A plugins that defines a custom menu

Creating a plugin that displays a custom menu

To run this example you must first create a C# project that refrecnes FTC.SDKInterface.dll, Feat.UIComposition.dll and Feat.Core.dll. The resulting dll from your project (along with any non .NET dependecies) must be placed in the Plugins directory in the FTC install location.

The Plugin class.

The first step is to create a class that implements the IPluginService interface. This is the type that will be instantiated by Scene Composer.

```
 public class CustomMenuPlugin : IPluginService
 {

 }
```

Implementing IPlugin

Next we must implement the IPluginService interface members.

```
public string Name
{
  get { return "Custom Menu Definition"; }
}

public string Description
{
   get { return "A plugin that defines a custom menu"; }
}

public string Version
{
  get { return "1.0"; }
}
public void Initialize(IUnityContainer unityContainer)
{
    //register the menu config interface
    var loader = unityContainer.Resolve<IConfigurationManager>().RegisterConfigurationInterface
    ...
}
```

Config and commands interfaces

We will need to provide an interface derived from IConfiguration and an interface derived from ICommandContainer that defines the available commands.

```
public interface IMenuPluginConfig : IConfiguration
{
  ObservableCollection<string> Data { get; set; }
}

public interface ICustomPluginCommands : ICommandContainer
{
  ManagedCommand AddItem { get; }
  ManagedCommand RemoveItem { get; }
}
```

Then in the Initialize method:

Register the menu config interface

```
...
var loader = unityContainer.Resolve<IConfigurationManager>().RegisterConfigurationInterface<IMe
loader.ConfigLoaded += loader_ConfigLoaded;
...
```

Register the menu commands

```
 var cmdCollection = new CommandDefinitionCollection<ICustomPluginCommands>()
 {
  {
     c=>c.AddItem,
     new ManagedCommand()
     {
        ExecuteHandler = o=>
        {
           var data = loader.Instance.Data;
           data.Add("Item " + data.Count);
        },
        Metadata = new MetadataCollection()
        {
           new MainMenuOption()
           {
              MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "Add" },
           }
         }
     }
  },
  {
     c=>c.RemoveItem,
     new ManagedCommand()
     {
        CanExecuteHandler = o=> loader.Instance.Data.Any(),
```

```
        ExecuteHandler = o=>
        {
           var data = loader.Instance.Data;
           data.RemoveAt(0);
        },
        Metadata = new MetadataCollection()
        {
           new MainMenuOption()
           {
              MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "Remove" },
           },
        }
     }
  }
};
cmdCollection.RegisterAll(unityContainer.Resolve<UICommandManager>());
```

And register a custom menu definition to hold the item list

```
  var menuDefMgr = unityContainer.Resolve<MenuDefinitionManager>();
  menuDefMgr.RegisterMenuDefinition(new CustomMenuDefinition()
  {
     MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "RegItems" },
  });

  loader.Instance.Data.CollectionChanged += Data_CollectionChanged;
```

The full code for the example:

CustomMenuPlugin.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using FTC.SDKInterface.Plugins;
using System.Collections.ObjectModel;
using Feat.UIComposition.Commanding;
using Feat.UIComposition.Config;
using Feat.UIComposition;
using Feat.UIComposition.Menus;
using FTC.SDKInterface.Constants;

namespace UIPlugins.CustomMenu
{
  public class CustomMenuPlugin : IPluginService
  {
     public string Name
     {
        get { return this.Description; }
     }
```

```
public string Version
{
   get { return "1.0"; }
}

public string Description
{
   get { return "Custom Menu Definition"; }
}

public IEnumerable<int> FeatureIds
{
   get { return null; }
}

public FTC.SDKInterface.Services.IUnityContainer UnityContainer { get; set; }

public void Initialize(FTC.SDKInterface.Services.IUnityContainer unityContainer)
{
   var loader = unityContainer.Resolve<IConfigurationManager>().RegisterConfigurationInterf
   loader.ConfigLoaded += loader_ConfigLoaded;
   this.UnityContainer = unityContainer;

   var cmdCollection = new CommandDefinitionCollection<ICustomPluginCommands>()
   {
      {
         c=>c.AddItem,
         new ManagedCommand()
         {
            ExecuteHandler = o=>
            {
               var data = loader.Instance.Data;
               data.Add("Item " + data.Count);
            },
            Metadata = new MetadataCollection()
            {
               new MainMenuOption()
               {
                  MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "Add" },
               }
            }
         }
      },
      {
         c=>c.RemoveItem,
         new ManagedCommand()
         {
            CanExecuteHandler = o=> loader.Instance.Data.Any(),
            ExecuteHandler = o=>
            {
               var data = loader.Instance.Data;
               data.RemoveAt(0);
            },
```

```
                Metadata = new MetadataCollection()
                {
                   new MainMenuOption()
                   {
                      MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "Remove" },
                   },
                }
             }
          }
       };
       cmdCollection.RegisterAll(unityContainer.Resolve<UICommandManager>());

       var menuDefMgr = unityContainer.Resolve<MenuDefinitionManager>();
       menuDefMgr.RegisterMenuDefinition(new CustomMenuDefinition()
       {
          MenuPath = new MenuPath() { MenuConstants.File_1, "Items", "RegItems" },
       });

       loader.Instance.Data.CollectionChanged += Data_CollectionChanged;
    }

    void Data_CollectionChanged(object sender, System.Collections.Specialized.NotifyCollectionC
    {
       var cmdContainer = this.UnityContainer.Resolve<UICommandManager>().GetCommandContainer<I
       cmdContainer.RemoveItem.RaiseCanExecuteChanged();
    }

    void loader_ConfigLoaded(object sender, ConfigLoadedEventArgs<IMenuPluginConfig> e)
    {
       if (!e.ConfigLoadedSuccessfully)
       {
          e.Instance.Data = new ObservableCollection<string>();
       }
    }
 }

 public interface IMenuPluginConfig : IConfiguration
 {
    ObservableCollection<string> Data { get; set; }
 }

 public interface ICustomPluginCommands : ICommandContainer
 {
    ManagedCommand AddItem { get; }
    ManagedCommand RemoveItem { get; }
 }
}
```

CustomMenuDefintion.cs

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

```
using Feat.UIComposition.Menus;
using System.Windows.Controls;
using Feat.UIComposition.Config;
using System.Collections.ObjectModel;
using Feat.UIComposition.Commanding;
using Feat.CAL.Services;
using Feat.Utils;
using System.Windows;

namespace UIPlugins.CustomMenu
{
  class CustomMenuDefinition : MenuDefinitionBase
  {
     protected override IMenuItemManager CreateMenuItemManger(MenuManager menuManger)
     {
        return new CustomMenuManager(this);
     }

     public override string UniqueName
     {
        get { return this.GetType().AssemblyQualifiedName + this.MenuPath; }
     }
  }

  class CustomMenuManager : IMenuItemManager
  {
     public ObservableCollection<string> items;
     private IDialogSvc dialogSvc;

     public CustomMenuDefinition Definition { get; set; }

     public CustomMenuManager(CustomMenuDefinition cmDef)
     {
        this.Definition = cmDef;
        this.dialogSvc = this.Definition.UnityContainer.Resolve<IDialogSvc>();
        this.items = this.Definition.UnityContainer.Resolve<IConfigurationManager>().
           GetConfiguration<IMenuPluginConfig>().Data;
        items.CollectionChanged += items_CollectionChanged;
     }

     void items_CollectionChanged(object sender, System.Collections.Specialized.NotifyCollectior
     {
        this.Invalidate();
     }

     MenuItem rootMenuItem;
     public void Attach(MenuManager menuMgr, ItemsControl parentMenuItem, object startingSibling
     {
        this.rootMenuItem = new MenuItem()
        {
           Header = "ItemList"
        };
        parentMenuItem.Items.Add(rootMenuItem);
        this.Invalidate();
```

```
    }

    public bool IsItemVisible
    {
       get
       {
          return this.items.Any();
       }
    }

    public event EventHandler HasItemsChanged;

    public void Invalidate()
    {
       this.rootMenuItem.Items.Clear();
       items.
          Select(s => new MenuItem()
          {
             Header = s,
             Command = new ManagedCommand()
             {
                ExecuteHandler = o =>
                   this.dialogSvc.ShowMessageBox(s, s, MessageBoxButton.OK, MessageBoxImage.Inf
             }
          }).
          ForEach(i => this.rootMenuItem.Items.Add(i));

       if (this.HasItemsChanged != null)
       {
          this.HasItemsChanged(this, EventArgs.Empty);
       }
    }
  }
}
```

Revision #1
Created 2023-02-15 07:56:18 UTC by Tsuyoshi.Kato
Updated 2023-02-15 07:58:33 UTC by Tsuyoshi.Kato