WPF 扩展库文档

Ping9719.WpfEx

V 0.1.0

目录

1. 开始使用

1.1. 安装

通过 NuGet 安装

```
Install-Package Ping9719.WpfEx
```

1.2. 开始

引用资源，在文件（App.xaml）中

```
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Themes/Theme.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
```

名称空间

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

2. 转换器

2.1. 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'}"/>
```

2.2. Enum2IntConverter（枚举转数字）

将枚举转为数字

2.3. Enum2StrConverter（枚举转字符串）

将枚举转为字符串（支持 Description 特性）

2.4. List2StrConverter（集合转字符串）

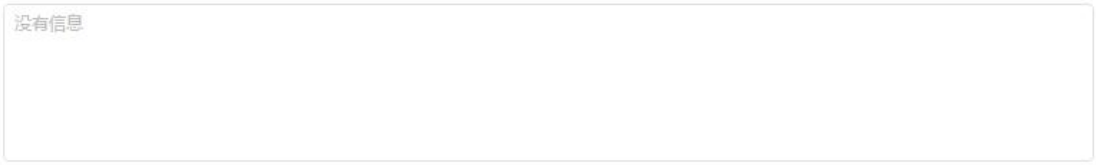
将数组、集合按照默认逗号分隔转为字符串。也可传入参数，按参数方式分割。

3. 控件

3.1. 原生控件

3.1.1. TextBox（文本框）

多行文本框样式




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

3.2. 扩展控件

3.2.1. UserControlBase（用户控件扩展）

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

3.2.2. IotState（工业传感器状态控件）

使用

```
<pi:IotState Text="传感器 1"></local:IotState>
```

属性

Text	string	文本
IsOk	bool	是否成功

方法

\	

事件

Click	单击图标时

3.2.3. 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

3.2.4. 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

3.2.5. 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

3.2.6. 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、DeviceUrnData、DeviceServo2Data。3 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

\		

方法

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

事件

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

4. 窗体

无

5. 绑定与命令

5.1. 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 { return var; }
            set { SetProperty(ref var, value); }
        }
    }
```

5.2. 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 { return $field$; }
        set { SetProperty(ref $field$, value); }
    }
    $end$]]>
            </Code>
        </Snippet>
    </CodeSnippet>
</CodeSnippets>
```

5.3. 命令

5.3.1.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
        }
    }
```

5.3.2.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
        }
    }
```

6. 其他

6.1. 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

7. 升级记录

版本	记录
0.1.0	<ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Theme.xaml"/> 改为 <ResourceDictionary Source="pack://application:,,,/Ping9719.WpfEx;component/Themes/Theme.xaml"/>