# WPF控件集成规范

由于画面系统使用的是基于GDI+的WinForm图形系统，但是由于现在各个子系统中使用WPF控件的情况广泛存在，因此，在现有画面系统上面支持WPF控件的需求就出现了，现在已经设计了WPF控件代理，用于在现有画面系统上面支持WPF控件，本文为大家介绍了WPF控件如何集成到画面系统中。

## WPF控件代理介绍

WPF控件代理是现有画面系统中为了承载WPF控件而设计的一种控件，对WPF控件的设计是为了让程序员能够更好的把精力集中在制作WPF控件的业务逻辑中，不用更多地关心如何与画面交互。

有关更多的WPF控件代理的设计，请参考[WPF控件代理设计框架图](http://rdsvr/sites/INSPEC_FX/Design%20Doc/Inspec%20FX/View/WPF_Surrogate.vsd)。

## 获取最新INSPECFX应用程序环境

首先，如果想使用画面上的WPF控件集成功能，必须获取最新的INSPECFX应用程序环境，此应用程序环境可以在[\\rdsvr\InspecFX\Output](file:///\\\\rdsvr\\InspecFX\\Output)中获取，有问题联系施成。

## WPF控件程序集位置

现在的画面集成WPF控件，支持两种WPF控件程序集的部署方式，画面内部程序及方式，以及单独程序及方式，下面对两种方式进行介绍。

### 画面内部程序集方式

所谓画面内部程序集方式，也就是将WPF控件的代码放置在画面内部的程序集中，比如GraphicsPresentation或者Drawing程序集中。

这种方式的优点是加载速度快，因为在画面内部的程序集中，所以省去了加载程序集的时间，缺点是收到了程序集依赖关系的限制，因为画面程序集，无法引用某些程序集，比如现阶段，画面没有引用消息与历史，因此，可能造成一些问题。

### 单独程序集方式

这种方式要求程序员将制作的WPF控件单独封装成为一个程序集，放置到Packages\View下，并将附带的引用程序集一并放入此文件夹（除AI级别与INSPEC级别的），这样能够解决引用问题。

注：建议以前做到INSPEC级别程序集中的WPF控件或者控件对应的ViewModel都单独封装到一个程序集中，并将此程序集放到Packages\View文件夹下面，这样能够解决程序集引用的问题。切勿将INSPEC层级的程序集直接放到Packages\View文件夹下面。

### 配置文件

第二部完成之后，就需要修改画面的Toolbox.config文件了，在其中添加你的WPF控件节点，让画面的工具箱能够识别你的WPF控件。

现在对于WPF控件的配置文件节点有两种配置方式，一种是普通WPF控件，另一种是支持MVVM的WPF控件配置方式。

<ToolboxItem Text="MockWpfControl" TipText="Create a Mock Wpf Control" ShapeAssembly="ControlEase.Inspec.Drawing" ShapeType="ControlEase.Inspec.Drawing.WpfSurrogate">

<InnerControl ControlAssembly="ControlEase.Inspec.Drawing" ControlType="ControlEase.Inspec.Drawing.TestWpfControl"/>

</ToolboxItem>

非MVVM方式WPF控件配置节点

<ToolboxItem Text="MockWpfDataContextControl" TipText="Create a Mock MVVM Wpf Control" ShapeAssembly="ControlEase.Inspec.Drawing" ShapeType="ControlEase.Inspec.Drawing.WpfSurrogate">

<InnerControl ControlAssembly="ControlEase.Inspec.Drawing"

ControlType="ControlEase.Inspec.Drawing.TestWpfDataContextControl"/>

<DataContext Assembly="ControlEase.Inspec.Drawing"

Type="ControlEase.Inspec.Drawing.TestWpfDataContextControlViewModel"/>

</ToolboxItem>

MVVM方式WPF控件配置节点

对比两种方式，其实就相差了一个DataContext节点，下面我们对每一个节点进行介绍。

#### ToolboxItem节点

ToolboxItem： 标示工具箱上面的工具项；

Text： 标示ToolboxItem的显示文本；

TipText： 标示ToolboxItem的气泡文本；

ShapeAssembly： 标示该工具项对应的图形所在的程序集（固定）；

ShapeType： 标示该工具项对应的图形对应的类的全名称（固定），WpfSurrogate表示此工具项使用WPF控件；

#### InnerControl节点

InnerControl： 标示WPF控件的节点；

ControlAssembly： 标示WPF控件所在的程序集；

ControlType： 标示WPF控件所对应的类的全名；

#### DataContext节点

DataContext： 标示WPF控件所对应的DataContext，用于为WPF控件使用MVVM设计模式（如果不是用MVVM模式，不用添加此节点）；

Assembly： 标示WPF控件对应的ViewModel所在的程序集；

Type： 标示WPF控件对应的ViewModel所对应的类的全名，ViewModel的要求是必须具有无参数的构造方法；

### 开发环境在PropertyGrid中显示属性

如果希望你的WPF控件或者WPF控件对应的ViewModel的属性在画面的PropertyGrid上显示，需要在该属性上标记DisplayableAttribute，该熟悉定义在Drawing中，因此，程序员制作的WPF控件需要引用Inspec.Drawing，程序集，如果认为DisplayableAttribute定义的位置不合理，可以修改，不过，这么设计的原因是因为画面不存在AI层级，因此就将Displayable定义在Inspec.Drawing中。

### 对WPF控件属性的保存

对于WPF控件或者WPF对应的ViewModel的属性保存，需要制作该控件或者ViewModel的Serializer，然后进行相应的标记，由于对于Serializer的描述已经介绍过很多次了，所以，在这里就不再介绍，有问题的请咨询黄硕。

## 支持非画面文件夹部署

在前一版本中，各子系统制作的WPF控件只允许部署在Packages\View（画面子系统文件夹）中。但是考虑到耦合性与各个子系统独立运行的原因，因此，需要支持子系统的控件部署在该子系统的文件夹中。

### 程序集部署

如果子系统的WPF控件是单独一个程序集，可以将该程序集放置在自己的子系统文件夹中，然后修改配置文件，配置文件格式将在下面介绍。

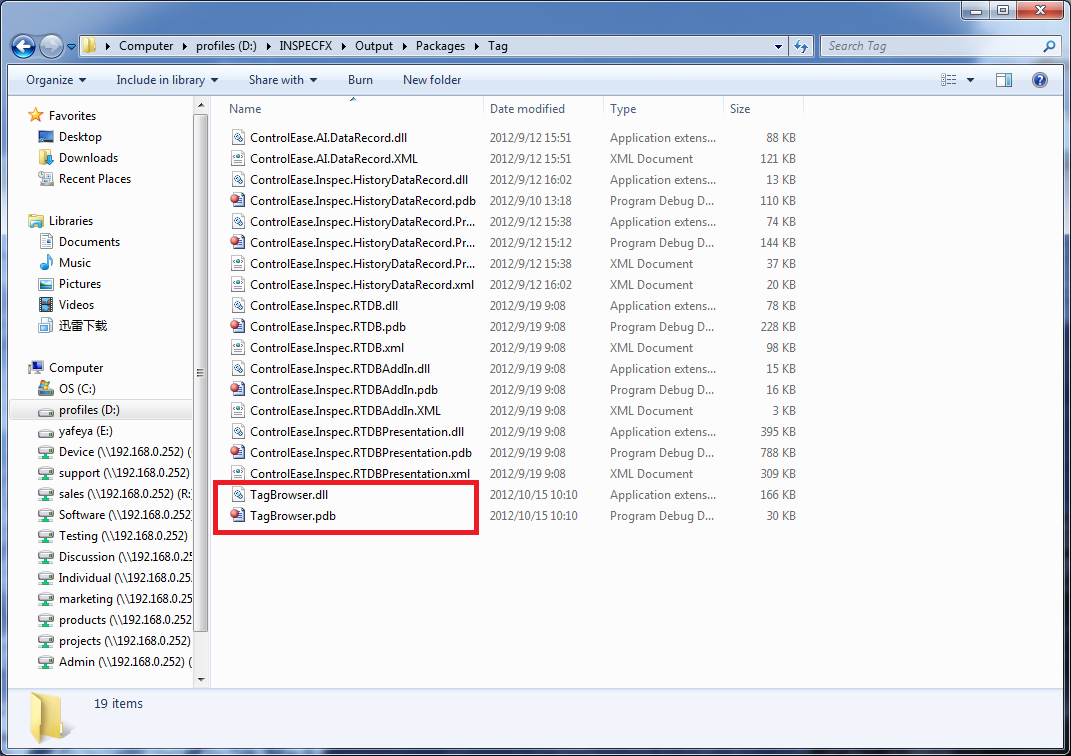


图1 WPF控件程序集部署

如图中所示，TagBrowser.dll为为了测试制作的WPF控件，此控件引用了INSPEC级别的Tag程序集，并在其中调用了一些Tag的INSPEC的逻辑，制作完成之后，不输在了Tag的文件夹中。

### 配置文件

在配置文件中需要添加程序集所在的文件夹信息，在配置文件节点中，使用Directory标示文件夹信息。

现阶段只支持将程序集文件部署在子系统的文件夹中，如下面节点中所示，” Directory=Packages\Tag”。

<ToolboxItem Text="MockTagBrowser" TipText="Create a Tag Browser"

ShapeAssembly="ControlEase.Inspec.Drawing" ShapeType="ControlEase.Inspec.Drawing.WpfSurrogate">

<InnerControl ControlAssembly="TagBrowser" ControlType="TagBrowser.TagViewer"

Directory="Packages\Tag"/>

<DataContext Assembly="TagBrowser" Type="TagBrowser.TagBrowserViewModel"

Directory="Packages\Tag"/>

</ToolboxItem>

TagBrowser部署配置节点