组件开发规范文档

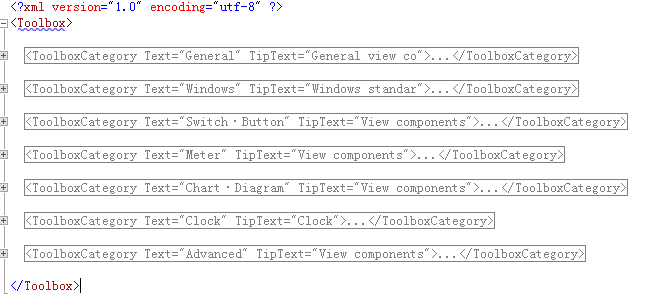
1. 支持的控件类型。

易控天地支持.Net平台开发的所有的程序，对于其他方式开发的程序（例如ActiveX控件，C/C++等），需要首先包装成.Net程序。对于.Net开发的控件的界面可以采用Winform或WPF进行呈现，其中WPF部分支持常规的界面开发，和MVVM方法开发。

1. 控件注册到易控天地中。

易控天地对控件采用开放的、松散的管理方式。控件的注册只需要将配置信息添加到修改工具箱配置文件Toolbox.config（中文版本的配置文件是Toolbox.zh-CN.config）即可。同时将控件所需要的所有的程序程序集拷贝到易控的安装目录。

ToolBox.config文件结构分析。文件结构如下图：



* + - * 1. 工具箱分类：上图中的一个ToolboxCategory是一个工具箱中的分类，Text属性是显示的文本，TipText是分类的提示信息（暂时不可用）。
        2. 控件配置：每一个ToolboxCategory下面是该工具分类下的控件的配置信息。控件的类型分成2中，一种是易控天地内部用的插件，另一种是控件。控件根据控件界面采用的开发技术不同，又分成Winform、WPF两种类型的控件。

Winform 控件配置信息。

<ToolboxItem Text="Keypad" TipText=" " ShapeAssembly="CE.NX.VWC" ShapeType="ControlEase.Inspec.ViewCore.CtrlSurrogate">

<InnerControl ControlAssembly="CE.NX.VWPS" ControlType="ControlEase.Inspec.ViewPluginShapes.KeyboardUserControl"/>

</ToolboxItem>

<ToolboxItem /> 的Text属性是控件在工具箱中的显示的文本信息，TipText是控件的提示信息。对于所有的Winform控件，<ToolboxItem />部分的ShapeAssembly，ShapeType属性都是不变的。<InnerControl />的属性ControlAssembly是控件的所有程序集的名称，ControlType是控件的类的全名称。

WPF控件配置信息。

<ToolboxItem Text="MsgBrowser" TipText=" " ShapeAssembly="CE.NX.VWC" ShapeType="ControlEase.Inspec.ViewCore.WpfSurrogate">

<InnerControl ControlAssembly="CE.NX.AEC" ControlType="ControlEase.Inspec.MessageControl.MessageControlViewModel" />

</ToolboxItem>

具体配置的含义参考Winform控件部分。不同不同的地方在于ShapeAssembly和ShapeType部分。

指定控件图标。在上面的配置信息中，没有配置控件所是有的图标，如果控件不指定图标，则采用默认的图标。如果需要指定自己的图标，需要在控件的入口类(InnerControl 的ControlType指定的类)的定义上加上[Icon]标注。要使用Icon标注，需要引用CE.AI.VWF.dll。

1. 控件基本功能开发。

我们可以使用Winform或Wpf开发控件，控件可以继承自任何其他的类。控件和画面的交互是通过一系列接口来实现的。

* 1. 控件数据的保存和加载。

控件序列化是通过控件实现接口ICustomSerializable来实现的，接口的定义如下：

public interface ICustomSerializable

{

bool IsFullCustomSerialize { get; }

void Decode(System.Xml.Linq.XElement element);

void Encode(System.Xml.Linq.XElement element);

}

1. 属性IsFullCustomSerialize 是用于指示控件是需要实现全部序列化还是实现部分。如果返回False，则表示需要画面自动序列化所有属性，返回True，表示所有序列化操作，在控件内部使用。如果控件没有实现此接口，和实现了此接口但是返回False效果一样，画面执行自动序列化，自动序列化只能针对基础类型的属性。在控件开发时推荐实现此接口，并IsFullCustomSerialize属性返回True。
2. 方法void Encode(System.Xml.Linq.XElement element)。画面在保存时，会调用此方法对控件的属性进行序列化。
3. 方法void Decode(System.Xml.Linq.XElement element)。画面加载时，调用此方式对控件属性进行反序列化。
   1. 控件对外暴露属性、方法。

控件中并不是所有的标明Public 的属性，在属性编辑器中都可见，所有标明Public 的属性和方法在脚本中可见（在编写脚本时不可见，但不代表这个方法不可访问）。

1. 属性在属性编辑器中可见。要想让属性在属性编辑器中可见，除了这个属性要标明Public外，还要对属性添加一个[Displayable (true)]标注。其他相关标注如下：

[Category("")]

[DisplayName("")]

[Description("")]

[Displayable(true)]

[Editor(typeof(Editor), typeof(UITypeEditor))]

[TypeConverter(typeof(Convert))]

因为易控天地是支持多语言的，在属性编辑器中显示的属性的名称的时候，易控天地会自动找该名称在控件的资源中对应的值。同时属性的描述信息，会在资源中自动找“属性名\_Description”的值用于属性的描述。如果不想用这种规则，可以手工的添加[DisplayName],[Description]这些标注。[Category()]用于指明属性在属性编辑器中所有的分类。[TypeConverter(typeof(Convert))]用于配置属性的值怎么在属性编辑器中显示，[Editor(typeof(Editor), typeof(UITypeEditor))]用于配置属性的值怎么编辑。具体的详细信息可以参考MSDN。

1. 属性在脚本编辑器中可见。

如果想让某个属性在脚本编辑器中可见，控件首先需要引用CE.AI.VWF.dll程序集，然后再属性上添加[ScriptProperty]标注。

1. 函数在脚本编辑器中可见。

如果想让某个属性在脚本编辑器中可见，控件首先需要引用CE.AI.VWF.dll程序集，然后再属性上添加[ScriptMethod]标注。

* 1. 自定义控件名称。

在画面中拖动创建一个控件时，默认会采用配种文件的中Text属性指定的名称，再加上一个序号用作控件对象的名称。如果想改变这个规则，可以通过实现IPluginCustomName接口来指定默认创建的名称的前缀。接口的定义如下：

public interface IPluginCustomName

{

string DefaultName { get; }

}

1. 控件高级功能开发

在控件中，可以使用易控天地中的变量，以及实现C#脚本功能。易控天地中所有的文本、图片等信息都在资源中统一管理，在控件中涉及这部分的属性，也要遵守这一规则。

* 1. 控件中使用变量以及C#脚本。

如果在控件中使用了变量表达式，脚本执行程序等控件需要通过实现IPropertyAnimateCompatible 接口告诉画面，该控件都用到了那些表达式以及脚本程序。IPropertyAnimateCompatible 接口定义如下：

public interface IPropertyAnimateCompatible

{

List<IPropertyAnimate> ListAnimate();

}

该接口定义在CE.NX.VWC程序集中。在编译、画面导入\导出、画面模板编译等时候，画面检索所有用用到的控件，并调用该方法。

1. 动画类型的属性。如果你有一个属性，需要通过变量表达式来驱动，可以定义一个PropertyLinkAnimate<T>类型的属性，在运行时只要响应该类型的ValueChanged事件，给相应的属性赋值即可。在开发时，会自动调用易控内置的变量表达式编辑器进行编辑。
2. 脚本类型的属性。脚本类型的属性，包括2种情况，一种是直接简单的只关联一个变量（Tag或IOTag），运行时将属性的值写入到变量。另一种是直接写一段脚本，运行时执行那一段脚本。对于第一种情况，实现方式也是编译成一段脚本进行处理。对于这两种情况，需要定义一个PropertyLinkEvent<T>类型的属性。对于第一种情况，运行时调用属性的SetValue下发值。对于第二种情况，定义一个PropertyLinkEvent<T>属性，同时设置这个这个属性的IsCodeOnly属性为True，同时指定这个属性的Editor为[Editor("ControlEase.Inspec.ViewPresentation.PropertyLinkScriptEditor,CE.NX.VWP", typeof(UITypeEditor))]，运行时通过PropertyLinkEvent<T>的方法SetValue()方法执行函数。
3. 不通过PropertyLinkAnimate<T>，PropertyLinkEvent<T>，通过变量的操作接口同样也可以获取变量，设置的变量的值，但是没有办法执行脚本程序，同时在属性编辑器中对这些属性的属性编辑器需要自己处理。变量操作的接口：ITagProvider，需要添加对CE.AI.TGF的引用。这个接口中提供了获取变量，设置变量的值等函数。获取这个接口通过ServiceLocator.Current.GetService<ITagProvider>()接口实现，同时需要添加对程序集CE.Nexus.ComponentModel的引用。
   1. 控件中使用资源（文本，图片，文件）。

易控天地中图片，文本，文件等是放在资源中统一处理的，所有我们在做控件时，也需要遵守这一规则。控件需要实现接口IResourceLinks，接口定义如下：

public interface IResourceLinks

{

/// <summary>

/// 获取资源引用信息

/// </summary>

/// <returns></returns>

List<ResourceRefernceItem> GetResourceId ( );

}

下面是一个例子：

/// <summary>

///

/// </summary>

/// <returns></returns>

public System.Collections.Generic.List<ResourceRefernceItem> GetResourceId()

{

var vv = new List<ResourceRefernceItem>();

if (!string.IsNullOrEmpty(mUrl))

{

vv.Add(new ResourceRefernceItem()

{

ResId = mUrl,

Type = ResourceTypes.File,

Value = mUrl,

UpdateResourceDelegate = new Func<string, bool>((name) =>

{

this.Url = name;

return true;

}

)

});

}

return vv;

}

如上面所示，返回的每一个ResourceRefernceItem，需要指定资源Id：ResId,资源的类型：Type，资源的当前值：Value以及更新这个资源ID的一个代理函数。

为了方便在资源的开发，对于所有引用了资源的属性，可以将属性定义成一下类型，这些类型自动内置了相应的属性编辑器，方便处理。

1. 文本类型：TextResourceItem。事例如下：

public TextResourceItem TextContent

{

get

{

if (mResourceItem == null)

{

mResourceItem = new TextResourceItem() { ResId = mModel.Text, Text = mModel.TextContent };

}

else

{

mResourceItem.ResId = mModel.Text;

mResourceItem.Text = mModel.TextContent;

}

return mResourceItem;

}

set

{

mResourceItem = value;

if (value != null)

{

//获取一个可用的资源ID

value.GetNewResourceId(mModel.TextContent, mModel.Text);

mModel.TextContent = value.Text;

mModel.Text = value.ResId;

}

else

{

mModel.TextContent = string.Empty;

mModel.Text = string.Empty;

}

OnPropertyChanged("TextContent");

}

}

1. 图片类型：ImageResourceItem。事例如下：

public ImageResourceItem ImageAdaptor

{

get

{

if (mImageAdaptor == null)

{

mImageAdaptor = new ImageResourceItem() { ImageId = mModel.Image };

}

if (mModel.InnerImage != null)

{

mImageAdaptor.Image = mModel.InnerImage.Content;

}

return mImageAdaptor;

}

set

{

var mg = new Image();

string oldName = mModel.Image;

if (value != null)

{

mModel.Image = value.ImageId;

if (value.Image == null && !string.IsNullOrEmpty(mModel.Image))

{

string stmp;

mg.Content = ImageCachManager.Manager.GetBitmapWithoutCach(mModel.Image, true, out stmp);

}

else

{

mg.Content = value.Image;

}

mg.Size = value.Image.Size;

Model.InnerImage = mg;

}

else

{

Model.InnerImage = null;

mModel.Image = null;

}

mImageAdaptor = value;

OnPropertyChanged("Image");

}

}

* 1. 控件支持运行时多语言。易控天地在运行时，是支持语言切换的，所有控件要实现相应的接口IRuntimeResource。接口定义如下：

public interface IRuntimeResource

{

bool SupportRuntimeChanged { get; }

void ResourceUpdate(IViewResourceService service);

}

在用户切换当前的语言环境时，画面会调用每个控件的ResourceUpdate函数，SupportRuntimeChanged 主要用于表示该控件支不支持运行时语言切换。

* 1. 控件和安全的结合。如果想要在控件中和易控天地的安全子系统进行结合，你需要通过ISecurityService，ISecurityProvider接口进行交互，使用该接口需要引用程序集CE.AI.SRF。开发时可以通过ISecurityProvider的ConfigPermissions弹出安全配置界面获取安全配置，这种方式需要自己做Editor和Converter在属性编辑器显示和编辑。你也使用系统内置的安全的Editor和Converter，你需要定义一个Tuple<string,List<string>,int> 类型的变量，同时标注

[TypeConverter("ControlEase.Inspec.ViewPresentation.PermissionConverter,CE.NX.VWP")]

[Editor ( "ControlEase.Inspec.ViewPresentation.PermissionEditor,CE.NX.VWP", typeof ( UITypeEditor ) )]；Tuple<string,List<string>,int>三部分的含义分别是服务器名称，权限列表，权限操作模式。

运行时可以通过接口ISecurityService的Validate方法进行校验。ISecurityService接口有个UserSwitched事件，表示当前用户改变，我们可以响应改事件，进行安全验证。

1. 程序集引用列表。
2. CE.AI.VWF
3. CE.NX.VWC
4. CE.AI.SPF
5. CE.NX.RSD
6. CE.AI.TGF
7. CE.AI.SRF