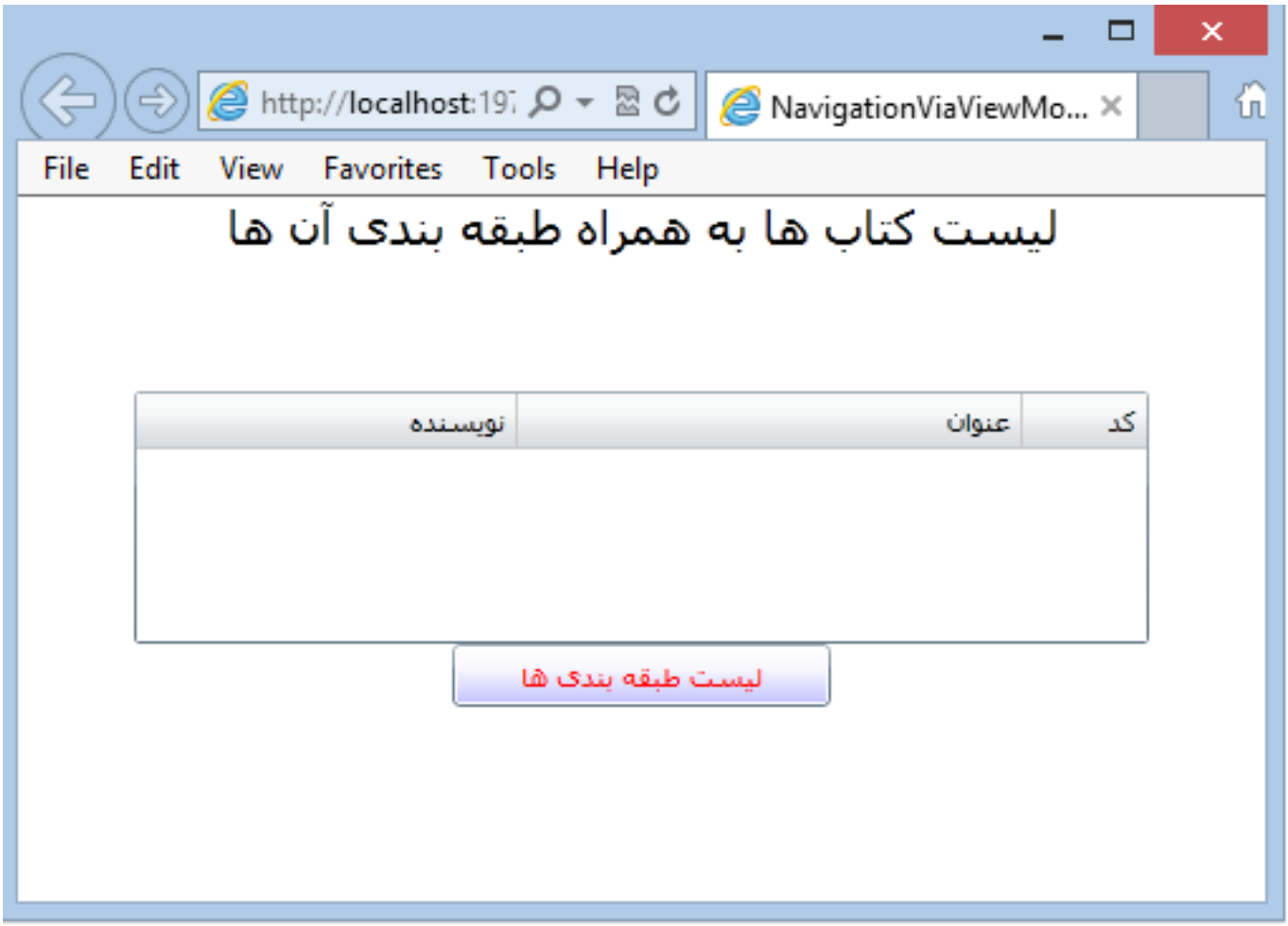
عنوان: **آموزش Prism #3**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۶ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, Silverlight, WPF, prism

در پست‌های قبلی با Prism و روش استفاده از آن آشنا شدیم (قسمت اول) و (قسمت دوم). در این پست با استفاده از Mef قصد ایجاد یک پروژه Silverlight رو به صورت ماژولار داریم. مثال پیاده سازی شده در پست قبلی را در این پست به صورت دیگر پیاده سازی خواهیم کرد.
تفاوت‌های پیاده سازی مثال پست قبلی با این پست:
در مثال قبل پروژه به صورت Desktop و با WPF پیاده سازی شده بود ولی در این مثال با Silverlight می‌باشد؛
در مثال قبل از UnityBootstrapper استفاده شده بود ولی در این مثال از MefBootstrapper؛
در مثال قبل هر View در یک ماژول قرار داشت ولی در این مثال هر دو View را در یک ماژول قرار دادم؛
در مثال قبل از Prism Libary 2.x استفاده شده بود ولی در این مثال از PrismLibrary 4.x؛
و...

نکته : برای فهم بهتر مفاهیم، آشنایی اولیه با MEF و مفاهیمی نظیر Export و Import و AggregateCatalog و AssemblyCatalog نیاز است. در صورتی که با این مطالب آشنایی ندارید می‌توانید از (^) شروع کنید.



برای شروع یک پروژه Silverlight ایجاد کنید. بعد از اضافه شدن دو پروژه Silverlight و Web، یک Silverlight Class

Library جدید بسازید.

ابتدا یک Page ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             x:Class="Module1.Module1View1"
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="50"  />
                <sdk:DataGridTextColumn Header="عنوان" Width="200" />
                <sdk:DataGridTextColumn Header="نویسنده" Width="150"  />
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Red"
                Background="Blue"
                Content="لیست طبقه بندی ها" />
    </StackPanel>
</UserControl>
```

بر روی Page مربوطه راست کلیک کنید و گزینه ViewCode را انتخاب کنید و کدهای زیر را در آن کپی کنید.

```
[Export(typeof(Module1View1))]
    public partial class Module1View1 : UserControl
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public Module1View1()
        {
            InitializeComponent();

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click(object sender, RoutedEventArgs e)
        {
            TheRegionManager.RequestNavigate
            (
                "MyRegion1",
                new Uri("Module1View2", UriKind.Relative),
                a => { }
            );
        }
    }
```

ابتدا خود این View باید حتما Export شود. در رویداد کلیک با استفاده از متد RequestNavigate می‌توانیم به View مورد نظر برای نمایش در Shell اشاره کنیم و این View در Region نمایش داده می‌شود. به دلیل اینکه در این کلاس به RegionManager نیاز داریم از ImportAttribute استفاده کردیم. این بدین معنی است که کلاس Module1View1 وابستگی مستقیم به IRegionManager دارد.

حال یک Page دیگر برای طبقه بندی کتاب‌ها ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
x:Class="Module1.Module1View2"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="150"/>
                <sdk:DataGridTextColumn Header="عنوان" Width="150"/>
```

```
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Green"
                Background="Yellow"
                Content="لیست کتاب ها" />
    </StackPanel>
</UserControl>
```

در Code Behind این Page نیز کدهای زیر را قرار دهید.

```
using Microsoft.Practices.Prism.Regions;
using System;
using System.ComponentModel.Composition;
using System.Windows;
using System.Windows.Controls;

namespace Module1
{
    [Export]
    public partial class Module1View2 : UserControl
    {
        IRegion _region1;

        [ImportingConstructor]
        public Module1View2( [Import] IRegionManager regionManager )
        {
            InitializeComponent();

            ViewModel viewModel = new ViewModel();
            DataContext = viewModel;

            viewModel.ShouldNavigateFromCurrentViewEvent += () => { return true; };

            _region1 = regionManager.Regions["MyRegion1"];

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click( object sender, RoutedEventArgs e )
        {
            _region1.RequestNavigate
            (
                new Uri( "Module1View1", UriKind.Relative ),
                a => { }
            );
        }
    }
}
```

در این ماژول برای اینکه بتوانیم حالت گردشی در فراخوانی ماژول‌ها را داشته باشیم ابتدا DataContext این کلاس را برابر با ViewModel ساخته شده قرار دادیم. با استفاده از رویداد ShoudlNavigateFromCurrentViewEvent که در کلاس ViewModel وجود دارد تعیین می‌کنیم که آیا باید از این View به View قبلی برگشت داشته باشیم یا نه. در صورتی که مقدار false برگشت داده شود خواهید دید که امکان فراخوانی View1 از View2 امکان پذیر نیست. در رویداد کلیک نیز همانند Page قبلی با استفاده از RegionManager و متد RequestNavigate به View مورد نظر راهبری کرده ایم.

نکته: اگر یک کلاس، سازنده با پارامتر داشته باشد باید با استفاده از ImportingConstructor حتما سازنده مورد نظر را هنگام وهله سازی مشخص کنیم در غیر این صورت با Exception مواجه خواهید شد.

حال قصد ایجاد کلاس ViewModel بالا را داریم:

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
```

```
using System.Windows.Shapes;
using System.ComponentModel.Composition;
using Microsoft.Practices.Prism.Regions;

namespace Module1
{
    public class ViewModel : IConfirmNavigationRequest
    {
        public event Func<bool> ShouldNavigateFromCurrentViewEvent;

        public bool IsNavigationTarget( NavigationContext navigationContext )
        {
            return true;
        }

        public void OnNavigatedTo( NavigationContext navigationContext )
        {

        }

        public void OnNavigatedFrom( NavigationContext navigationContext )
        {

        }

        public void ConfirmNavigationRequest( NavigationContext navigationContext, Action<bool>
continuationCallback )
        {
            bool shouldNavigateFromCurrentViewFlag = false;

            if ( ShouldNavigateFromCurrentViewEvent != null )
                shouldNavigateFromCurrentViewFlag = ShouldNavigateFromCurrentViewEvent();

            continuationCallback( shouldNavigateFromCurrentViewFlag );
        }
    }
}
```

توضیح متدهای بالا:

IsNavigateTarget : برای تعیین اینکه آیا کلاس پیاده سازی کننده اینترفیس، می‌تواند عملیات راهبری را مدیریت کند یا نه.

OnNavigateTo : زمانی عملیات راهبری وارد View شود(بهتره بگم View مورد نظر در Region صفحه لود شود) این متد فراخوانی می‌شود.

OnNavigateFrom : زمانی که راهبری از این View خارج می‌شود (View از حالت لود خارج می‌شود) این متد فراخوانی خواهد شد.

ConfirmNavigationRequest : برای تایید عملیات راهبری توسط کلاس پیاده سازی کننده اینترفیس استفاده می‌شود.

حال یک کلاس برای پیاده سازی و مدیریت ماژول می‌سازیم.

```
using Microsoft.Practices.Prism.MefExtensions.Modularity;
using Microsoft.Practices.Prism.Modularity;
using Microsoft.Practices.Prism.Regions;
using System.ComponentModel.Composition;

namespace Module1
{
    [ModuleExport(typeof(Module1Impl))]
    public class Module1Impl : IModule
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public void Initialize()
        {
            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View1));

            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View2));
        }
    }
}
```

همان طور که مشاهده می‌کنید از ModuleExportAttribute برای شناسایی ماژول توسط MefBootstrapper استفاده کردیم و نوع آن را ModuleImpl قرار دادیم. ImportAttribute استفاده شده در این کلاس و خاصیت TheRegionManager برای این است که در هنگام ساخت Instance از این کلاس IRegionManager موجود در Container باید در اختیار این کلاس قرار گیرد(نشان دهنده وابستگی مستقیم این کلاس با IRegionManager است). روش دیگر این است که در سازنده این کلاس هم این اینترفیس را تزریق کنیم.

در متد Initialize برای RegionManager دو View ساخته شده را رجیستر کردیم. این کار باید به تعداد Viewهای موجود در ماژول انجام شود.

Shell

در پروژه اصلی یک Page به نام Shell ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="NavigationViaViewModel.Shell"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:prism="http://www.codeplex.com/prism" FlowDirection="RightToLeft"
FontFamily="Tahoma">

    <Grid x:Name="LayoutRoot"
          Background="White">
        <TextBlock Text="لیست کتاب‌ها به همراه طبقه بندی آن ها"
                   FontSize="19"
                   Foreground="Black"
                   HorizontalAlignment="Center"
                   VerticalAlignment="Top" />
        <ContentControl HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        prism:RegionManager.RegionName="MyRegion1" />
    </Grid>
</UserControl>
```

همانند مثال قبلی یک ContentControl داریم و به وسیله RegionName که یک AttachedProperty است یک Region به نام MyRegion1 ایجاد کردیم. تمام ماژول‌های این مثال در این محدوده نمایش داده خواهند شد.

Bootstrapper

حال نیاز به یک Bootstrapper داریم. برای این کار یک کلاس به نام TheBootstrapper بسازید:

```
using Microsoft.Practices.Prism.MefExtensions;
using Microsoft.Practices.Prism.Modularity;
using System.ComponentModel.Composition.Hosting;
using System.Windows;

namespace NavigationViaViewModel
{
    public class TheBootstrapper : MefBootstrapper
    {
        protected override void InitializeShell()
        {
            base.InitializeShell();

            Application.Current.RootVisual = (UIElement)Shell;
        }

        protected override DependencyObject CreateShell()
        {
            return Container.GetExportedValue<Shell>();
        }

        protected override void ConfigureAggregateCatalog()
        {
            base.ConfigureAggregateCatalog();
            AggregateCatalog.Catalogs.Add(new AssemblyCatalog(this.GetType().Assembly));
        }

        protected override IModuleCatalog CreateModuleCatalog()
        {
            ModuleCatalog moduleCatalog = new ModuleCatalog();

            moduleCatalog.AddModule
            (
                new ModuleInfo
                {
```

```
                    InitializationMode = InitializationMode.WhenAvailable,
                    Ref = "Module1.xap",
                    ModuleName = "Module1Impl",
                    ModuleType = "Module1.Module1Impl, Module1, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null"
                }
            );

            return moduleCatalog;
        }
    }
}
```

متد CreateShell اولین متد در این کلاس است که اجرا خواهد شد. بعد از متد CreateShell، متد InitializeShell اجرا خواهد شد. خاصیت Shell دقیقا به مقدار برگشتی متد CreateShell اشاره خواهد کرد. در متد InitializeShell مقدار خاصیت Shell به RootVisual این پروژه اشاره می‌کند(مانند MainWindow در کلاس Application پروژه‌های WPF).

متد ConfigureAggregateCatalog برای مدیریت کاتالوگ‌ها و ماژول‌ها که هر کدام در یک اسمبلی جدا وجود خواهند شد استفاده می‌شود. در این متد من از AssemblyCatalog استفاده کردم. AssemblyCatalog تمام کلاس هایی که ExportAttribute را به همراه دارند شناسایی می‌کند و آن‌ها را در Container نگهداری خواهد کرد(^). مانند یک ServiceLocator در Microsoft unity Service Locator(^) .

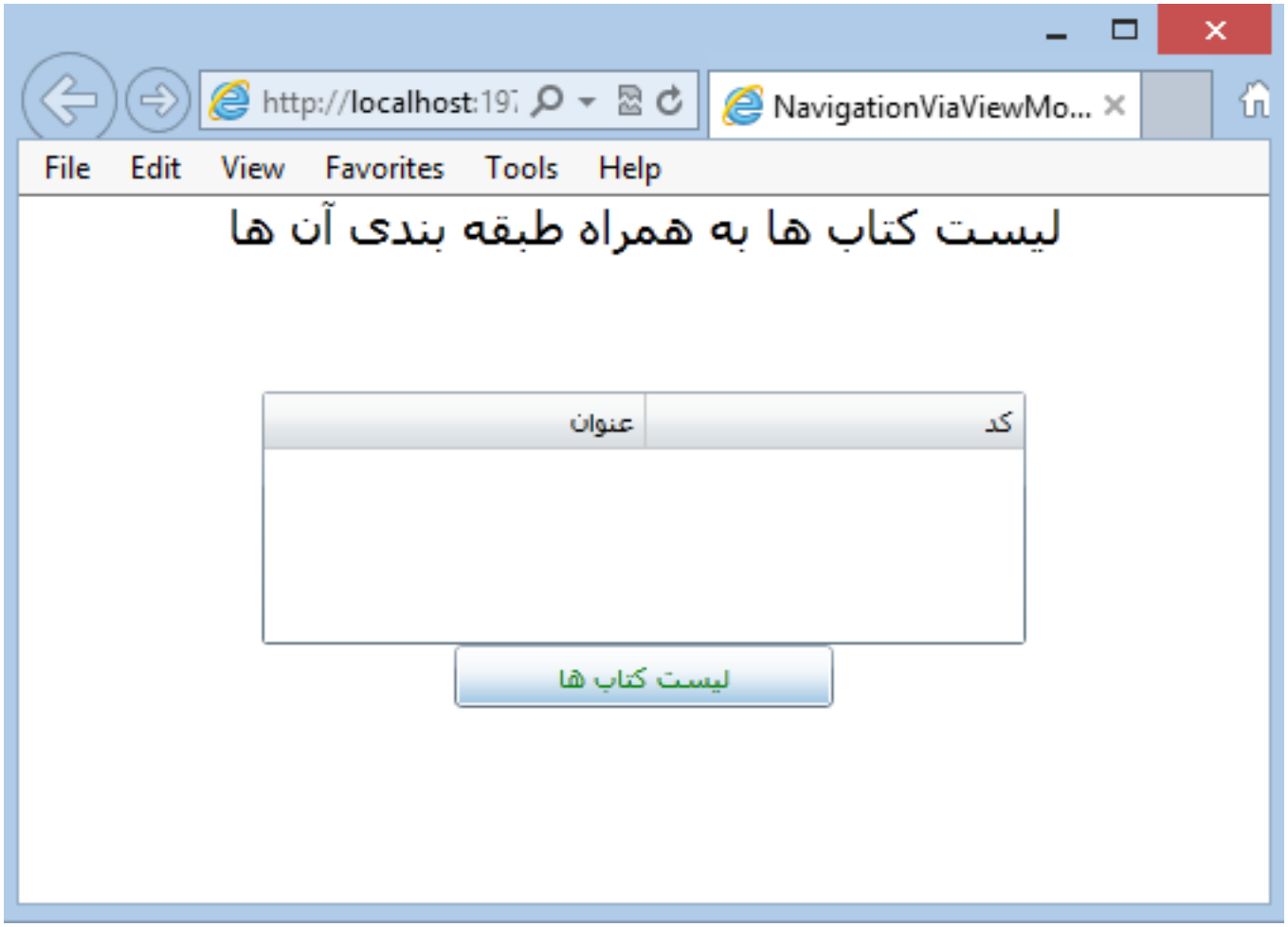
متد آخر به نام CreateModuleCatalog است و باید در آن تمام ماژول‌های برنامه را به کلاس ModuleCatalog اضافه کنیم. در مثال پست قبلی به دلیل استفاده از UnityBootstrapper باید این کار را از طریق BuildEvent ها مدیریت می‌کردیم ولی در این جا Mef به راحتی این کار را انجام خواهد داد.

تغییرات زیر را در فایل App.Xaml قرار دهید و پروژه را اجرا کنید.

```
public partial class App : Application
 {
        public App()
        {
            this.Startup += this.Application_Startup;
            InitializeComponent();
        }

        private void Application_Startup(object sender, StartupEventArgs e)
        {
           var bootstrapper = new TheBootstrapper();
            bootstrapper.Run();
        }
}
```

با کلیک بر روی ماژول عملیات راهبری برای ماژول انجام خواهد شد.



دریافت سورس پروژه

ادامه دارد..

نظرات خوانندگان

نویسنده: javad
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۲:۱

سلام
اگه می‌شه آموزش استفاده از Entity Framwork در prism را نیز قرار دهید . می‌خوام ماژول‌های مختلف از یک دیتا بیس استفاده کنند و یک EF مشترک داشته باشند ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۳:۱۰

بسیار ساده است. شما نیاز به طراحی یک UnitOfWork بر اساس EF دارید(^). بعد از آن کافیست کدهای مورد نظر برای عملیات CRUD رو در ViewModelهای هر ماژول بنویسید. در پروژه‌های Silverlight هم می‌تونید از RIA Service و EF استفاده کنید. سعی می‌کنم در صورت داشتن زمان کافی یک پست را به این مطلب اختصاص بدم.

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۶:۳۴

سلام . دستتون درد نکنه اقای پاکدل . فقط یه چیزی!
یکی اینکه این اموزشتونو اگه میشه یکم سریعتر بدید . اون روش قبلیه که گفتید رو من خوندم خیلی واضح‌تر توضیح داده بودین . اما از این یکی زیاد نمیتونم درکش کنم.
اگه میشه لطفا رو یه ساختار کنین . یعنی مثلا همین Prism رو با همون الگویه MVVM ای که داره تویه WPF بگین که ما هم بتونیم استفاده کنیم . شما یکی شو با یه روش، یکی دیگشو با یه روشه دیگه و باز اینارو هر کدوم یکی تو Silver و اون یکی تو WPF . این نظر منه . اگه شما یه دونشونو انتخاب کنید و همینطوری ادامه بدین بهتره که ما هم بتونیم برای خودمون یه جمع بندی و یه راه مشخص پیدا کنیم . سایت واقعا عالی دارین . خیلی چیزا من از این سایت یاد گرفتم . این ماژولار بودن تو این سبک و تا این سطح خیلی برام کاربردی و مهمه . میخوام پایه پروژه‌های شرکتو بر همین روال قرار بدم . اگه میشد شما از همین Prism و این MEF یه پروژه WPF بسازین فقط یکی دوتا ماژول ساده براش پیاده سازه کنین و یه فیلم بگیرین خیلی ممنون میشم . میخوام ا این روش استفاده کنم اما روال کار برام مبهمه . اگه کتاب یا سری آموزشی در این باره هم دارین بزارین ما استفاده کنیم . اموزش هاتونم من هر روز میام میخونم و چک میکنم اما خیلی دیر دیر مطلب میزارین . حتما این اموزشو ادامه بدین . مخصوصا Prism With MEF In WPF . خلیی باحالین....

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۱۵

سلام . خسته نباشید . من اگه بخوام تمام ماژول‌ها به صورت داینامیک از تو یک فولدر بخونه باید چیکار کرد. داخل WPF از کلاس DirectoryCatalog استفاده میشه کرد . اما برای سیلورلایت این کلاس وجود نداره . اگه میشه راهنمایی بفرمایین .

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۲۸

ابتدا اسمبلی System.ComponentModel.Composition را به پروژه خود اضافه نمایید. در فضای نام System.ComponentModel.Composition.Hosting کلاس DirectoryCatalog موجود است.

نویسنده: imo0

تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۴۲

با تشکر ولی به نظر سیلورلایت نداره . لطفا اینجا رو یه چک بکنید . نوشته که

"**Note:** DirectoryCatalog is not supported in Silverlight ".

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۲۵ ۲۲:۴۴

در همون لینکی که دادید یک پیاده سازی کمکی ذکر شده: A DirectoryCatalog class for Silverlight

DeploymentCatalog هم هست