Prism For Windows Runtime

J. ROMAGNY

Page CodePlex, Exemple , documentation de l'exemple

Ne pas confondre « **Prism** » (dédié aux applications WPF dans sa version 5.0) et **Prism For Windows Runtime** (dédié aux applications du Windows Store).

1. Création d'un projet « Prism For Windows Runtime »

Créer un projet vide (par exemple).

Installer « **Prism.StoreApps** » avec les packages **NuGet**.

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

Installer «**Unity** » avec les packages **NuGet**

```
PM> Install-Package Unity
```

2. Convention

Dossiers

- ➔ Les « Views » dans un dossier « Views » (namespace « *.Views »)
- ➔ Les « ViewModels » dans un dossier « ViewModels » (namespace « *.ViewModels »)

Noms

- Les « Views » se terminent avec « Page » (exemple : MainPage)
- Correspondance de nom de « View » et « ViewModel » : le nom du ViewModel d'une vue = nom de la vue + « ViewModel ».
  Exemple pour « MainPage » le ViewModel » se nomme « MainPageViewModel »

Héritage

- ✓ Faire hériter « **App** » de « **MvvmAppBase** »
- ✓ Faire hériter les **pages** de « **VisualStateAwarePage** »
- ✓ Faire hériter les « **ViewModels** » de la classe de base « **ViewModel** »
- ✓ Faire hériter les « **Models** » de la classe « **BindableBase** » (pour la notification)

3. « App »

```
<prism:MvvmAppBase
    …
    xmlns:prism="using:Microsoft.Practices.Prism.Mvvm">

</prism:MvvmAppBase>
```

```
using Microsoft.Practices.Prism.Mvvm;
using System;
using System.Threading.Tasks;
using Windows.ApplicationModel.Activation;
using Windows.UI.Xaml;

namespace PrismForWindowsRuntimeDemo
{

    sealed partial class App : MvvmAppBase
    {

        protected override Task OnLaunchApplicationAsync(LaunchActivatedEventArgs args)
        {
            if (args != null && !string.IsNullOrEmpty(args.Arguments))
            {
                //
            }
            else
            {
                NavigationService.Navigate("Main", null);
            }

            Window.Current.Activate();
            return Task.FromResult<object>(null);
        }
    }
}
```

Faire hériter « App » de « MvvmAppBase »

Se produit au clic sur une vignette secondaire. On pourrait naviguer vers une page détails en passant le paramètre

a. Injection de dépendance

```
sealed partial class App : MvvmAppBase
{
        IUnityContainer container = new UnityContainer();

        protected override Task OnInitializeAsync(IActivatedEventArgs args)
        {
            container.RegisterInstance<INavigationService>(this.NavigationService);

            return base.OnInitializeAsync(args);
        }

        protected override object Resolve(Type type)
        {
            return container.Resolve(type);
        }

        // etc.
}
```

Utilisation d'un conteneur Unity

Enregistrement des services et instances dans « OnInitialize »

Permet de résoudre les instances à injecter

Exemple

Injection dans le constructeur d'un « ViewModel »

```
public class MainPageViewModel : ViewModel
{
        private INavigationService _navigationService;

        public MainPageViewModel(INavigationService navigationService)
        {
            _navigationService = navigationService;
        }
}
```

b. Paramètres d'application

Apparaissent dans le volet Windows « **paramètres** » pour l'application. Dans « App »

```
protected override IList<SettingsCommand> GetSettingsCommands()
{
            var commands = new List<SettingsCommand>();

            commands.Add(new SettingsCommand(Guid.NewGuid().ToString(), "Un flyout",
                                                    (handler) => new SettingsFlyout1().Show()));
            commands.Add(new SettingsCommand(Guid.NewGuid().ToString(), "Un lien",
                            async (handler) => await Launcher.LaunchUriAsync(new
                                                                    Uri("http://www.msn.com/"))));

            return commands;
}
```

4. Pages

Les pages sont à ajouter dans le dossier « **Views** »

```
<prism:VisualStateAwarePage
    …
     xmlns:prism="using:Microsoft.Practices.Prism.StoreApps">

     <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

     </Grid>
</prism:VisualStateAwarePage>
```

Hérite de « VisualStateAwarePage »

```
public sealed partial class MainPage : VisualStateAwarePage
{
     // etc.
}
```

Ou simplement

```
public sealed partial class MainPage
{
     // etc.
}
```

ViewModelLocator

Prism retrouvera automatiquement le VIewModel de la page (si la convention est respectée).A indiquer sur toutes les pages ayant besoin de retrouver son « ViewModel »

```
<prism:VisualStateAwarePage
     xmlns:prism="using:Microsoft.Practices.Prism.StoreApps"
     xmlns:prismmvvm="using:Microsoft.Practices.Prism.Mvvm"
     prismmvvm:ViewModelLocator.AutoWireViewModel="True">

     <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

     </Grid>
</prism:VisualStateAwarePage>
```

Modifier la convention. « App »

```
protected override Task OnInitializeAsync(IActivatedEventArgs args)
{
      // enregistrement des services dans le conteneur Unity

      ViewModelLocationProvider.SetDefaultViewTypeToViewModelTypeResolver((viewType) =>
      {

           var viewModelTypeName = string.Format(CultureInfo.InvariantCulture,
"PrismDemo.ViewModels.{0}ViewModel, PrismDemo, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null",
viewType.Name);
           var viewModelType = Type.GetType(viewModelTypeName);
           return viewModelType;
       });

      return base.OnInitializeAsync(args);
}
```

On reçoit le type de la vue

Retourner le type du « ViewModel » correspondant

5. Models

```
public class User : BindableBase
{
        // etc.

        private string _email;
        public string Email
        {
            get { return _email; }
            set { SetProperty(ref _email, value); }
        }
}
```

Méthode permettant de modifier la valeur et notifier le changement de valeur de la propriété

On peut également utiliser « **OnPropertyChanged** » pour notifier d'un changement de valeur de propriété

```
OnPropertyChanged("FullName");
```

6. ViewModels

```
public class MainPageViewModel : ViewModel
{
        // etc.
}
```

a. Commandes

Utilisation de « **DelegateCommand** »

```
public ICommand MyCommand { get; private set; }
```

```
MyCommand = new DelegateCommand(() =>
{

});
```

Commande avec Condition d'exécution

```
public ICommand MyCommandWithCanExecute { get; private set; }
```

```
MyCommandWithCanExecute = new DelegateCommand(() =>
            {

            }, () =>
            {
                return true;
            });
```

Evaluer la condition d'exécution

Utiliser la méthode « **RaiseCanExecuteChanged** » pour déclencher l'évaluation d'exécution de la commande

```
MyCommandWithCanExecute.RaiseCanExecuteChanged();
```

Commande avec paramètre

```
public DelegateCommand<string> MyCommandWithParameter { get; private set; }
```

```
MyCommandWithParameter = new DelegateCommand<string>((parameter) =>
            {

            });
```

Composite commande

```
var command1 = new DelegateCommand(Save, CanSave);
var  command2 = new DelegateCommand(Save, CanSave);

var compositeCommand = new CompositeCommand();
compositeCommand.RegisterCommand(command1);
compositeCommand.RegisterCommand(command2);
```

La commande composite ne peut s'exécuter que si les commandes enregistrées le peuvent

Exécution

```
compositeCommand.Execute("");
```

b. Behaviors Blend (comportements)

Utiliser les comportements Blend pour certains cas, exemple « InvokeCommandAction » qui appellera la commande associée du ViewModel pour le clic sur un élément de ListView ou GridView

```
<ListView x:Name="itemsListView" ItemsSource="{Binding Items}">
       <Interactivity:Interaction.Behaviors>
           <Core:EventTriggerBehavior EventName="SelectionChanged">
               <Core:InvokeCommandAction Command="{Binding GoToCurrentCommand}"
                         CommandParameter="{Binding ElementName=itemsListView,Path=SelectedItem}"/>
             </Core:EventTriggerBehavior>
          </Interactivity:Interaction.Behaviors>
</ListView>
```

Namespaces ajoutés

```
xmlns:Interactivity="using:Microsoft.Xaml.Interactivity"
xmlns:Core="using:Microsoft.Xaml.Interactions.Core"
```

c. Navigation

Utilisation de « **INavigationService** » (« FrameNavigationService » l'implémentant) **que l'on injecte** dans les « **ViewModels** » ayant besoin de l'utiliser.

Navigation sans paramètre

```
_navigationService.Navigate("Second",null);
```

« pageToken » exemple pour « SecondPage » on n'indique que « Second »

Navigation avec passage de paramètre

```
_navigationService.Navigate("Second", "Mon paramètre");
```

Navigation retour (bouton retour par exemple)

```
_navigationService.GoBack();
```

Exemple de « ViewModel » implémentant la navigation vers une autre page

```
public class MainPageViewModel : ViewModel
{
        public ICommand GoToSecondPageCommand { get; private set; }

        private INavigationService _navigationService;

        public MainPageViewModel(INavigationService navigationService)
        {
            _navigationService = navigationService;

            GoToSecondPageCommand = new DelegateCommand(() =>
            {
                _navigationService.Navigate("Second", "Mon paramètre");
            });
        }
}
```

Injection de « navigationService »

Récupération du paramètre dans le ViewModel de la page vers laquelle naviguer.

```
public class SecondPageViewModel : ViewModel
{
        public override void OnNavigatedTo(object navigationParameter, NavigationMode navigationMode,
                                                                    Dictionary<string, object> viewModelState)
        {
           base.OnNavigatedTo(navigationParameter, navigationMode, viewModelState);

            if (navigationParameter != null)
            {

            }
        }

        public override void OnNavigatedFrom(Dictionary<string, object> viewModelState,
                                                                                bool suspending)
        {
            base.OnNavigatedFrom(viewModelState, suspending);

        }
}
```

Récupération du paramètre passé dans « OnNavigatedTo » du « ViewModel »

7. Cycle de vie

a. PageState, sauvegarder les états de la page dans le code-behind

```
<CheckBox x:Name="checkBox1">Case à cocher avec pageState</CheckBox>
```

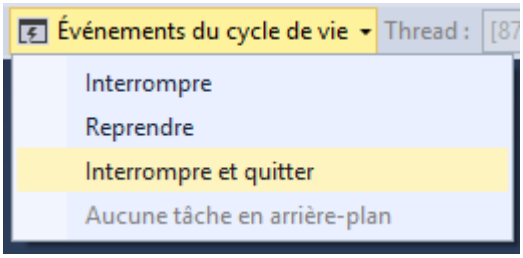
```
public sealed partial class SecondPage
{
        // etc.
        protected override void SaveState(Dictionary<string, object> pageState)
        {
            if (pageState == null) return;

            base.SaveState(pageState);

            pageState["checkBox1CheckedState"] = checkBox1.IsChecked;
        }
        protected override void LoadState(object navigationParameter, Dictionary<string, object>
pageState)
        {
            base.LoadState(navigationParameter, pageState);

            if (pageState!=null && pageState.ContainsKey("checkBox1CheckedState"))
            {
                checkBox1.IsChecked = (bool)pageState["checkBox1CheckedState"];
            }
        }
}
```

Sauvegarde de l'état de la case à cocher lorsque l'application est suspendue

Restauration des états de la page au relancement de l'application

Test

Lancer l'application, naviguer vers la page, changer les valeurs des champs puis « Interrompre et quitter » … relancer l'application, normalement la page sur laquelle on était avant d'interrompre est affichée et les états rechargés



b. RestorableState, attribut pour les propriétés de « ViewModels »

Dans un « **ViewModel** »

```
[RestorableState]
public bool IsChecked { get; set; }
```

On lie par exemple une case à cocher dans la page

```
<CheckBox IsChecked="{Binding IsChecked,Mode=TwoWay}">Case à cocher avec attribut
RestorableState</CheckBox>
```

c. ViewModelSate, pour sauver des états du « ViewModel »

```
public class DetailsPageViewModel : ViewModel
{

        public string Message { get; set; }
        public SampleDataItem CurrentItem { get; set; }

        // constructeur, etc.

        public override void OnNavigatedTo(object navigationParameter, NavigationMode navigationMode,
Dictionary<string, object> viewModelState)
        {
            base.OnNavigatedTo(navigationParameter, navigationMode, viewModelState);

            if (navigationMode == NavigationMode.Refresh)
            {
                if (viewModelState.ContainsKey("Message"))
                {
                    Message = (string)viewModelState["Message"];
                    OnPropertyChanged("Message");
                }
                if (viewModelState.ContainsKey("CurrentItem"))
                {
                    CurrentItem = (SampleDataItem)viewModelState["CurrentItem"];
                    OnPropertyChanged("CurrentItem");
                }
            }

        }

        public override void OnNavigatedFrom(Dictionary<string, object> viewModelState, bool
suspending)
        {
            base.OnNavigatedFrom(viewModelState, suspending);

            if (suspending)
            {
                viewModelState.Add("Message", Message);
                viewModelState.Add("CurrentItem", CurrentItem);
            }
        }
}
```

« NavigationMode » :

- New : premier chargement
- Back : un retour
- Forward
- Refresh

Récupération des états du « ViewModel »

Enregistrement de variable

Enregistrement d'objet

```
public class SampleDataItem
{
        public SampleDataItem() { }
        public SampleDataItem(String title)
        {
            this.Title = title;

        }
        public string Title { get; set; }

        // etc.
}
```

Constructeur par défaut requis pour la sérialisation

Dans la page

```
<TextBox Text="{Binding Message,Mode=TwoWay}" />
<TextBox Text="{Binding CurrentItem.Title,Mode=TwoWay}" />
```

Activer le mode « TwoWay » pour reporter les modifications des champs

Dans « App »

```
protected override void OnRegisterKnownTypesForSerialization()
{
            SessionStateService.RegisterKnownType(typeof(SampleDataItem));

            base.OnRegisterKnownTypesForSerialization();
}
```

Enregistrement de la classe pour la sérialisation de l'objet lors de la suspension

d. SessionStateService

« App »

```
sealed partial class App : MvvmAppBase
{
        IUnityContainer container = new UnityContainer();

        protected override Task OnInitializeAsync(IActivatedEventArgs args)
        {
            // autres services enregistrés
            container.RegisterInstance<ISessionStateService>(this.SessionStateService);

            return base.OnInitializeAsync(args);
        }
        // etc.
}
```

Enregistrement de « SessionStateService » dans le conteneur

..Injecter ISessionStateService dans les services, etc. l'utilisant

Enregistrement

```
_sessionStateService.SessionState.Add("CurrentItem", CurrentItem);
```

Récupération

```
if (_sessionStateService.SessionState.ContainsKey("CurrentItem"))
{
     CurrentItem = (SampleDataItem)_sessionStateService.SessionState["CurrentItem"];
}
```

Requiert également d'enregistrer les classes pour la sérialisation dans « App »

8. Echange de messages entre « ViewModels » ,etc. (« PubSubEvents »)

Documentation

« Prism.PubSubEvents » est une bibliothèque portable (PCL) de « **Prism** »

Installer avec les packages **NuGet**

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

Créer une classe héritant de « PubSubEvent »

Type du paramètre passé

```
public class MyEvent : PubSubEvent<string> { }
```

« App»
Enregistrement de l'eventAggregator

```
container.RegisterType<IEventAggregator, EventAggregator>(new ContainerControlledLifetimeManager());
```

Injecter « IEventAggregator » dans les « ViewModels », etc. l'utilisant

Abonnement

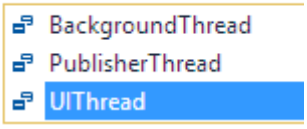
```
_eventAggregator.GetEvent<MyEvent>().Subscribe((parameter) =>
{
    //
});
```

Publication

```
_eventAggregator.GetEvent<MyEvent>().Publish("Mon message!");
```

ThreadOption

```
ThreadOption.);
```



keepSubscriberReferenceAlive (false par défaut)

Filtre

```
_eventAggregator.GetEvent<MyEvent>().Subscribe((parameter) =>
            {
                //
            }, ThreadOption.UIThread, false,
            (message) =>
            {
                //
                return true;
            });
```