**WPF扩展库文档**

Ping9719.WpfEx

V 0.1.14

目录

[1. 开始使用 3](#_Toc15615)

[1.1. 安装 3](#_Toc10446)

[1.2. 开始 3](#_Toc12187)

[2. 转换器 4](#_Toc32059)

[2.1. Enum2BoolConverter（枚举转布尔） 4](#_Toc27703)

[2.2. Enum2IntConverter（枚举转数字） 4](#_Toc24030)

[2.3. Enum2StrConverter（枚举转字符串） 4](#_Toc14978)

[2.4. List2StrConverter（集合转字符串） 4](#_Toc1912)

[3. 控件 5](#_Toc20707)

[3.1. 原生控件 5](#_Toc6384)

[3.1.1. TabControl（选项卡） 5](#_Toc29775)

[3.1.2. TextBox（文本框） 6](#_Toc9802)

[3.2. 扩展控件 6](#_Toc26626)

[3.2.1. UserControlBase（用户控件扩展） 6](#_Toc28691)

[3.2.2. RunStateButton（运行状态按钮） 7](#_Toc13218)

[3.2.3. SyncProgressBar（同步进度条） 8](#_Toc19295)

[3.2.4. IotState（工业传感器状态控件） 9](#_Toc15319)

[3.2.5. IotUrn（工业气缸控件） 9](#_Toc26259)

[3.2.6. IotServo（工业伺服控件） 10](#_Toc18534)

[3.2.7. IotServo2（工业伺服控件） 11](#_Toc14727)

[3.2.8. IotDevice（工业设备控件） 11](#_Toc11336)

[4. 窗体 13](#_Toc9165)

[5. 绑定与命令 14](#_Toc10070)

[5.1. 绑定 14](#_Toc4640)

[5.2. VS编译器-代码片段 14](#_Toc5764)

[5.3. 命令 15](#_Toc3137)

[5.3.1. 不带参数命令 15](#_Toc28351)

[5.3.2. 带参数命令 16](#_Toc22225)

[6. 扩展 16](#_Toc8168)

[6.1. QueueTaskTime（简单定时任务队列执行器） 16](#_Toc28905)

[6.2. 将WPF控件保存为图片 18](#_Toc10375)

[7. 升级记录 19](#_Toc13180)

# 开始使用

## 安装

通过NuGet安装

|  |
| --- |
| Install-Package Ping9719.WpfEx |

## 开始

引用资源，在文件（App.xaml）中

|  |
| --- |
| <Application.Resources>  <ResourceDictionary>  <ResourceDictionary.MergedDictionaries>  <!--Ping9719.WpfEx-->  <ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Themes/Theme.xaml"/>  <!--HandyControl-->  <ResourceDictionary Source="pack://application:,,,/HandyControl;component/Themes/SkinDefault.xaml"/>  <ResourceDictionary Source="pack://application:,,,/HandyControl;component/Themes/Theme.xaml"/>  </ResourceDictionary.MergedDictionaries>  </ResourceDictionary>  </Application.Resources> |

名称空间

|  |
| --- |
| xmlns:pi="https://github.com/ping9719/wpfex" |

# 转换器

## Enum2BoolConverter（枚举转布尔）

将枚举转为布尔，一般用于单选框。

|  |
| --- |
| <RadioButton Content="男" IsChecked="{Binding Sex, Converter={StaticResource Enum2BoolConverter}, ConverterParameter='Man'}"/> |

## Enum2IntConverter（枚举转数字）

将枚举转为数字

## Enum2StrConverter（枚举转字符串）

将枚举转为字符串（支持Description特性）

## List2StrConverter（集合转字符串）

将数组、集合按照默认逗号分隔转为字符串。也可传入参数，按参数方式分割。

# 控件

## 原生控件

### TabControl（选项卡）

#### TabControlBaseStyle.MouseOver（基础悬停鼠标样式）



|  |
| --- |
| <TabControl Margin="10" Style="{StaticResource TabControlBaseStyle.MouseOver}">  <TabItem Header="选项卡1">  </TabItem>  <TabItem Header="选项卡2">  </TabItem>  <TabItem Header="选项卡3">  </TabItem>  </TabControl> |

#### TabControlInLine.MouseOver（线条铺满悬停鼠标样式）



|  |
| --- |
| <TabControl Margin="10" Style="{StaticResource TabControlInLine.MouseOver}">  <TabItem Header="选项卡1">  </TabItem>  <TabItem Header="选项卡2">  </TabItem>  <TabItem Header="选项卡3">  </TabItem>  </TabControl> |

#### TabControlRectangle（矩形样式）



|  |
| --- |
| <TabControl Margin="10" Style="{StaticResource TabControlRectangle}">  <TabItem Header="选项卡1">  </TabItem>  <TabItem Header="选项卡2">  </TabItem>  <TabItem Header="选项卡3">  </TabItem>  </TabControl> |

### TextBox（文本框）

#### TextBoxExtend.Multi（多行文本框样式）



|  |
| --- |
| <TextBox Margin="10,5" Style="{StaticResource TextBoxExtend.Multi}"></TextBox> |

## 扩展控件

### UserControlBase（用户控件扩展）

UserControlBase:UserControl

**属性**

|  |  |  |
| --- | --- | --- |
| IsLoadedVisible | bool | 是否已加载并显示界面 |
| IsInDesignMode | bool | 是否处于设计模式 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| LoadedVisibleFirst | 首次加载并显示控件时发生 |
| LoadedVisible | 加载并显示控件时发生 |

### RunStateButton（运行状态按钮）

用来启用，停止设备。

**基础功能只包含“启动”“停止”。**

**如需使用“暂停”“继续”，请使用样式“RunStateButtonPause”**



**使用**

|  |
| --- |
| <StackPanel>  <pi:RunStateButton Margin="3" RunState="Stop" IsVisRset="False"/>  <pi:RunStateButton Margin="3" RunState="Run" IsVisRset="True"/>  <pi:RunStateButton Margin="3" RunState="Stop" IsVisRset="False" Style="{StaticResource RunStateButtonPause}"/>  <pi:RunStateButton Margin="3" RunState="Run" IsVisRset="True" Style="{StaticResource RunStateButtonPause}"/>  <pi:RunStateButton Margin="3" FontSize="20" Padding="30,10" RunState="Stop" IsVisRset="True" Style="{StaticResource RunStateButtonPause}"/>  <pi:RunStateButton Margin="3" FontSize="20" Padding="30,10" RunState="Stop" IsVisRset="True" Style="{StaticResource RunStateButtonPause.NotHide}"/>  </StackPanel> |

**属性**

|  |  |  |
| --- | --- | --- |
| IsVisRset | bool | 是否显示复位按钮 |
| RunState | RunState | 运行状态（枚举值） |
| RunStateButtonClick | RunStateButtonClick | 最近一次单击的按钮 |
| StartButContent | object | 启动按钮内容 |
| PauseButContent | object | 暂停按钮内容 |
| ContinueButContent | object | 继续按钮内容 |
| StopButContent | object | 停止按钮内容 |
| RsetButContent | object | 复位按钮内容 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| Click | 单击按钮时。返回的OriginalSource参数为枚举值（RunStateButtonClick），也可以通过属性获取 |

### SyncProgressBar（同步进度条）



**使用**

|  |
| --- |
| //XAML  <StackPanel Width="200">  <pi:SyncProgressBar SyncState="OkVisible"/>  <pi:SyncProgressBar SyncState="SyncIn" QueueTask="{Binding QueueTask}"/>  <pi:SyncProgressBar SyncState="SyncErr"/>  <pi:SyncProgressBar SyncState="SyncIn" QueueTaskOkVisible="True" Value="30"/>  </StackPanel>  //ViewModel  public QueueTaskTime QueueTask { get; set; } = new QueueTaskTime(); |

**属性**

|  |  |  |
| --- | --- | --- |
| QueueTask | QueueTaskTime | 任务，添加的任务将会自动更新到界面，强烈推荐！参考[QueueTaskTime](#_QueueTaskTime（简单定时任务队列执行器）)的使用。 |
| InteriorHeight | double | 内部显示高度 |
| SyncState | SyncProgressBarState | 同步状态（枚举） |
| SyncTextOk | string | 同步成功描述文本 |
| SyncTextIn | string | 同步中描述文本 |
| SyncTextErr | string | 同步错误描述文本 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| \ |  |

### IotState（工业传感器状态控件）

如需加边框，请使用“BorderThickness”



**使用**

|  |
| --- |
| <StackPanel Width="150">  <pi:IotState Text="传感器1"/>  <pi:IotState Text="传感器1" IsOk="True" BorderThickness="1" HorizontalContentAlignment="Left" />  <pi:IotState Text="传感器1" IsOk="True" OkBrush="Red" BorderThickness="1"/>  <pi:IotState Text="传感器1" IsOk="True" InteriorHeight="10" />  </StackPanel> |

**属性**

|  |  |  |
| --- | --- | --- |
| Text | string | 文本 |
| IsOk | bool | 是否成功 |
| OkBrush | Brush | 成功状态下的颜色 |
| NotOkBrush | Brush | 不成功状态下的颜色 |
| InteriorHeight | double | 内部高度 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| Click | 单击图标时 |

### IotUrn（工业气缸控件）



**使用**

|  |
| --- |
| <pi:IotUrn Text="气缸1" IsButBadge1="True"></local:IotUrn> |

**属性**

|  |  |  |
| --- | --- | --- |
| Text | string | 文本 |
| IsButBadge1 | bool | 是否显示按钮1上面的标记 |
| IsButBadge2 | bool | 是否显示按钮2上面的标记 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| ButClick1 | 单击按钮1 |
| ButClick2 | 单击按钮2 |

### IotServo（工业伺服控件）



**使用**

|  |
| --- |
| <pi:IotServo Text="伺服1" x:Name="ser1"></pi:IotServo>  ser1.ModelSpeeds = new List<ServoSpeed>()  {  new ServoSpeed (){ Name="手动模式",Speed=10,},  new ServoSpeed (){ Name="自动模式",Speed=100,},  new ServoSpeed (){ Name="测试模式",Speed=15,},  }; |

**属性**

|  |  |  |
| --- | --- | --- |
| Text | string | 文本 |
| IsFold | bool | 是否折叠 |
| Location | double | 当前位置 |
| ModelSpeedHome | ServoSpeed | 主页显示的速度模式 |
| ModelSpeeds | List<ServoSpeed> | 全部的速度模式 |

**方法**

|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| LocationChange | 尝试改变伺服的位置时 |
| SpeedChange | 尝试改变伺服的速度时 |

### IotServo2（工业伺服控件）

此控件只支持2种速度模式，并且可以点击主页的模式可以切换，能满足大多数的场景。默认是：手动模式，自动模式。



**使用**

|  |
| --- |
| <pi:IotServo2 Text="伺服1"/> |

**属性**

|  |  |  |
| --- | --- | --- |
| Text | string | 文本 |
| IsFold | bool | 是否折叠 |
| Location | double | 当前位置 |
| IsVisSpeed1 | bool | 是否主页中显示速度1和模式，默认true |
| Speed1 | int | 速度1，默认手动速度 |
| Speed2 | int | 速度2，默认自动速度 |

**方法**

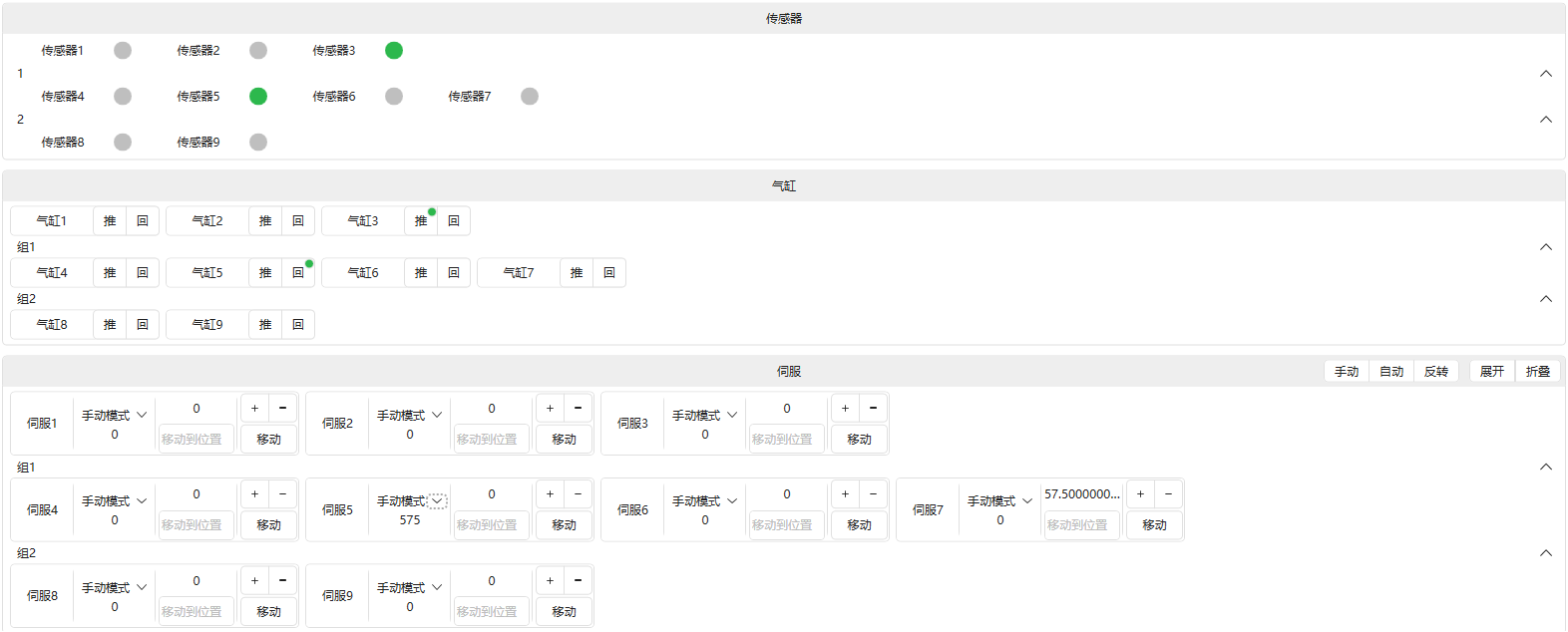
|  |  |
| --- | --- |
| \ |  |

**事件**

|  |  |
| --- | --- |
| LocationChange | 尝试改变伺服的位置时 |
| SpeedChange | 尝试改变伺服的速度时 |

### IotDevice（工业设备控件）

可以将工业控件按照一定规律分组进行排列，并可以进行折叠等操作。目前只支持DeviceStateData、DeviceStateSetData、DeviceUrnData、DeviceServo2Data。4个工业控件，以后会更多。



**使用**

|  |
| --- |
| <Border Style="{StaticResource BorderRegion}" Grid.Row="1" Margin="10,5" Padding="0">  <pi:IotDevice x:Name="dev" UrnClick="clike" ServoClick="clikeser"/>  </Border>  //使用LoadUi()方法加载一个或多个  List<DeviceStateData> deviceStateDatas = new List<DeviceStateData>()  {  new DeviceStateData (){ Name="传感器1",GroupName="",IsOk=false},  new DeviceStateData (){ Name="传感器4",GroupName="1",IsOk=false},  };  List<DeviceUrnData> deviceUrnDatas = new List<DeviceUrnData>()  {  new DeviceUrnData (){Name="气缸1",GroupName="" },  new DeviceUrnData (){Name="气缸5",GroupName="组1" },  };  List<DeviceServo2Data> deviceServoDatas = new List<DeviceServo2Data>()  {  new DeviceServo2Data (){Name="伺服1",GroupName="" },  new DeviceServo2Data (){Name="伺服4",GroupName="组1" },  };  dev.LoadUi(deviceStateDatas, deviceUrnDatas, deviceServoDatas);  //操作Ui直接改变模型就行了，使用的为双向绑定  deviceStateDatas[2].IsOk = !deviceStateDatas[2].IsOk;  deviceUrnDatas[2].IsGoTo = !deviceUrnDatas[2].IsGoTo;  deviceServoDatas[2].AutoSpeed++;  //事件包含的参数需要转为object[]使用  private void clike(object sender, RoutedEventArgs e)  {  var aaa = (object[])e.OriginalSource;  } |

**属性**

|  |  |  |
| --- | --- | --- |
| StateHeader | string | 传感器标题文本 |
| StateSetHeader | string | 传感器（可控制）标题文本 |
| UrnHeader | string | 气缸标题文本 |
| Servo2Header | string | 伺服标题文本 |

**方法**

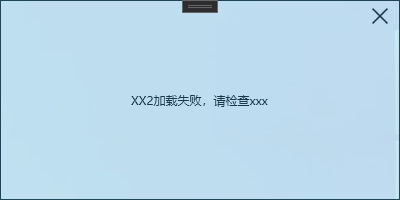
|  |  |
| --- | --- |
| LoadUi(IEnumerable<IDeviceDataBase> deviceDatas) | 加载Ui |
| LoadUi(params IEnumerable<IDeviceDataBase>[] deviceDatass) | 加载多个Ui |

**事件**

|  |  |
| --- | --- |
| StateSetClick | 可点击的状态点击。返回的OriginalSource参数为原绑定数据 |
| UrnClick | 气缸点击推或回。返回的OriginalSource参数为object[]，1为bool（true为推）；2为原绑定数据 |
| ServoClick | 伺服点击的操作。返回的OriginalSource参数为object[]，1为枚举；2为新数据；3为原绑定数据 |

# 窗体

## 入口页面加载等待窗体（MainLoadWindow）



**使用**

|  |
| --- |
| var isok = MainLoadWindow.Show(new Tuple<string, string, Action>("加载XX1中..", "XX1加载失败", () =>  {  }), new Tuple<string, string, Action>("加载XX2中..", "XX2加载失败，请检查xxx", () =>  {  var aaa = Convert.ToInt32("abc");  }), new Tuple<string, string, Action>("加载XX3中..", "XX3加载失败", () =>  {  }));  if (isok)  {  } |

# 绑定与命令

## 绑定

在XMAL中

|  |
| --- |
| <TextBlock Text="{Binding Var}"></TextBlock> |

在cs文件中

|  |
| --- |
| UcHomeVModel ucHomeVModel = new UcHomeVModel();  public UcHome()  {  InitializeComponent();  this.DataContext = ucHomeVModel;  } |

在ViewModel 文件夹中

|  |
| --- |
| using Ping9719.WpfEx.Mvvm;  public class UcHomeVModel : BindableBase  {  private string var;  public string Var { get => var; set { SetProperty(ref var, value); } }  } |

## VS编译器-代码片段



提供“VS编译器-代码片段”来提高工作效率，操作如下：

打开VS-->工具-->代码片段管理器-->Visual C#-->添加

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">  <CodeSnippet Format="1.0.0">  <Header>  <Title>propmvvm</Title>  <Shortcut>propmvvm</Shortcut>  <Description>属性和支持MVVM模式字段的代码片段</Description>  <Author>夏诗评</Author>  <SnippetTypes>  <SnippetType>Expansion</SnippetType>  </SnippetTypes>  </Header>  <Snippet>  <Declarations>  <Literal>  <ID>type</ID>  <ToolTip>属性类型</ToolTip>  <Default>int</Default>  </Literal>  <Literal>  <ID>property</ID>  <ToolTip>属性名</ToolTip>  <Default>MyProperty</Default>  </Literal>  <Literal>  <ID>field</ID>  <ToolTip>支持此属性的变量</ToolTip>  <Default>myVar</Default>  </Literal>  </Declarations>  <Code Language="csharp"><![CDATA[private $type$ $field$;  public $type$ $property$ { get => $field$; set { SetProperty(ref $field$, value); } }  $end$]]>  </Code>  </Snippet>  </CodeSnippet>  </CodeSnippets> |

## 命令

### 不带参数命令

在XMAL中

|  |
| --- |
| <Button Command="{Binding MyCommand}" CommandParameter="abc" Content="按钮"/> |

在ViewModel 文件夹中

|  |
| --- |
| public class MainWindowViewModel : BindableBase  {  public ICommand MyCommand { get => new DelegateCommand<string>(My); }  //执行方法  public void My(string obj)  {  //code  }  } |

### 带参数命令

在XMAL中

|  |
| --- |
| <Button Command="{Binding MyCommand}" Content="按钮"/> |

在ViewModel 文件夹中

|  |
| --- |
| public class MainWindowViewModel : BindableBase  {  public ICommand MyCommand { get => new DelegateCommand(My); }  //执行方法  public void My()  {  //code  }  } |

# 扩展

## QueueTaskTime（简单定时任务队列执行器）

任务队列管理

**属性**

|  |  |  |
| --- | --- | --- |
| State | QueueTaskTimeState | 运行状态枚举 |

**方法**

|  |  |
| --- | --- |
| Start | 启动 |
| Stop | 暂停 |
| AddForTask | 添加循环任务 |
| AddQueueTask | 添加队列任务 |

**事件**

|  |  |
| --- | --- |
| LoadedVisibleFirst | 首次加载并显示控件时发生 |
| LoadedVisible | 加载并显示控件时发生 |

## 将WPF控件保存为图片

\*发现特定情境下有bug，使用时做好测试。

|  |  |
| --- | --- |
| void WpfHelp.SaveToImg(this FrameworkElement visual, string fileName) | 保存为本地文件 |
| MemoryStream SaveToImg(this FrameworkElement visual) | 保存到内存流 |

# 升级记录

升级记录只记载影响版本兼容性的问题。

|  |  |
| --- | --- |
| 版本 | 记录 |
| 0.1.14 | SyncProgressBar.QueueTask改为可绑定的方式 |
| 0.1.0 | <ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Theme.xaml"/>  改为  <ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Themes/Theme.xaml"/> |