历史数据的示例。新建 WPF 工程。



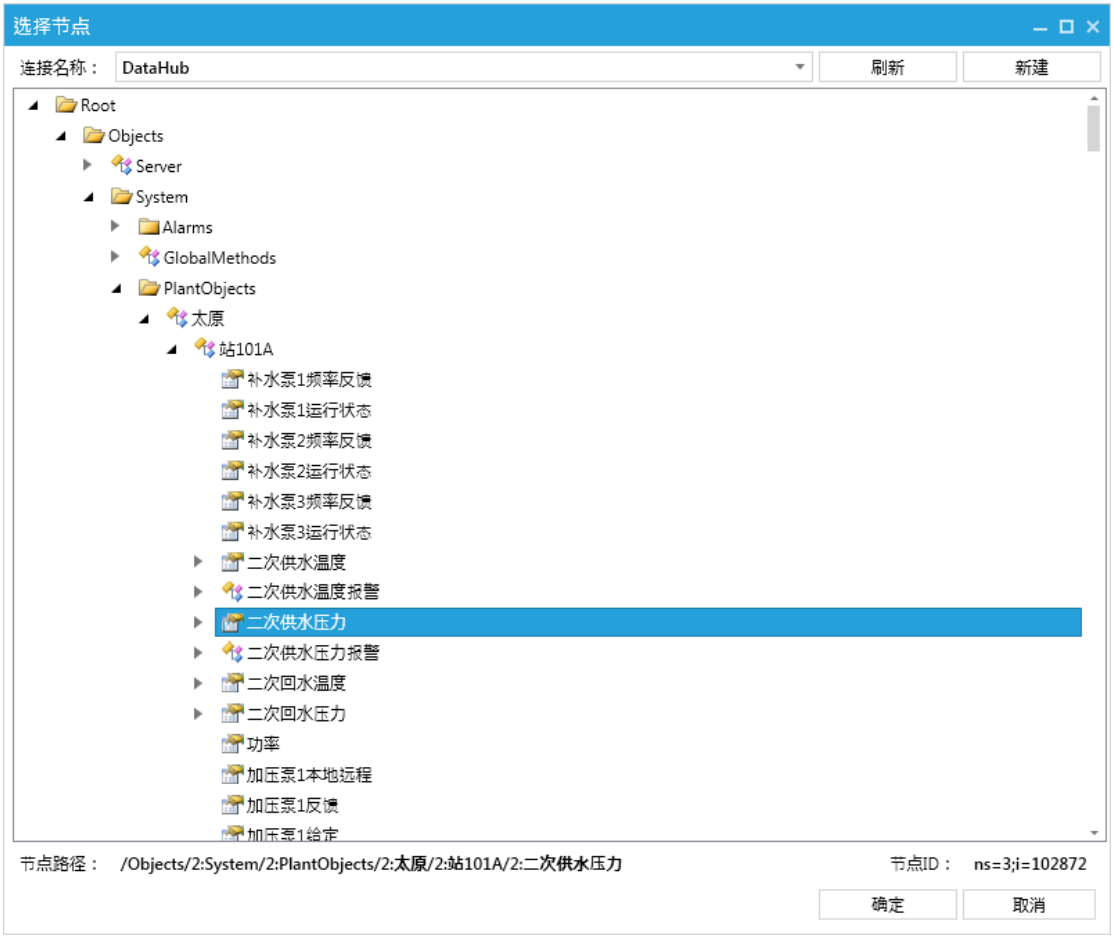
在 Xaml（标记文本）界面添加如下控件，2 个 DatePicker 用来选择读取历史数据的起始时间和结束时间，一个 TextBox 用来输入读取的数据量（自起始时间开始选中的 DataHub 节

点存储的历史数据的个数），一个 Button 用于在单击时触发读取历史数据，一个 ListBox 用于显示读取到的所有数据。



添加按钮单击事件方法，在按钮（读取历史数据）的 Click 方法中，右键单击选择插入代码片段->读取变量历史数据。

然后选择要读取的变量节点



点击确定后，即添加代码段完成，然后进行对代码段的修改。

修改一、在建立连接时，线程挂起 1 秒(如果调试出现异常，可将此时间延长)，等待建立连接的线程连接成功。（若工程已有相关 OPC UA 的相对绑定或者绝对绑定，那么连接会被默认建立，此时不需要将线程进行挂起）

```
var connection =  UaConnectionManager.Current.GetConnection(configuration);
Thread.Sleep(1000);
var session = (connection == null) ? null : connection.Connection;
if (session == null)
{
    throw new ServiceResultException(StatusCodes.BadNotConnected);
}
```

修改二、将起始时间和结束时间对应到相应的控件上去，startDate 和 endDate 分别是 2 个 DatePicker 控件的名称。

```
var startTime = (DateTime)this.startDate.SelectedDate;
var endTime = (DateTime)this.endDate.SelectedDate;

//var startTime = DateTime.UtcNow.Subtract(TimeSpan.FromMinutes(60)); // TODO: replace with real start time
//var endTime = DateTime.UtcNow; // TODO: replace with real end time
```

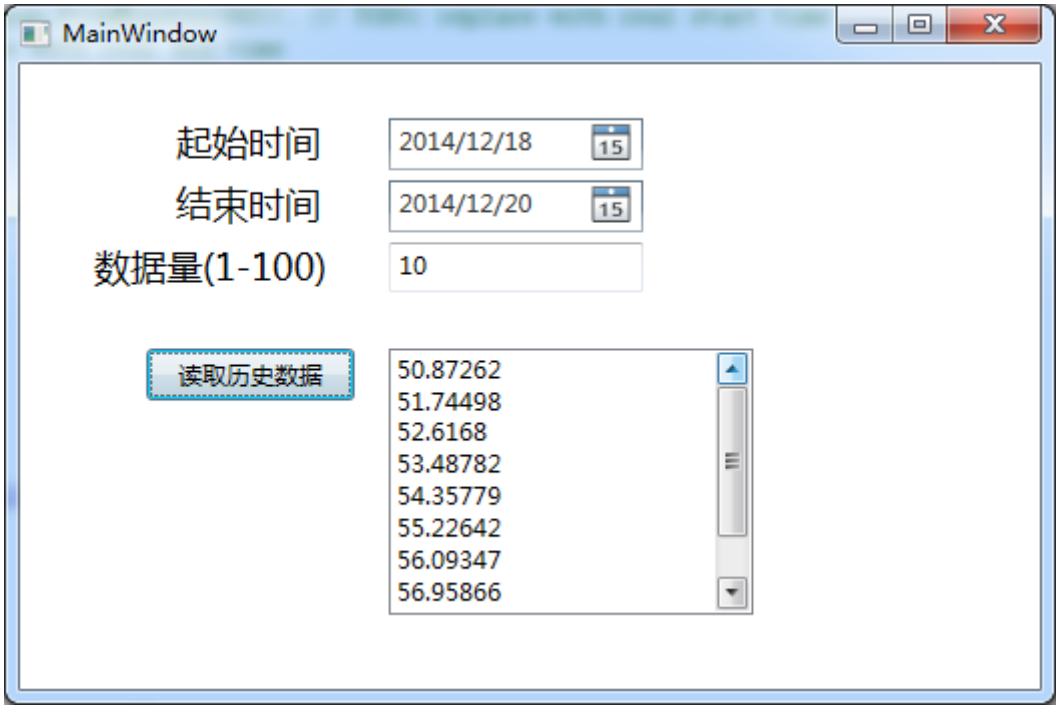
修改三、将 TextBox 的输入值作为数据量参数，最大值为 100

```
session.HistoryReadRawDataAsync(
    new NodeId[] { variableId },
    startTime,
    endTime,
    UInt32.Parse(this.dataNum.Text), // maximum records per variable of each call
    false,
    null).ContinueWith((t) => {
```

修改四、在处理数据(Process Data)处，将读取的数据以字符串集合(List<string>)的形式作为 ListBox 控件的 ItemSource，使数据在 ListBox 上显示出来。

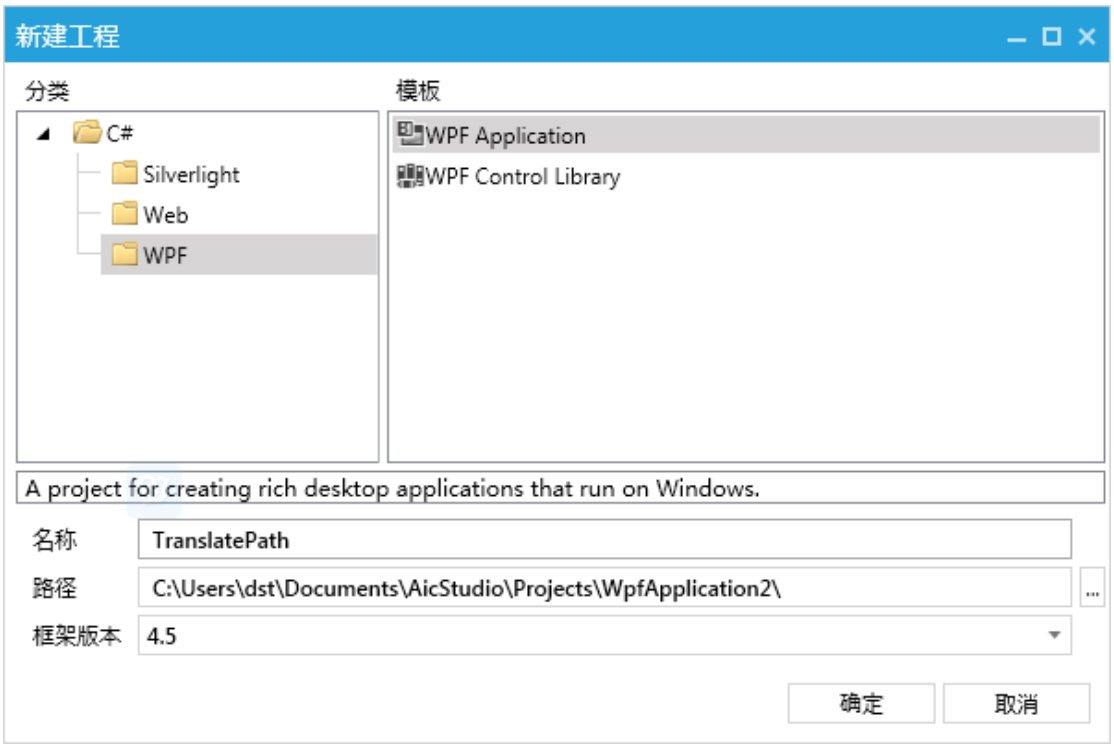
```
    dataValues = result.Result;
     if (dataValues == null || dataValues.Count == 0) {
         // TODO: NO DATA
     } else {
         // TODO: Process Data
List<string> listString = new List<string>();
if (dataValues != null)
{
   foreach (DataValue dv in dataValues)
   {
      listString.Add(dv.Value.ToString());
   }
}

this.listValue.ItemsSource = listString;
     }
}
```

修改完毕后点击运行，进行测试。测试需要 DataHub 已经存储了历史数据，并打开 DataHub 服务。

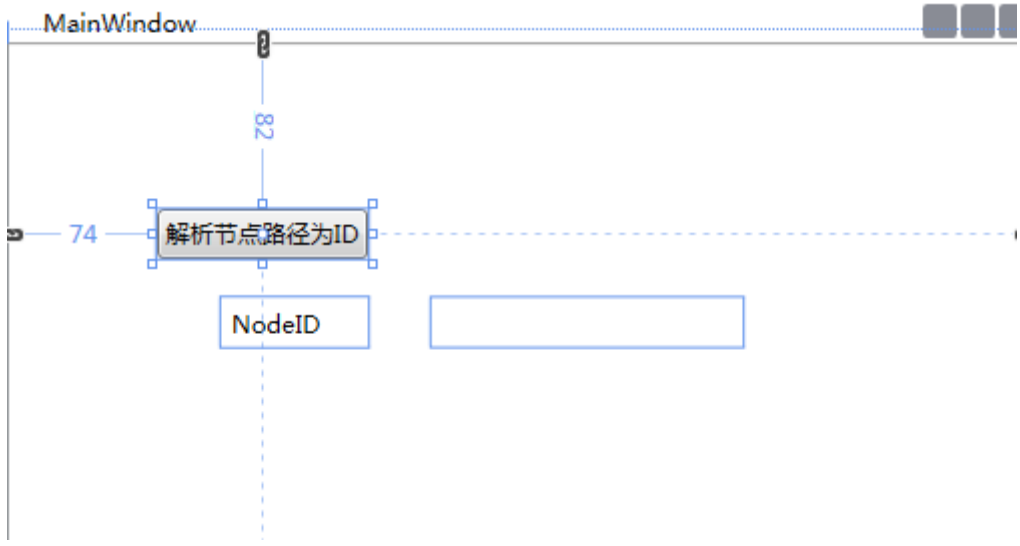


5.2.6 解析节点路径为 ID(Translate path)

先建 WPF 工程进行此代码段的测试。

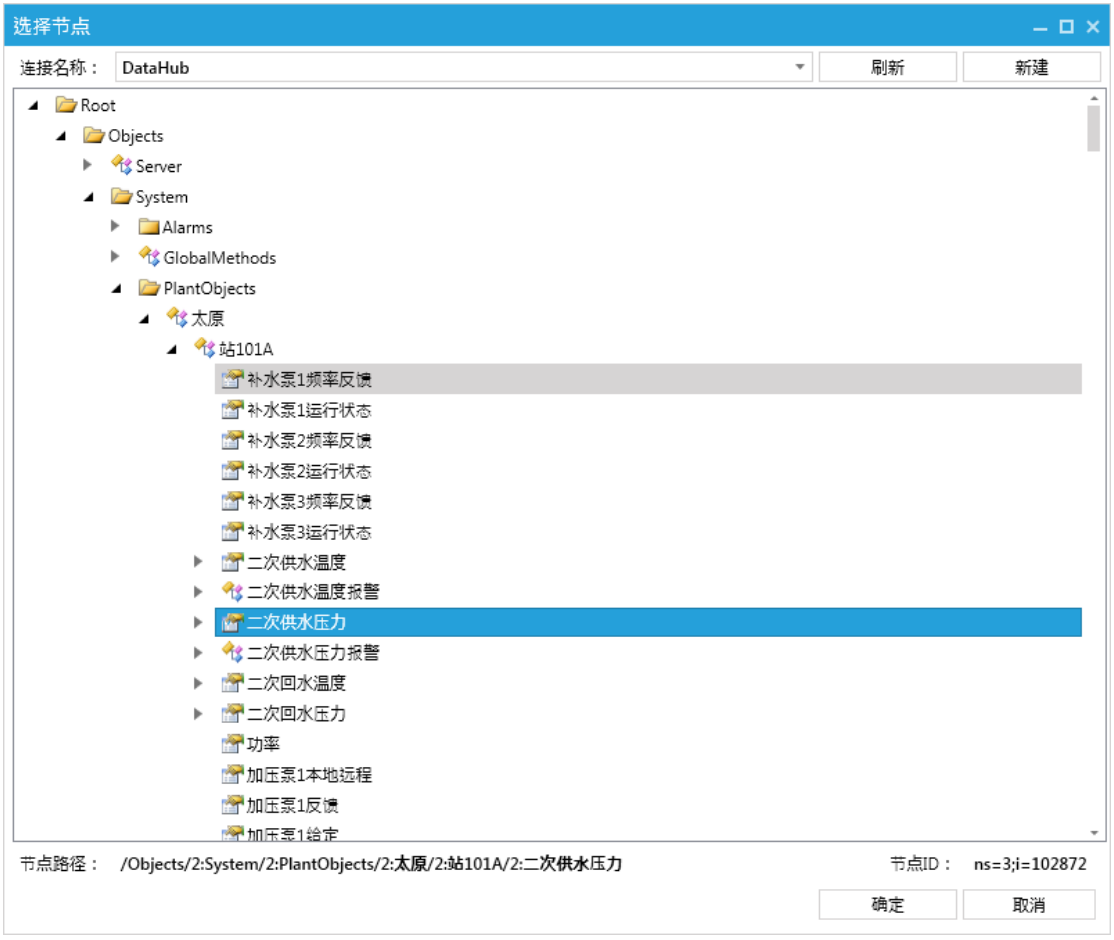


在 Xaml(标记文本)界面添加控件。



在按钮的 Click 方法中插入代码段->解析节点路径为 ID

选择连接的节点



点击确定添加代码段

```
        private void btReadHistory_Click(object sender, System.Windows.RoutedEventArgs e)
        {
            // TODO: Add event handler implementation here.
            var configuration = Application.Current.Resources["DataHub"] as IUaConnectionConfiguration;
            if (configuration == null || !configuration.IsValid) {
                throw new ServiceResultException(StatusCodes.BadConfigurationError);
            }

            var connection = UaConnectionManager.Current.GetConnection(configuration);
            var session = (connection == null) ? null : connection.Connection;
            if (session == null) {
                throw new ServiceResultException(StatusCodes.BadNotConnected);
            }

            session.TranslateBrowsePathAsync(
                Opc.Ua.ObjectIds.RootFolder, // TODO: replace with real base node id
                "/Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力") // TODO: replace with relative path to base node
        .ContinueWith((t) => {
                if (t.Exception != null) {
                    // TODO: Error
                } else {
                    var result = t.Result;
                    if (result.IsGood) {
                        // TODO: OK
                        // NodeId nodeId = result.Result;


                    } else {
                        // TODO: Error
                        // ServiceResult sr = result.Error;


                    }
                }
            }, TaskScheduler.FromCurrentSynchronizationContext());
```

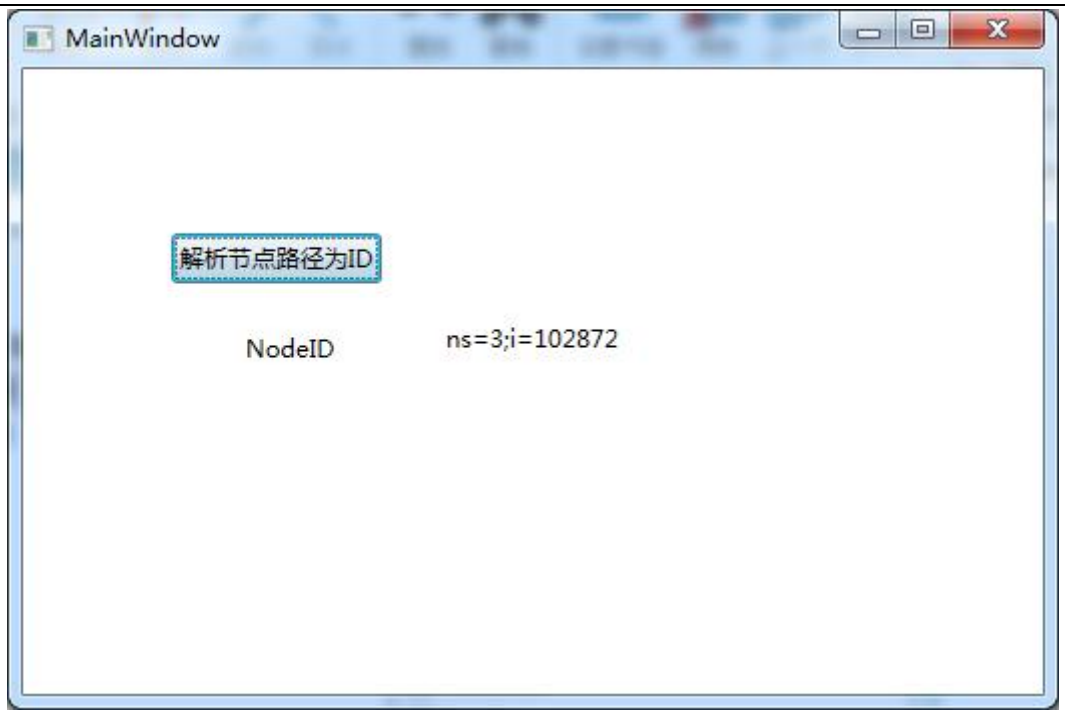
添加完毕后进行相关的修改。

修改一、与 5）小节中一样在建立连接的过程中进行当前线程的挂起。

修改二、蓝色背景部分代码，将获取的 NodeID 在 TextBlock 控件上显示。

```
            var connection = UaConnectionManager.Current.GetConnection(configuration);
            Thread.Sleep(2000);
            var session = (connection == null) ? null : connection.Connection;
            if (session == null) {
                throw new ServiceResultException(StatusCodes.BadNotConnected);
            }

            session.TranslateBrowsePathAsync(
                Opc.Ua.ObjectIds.RootFolder, // TODO: replace with real base node id
                "/Objects/2:System/2:PlantObjects/2:太原/2:站101A/2:二次供水压力") // TODO: replace with relative path to base node
        .ContinueWith((t) => {
                if (t.Exception != null) {
                    // TODO: Error
                } else {
                    var result = t.Result;
                    if (result.IsGood) {
                        // TODO: OK
                        NodeId nodeId = result.Result;
                        this.textBlock.Text = nodeId.ToString();


                    } else {
                        // TODO: Error
                        // ServiceResult sr = result.Error;


                    }
                }
            }, TaskScheduler.FromCurrentSynchronizationContext());
```

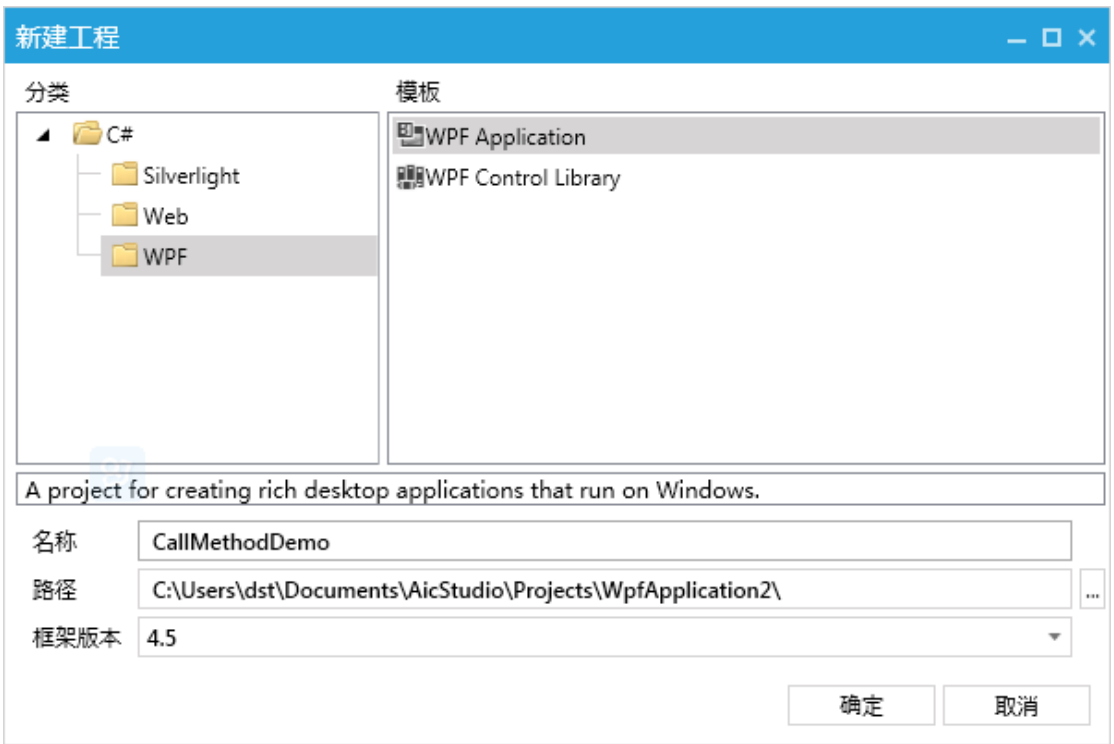
修改完毕后进行运行调试。



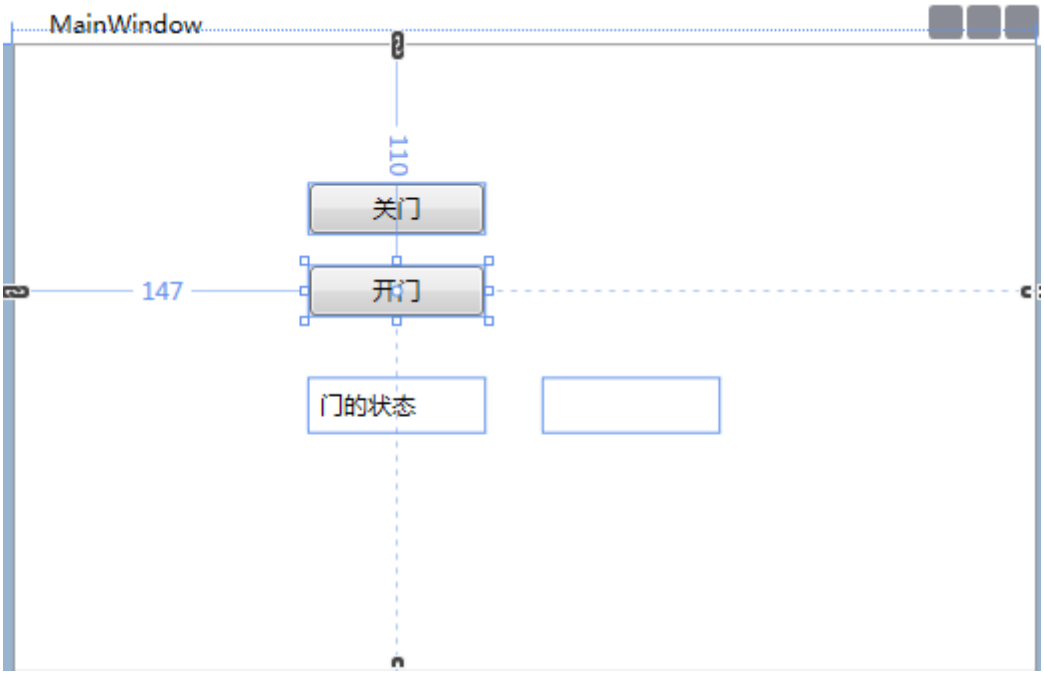
5.2.7 调用方法(Call method)

这里调用的方法是在 DataHub 中数据类型中定义的方法，用于修改或者读取数据类型对象的变量值（更多详细的介绍请参见 AicDataHub 用户手册）。

下面是调用方法的示例。首先新建 WPF 工程 CallMethodDemo;

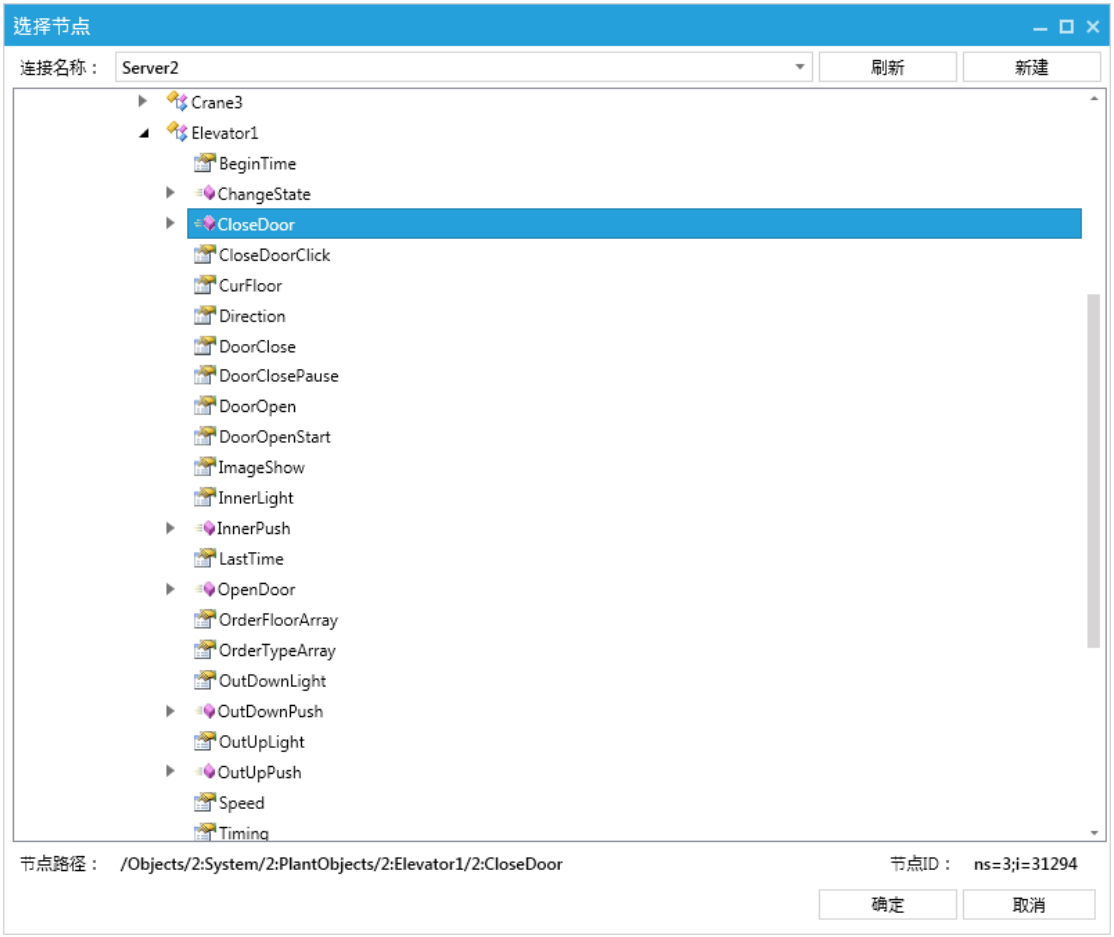


在 XAML 用户界面(标记文本)上添加示例所用的控件，2 个按钮分别用于关门和开门，一个 TextBlock 用于显示门的状态。



在开门与关门按钮的单击事件 Click 方法中，插入调用方法的代码段。

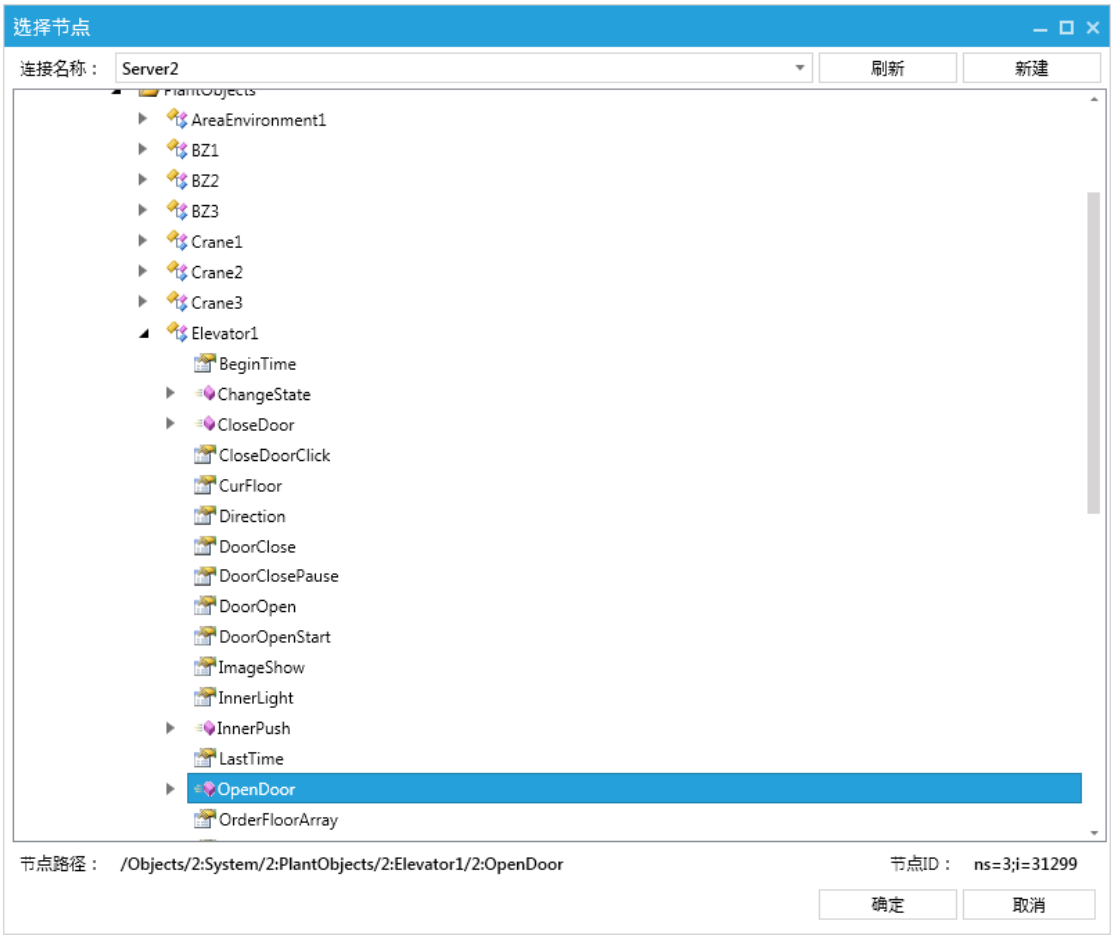
关门按钮 Click 方法中，选择 DataHub 服务器（此实例用的是电梯的类型），选择对象 Elevator 的方法节点 CloseDoor，点击 OK，添加关门的方法代码段。



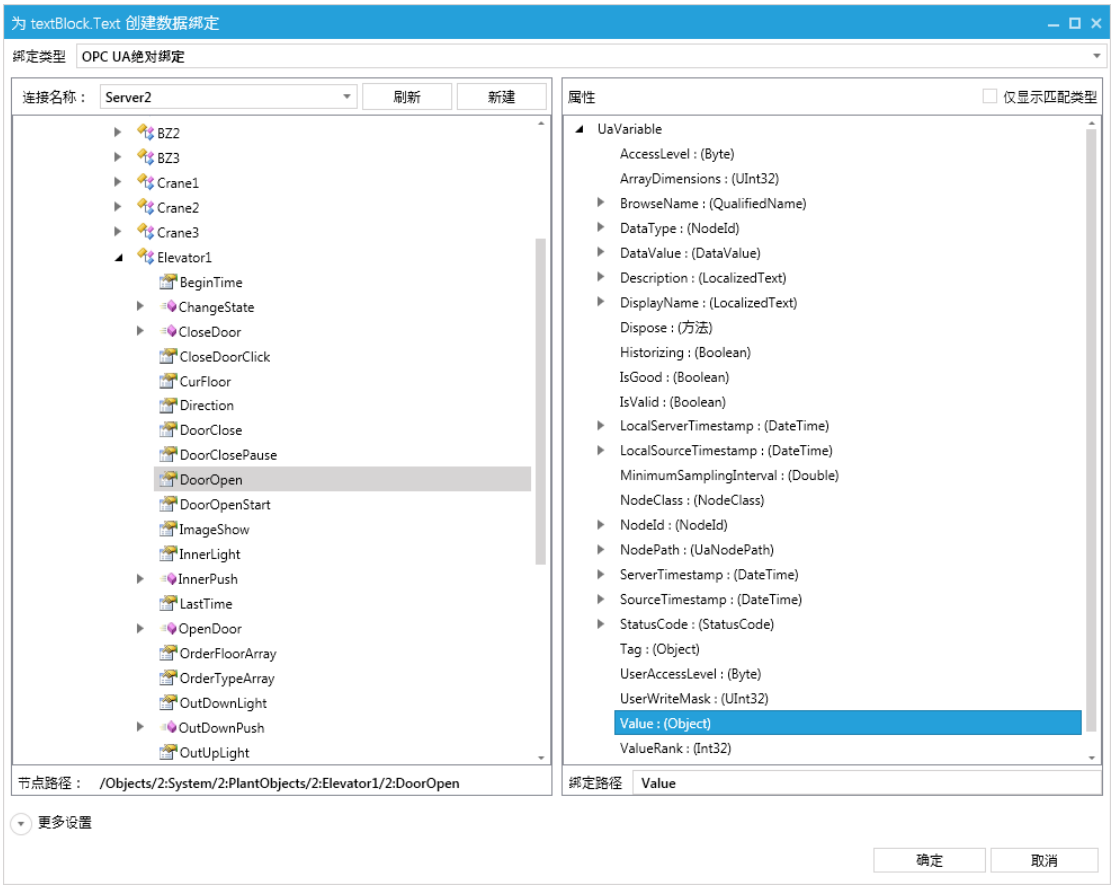
此代码由于没有输出参数，所以不需要进行进一步的修改。

```
private void btClose_Click(object sender, System.Windows.RoutedEventArgs e)
    {
        // TODO: Add event handler implementation here.
        var configuration = Application.Current.Resources["Server2"] as IUaConnectionConfiguration;
        if (configuration == null || !configuration.IsValid) {
            throw new ServiceResultException(StatusCodes.BadConfigurationError);
        }

        var connection = UaConnectionManager.Current.GetConnection(configuration);
        var session = (connection == null) ? null : connection.Connection;
        if (session == null) {
            throw new ServiceResultException(StatusCodes.BadNotConnected);
        }

        session.CallAsync(
            new NodeId(31365U, 3), // TODO: replace with real object node id which this method called on
            new NodeId(31294U, 3) // Method: /Objects/2:System/2:PlantObjects/2:Elevator1/2:CloseDoor
// TODO: Method input arguments here
        ).ContinueWith((t) => {
            if (t.Exception != null) {
                // TODO: Error

            } else {
                // TODO: OK
                // IList<object> outputArguments = t.Result;

            }
        }, TaskScheduler.FromCurrentSynchronizationContext());
```

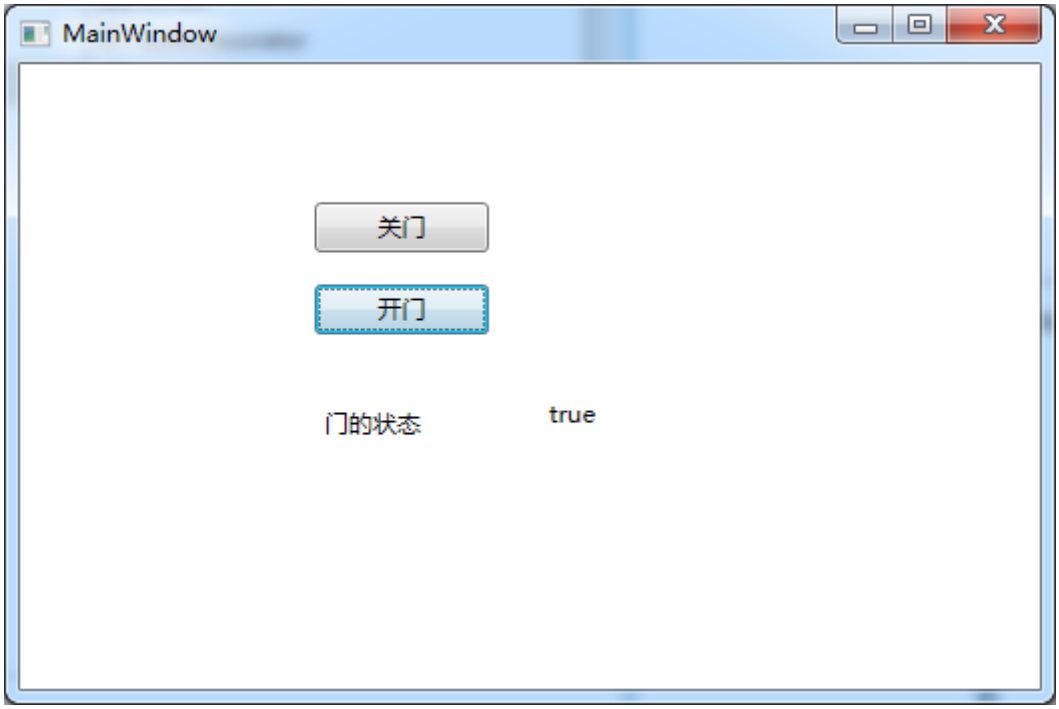
开门按钮的 Click 方法添加和关门类似，选择节点时选择 Elevator1 对象下的 OpenDoor 方法节点。



然后将 TextBlock.Text 属性 OPC UA 绝对绑定到 Elevator1 对象的 DoorOpen 变量上，用于显示当前的电梯门的状态。(绑定的细节参见 4.2 节数据绑定)

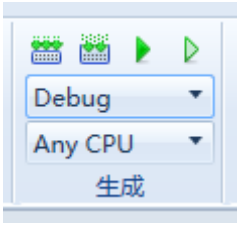


然后编译运行调试。关门所需要的时间会比较长(3s 左右)



6 编译与调试

在 AicStudio 的主页菜单中有下图所示功能区：支持 Debug 和 Release，Debug 是调试用的，可以打断点执行，Release 是发布用的，不能打断点执行。下图中第一行的四个小按钮从左到右依次是：生成解决方案；生成选中的工程（在解决方案浏览器中选中想要生成的工程）；调试模式运行（断点有效）；不调试运行（断点无效）。



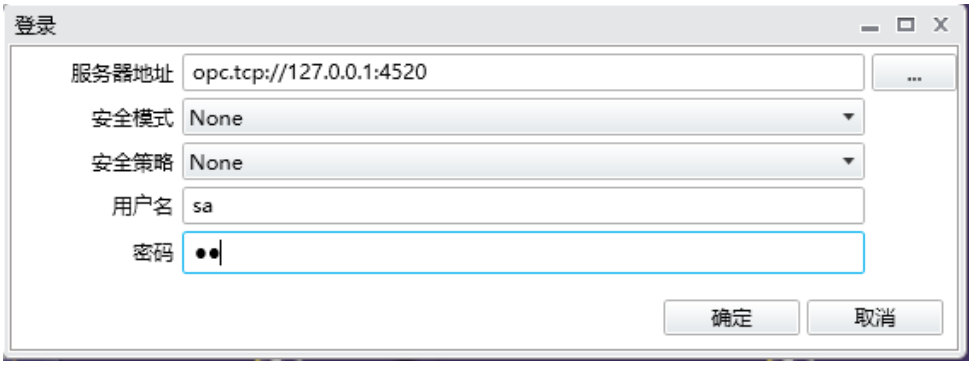
工程菜单中有如下常用的编译和调试功能，基本和 VisualStudio 中的常用调试功能类似。

7 快速入门示例

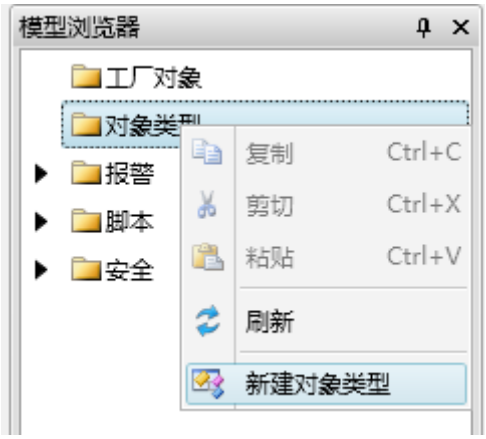
本节中，将使用 AicStudio 新建一个 WPF 项目，结合 DataHub 和 IOServer 具体实现一个简单的监控工。在此简略说明一点：IOServer 是用于现场采集数据的，并将数据主动或者被动的发送给 DataHub，DataHub 进行对数据的存储、数据下发（下发给 IOServer）作为 UI 界面的数据源（通过界面 Binding 至 DataHub 某对象实例的某变量），IOServer 和 DataHub 的具体可以查看 AicTech 的 IOServer 用户手册和 DataHub 用户手册。

7.1 使用 DataHub 建立对象模型和对象实例

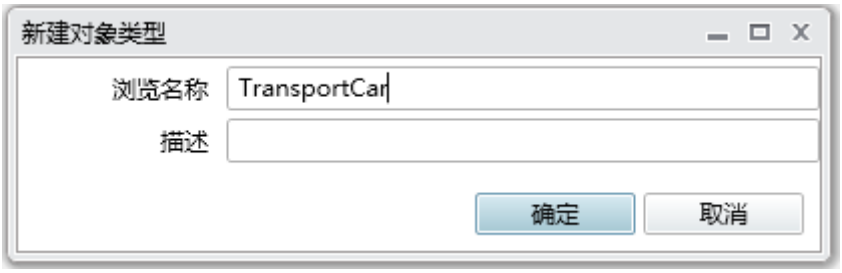
打开 AicTechDataHub Configurator，打开之前应确保 AicTechDataHub Server 正在运行，可以在电脑的服务中查看开启。下图登陆用户名、密码都为 sa。



新建对象类型，右击模型浏览器中的对象类型选择新建对象类型如下图。

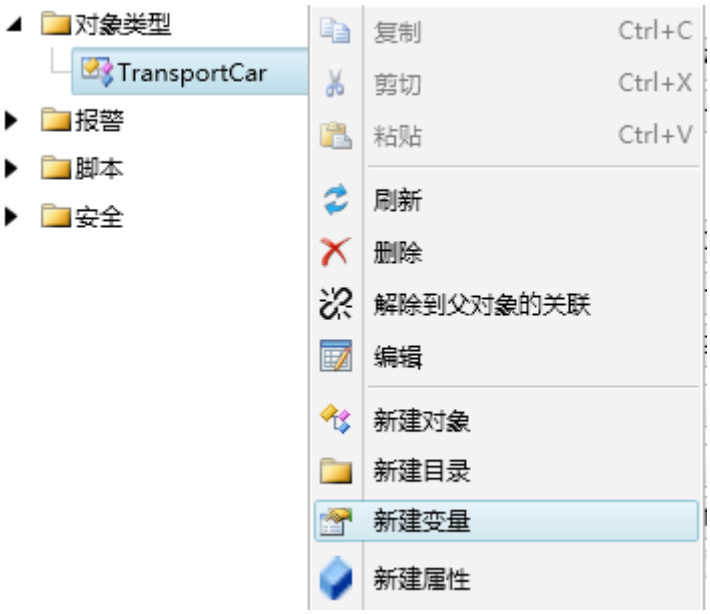


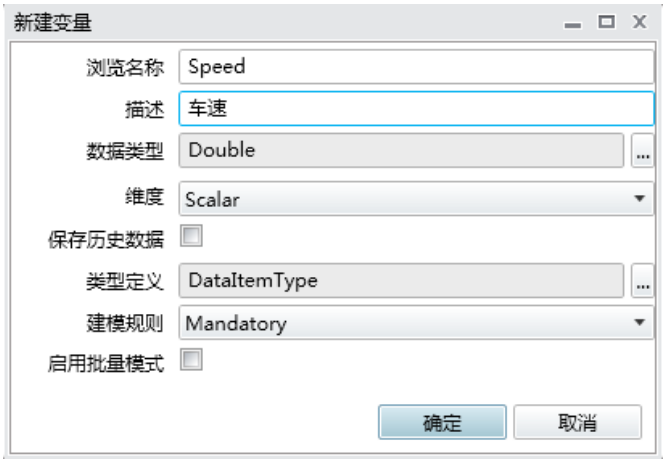
自定义浏览名称，如下图。



对于运输车这个模型，这里简略定义这样几个变量：车速 Speed，运输车钢罐中物料的物位，钢罐内部的温度（也应该有压力，这里省略）。

右击新建完成的 TransportCar 选择新建变量，如下图。建 Speed 变量，MaterialLeval 变量，Tempurature 变量，数据类型都为 Double 即可。

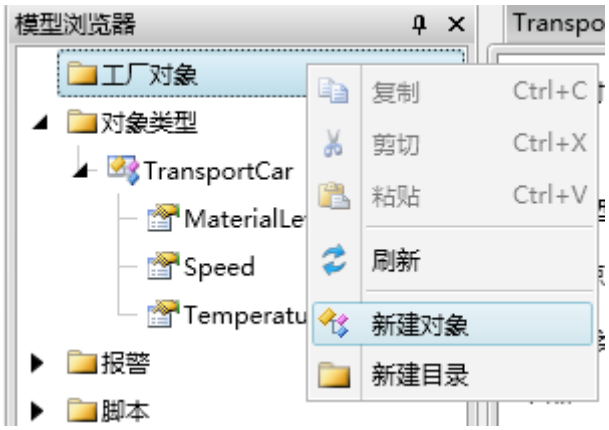

对象类型
TransportCar
报警
脚本
安全
复制 Ctrl+C
剪切 Ctrl+X
粘贴 Ctrl+V
刷新
删除
解除到父对象的关联
编辑
新建对象
新建目录
新建变量
新建属性

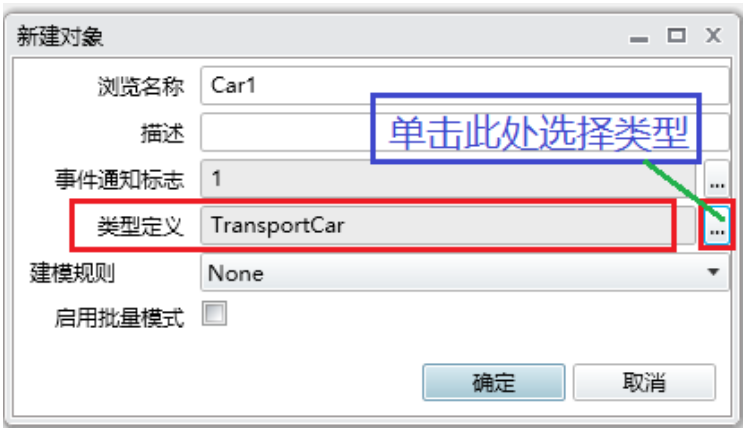

新建变量
浏览名称 Speed
描述 车速
数据类型 Double
维度 Scalar
保存历史数据
类型定义 DataItemType
建模规则 Mandatory
启用批量模式
确定
取消

新建完成如下图：


对象类型
TransportCar
MaterialLevel
Speed
Temperature
报警

右击工厂对象（PlantObject）选择新建对象（如下图），类型定义选择刚才新建的 TransportCar，新建两个名为 Car1 和 Car2 的对象实例。



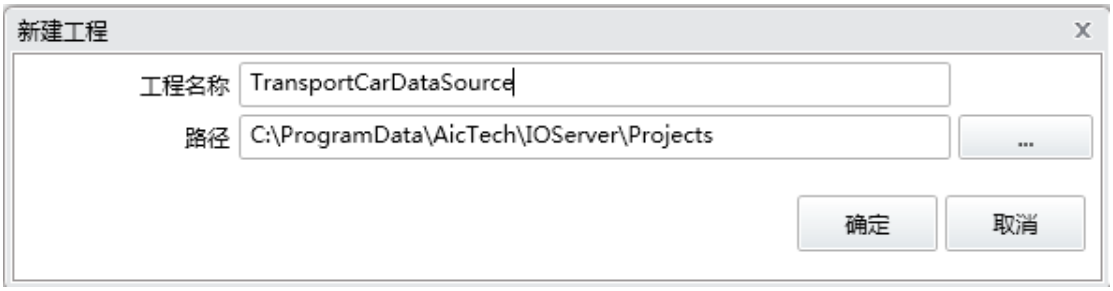


新建完成后如下图：

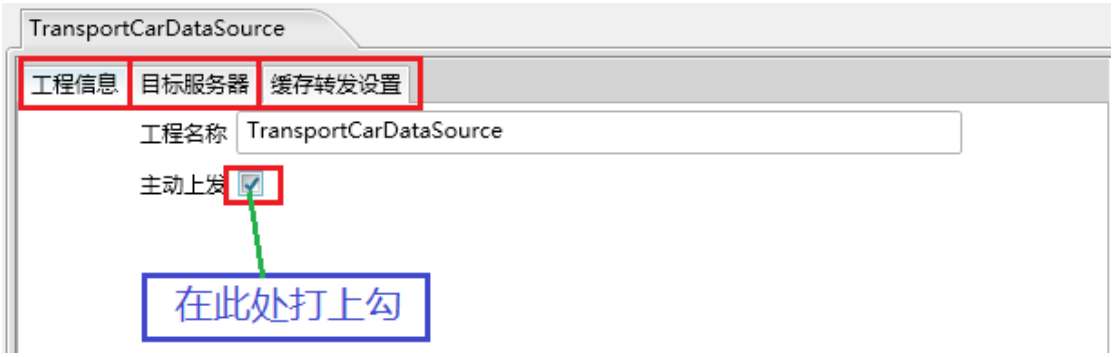
至此运输车的对象模型和实例均已建立完成，后续的工作就是将 IOServer 中采集上来的对应数据和 DataHub 实例中对应的变量相互绑定以及将 UI 界面上的变量和 DataHub 对象模型中对应的变量进行绑定。

7.2 使用 IOServer 为 DataHub 提供数据

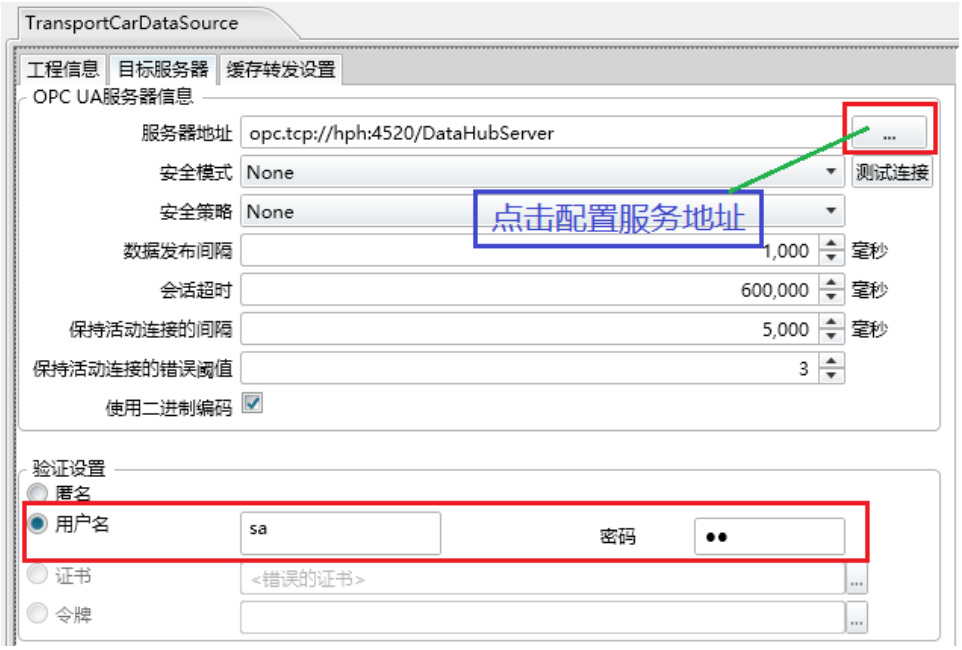
打开 IOServer（以管理员方式），新建工程如下图：



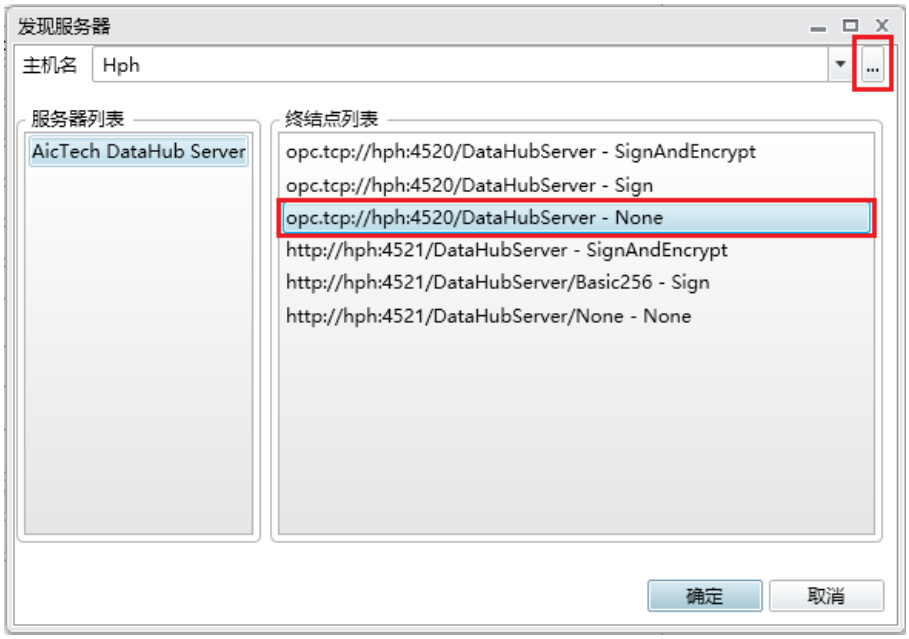
点击确定，右击新建的工程名选择修改工程，右边的窗口出现此工程信息（如下图），在主动上发出打上勾后目标服务器和缓存转发设置被激活。



点击目标服务器，配置如下。

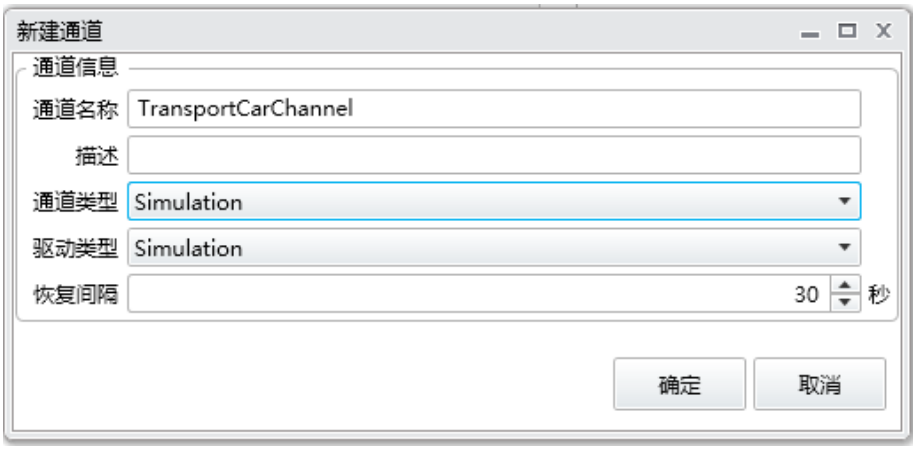


点击配置服务器地址按钮，窗口如下，点击右边红框的按钮，选择红框条目，其中 hph 是本人的计算机名，不同计算机名字可能不同，点击确定。

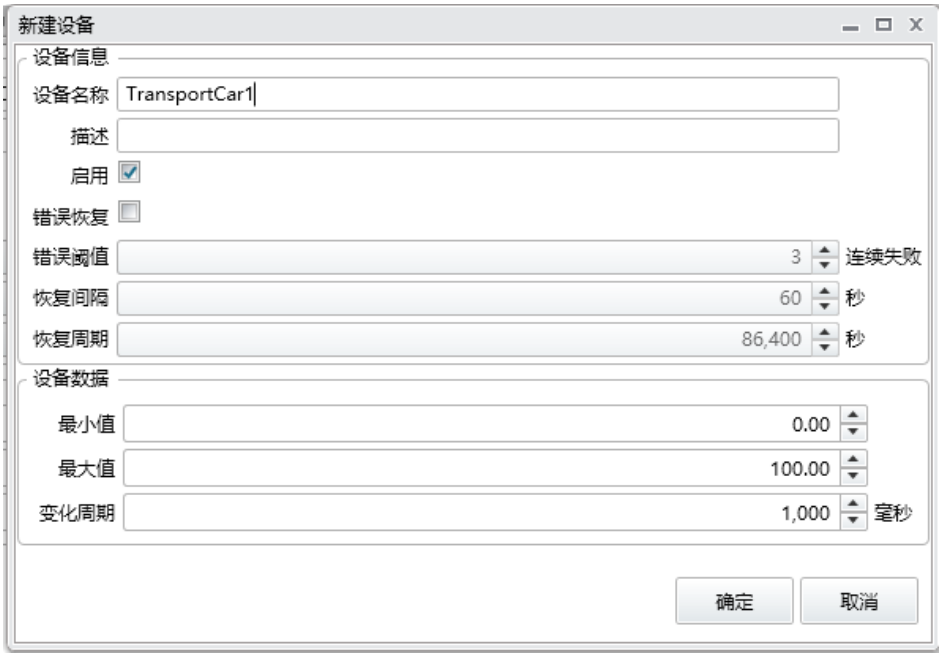


目标服务器配置窗口中的验证设置选择用户名 sa，密码 sa。

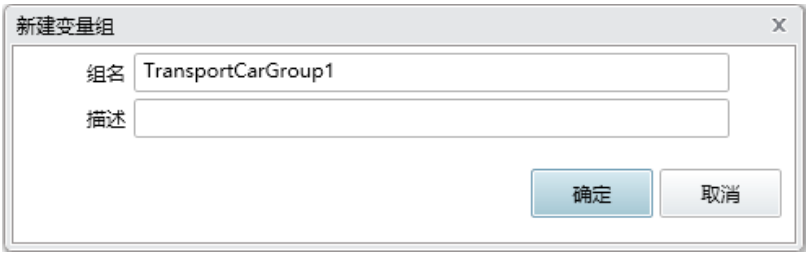
右击工程名选择新建通道，配置如下图，通道名称可自定义，通道类型选择 Simulation 驱动类型选择 Simulation，点击确定。在此说明一点：Simulation 驱动是一个模拟的数据来源，可以按照一定过得规律产生模拟数据，实际中需要从现场的设备中采集数据，还需要开发相应的 IOServer 驱动，具体见 IOServer 驱动开发文档。



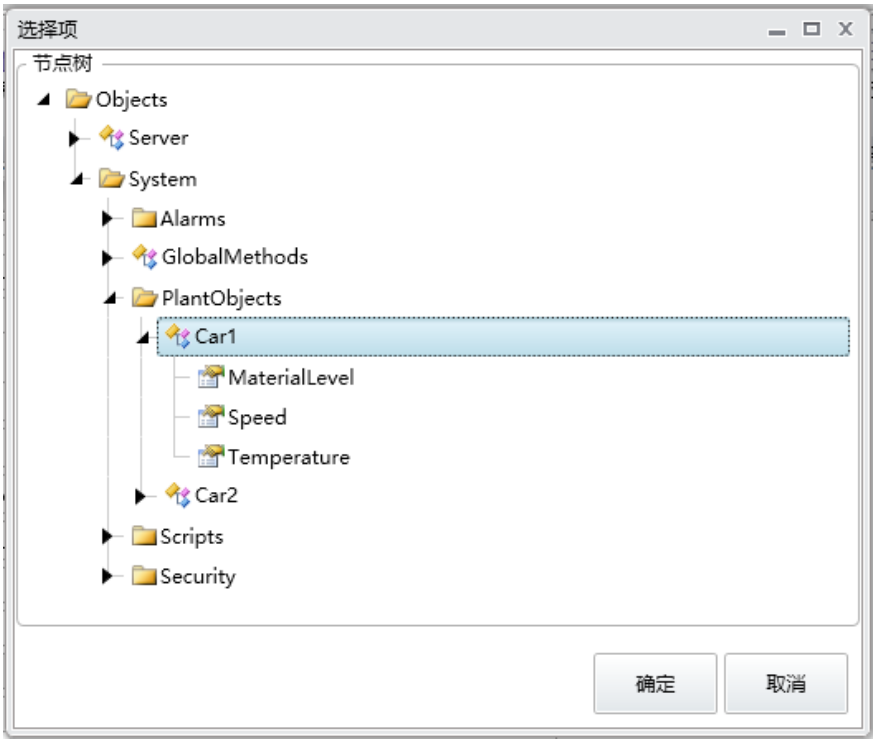
右击通道名称选择新建设备，如下图，设备名称可自定义，设备数据中最大值和最小值在此选择默认的即可，变化周期用于设置数值大小变化 1 所间隔的时间，按默认设置即可（1000ms）也可自行设置。



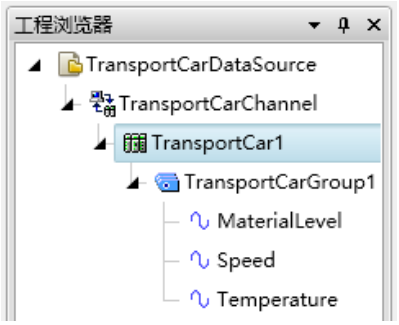
右击设备名称选择新建变量组，变量组名称自定义。



为方便起见，我们选择则从 DataHub 导入变量（当然也可以自行添加相应的变量），右击变量组名选择从服务器导入，在弹出的窗口进行如下选择，点击确定即可。

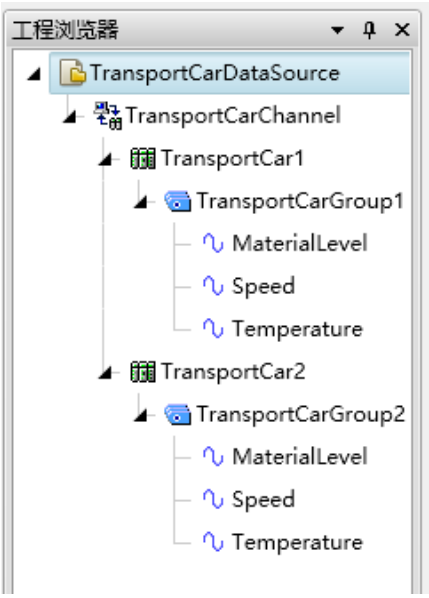


导入后结果如下图：



导入成功后再设置一下各个变量模拟采集的方式。双击变量名，右边会显示变量模拟采集的设置窗口，变量数据窗口中的寄存器名称选项用于设置变量模拟采集方式，此处将其选择设置为三角波形式“Triangle”，这样变量的值会从最小值以 1/s 的变化频率增加到最大值之后再从最大值均匀的减小到最小值，循环进行，此处我对这三个变量的寄存器名都设置为“Triangle”，可自定义。

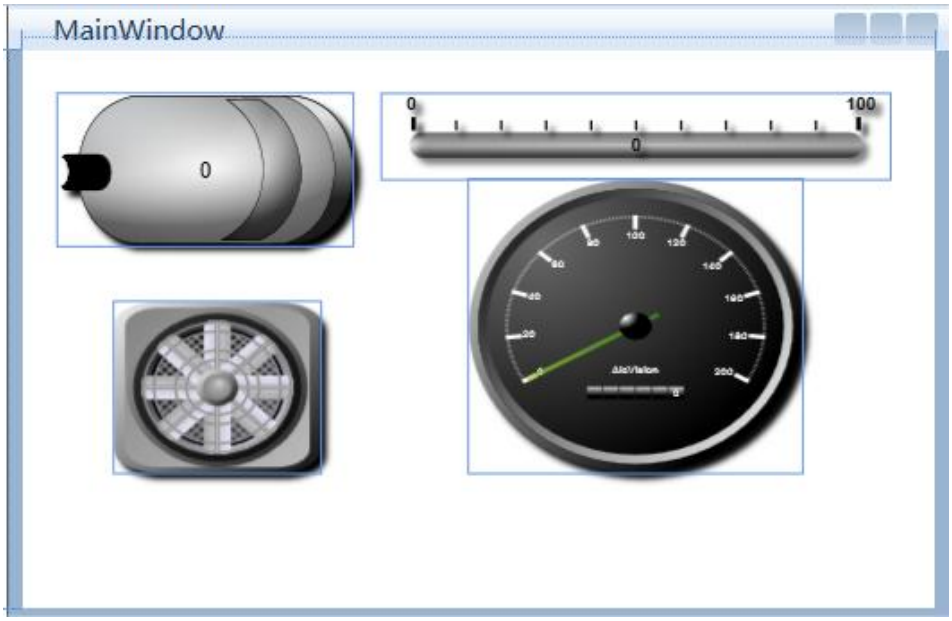
至此我们已经配置完 TransportCar1 的变量了，类似我们在 TransportCarChannel 下再建一个设备 TransportCar2 将在 DataHub 中的 Car2 这个对象实例中的变量导入到 IOServer 中，为了以示区分，可以改变变量模拟采集方式，这里对 TransportCar2 的变量的模拟采集方式改为“Increase”、修改 TransportCar2 将最小值改为 20，最大值改为 90，变化周期改为 500ms，可自定义。



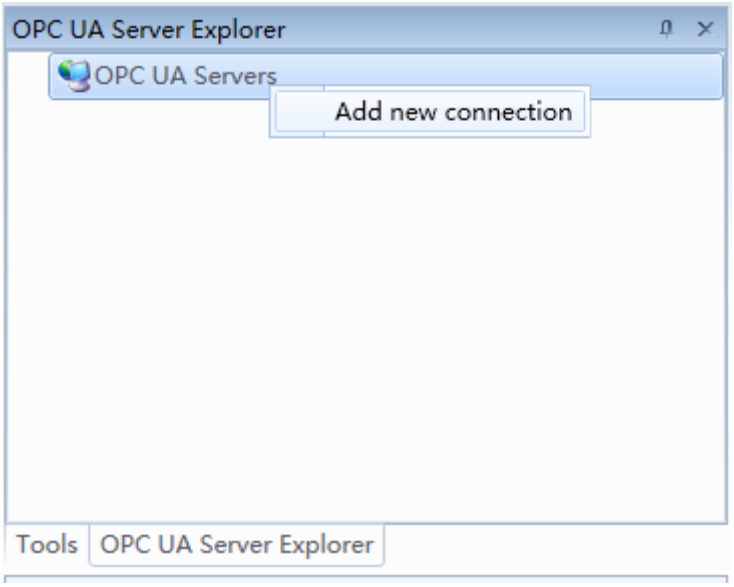
导入完成后点击保存（必须先保存），右击工程名选择“设为启动工程（本机）”（若设置失败，那么可能是打开 IOServer 时不是以管理员方式启动的）。在服务中开启 IOServer，若 IOServer 服务正在运行则需重启服务（此时也应该已经打开 DataHub 服务）。完成后 DataHub 中 Car1 和 Car2 两个实例中的变量已经随着 IOServer 中相应的变量的变化而变化了。从 4.2 中我们已经知道 UI 界面上控件的属性已经可以绑定到 OPC UA Data Node（DataHub 中已经定义好的变量），这样就可以在 UI 界面上实时显示我们需要监测的变量。

7.3 使用 AicStudio 构建监控画面

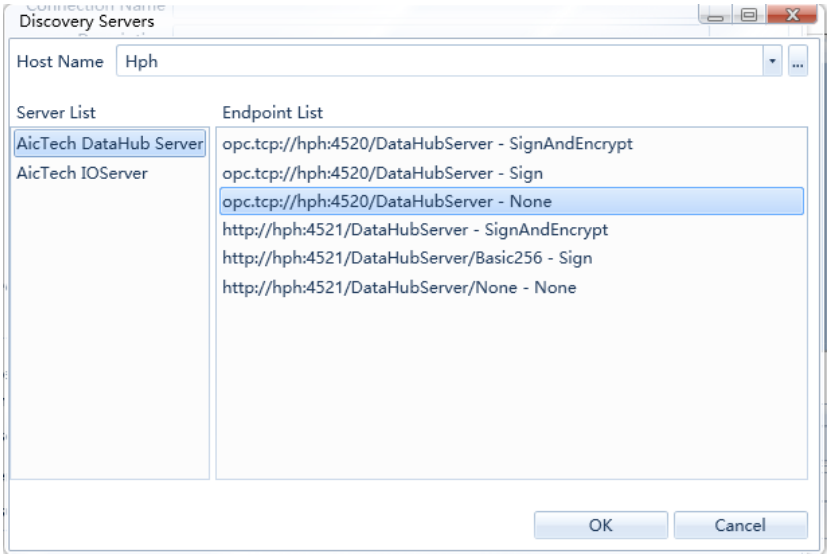
打开 AicStudio 新建 WPF 项目，命名为“AicTech 快速入门”从工具窗口拖入艾克信控控件：HorizontalLinearGauge，Fans003，VesselTank，RacingGauge，对每个控件可以用上文中的特效修饰。HorizontalLinearGauge 用于表示温度，RacingGauge 用于表示车速，VesselTank 用于表示钢罐，控件本身可以显示钢罐的物位，Fans003 显示的风扇用于表示制冷机，其功能为在温度高于 40 时进行制冷（叶扇的转动），并且温度越高，叶扇旋转越快。

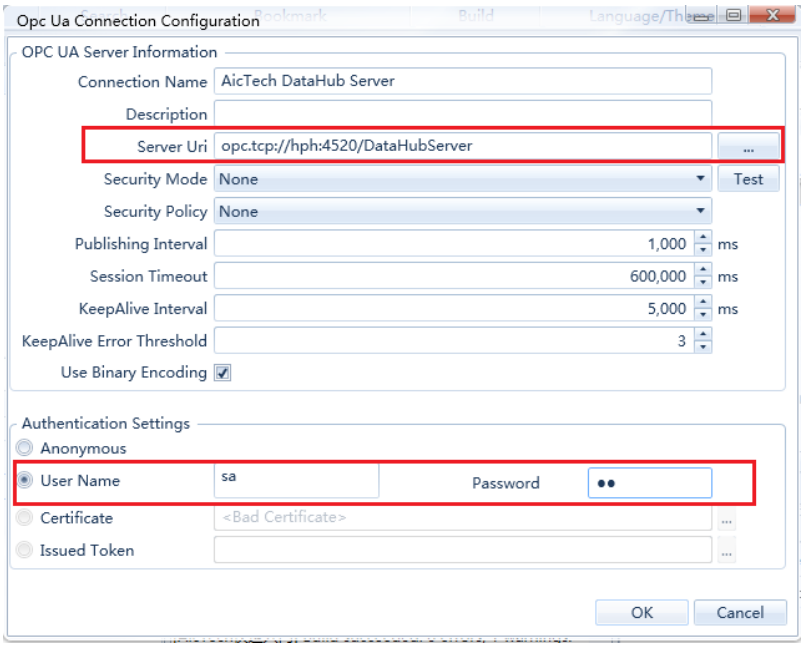


在工具窗口处选择 OPC UA 服务浏览器添加一个新的连接：

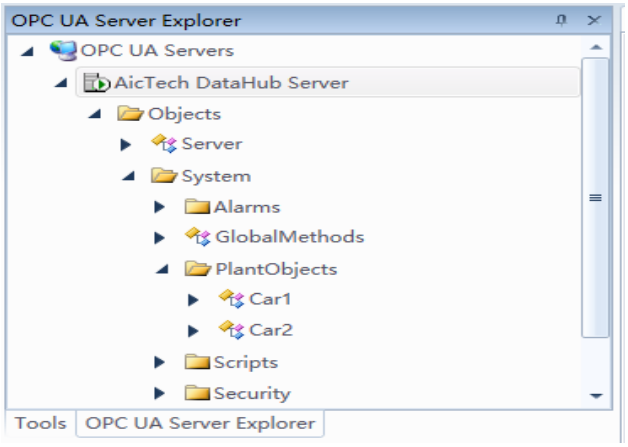


此处添加的而服务器连接和上文 IOServer 中配置目标服务器相同：

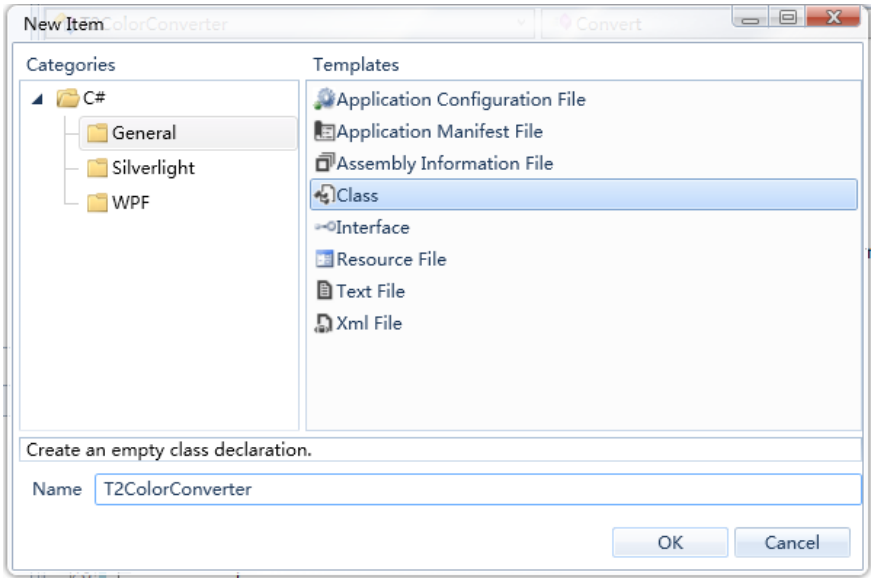




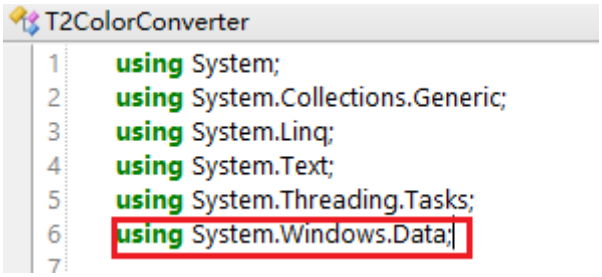
点击确定后双击这个新建的连接，这时在 AicStudio 中已经可以看到之前新建的两个实例：



在浏览器窗口右击项目名称，选择“Add New Item”，添加一个类，命名为T2ColorConverter，如图。



在 T2ColorConverter 文件中添加引用 usingSystem.Windows.Data;



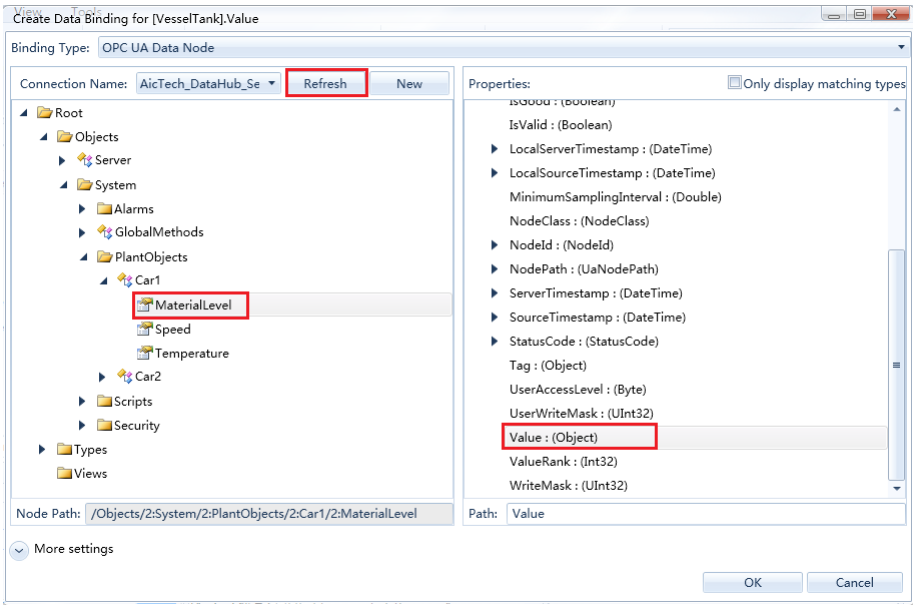
在 T2ColorConverter 文件中添加如下代码：

```
namespace AicTech快速入门
{
    /// <summary>
    /// T2ColorConverter.
    /// </summary>
    public class T2ColorConverter:IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
            if(value!=null)
            {
                int a =System.Convert.ToInt32(value);
                if(a>=0&&a<40)
                    return "Green";
                if(a>=40&&a<80)
                    return "Orange";
                if(a>=80&&a<=100)
                    return "Red";

                return "Black";
            }

            return "Black";
        }

        public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

此文件实现了一个值转换器，用于 HorizontalLinearGauge 控件在温度低于 40 度时温度条的颜色显示绿色，40 到 80 之间显示橘黄色，高于 80 显示红色。

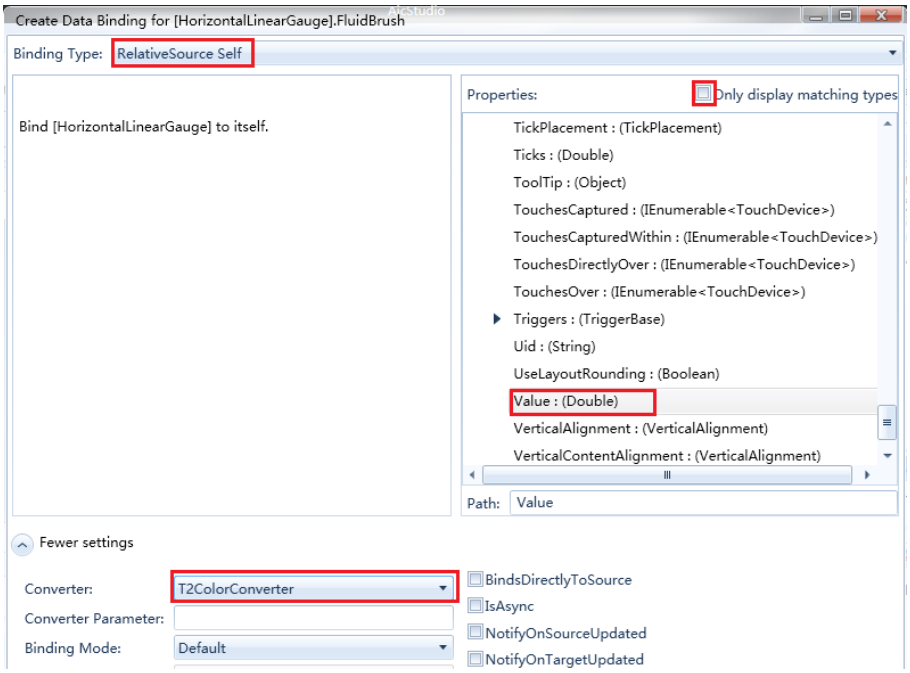
接下来将 DataHub 中的值绑定到特定控件的属性中:

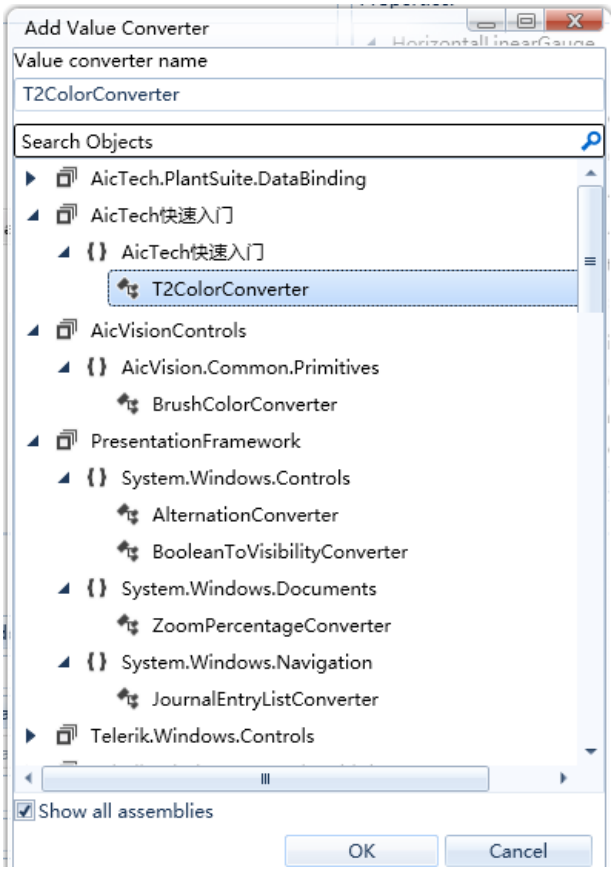
对 VesselTank 控件的 Value 属性进行数据绑定：选择 OPCUA 绝对绑定，点击刷新按钮，绑定到已经建好的实例 Car1 中的 MaterialLevel 属性的 Value 属性：



同样，将 RacingGauge 控件的 Value 属性绑定到 Car1 中的 Speed 属性的 Value 属性：将 HorizontalLinearGauge 控件的 Value 属性绑定到 Car1 中的 Temperature 属性的 Value 属性。

现在实现上文值转换器的功能：我们为 HorizontalLinearGauge 控件的 FluidBrush 属性创建一个数据绑定：绑定类型选择 RelativeSource Self，去掉仅显示匹配类型的勾（否则不能选择 Value 属性），选择 Value 属性，点开更多设置选择 T2ColorConverter 这个转换器（如果没有这个选项的话选择增加值转换器来创建一个 T2ColorConverter 即可、检查之前有没有正确写完这个转换器，）。

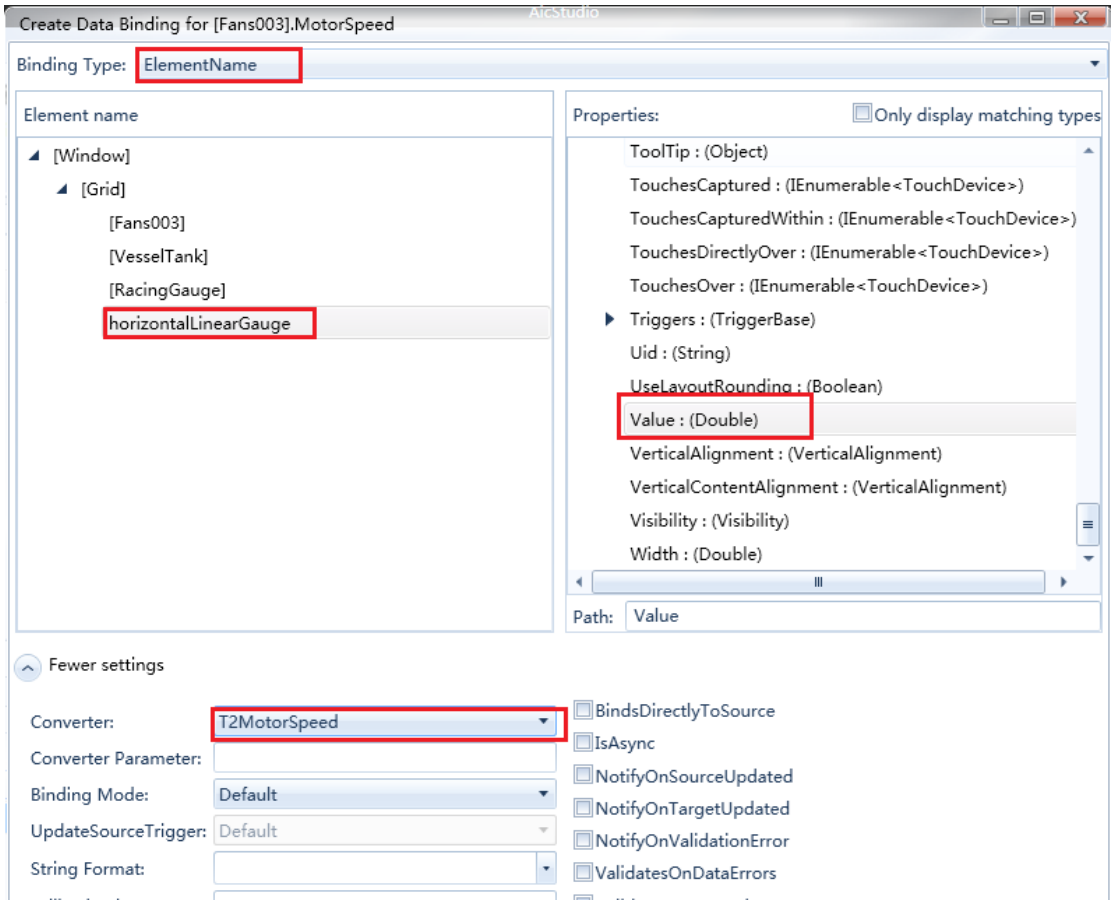




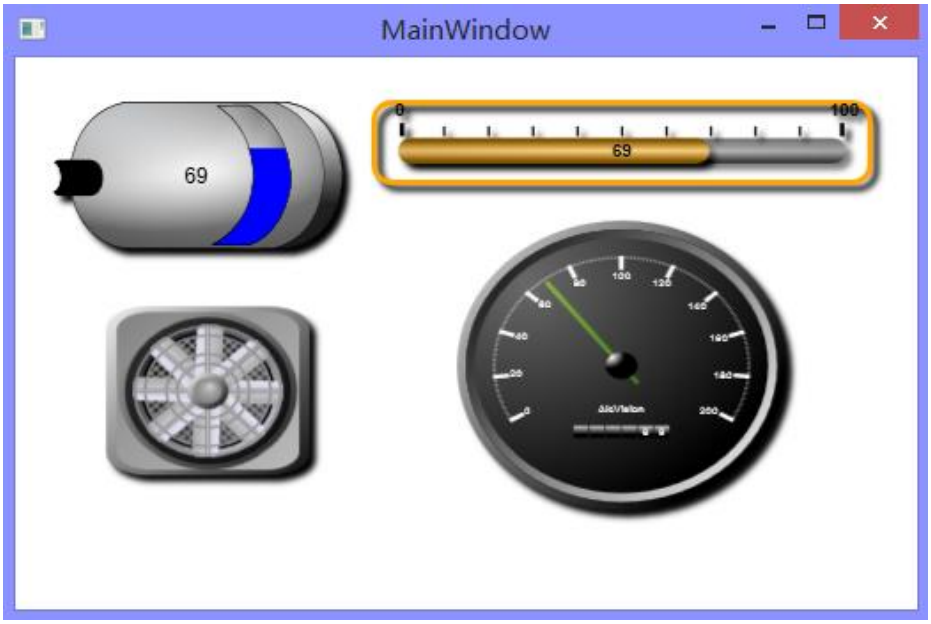
现在我们再来实现一个效果，让 Fans003 这个控件随着温度的变化而变化，在温度小于 40 度时，风扇不转动，温度大于 40 度小于 80 度时温度越高转动速度越快，大于 80 度时以最高转速转动（100），风扇的转速通过 Fans003 这个控件的 MotorSpeed 这个属性控制，为实现这个效果我们还需要实现一个值转换器。类似 T2ColorConverter 文件的创建，创建一个名为 T2MotorSpeed 的文件，添加引用 usingSystem.Windows.Data，添加如下代码：

```
public class T2MotorSpeed:IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        if(value!=null)
        {
            int a =System.Convert.ToInt32(value);
            if(a>=0&&a<40)
                return 0;
            else if(a>=40&&a<80)
                return 2*(a-30);
            else
                return 100;
        }

        return 0;
    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

现在添加 Fans003 控件的 MotorSpeed 属性的数据绑定：绑定类型选择 ElementName，绑定到 horizontalLinearGauge 控件的 Value 属性上，值转换器新建一个 T2MotorSpeed 并选择这个值转换器。



至此，我们已经完成了界面的基本控件属性和实际采集数据的绑定，点击运行即可看到实时监控效果。

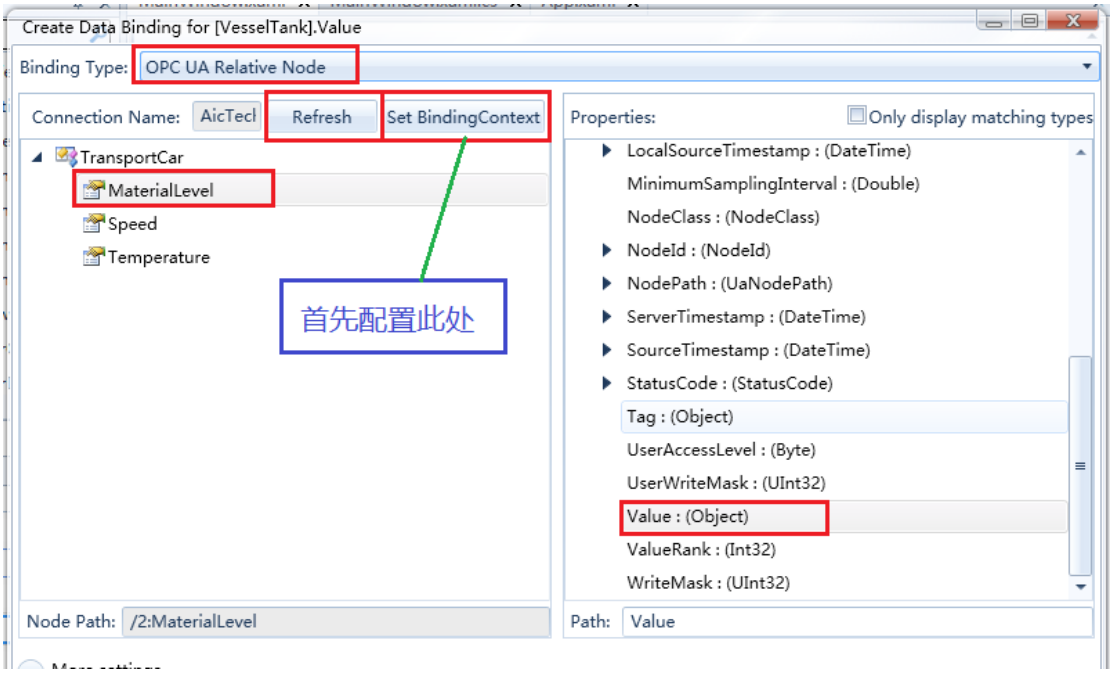


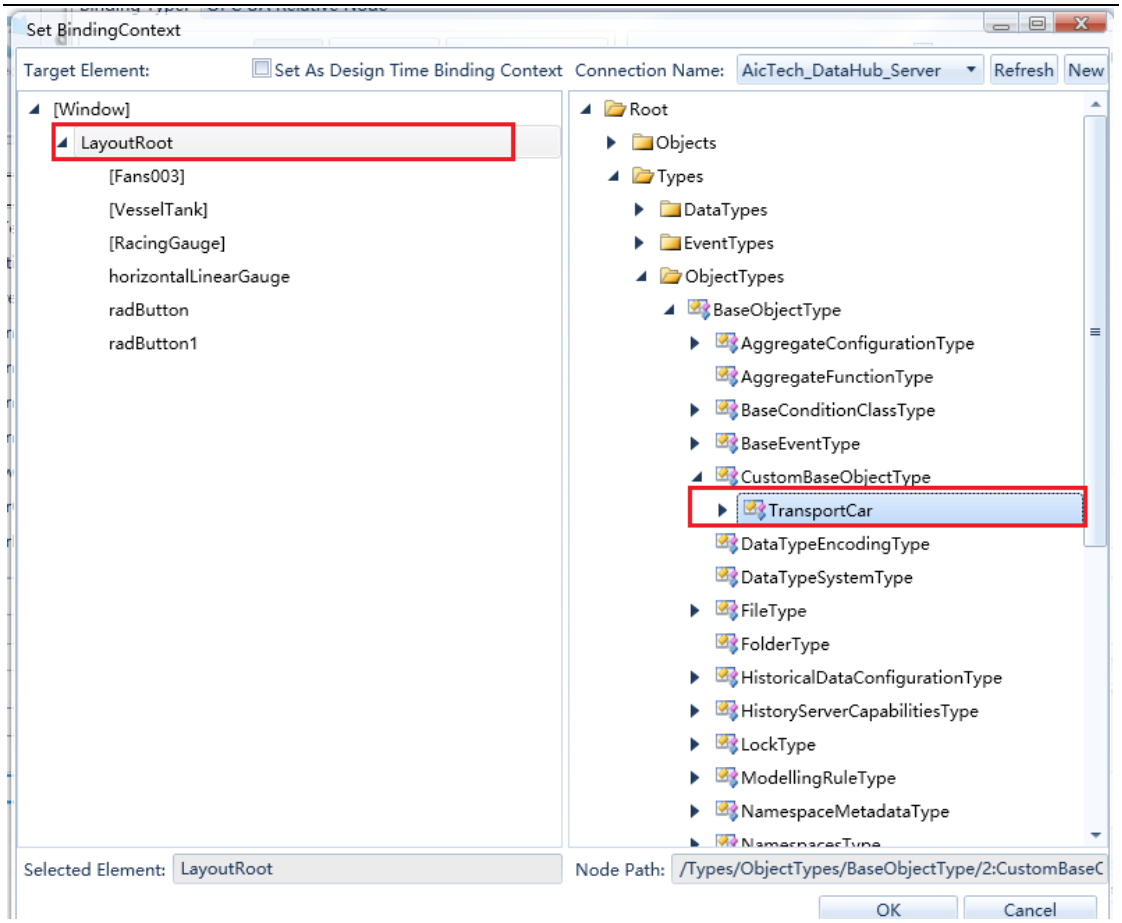
在实际情况下，我们会遇到检测相同模型的若干个实例，比如我们现在已经创建的 Car1 和 Car2，对于这种情况，实际的模型一般只需要创建一套显示控件即可（如此例子中简单的四个控件），用这一套控件切换着去显示不同实例的实时监控数据，切换的实现可以有很多种（这里中按钮实现），这种可以切换着去显示不同实例数据的数据绑定类型和至此本例的所采用的数据绑定类型有所不同，至此本例中控件属性和 OPCUA 模型实例的属性数据绑定方式为 OPCUA 绝对绑定，切换显示不同实例监控数据的实现是依赖于 OPCUA 相对绑定。下文详细讲述。

从工具窗口拖入两个 RadButton 控件，将其 Content 属性改为 Car1 和 Car2，将其 Name 属性改为 btnCar1 和 btnCar2。将大纲中的 Grid 控件的 Name 属性改为 LayoutRoot。

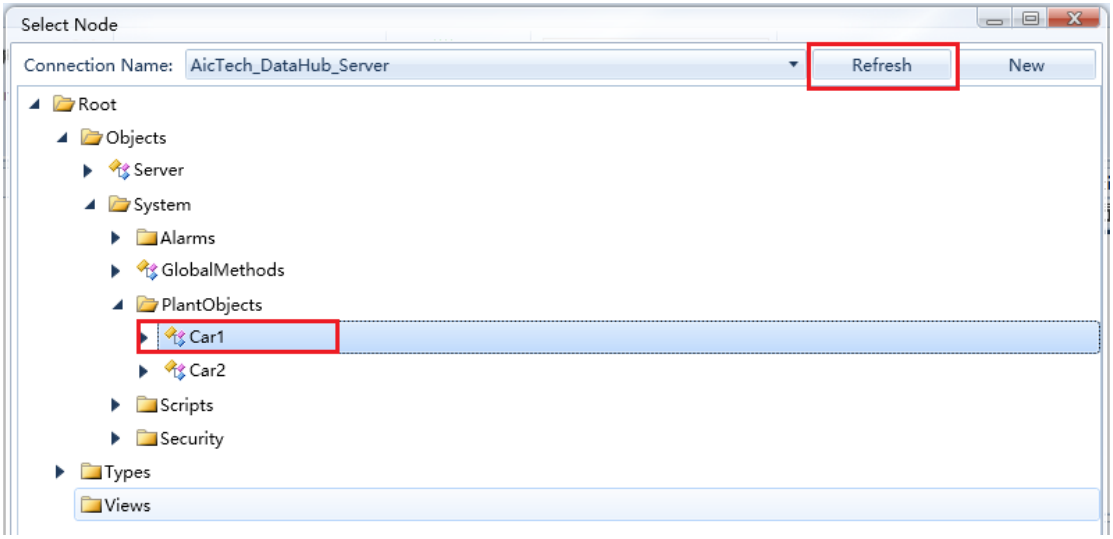
现在需要修改 HorizontalLinearGauge，VesselTank，RacingGauge 这三个控件的 Value 属性的数据绑定类型，先修改 VesselTank 的 Value 属性的数据绑定，如下图：数据绑定类型选择 OPCUA 相对绑定，然后应首先配置数据上下文（如果已经配置过就不需要再次配置），即图中的 Set Binding Context 按钮点击弹出的窗口配置如下图，

图中左窗口中选中的 LayoutRoot 是包含这一套控件的父级容器，右窗口选中我们已经建好的 DataHub 中的 TransportCar 这个类型，点击 OK。配置完 Set Binding Context 后点击刷新选择 MaterialLevel 属性的 Value 属性进行绑定，点击 OK。





同样我们完成 HorizontalLinearGauge 控件和 RacingGauge 控件的数据绑定，分别绑定到 Temperature 的 Value 属性和 Speed 的 Value 属性。至此，真正的数据还没法到达控件的属性上，参考 4.3.1 中为 btnCar1 和 btnCar2 两个按钮的 Click 事件分别添加 btnCar1_Click 和 btnCar2_Click 处理方法，参考 5.2 中分别在插入 Set Relative Binding Context 对应的代码段，配置如下图，点击刷新按钮，在 btnCar1_Click 中选择 Car1 对象，在 btnCar2_Click 中选择 Car2 对象。参考 4.2 中的 OPCUA 相对绑定，将 SetBindingContext 方法中的 this 参数改为 this.LayoutRoot。



至此，已经实现了 OPCUA 的相对绑定，运行点击 Car1 和 Car2 按钮就可以进行 Car1 实例和 Car2 实例监测数据的切换。