عنوان: **آموزش Prism #2**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۳ ۹:۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, prism

در پست قبلی توضیح کلی درباره فریم ورک Prism داده شد. در این بخش قصد داریم آموزش‌های داده شده در پست قبلی را با هم در یک مثال مشاهده کنیم. در پروژه‌های ماژولار طراحی و ایجاد زیر ساخت قوی برای مدیریت ماژول‌ها بسیار مهم است. Prism فریم ورکی است که فقط چارچوب و قواعد اصول طراحی این گونه پروژه‌ها را در اختیار ما قرار می‌دهد. در پروژه‌های ماژولار هر ماژول باید در یک اسمبلی جدا قرار داشته باشد که ساختار پیاده سازی آن می‌تواند کاملا متفاوت با پیاده سازی سایر ماژول‌ها باشد.
برای شروع باید فایل‌های اسمبلی Prism رو دانلود کنید(لینک دانلود).

تشریح پروژه:

می‌خواهیم برنامه ای بنویسیم که دارای سه ماژول زیر است.:

ماژول Navigator : برای انتخاب و Switch کردن بین ماژول‌ها استفاده می‌شود؛
ماژول طبقه بندی کتاب‌ها : لیست طبقه بندی کتاب‌ها را به ما نمایش می‌دهد؛
ماژول لیست کتاب‌ها : عناوین کتاب‌ها به همراه نویسنده و کد کتاب را به ما نمایش می‌دهد.

*در این پروژه از UnityContainer برای مباحث Dependency Injection استفاده شده است.
ابتدا یک پروژه WPF در Vs.Net ایجاد کنید(در اینجا من نام آن را FirstPrismSample گذاشتم). قصد داریم یک صفحه طراحی کنیم که دو ماژول مختلف در آن لود شود. ابتدا باید Shell پروژه رو طراحی کنیم. یک Window جدید به نام Shell بسازید و کد زیر را

در آن کپی کنید.

```
<Window x:Class="FirstPrismSample.Shell"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:com="http://www.codeplex.com/CompositeWPF"
    Title="Prism Sample By Masoud Pakdel" Height="400" Width="600"
WindowStartupLocation="CenterScreen">
    <DockPanel>
      <ContentControl com:RegionManager.RegionName="WorkspaceRegion" Width="400"/>
      <ContentControl com:RegionManager.RegionName="NavigatorRegion"  DockPanel.Dock="Left" Width="200"
/>
    </DockPanel>
</Window>
```

در این صفحه دو ContentControl تعریف کردم یکی به نام Navigator و دیگری به نام Workspace. به وسیله RegionName که یک AttachedProperty است هر کدوم از این نواحی را برای Prism تعریف کردیم. حال باید یک ماژول برای Navigator و دو ماژول دیگر یکی برای طبقه بندی کتاب‌ها و دیگری برای لیست کتاب‌ها بسازیم.

#پروژه Common

قبل از هر چیز یک پروژه Common می‌سازیم و مشترکات بین ماژول‌ها رو در آن قرار می‌دهیم(این پروژه باید به تمام ماژول‌ها رفرنس داده شود). این مشترکات شامل :

کلاس پایه ViewModel

کلاس ViewRequestEvent

کلاس ModuleService

کد کلاس ViewModelBase که فقط اینترفیس INotifyPropertyChanged رو پیاده سازی کرده است:

```
using System.ComponentModel;

namespace FirstPrismSample.Common
{
    public abstract class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void RaisePropertyChangedEvent( string propertyName )
        {
            if ( PropertyChanged != null )
            {
                PropertyChangedEventArgs e = new PropertyChangedEventArgs( propertyName );
                PropertyChanged( this, e );
            }
        }
    }
}
```

کلاس ViewRequestEvent که به صورت زیر است:

```
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.Common.Events
{
    public class ViewRequestedEvent : CompositePresentationEvent<string>
    {
    }
}
```

توضیح درباره CompositePresentationEvent :

در طراحی و توسعه پروژه‌های ماژولار نکته ای که باید به آن دقت کنید این است که ماژول‌های پروژه نباید به هم وابستگی مستقیم داشته باشند در عین حال ماژول‌ها باید بتوانند با هم در ارتباط باشند. CPE یا **C**omposite **P**resentation **E**vent دقیقا برای این

منظور به وجود آمده است. CPE که در این جا طراحی کردم فقط کلاسی است که از CompositePresentationEventارث برده است و دلیل آن که به صورت string generic استفاده شده است این است که می‌خواهیم در هر درخواست نام ماژول درخواستی را داشته باشیم و به همین دلیل نام آن را ViewRequestedEvent گذاشتم.

توضیح درباره EventAggregator

EventAggregator یا به اختصار EA مکانیزمی است در پروژهای ماژولار برای اینکه در Composite UIها بتوانیم بین کامپوننت‌ها ارتباط برقرار کنیم. استفاده از EA وابستگی بین ماژول‌ها را از بین خواهد برد. برنامه نویسانی که با MVVM Light آشنایی دارند از قابلیت Messaging موجود در این فریم ورک برای ارتباط بین View و ViewModel استفاده می‌کنند. در Prism این عملیات توسط EA انجام می‌شود. یعنی برای ارتباط با Viewها باید از EA تعبیه شده در Prism استفاده کنیم. در ادامه مطلب، چگونگی استفاده از EA را خواهید آموخت.

اینترفیس IModuleService که فقط شامل یک متد است:

```
namespace FirstPrismSample .Common
{
    public interface IModuleServices
    {
        void ActivateView(string viewName);
    }
}
```

کلاس ModuleService که اینترفیس بالا را پیاده سازی کرده است:

```
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;

namespace FirstPrismSample.Common
{
    public class ModuleServices : IModuleServices
    {
        private readonly IUnityContainer m_Container;

        public ModuleServices(IUnityContainer container)
        {
            m_Container = container;
        }

        public void ActivateView(string viewName)
        {
            var regionManager = m_Container.Resolve<IRegionManager>();

            // غیر فعال کردن ویو
            IRegion workspaceRegion = regionManager.Regions["WorkspaceRegion"];
            var views = workspaceRegion.Views;
            foreach (var view in views)
            {
                workspaceRegion.Deactivate(view);
            }

            //فعال کردن ویو انتخاب شده
            var viewToActivate = regionManager.Regions["WorkspaceRegion"].GetView(viewName);
            regionManager.Regions["WorkspaceRegion"].Activate(viewToActivate);
        }
    }
}
```

متد ActivateView نام view مورد نظر برای فعال سازی را دریافت می‌کند. برای فعال کردن View ابتدا باید سایر viewهای فعال در RegionManager را غیر فعال کنیم. سپس فقط view مورد نظر در RegionManager انتخاب و فعال می‌شود.

*نکته: در هر ماژول ارجاع به اسمبلی‌های Prism مورد نیاز است.

#ماژول طبقه بندی کتاب ها:

برای شروع یک Class Library جدید به نام ModuleCategory به پروژه اضافه کنید. یک UserControl به نام CategoryView

بسازید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="FirstPrismSample.ModuleCategory.CategoryView "
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             Background="LightGray" FlowDirection="RightToLeft" FontFamily="Tahoma">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text=" طبقه بندی ها"/>
        <ListView Grid.Row="1"  Margin="10" Name="lvCategory">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50" />
                    <GridViewColumn Header="عنوان" Width="200"  />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

یک کلاس به نام CategoryModule بسازید که اینترفیس IModule رو پیاده سازی کند.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.ModuleCategory
{
    [Module(ModuleName = "ModuleCategory")]
    public class CategoryModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleCategory";

        public CategoryModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~CategoryModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            regionManager.Regions["WorkspaceRegion"].Add(new CategoryView(), moduleName);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(moduleName);
        }
    }
}
```

چند نکته :

*ModuleAttribute استفاده شده در بالای کلاس برای تعیین نام ماژول استفاده می‌شود. این Attribute دارای دو خاصیت دیگر

هم است :

OnDemand : برای تعیین اینکه ماژول باید به صورت OnDemand (بنا به درخواست) لود شود.
StartupLoaded : برای تعیین اینکه ماژول به عنوان ماژول اول پروژه لود شود.(البته این گزینه Obsolute شده است)

*برای تعریف ماژول کلاس مورد نظر حتما باید اینترفیس IModule را پیاده سازی کند. این اینترفیس فقط شامل یک متد است به نام Initialize.

*در این پروژه چون Viewهای برنامه صرفا جهت نمایش هستند در نتیجه نیاز به ایجاد ViewModel برای آن‌ها نیست. در پروژه‌های اجرایی حتما برای هر View باید ViewModel متناظر با آن تهیه شود.

توضیح درباره متد Initialize

در این متد ابتدا با استفاده از Container موجود RegionManager را به دست می‌آوریم. با استفاده از RegionManager می‌تونیم یک CompositeUI طراحی کنیم. در فایل Shell مشاهده کردید که یک صفحه به دو ناحیه تقسیم شد و به هر ناحیه هم یک نام اختصاص دادیم. دستور زیر به یک ناحیه اشاره خواهد داشت:

```
regionManager.Regions["WorkspaceRegion"]
```

در خط بعد با استفاده از EA یا Event Aggregator توانستیم CPE را بدست بیاوریم. متد Subscribe در کلاس CPE یک ارجاع قوی به delegate مورد نظر ایجاد می‌کند(پارامتر دوم این متد که از نوع boolean است) که به این معنی است که این delegate هیچ گاه توسط GC جمع آوری نخواهد شد. در نتیجه، قبل از اینکه ماژول بسته شود باید به صورت دستی این کار را انجام دهیم که مخرب را برای همین ایجاد کردیم. اگر به کدهای مخرب دقت کنید می‌بینید که با استفاده از EA توانستیم ViewRequestEventHandler را Unsubscribe کنیم به دلیل اینکه از ارجاع قوی با strong Reference در متد Subscribe استفاده شده است. دستور moduleService.ActiveateView ماژول مورد نظر را در region مورد نظر هاست خواهد کرد.

#ماژول لیست کتاب ها:

ابتدا یک Class Library به نام ModuleBook بسازید و همانند ماژول قبلی نیاز به یک Window و یک کلاس داریم:
BookWindow که کاملا مشابه به CategoryView است.

```
<UserControl x:Class="FirstPrismSample.ModuleBook.BookView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Background="LightGray" FontFamily="Tahoma" FlowDirection="RightToLeft">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text="لیست کتاب ها"/>
        <ListView Grid.Row="1" Margin="10" Name="lvBook">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50"  />
                    <GridViewColumn Header="عنوان" Width="200" />
                    <GridViewColumn Header="نویسنده" Width="150" />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

کلاس BookModule که پیاده سازی و توضیحات آن کاملا مشابه به CategoryModule می‌باشد.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Presentation.Events;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;

namespace FirstPrismSample.ModuleBook
{
    [Module(ModuleName = "moduleBook")]
    public class BookModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleBook";

        public BookModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~BookModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();

            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            var view = new BookView();
            regionManager.Regions["WorkspaceRegion"].Add(view, moduleName);
            regionManager.Regions["WorkspaceRegion"].Deactivate(view);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(m_WorkspaceBName);
        }
    }
}
```

#ماژول Navigator

برای این ماژول هم ابتدا View مورد نظر را ایجاد می‌کنیم:

```
<UserControl x:Class="FirstPrismSample.ModuleNavigator.NavigatorView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <Grid>
        <StackPanel VerticalAlignment="Center">
            <TextBlock Text="انتخاب ماژول" Foreground="Green" HorizontalAlignment="Center"
                VerticalAlignment="Center" FontFamily="Tahoma" FontSize="24" FontWeight="Bold" />
        <Button Command="{Binding ShowModuleCategory}" Margin="5" Width="125">طبقه بندی کتاب
ها</Button>
        <Button Command="{Binding ShowModuleBook}" Margin="5" Width="125">لیست کتاب ها</Button>
        </StackPanel>
        </Grid>
</UserControl>
```

حال قصد داریم برای این View یک ViewModel بسازیم. نام آن را INavigatorViewModel خواهیم گذاشت:

```
public interface INavigatorViewModel
    {
        ICommand ShowModuleCategory { get; set; }
        ICommand ShowModuleBook { get; set; }
```

```
        string ActiveWorkspace { get; set; }

        IUnityContainer Container { get; set; }

        event PropertyChangedEventHandler PropertyChanged;
    }
```

*در اینترفیس بالا دو Command داریم که هر کدام وظیفه لود یک ماژول را بر عهده دارند.
*خاصیت ActiveWorkspace برای تعیین workspace فعال تعریف شده است.

حال به پیاده سازی مثال بالا می‌پردازیم:

```
public class NavigatorViewModel : ViewModelBase, INavigatorViewModel
    {
        public NavigatorViewModel(IUnityContainer container)
        {
            this.Initialize(container);
        }

        public ICommand ShowModuleCategory { get; set; }

        public ICommand ShowModuleBook { get; set; }

        public string ActiveWorkspace { get; set; }

        public IUnityContainer Container { get; set; }

        private void Initialize(IUnityContainer container)
        {
            this.Container = container;
            this.ShowModuleCategory = new ShowModuleCategoryCommand(this);
            this.ShowModuleBook = new ShowModuleBookCommand(this);
            this.ActiveWorkspace = "ModuleCategory";
        }
    }
```

تنها نکته مهم در کلاس بالا متد Initialize است که دو Command مورد نظر را پیاده سازی کرده است. ماژول پیش فرض هم ماژول طبقه بندی کتاب‌ها یا ModuleCategory در نظر گرفته شده است. همان طور که می‌بینید پیاده سازی Commandها بالا توسط دو کلاس ShowModuleCategoryCommand و ShowModuleBookCommand انجام شده که در زیر کدهای آن‌ها را می‌بینید.

#کد کلاس ShowModuleCategoryCommand

```
public class ShowModuleCategoryCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private const string workspaceName = "ModuleCategory";

        public ShowModuleCategoryCommand(NavigatorViewModel viewModel)
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute(object parameter)
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute(object parameter)
        {
            CommandServices.ShowWorkspace(workspaceName, viewModel);
        }
    }
```

#کد کلاس ShowModuleBookCommand

```
public class ShowModuleBookCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private readonly string workspaceName = "ModuleBook";

        public ShowModuleBookCommand( NavigatorViewModel viewModel )
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute( object parameter )
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute( object parameter )
        {
            CommandServices.ShowWorkspace( workspaceName , viewModel );
        }
    }
```

با توجه به این که فرض است با متدهای Execute و CanExecute و CanExecuteChanged آشنایی دارید از توضیح این مطالب خودداری خواهم کرد. فقط کلاس CommandServices در متد Execute دارای متدی به نام ShowWorkspace است که کدهای زیر را شامل می‌شود:

```
public static void ShowWorkspace(string workspaceName, INavigatorViewModel viewModel)
  {
            var eventAggregator = viewModel.Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Publish(workspaceName);

            viewModel.ActiveWorkspace = workspaceName;
 }
```

در این متد با استفاده از CPE که در پروژه Common ایجاد کردیم ماژول مورد نظر را لود خواهیم کرد. و بعد از آن مقدار ActiveWorkspace جاری در ViewModel به نام ماژول تغییر پیدا می‌کند. متد Publish در CPE این کار را انجام خواهد دارد.

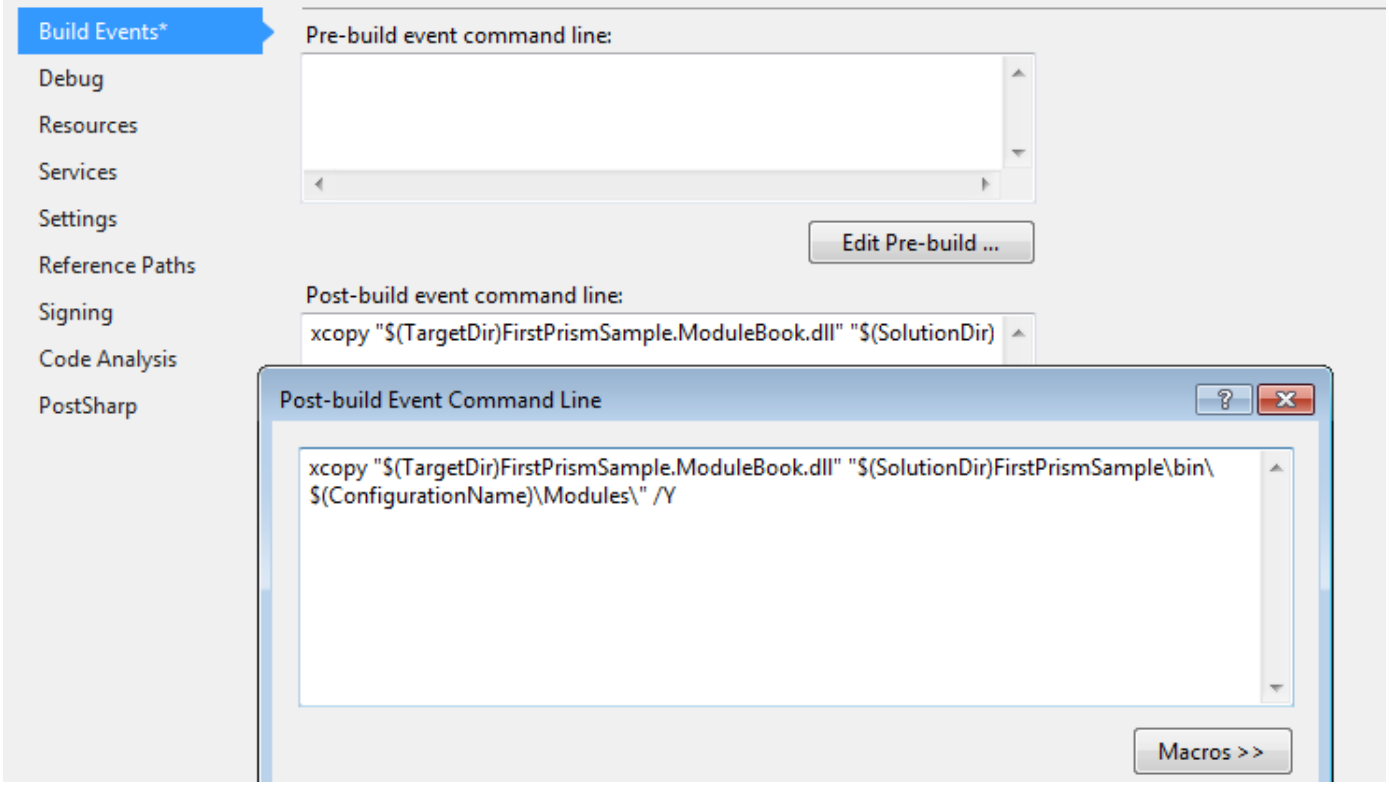
عدم وابستگی ماژول ها

همان طور که می‌بینید ماژول‌های پروژه به هم Reference داده نشده اند حتی هیچ Reference هم به پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد، داده نشده است ولی در عین حال باید با هم در ارتباط باشند. برای حل این مسئله این ماژول‌ها باید در فولدر bin پروژه اصلی خود را کپی کنند. بهترین روش استفاده از Pre-Post Build Event خود VS.Net است. برای این کار از پنجره Project Properties وارد برگه Build Events شوید و از قسمت Post Build Event Command Line استفاده کنید و کد زیر را در آن کپی نمایید:

```
xcopy "$(TargetDir)FirstPrismSample.ModuleBook.dll"
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

قطعا باید به جای FirstPrismSample نام Solution خود و به جای ModuleBook نام ماژول را وارد نمایید.

مانند:



مراحل بالا برای هر ماژول باید تکرار شود(ModuleNavigation , ModuleBook , ModuleCategory). بعد از Rebuild پروژه در فولدر bin پروژه اصلی یک فولدر به نام Module ایجاد می‌شود که اسمبلی هر ماژول در آن کپی خواهد شد.

ایجاد Bootstrapper

حال نوبت به Bootstrapper میرسد(در پست قبلی در باره مفهوم Bootstrapper شرح داده شد). در پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد کلاس زیر را ایجاد کنید.

```
public class Bootstrapper : UnityBootstrapper
    {
        protected override void ConfigureContainer()
        {
            base.ConfigureContainer();
            Container.RegisterType<IModuleServices, ModuleServices>();
        }

        protected override DependencyObject CreateShell()
        {
            var shell = new Shell();
            shell.Show();
            return shell;
        }

        protected override IModuleCatalog GetModuleCatalog()
        {
            var catalog = new DirectoryModuleCatalog();
            catalog.ModulePath = @".\Modules";
            return catalog;
        }
    }
```

متد ConfigureContainer برای تزریق وابستگی به وسیله UnityContainer استفاده می‌شود. در این متد باید تمامی Registrationهای مورد نیاز برای DI را انجام دهید. نکته مهم این است که عملیات وهله سازی و Initialization برای Container در متد base کلاس UnityBootstrapper انجام خواهد شد پس همیشه باید متد base این کلاس در ابتدای این متد فراخوانی شود در غیر این صورت با خطا متوقف خواهید شد.

متد CreateShell برای ایجاد و وهله سازی از Shell پروژه استفاده می‌شود. در این جا یک وهله از Shell Window برگشت داده می‌شود.
متد GetModuleCatalog برای تعیین مسیر ماژول‌ها در پروژه کاربرد دارد. در این متد با استفاده از خاصیت ModulePath کلاس DirectoryModuleCatalog تعیین کرده ایم که ماژول‌های پروژه در فولدر Modules موجود در bin اصلی پروژه قرار دارد. اگر به دستورات کپی در Post Build Event قسمت قبل توجه کنید می‌بینید که دستور ساخت فولدر وجود دارد.

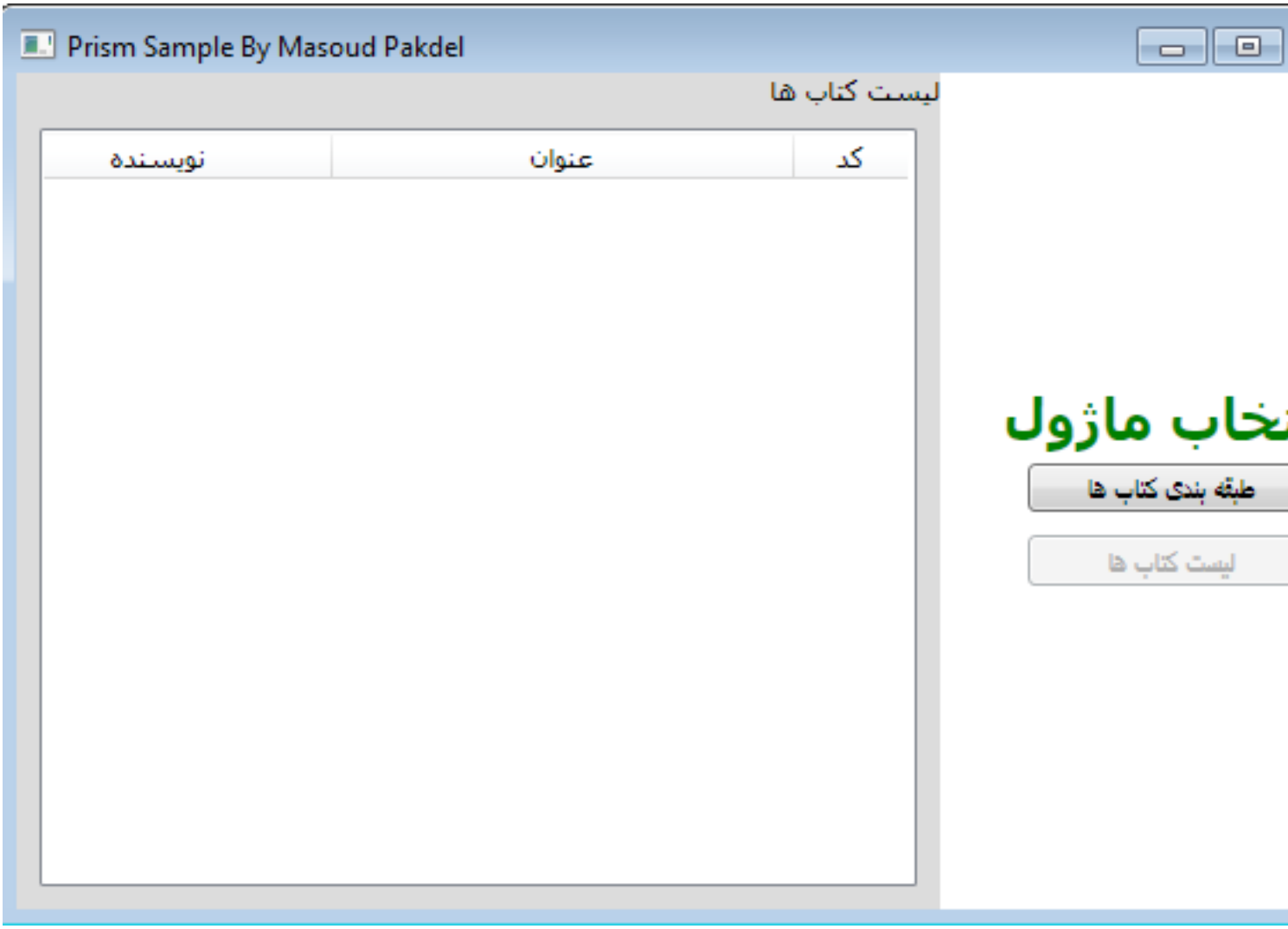
```
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

***نکته:** اگر استفاده از این روش برای شناسایی ماژول‌ها توسط Bootstrapper را چندان جالب نمی‌دانید می‌تونید از MEF استفاده کنید که اسمبلی ماژول‌های پروژه را به راحتی شناسایی می‌کند و در اختیار Bootsrtapper قرار می‌دهد(از آن جا در مستندات مربوط به Prism، بیشتر به استفاده از MEF تاکید شده است من هم در پست‌های بعدی، مثال‌ها را با MEF پیاده سازی خواهم کرد)

در پایان باید فایل App.xaml را تغییر دهید به گونه ای که متد Run در کلاس Bootstapper ابتدا اجرا شود.

```
public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            var bootstrapper = new Bootstrapper();
            bootstrapper.Run();
        }
    }
```

اجرای پروژه:
بعد از اجرا، با انتخاب ماژول مورد نظر اطلاعات ماژول در Workspace Content Control لود خواهد شد.


Prism Sample By Masoud Pakdel
لیست کتاب ها
کد
عنوان
نویسنده
طبقه بندی کتاب ها
لیست کتاب ها

ادامه دارد...

نظرات خوانندگان

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۰:۲۷

با سلام مهندس
خیلی عالیه امیدوارم ادامه بدید . با تشکر

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۹:۵۶

ممنون از آموزش خوبتون ، نظرتون در مورد استفاده از Prism به همراه StructerMap چیه ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۳ ۲۲:۲۳

شدنی است. فقط همانند UnityBootstrapper نیاز به یک StructureMapBootstrapper دارید. این کار قبلا توسط Richard Cerirol انجام شده. می‌تونید از nuget استفاده کنید:

```
PM> Install-Package Prism.StructureMapExtensions
```

نویسنده: بهنام
تاریخ: ۱۳۹۲/۰۴/۰۵ ۱:۲۶

با سلام و با تشکر مطلب مفیدتان
چند اصلاح کوچک در مطلب هست که اینجا بیان می‌کنم
بخش اول (مبدا)دستور xcopy باید به دستور زیر تبدیل شود:

```
xcopy "$(SolutionDir)\PrismProject.ModuleBook\bin\$(ConfigurationName)\PrismProject.ModuleBook.dll"
"$(SolutionDir)PrismProject\bin\$(ConfigurationName)\Modules\" /Y
```

همچنین متد GetModuleCatalog به CreateModuleCatalog تبدیل شده است.
با تشکر مجدد

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۵ ۹:۳۰

ممنونم دوست عزیز.
در مورد دستور اول روش ذکر شده کاملا صحیح است و نیازی به اصلاح نیست.

$TargetDir دقیقا به مسیر فایل‌های اجرایی اشاره می‌کند و $ConfigurationName را در خودش پشتیبانی می‌کنه. یعنی اگر پروژه در حال Relase باشد با استفاده از $TargetDir دقیقا به فایل‌های موجود در فولدر Release در bin پروژه اشاره می‌کند و در حالت Debug به فایل‌های موجود در فولدر Debug در bin پروژه. با استفاده از گزینه Macros درقسمت Edit Post-Build مشاهده می‌کنید که مقدار $TargetDir دقیقا صحیح است. اما دلیل اینکه چرا در بخش دوم دستور از $SolutionDir استفاده شده است به این دلیل است که می‌خواهیم به فولدر bin پروژه اصلی اشاره داشته باشیم و چون این پروژه حتما در مسیر Solution جاری خواهد بود در نتیجه از این آدرس استفاده شده است.(در این جا TargetDir و TargetPath نمی‌تواند کمکی به ما بکند). به تصویر زیر دقت کنید:(چون پروژه در حالت release است در نتیجه مقادیر TargetDir و TargetPath به release ختم می‌شود)

Macro	Value
OutDir	bin\Release\
ConfigurationName	Release
ProjectName	XLIFFProject
TargetName	WpfApplication١
TargetPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\WpfApplication١.exe
ProjectPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\XLIFFProject.csproj
ProjectFileName	XLIFFProject.csproj
TargetExt	.exe
TargetFileName	WpfApplication١.exe
DevEnvDir	C:\Program Files (x٨٦)\Microsoft Visual Studio ١٠,٠\Common٧\IDE\
TargetDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\
ProjectDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\
SolutionFileName	XLIFFProject.sln
SolutionPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject.sln
SolutionDir	E:\Workspace\Projects\XLIFFProject\
SolutionName	XLIFFProject
PlatformName	x٨٦
ProjectExt	.csproj
SolutionExt	.sln

به تفاوت مقادیر بین $TargetDir و $TargetPath و $SolutionDir و ... دقت کنید.

در مورد متد GetModuleCatalog هم باید عنوان کنم که این متد در اسمبلی Microsoft.Practices.Composite.UnityExtensions ورژن 2.0.1.0 وجود دارد. در ورژن 4 نسخه Prism این متد به این نام تغییر کرده است. در این جا می‌تونید تغییرات بین Prism Library 2 و Prism Library 4 رو ببینید

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۴/۲۲ ۱۹:۴۹

درود؛

لطفاً سورس پروژهٔ مثال را هم جهت دانلود اینجا بذارین، چون توی مقاله اشاره‌ای به اینکه پروژه‌ها از چه نوعی باشند و کدوم رفرنس‌ها را لازم دارند نشده و برای یکی مثل من که کلاً آشناییش با مقالات شما آغاز شده پیشرفت کار خیلی کند میشه.
سپاسگزارم.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۲۲ ۲۰:۱۶

در قسمت سوم ، سورس پیوست شده