AVEVA .Net 菜单自定义

AVEVA .Net Command and Menu Customisation

eryar@163.*com*

摘要 ***Abstract***：以一个具体实例详细介绍 ***AVEVA .Net*** 中自定义菜单的原理及方法。掌握了自定义菜单的方法就可以对二次开发的程序进行分类和整理，便于用户使用。

关键字 ***Key words***：***AVEVA .Net***、***Addin Commands***、***Menu Customisation***、PDMS、二次开发

一、概述 ***Introduction***

通过《***AVEVA .Net Quick Start Guide***》这篇文章，大家对 ***AVEVA .Net*** 的二次开发有了个大概认识，但是这样并不能编写出实际有用的二次开发插件（***Addin***）。本文再深入一步，介绍自定义菜单方法，在菜单中调用二次开发的插件，便于用户操作。

用户通过菜单、右键菜单命令、工具条来调用开发的插件。通用程序框架 ***CAF***（***The Common Application Framework***）通过类 ***CommandBarManager*** 来提供了创建菜单、工具条等的接口。也提供了与这些工具交互事件的接口，这种方法 ***CAF*** 虽然支持但不推荐。

通用程序框架 ***CAF*** 还支持通过 ***UIC***（用户界面自定义文件 ***User Interface Customisation***）文件来自定义菜单、工具条等，***UIC*** 文件是 ***XML*** 格式的。

基于 ***XML*** 定义的菜单及工具条需要有个机制将用户界面实体与需要调用的相关功能关联上。为了实现这人目的，插件（***Addins***）功能通过一些命令对象（***command objects***）暴露出来。插件通过类 ***CommandManager*** 来加载这些命令对象。***XML*** 定义中可以包含用户界面实体如菜单入口、工具条的按钮及与其相关的命令对象。选择相应的菜单或点击工具条上的按钮都可以导致相关联的命令被执行。

基于命令模型的一个好处就是将用户界面或显示层与逻辑层解耦。程序逻辑层就与显示层没有什么直接关系了。若程序的状态需要对用户界面上的操作做出反映，程序只需要修改命令的状态即可。命令对象知道它与哪个用户界面实体关联，所以确保其内部状态反映相应的用户界面。这对一些状态是很简单的，如“***enabled***”、“***visible***、“***checked***”；但对动态程序状态就有点复杂了，如“***combo-box***”。

二、编写命令类 ***Writing a Command Class***

如下所示的代码是一个简单的命令类示例，用来管理一个可停靠窗口的可见性，代码中是控制 ***AttributeBrowser*** 可停靠窗口。通用程序框架 ***CAF*** 提供了一个抽象的命令类 ***Command***，所有的命令类都必须派生自 ***Command*** 类。在其构造函数中应该设置属性 ***Key***，在 ***UIC*** 文件中是通过 ***Key*** 这个属性来找到相应的命令，后面将会以一个实例来说明。

```
using System;
using System.Collections.Generic;
using System.Text;
using Aveva.ApplicationFramework.Presentation;

namespace Aveva.Presentation.AttributeBrowserAddin
{
    /// <summary>
    /// Class to manage the visibility state of the AttributeBrowser docked window
    /// This command should be associated with a StateButtonTool.
    /// </summary>
    public class ShowAttributeBrowserCommand : Command
    {
        private DockedWindow _window;
        /// <summary>
        /// Constructor for ShowAttributeBrowserCommand
        /// </summary>
        /// <param name="window">The docked window whose visibilty state will be managed.</param>
        public ShowAttributeBrowserCommand(DockedWindow window)
        {
            // Set the command key
            this.Key = "Aveva.ShowAttributeBrowserCommand";
            // Save the docked window
            _window = window;
            // Create an event handler for the window closed event
            _window.Closed += new EventHandler(_window_Closed);
            // Create an event handler for the WindowLayoutLoaded event
            WindowManager.Instance.WindowLayoutLoaded                    +=                    new
EventHandler(Instance_WindowLayoutLoaded);
        }

        void Instance_WindowLayoutLoaded(object sender, EventArgs e)
        {
            // Update the command state to match initial window visibility
            this.Checked = _window.Visible;
        }

        void _window_Closed(object sender, EventArgs e)
        {
```

```
            // Update the command state when the window is closed
            this.Checked = false;
        }

        /// <summary>
        /// Override the base class Execute method to show and hide the window
        /// </summary>
        public override void Execute()
        {
            if (this.Checked)
            {
                _window.Show();
            }
            else
            {
                _window.Hide();
            }
        }
    }
}
```

抽象基类 ***Command*** 提供了以下方法，其派生类中可以重载：

- ***void Execute()***：必须被重载，在执行命令时调用这个函数；
- ***CommandState GetState()***：这个函数由通用程序框架 ***CAF*** 调用，以便更新用户界面或上下文菜单。返回值是一个 ***CommandState*** 枚举值，反应了命令的状态。枚举值用来位来表示命令状态，这些状态可以使用位的或 ***OR*** 操作。
- ***String Description***：命令的描述；
- ***void Refresh(string context)***：当 ***CommandManager.ApplicationContext*** 的属性更改时都会调用这个方法。这也就给了一个更新其 ***Enabled*** 或 ***visible*** 状态的机会。

抽象基类 ***Command*** 也有一些属性可以用来更新用户界面上的状态：

- ***bool Checked***：若命令与像 ***StateButtonTool*** 这样的用户界面关联，则此属性值与用户界面上的状态会同步；
- ***bool Enabled***：此属性值的改变将会影响所有与其关联的用户界面；
- ***ArrayList List***：此属性允许命令可以与有字符列表的用户界面交流。如 ***ComboBoxTool***；
- ***int SelectedIndex***：此属性表示被用记选中的列表中的索引值；
- ***object Value***：此属性保存了当前关联的用户界面实体；
- ***bool ValueChanged***：通用程序框架 ***CAF*** 在调用命令类的 ***Execute*** 方法之前，若用户界面的值改变时会设置此属性的值。当命令执行结束后此值会被清除；
- ***bool Visible***：此属性值的改变将会影响所有与其关联的用户界面；

在通用程序框架 ***CAF*** 中注册一个命令是通过向 ***CommanManagers.Commands*** 的集合中添加一个命令实例来实现的。程序代码如下所示：

```
// Create and register addins commands
```

```
// Get the CommandManager
CommandManager commandManager = (CommandManager)serviceManager.GetService(typeof(CommandManager));
ShowAttributeBrowserCommand showCommand = new ShowAttributeBrowserCommand(attributeListWindow);
commandManager.Commands.Add(showCommand);
```

三、命令事件 ***Command Events***

命令基类 ***Command*** 提供了两个事件 ***BeforeCommandExecute*** 和 ***CommandExecuted*** 给程序员，用来响应命令的执行，即在命令执行前后做些处理。示例代码如下：

```
Command anotherCommand = commandManager.Commands["KeyOfCommand"];
anotherCommand.BeforeCommandExecute += new
System.ComponentModel.CancelEventHandler(anotherCommand_BeforeCommandExecute);
anotherCommand.CommandExecuted += new
EventHandler(anotherCommand_CommandExecuted);

//The BeforeCommandExecute event handler is of type
//CancelEventHandler and is passed a CancelEventArgs object
//which enables the command execution to be cancelled by setting
//the Cancel property to true.
void anotherCommand_BeforeCommandExecute(object sender,
System.ComponentModel.CancelEventArgs e)
{
    e.Cancel = true;
}
```

四、菜单自定义 ***Menu Customisation***

用户使用插件的方式通常都是通过使用菜单或命令栏上的按钮。如前文所述，通用程序框架 ***CAF*** 提供了自定义菜单和命令栏的机制，即通过配置“***User Interface Customisation***”（***UIC***）文件。下面将会对 ***UIC*** 文件的细节进行描述，如怎样配置通用程序框架 ***CAF*** 来加载 ***UIC*** 文件，通过用户交互的自定义工具来编辑 ***UIC*** 文件等等。

五、配置要加载 ***UIC*** 文件的模块 ***Configuring a Module to Load a UIC File***

每个基于通用程序框架 ***CAF*** 的插件都有一个 ***XML*** 格式的配置文件，包含了程序启动时需要加载的一系列 ***UIC*** 文件。此文件的默认路径就是安装目录，文件名的命令方式为<模块名>***Customisation.xml***。例如 ***AVEVA Marine*** 模块 ***Hull Design*** 就有一个名为 ***HullDesignCustomisation.xml*** 的配置文件，文件内容如下所示。默认情况下 ***UIC*** 文件也希望在相同的目录下。标专“$1”表示 ***UIC*** 文件的路径将会由当前工程名代替。

```
<?xml version="1.0" encoding="utf-8"?>
<UICustomizationSet                                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <UICustomizationFiles>
    <CustomizationFile Name="Hull General" Path="AVEVA.Marine.UI.HullGeneral.uic" />
    <CustomizationFile Name="Hull Design" Path="AVEVA.Marine.UI.HullDesign.uic" />
    <CustomizationFile Name="Project" Path="$1.uic" />
    <CustomizationFile Name="StatusControllerAddin" Path="StatusController.uic" />
  </UICustomizationFiles>
</UICustomizationSet>
```

配置文件中 ***UIC*** 文件的顺序是很重要的，它表示了程序加载这些自定义界面的顺序。一个新的 ***UIC*** 文件可以加载到一个模块中，通过在其配置文件中简单添加一行即可。

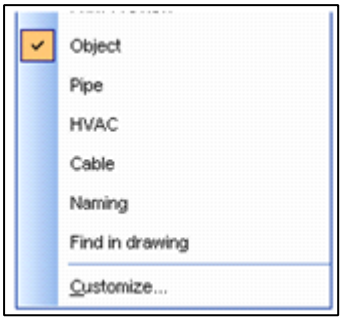
与在模块的配置文件中添加 ***UIC*** 文件的功能类似的方法是在程序中编码实现对 ***UIC*** 文件的添加，通过 ***CommandBarManager*** 的方法 ***AddUICustomisationFile*** 来编码实现 ***UIC*** 文件的添加。程序代码如下所示：

```
// Load a UIC file for the AttributeBrowser.
CommandBarManager                          commandBarManager                          =
(CommandBarManager)serviceManager.GetService(typeof(CommandBarManager));
    commandBarManager.AddUICustomizationFile("AttributeBrowser.uic",
"AttributeBrowser");
```

通过具体实例将会看出，通过自定义对话框，配置文件中的 UIC 文件将会被管理，不用自己手动更改 UIC 文件中的内容。

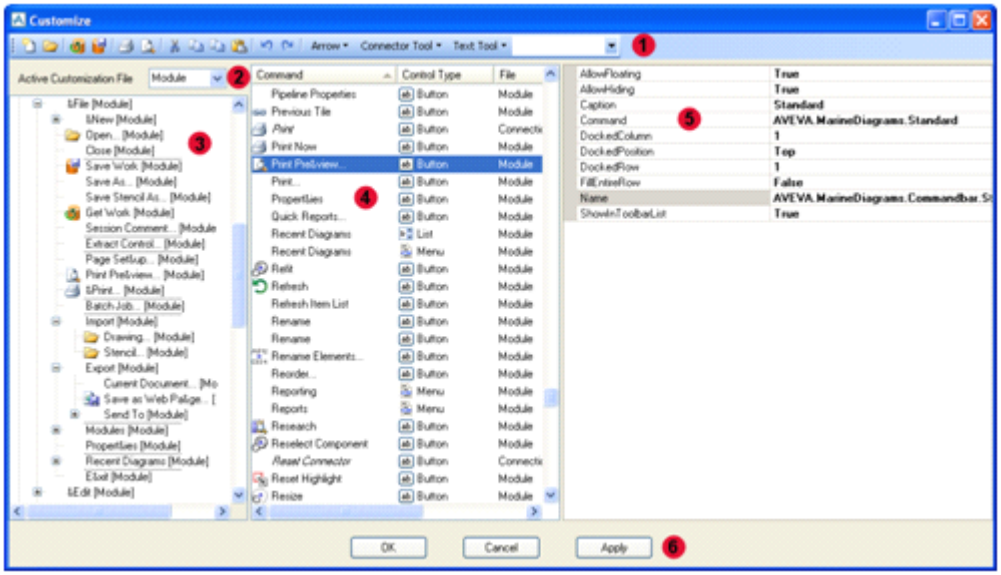
六、***UIC***文件的编辑 ***Editing UIC File***

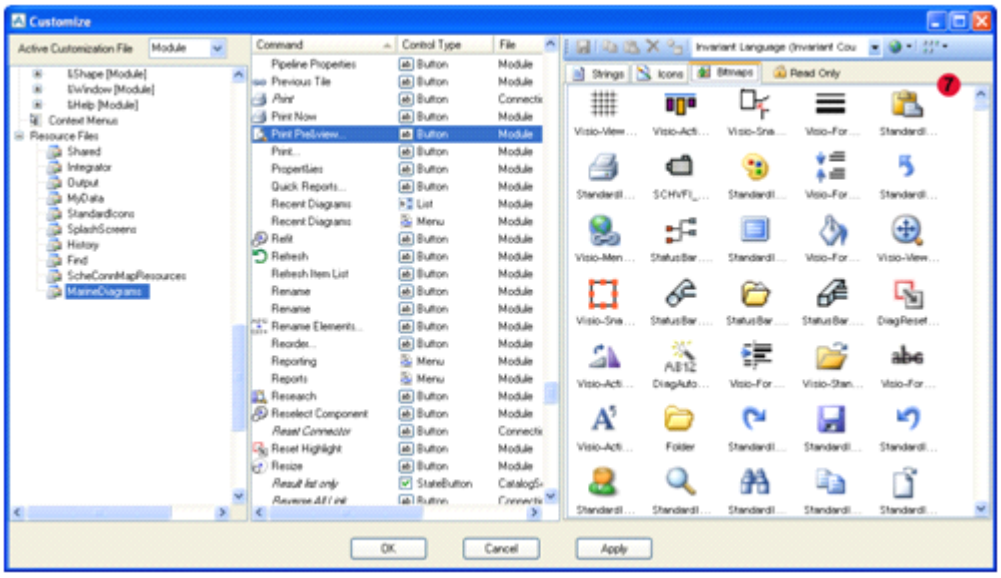
UIC文件是通过通用程序框架 ***CAF*** 的内置交互工具来创建与编辑的，不需要亲自更其中内容。通过在工具栏上右键，选择 ***Customize*** 来启动自定义对话框，如下图所示：



自定义对话框主要由七个部分组成：

① ***CommandBar Preview Area***
② ***Active Customisation File***
③ ***Tree showing CommandBars***
④ ***List of tools***
⑤ ***Property grid***
⑥ ***Action buttons***
⑦ ***Resource Editor***



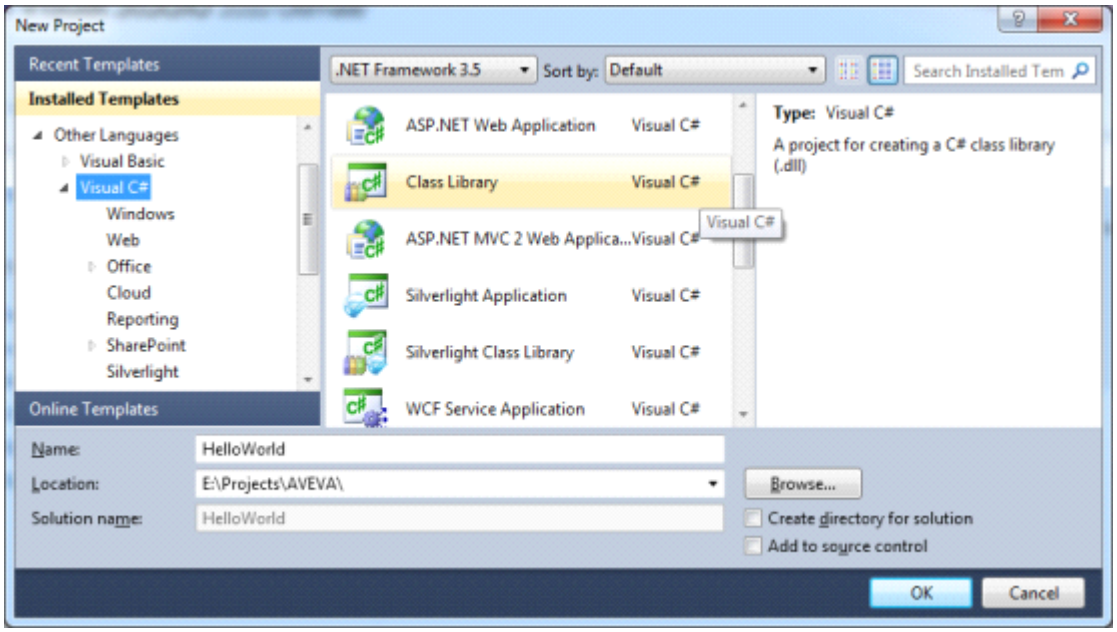


以下将会以具体实例来介绍相应的用法。更多细节请参考《***AVEVA .NET Customisation User Guide***》。通过自定义对话框，UIC 文件中的内容将会被 AVEVA 管理，所以创建自定义的菜单或工具栏按钮，都从这个自定义对话框中编辑，还是很方便。

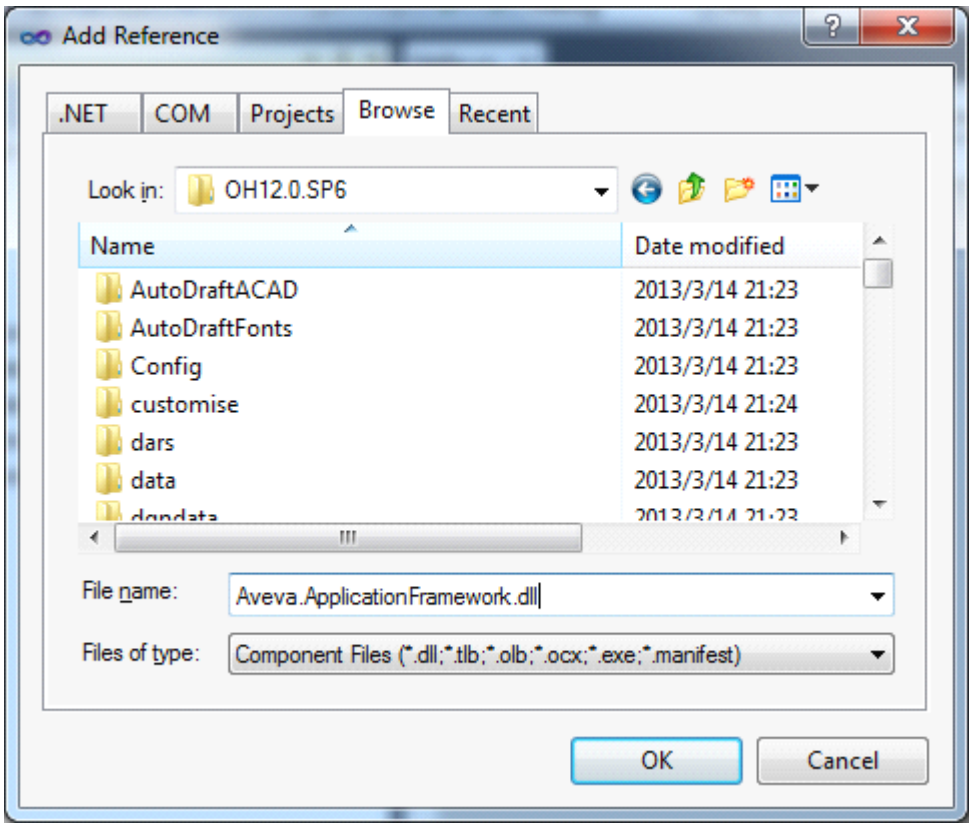
七、程序实例 ***Codes***

本节以一个具体实例来说明 ***AVEVA .Net*** 的菜单的自定义及与相关命令的关联方法。步骤如下：

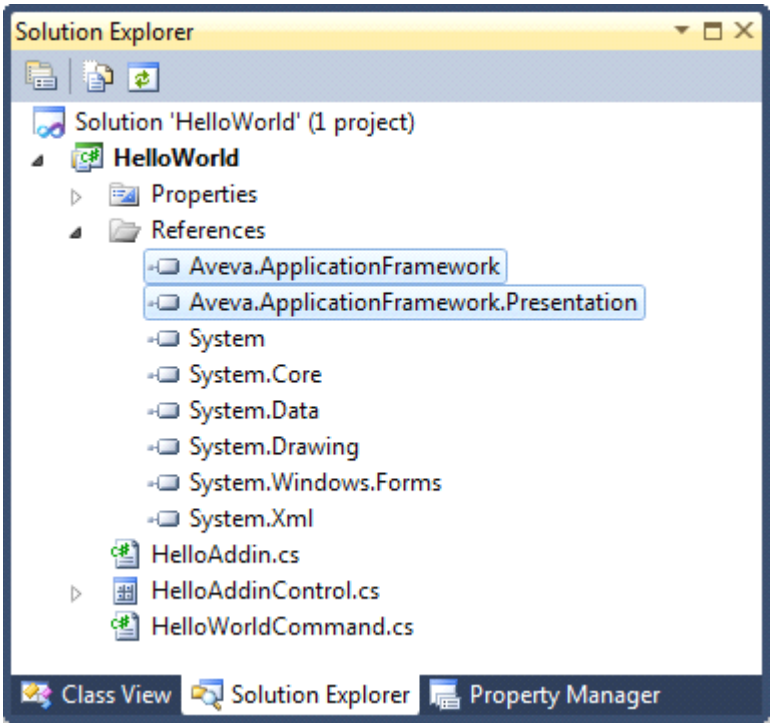
1. 创建一个 ***C#***的类库（***Class Library***），工程名为 ***HelloWorld***，并选择 ***.NET Framework*** 3.5，如下图所示：



2. 添加引用 ***Aveva.ApplicationFramework.dll*** 和 ***Aveva.ApplicationFramework.Presentation.dll***，如下图所示：



添加完后的引用库如下图所示：

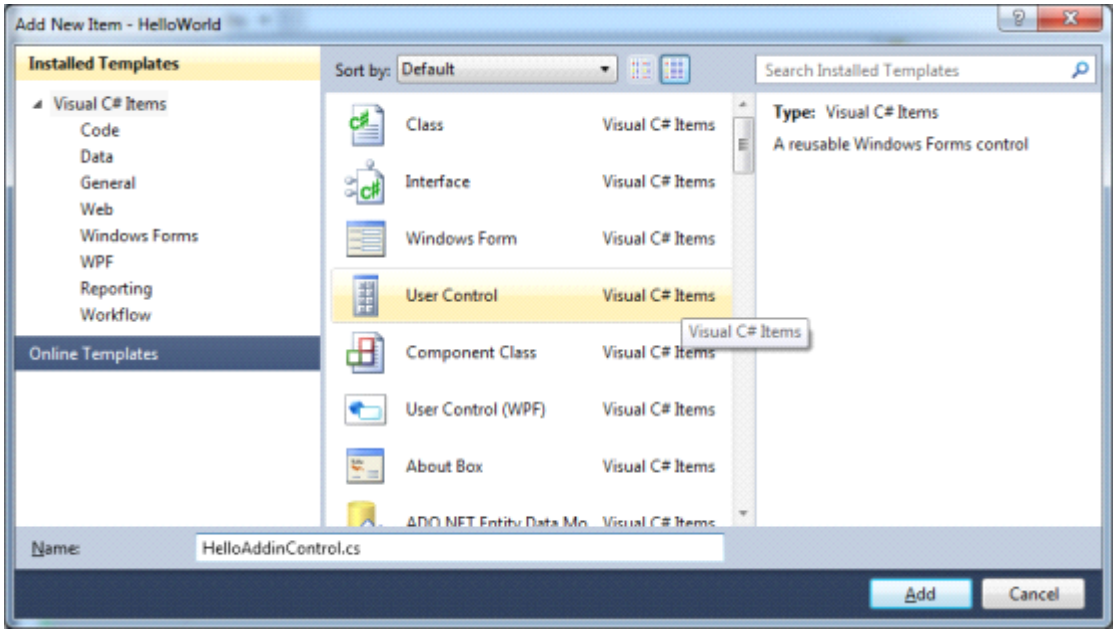


3. 编写一个插件类 ***HelloAddin***，派生自 ***IAddin***，并实现那四个虚函数，程序代码如下所示：

```
using System;
using System.Collections.Generic;
using System.Text;

using Aveva.ApplicationFramework;
using Aveva.ApplicationFramework.Presentation;

namespace HelloWorld
{
    public class HelloAddin : IAddin
    {
        #region IAddin Members

        public string Description
        {
            get
            {
                return "AVEVA .Net Menu Customisation Demo.";
            }
        }

        public string Name
        {
            get
```

```
            {
                return "HelloAddin";
            }
        }

        public void Start(ServiceManager serviceManager)
        {
            // Stop for attaching to the process.
            System.Windows.Forms.MessageBox.Show("Debug...");
        }

        public void Stop()
        {

        }

        #endregion
    }
}
```

4. 添加一个用户控件（***User Control***）而不是一个窗口（***Windows Form***），如下图所示：



5. 自定义用户控件上内容，本例仅在其上添加最简单的 ***label*** 为例，添加后并将其内容改为“***Hello World***”，如下图所示：

6. 添加命令类 ***HelloWorldCommand***，派生自抽象基类 ***Command***，并实现几个关键的方法。如在其构造函数中添加 ***Key*** 属性，***Key*** 属性在后面添加菜单时很重要。还有必须实现的 ***Execute*** 方法。程序代码如下所示：

```
using System;
using System.Collections.Generic;
using System.Text;

using Aveva.ApplicationFramework.Presentation;

namespace HelloWorld
{
    class HelloWorldCommand : Command
    {
        private DockedWindow myWindow;

        // Default Constructor.
        public HelloWorldCommand(DockedWindow window)
        {
            // Set the command key.
            // This is very import!!!
            this.Key = "Aveva.HelloWorldCommand";

            // Save the docked window.
            myWindow = window;

            // Create an event handler for the window closed event.
            myWindow.Closed += new EventHandler(_window_Closed);

            // Create an event handler for the windowLayoutLoaded event.
            WindowManager.Instance.WindowLayoutLoaded += new
EventHandler(Instance_WindowLayoutLoaded);
        }
```

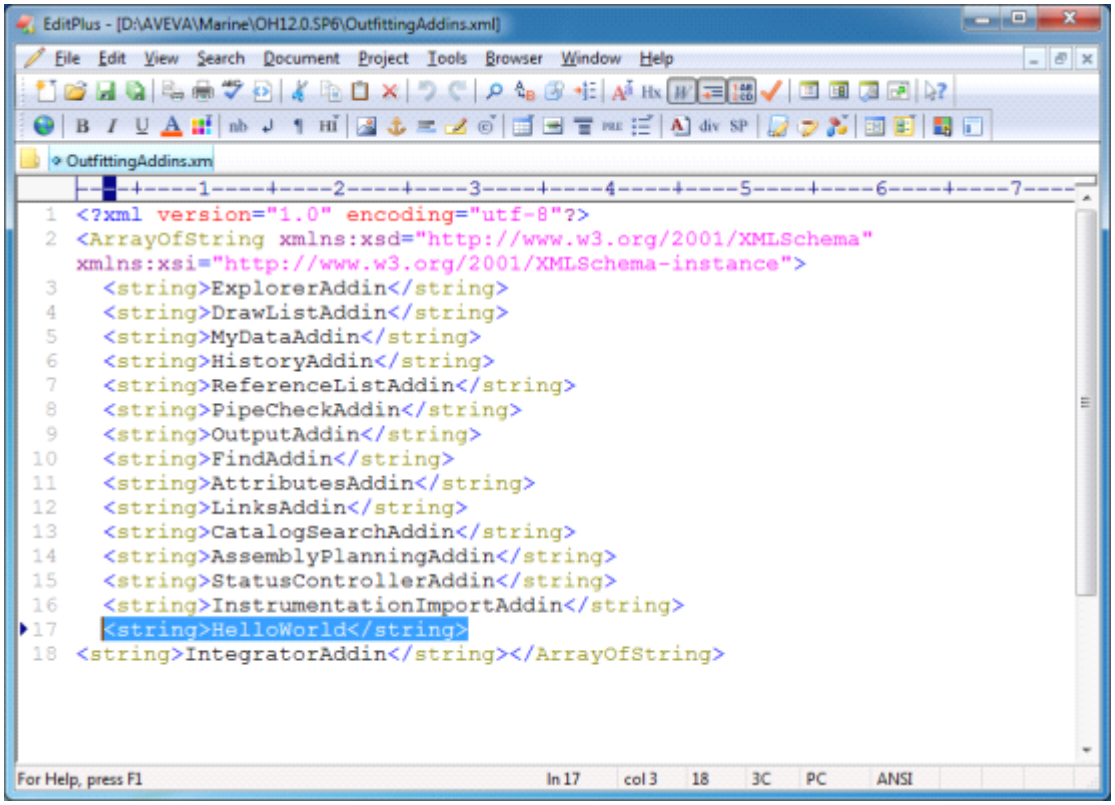
```
        void _window_Closed(object sender, EventArgs e)
        {
            this.Checked = false;
        }

        void Instance_WindowLayoutLoaded(object sender, EventArgs e)
        {
            // Update the command state to match initial window visibility.
            this.Checked = myWindow.Visible;
        }

        // This method must be overriden to provide the command execution
functionality.
        public override void Execute()
        {
            // Always show the DockedWindow.
            myWindow.Show();
        }
    }
}
```

7. 在派生自 ***IAddin*** 的类中的 ***Start*** 方法中添加创建可停靠窗口的代码，并将命令添加通用程序框架的命令集合中去，程序如下所示：

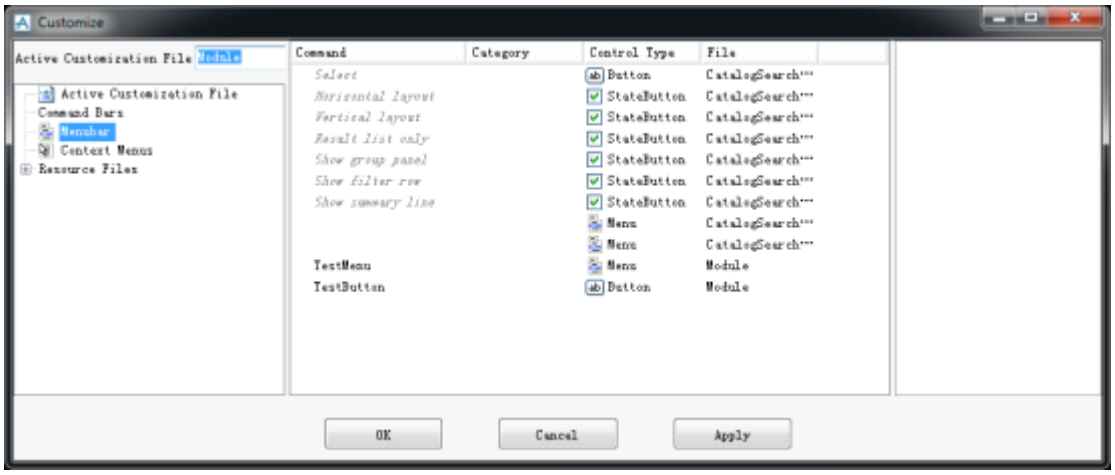
```
public void Start(ServiceManager serviceManager)
        {
            // Stop for attaching to the process.
            System.Windows.Forms.MessageBox.Show("Debug...");

            // Get the WindowManager service.
            WindowManager                    windowManger                    =
(WindowManager)serviceManager.GetService(typeof(WindowManager));

            // Create a docked window to host and Hello World User control.
            DockedWindow                     myWindow                        =
windowManger.CreateDockedWindow("Aveva.HelloAddin.UserControl",
                "HelloWorld",
                new HelloAddinControl(),
                DockedPosition.Right);

            // Create and register Addin's command.
            CommandManager                   commandManager                  =
(CommandManager)serviceManager.GetService(typeof(CommandManager));

            HelloWorldCommand showCommand = new HelloWorldCommand(myWindow);
```

```
            commandManager.Commands.Add(showCommand);
        }
```

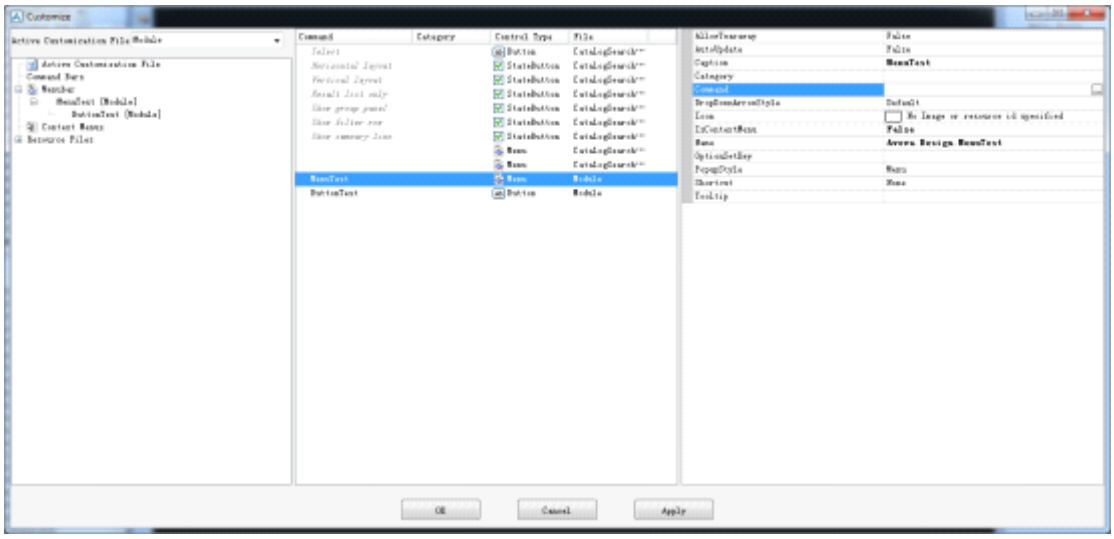
8. 编译工程（***Build HelloWorld***），并将生成的动态库（***HelloWorld.dll***）复制到 ***AVEVA*** 的安装目录；
9. 修改配置文件，使其加载该插件，修改方法如下图所示：



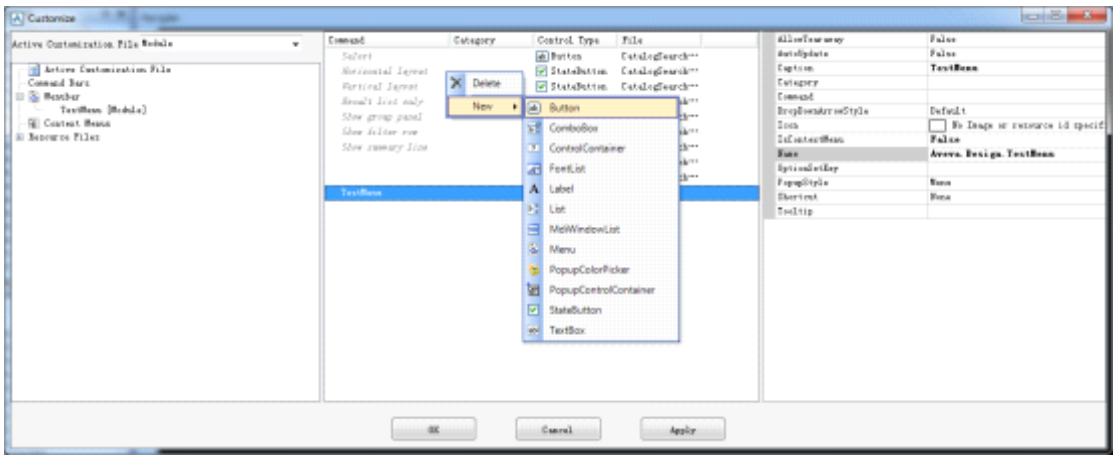
10. 添加自定义菜单，方便插件的调用。在工具栏的任意位置点击鼠标右键，选择 ***Customize***，出现自定义的对话框，并在 ***Active Customization File*** 的选项框中选择 ***Module***，如下图所示：



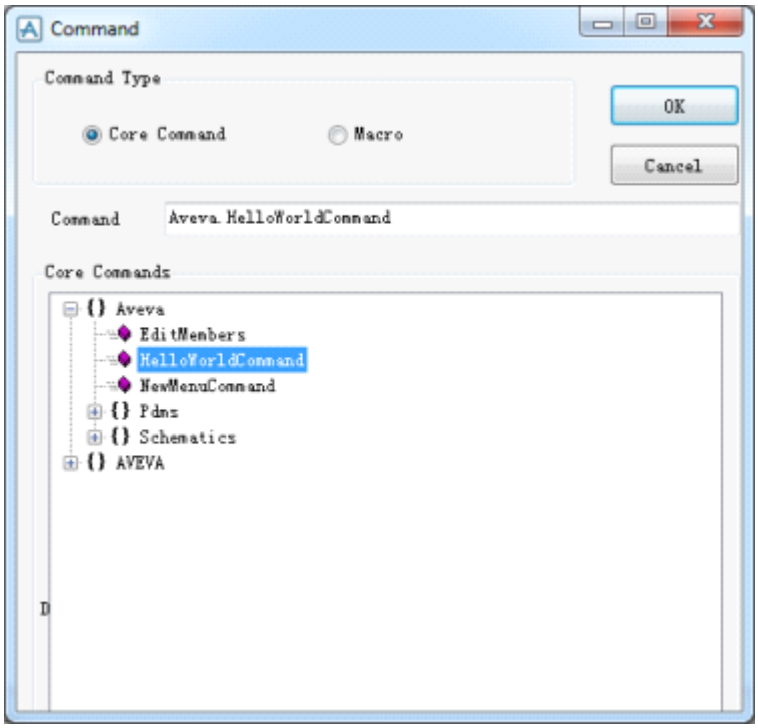
在 ***Menubar*** 中添加自定义菜单项，将其名称改为 ***MenuTest***，如下图所示：



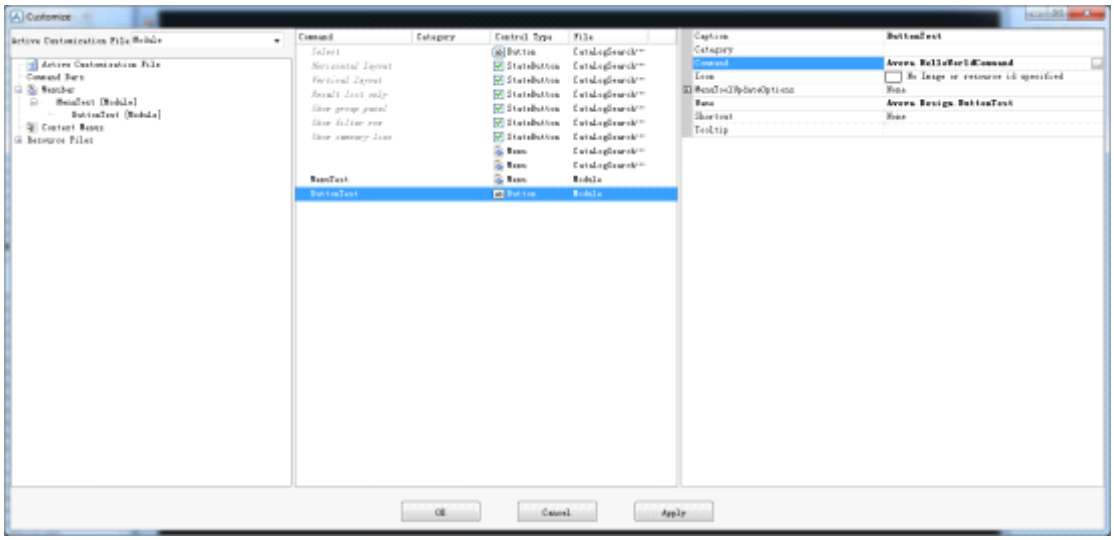
在对话框中间任意位置点鼠标右键，添加一个按钮（***button***），如下图所示：



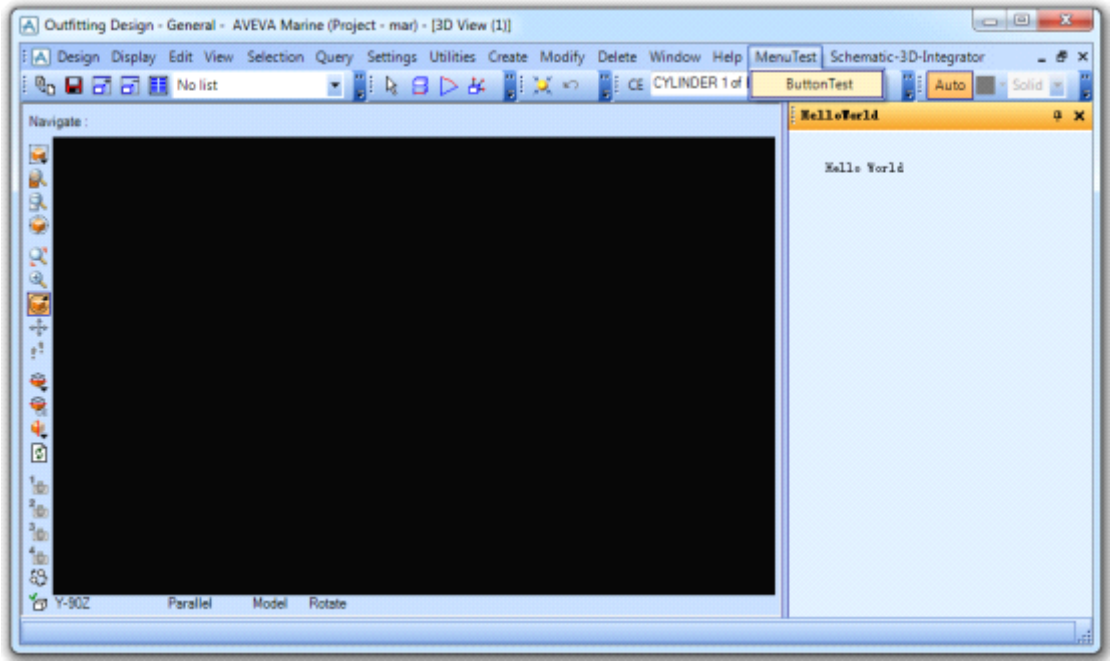
修改按钮的标题为 ***ButtonTest***，修改其命令（***Command***），在 ***Core Command*** 中选择我们创建的命令类的 ***Key***，如下图所示：



将按钮拖到左边的菜单下后，***Apply*** 结束。如下图所示：



11. 自定义后的菜单如下图所示：



八、结论 ***Conclusion***

通过实例对 ***AVEVA .Net*** 的二次开发有了便全面的认识，可以通过自定义菜单方便调用二次开发的插件。自定义菜单或者工具栏按钮主要是通过自定义对话框来实现，不需要程序员手动修改 UIC 文件，交互操作使用方便。

若有术语用词不当之处，敬请指出。eryar@163.com

九、参考资料 ***Bibliography***

1. ***AVEVA .Net Customisation User Guide*** 本文基本上是对这个 ***Guide*** 的部分翻译。
2. ***GoF book. Design Patterns Elements of Reusable Object-Oriented Software*** 对 ***Command*** 模式感兴趣的可以一读。