DEVEXPRESS WPF MVVM FRAMEWORK(DXMVVM)을 이용하여 MVVM을 지켜보자

닷넷데브 VINCENT

MVVM(MODEL-VIEW-VIEWMODEL)

MVVM(MODEL-VIEW-VIEWMODEL) (1)

- Mvvm은 WPF를 만든 Microsoft에서 처음 생겨났다.
- MVVM은 많은 WPF 개발자들 또는 WPF를 시작하고자 하는 사람들이 궁금해하는 것이고 도대체 어떻게 하는지 같은 질문들이 생겨난다.
- 내가 생각하는 MVVM은 MODEL이 들어가는, 객체지향설계 중 책임주도설계에 기반한 확장형이다.
- MVVM을 지켜서 얻는 이득이 무엇인지는 검색하면 자료가 많이 있다.
- 다음 장에서 내가 생각하는 MVVM에 대한 FAQ를 나열했다.

MVVM(MODEL-VIEW-VIEWMODEL) (2)

- ViewModel과 XAML이 Binding으로 상호작용하면 MVVM 이다.
  - Vincent: 단순히 ViewModel과 XAML이 Binding 하는 것은 WPF의 Binding 기능을 이용하는 것이고, 더 나아가 ViewModel와 View이 서로 의존관계를 가져서는 안되는 것이 핵심이다. 하지만 개발 스타일에 Trade Off는 존재하므로 팀원의 합의간 적당히 하면 된다.
- View (UserControl, Page, Window)의 Code-Behind에 소스코드가 들어있으면 MVVM이 아니다.
  - Vincent: MVVM과 Zero-Code-Behind를 관련 지어서는 안된다. 그것은 그냥 개발 Style, 또는 개발 정책인 것이고, View가 View의 기능을 하기 위하여 Code Behind에 Event를 사용하는 것은 MVVM 위반이 아니다.
- ViewModel에서 MessageBox를 사용하면 안된다.
  - Vincent: ViewModel은 View를 위한 Model이다. 즉, 그냥 .NET의 객체이므로 Rendering이 필요한 MessageBox는 사용해서는 안된다. 그래서 ViewModel이 못하는 이런 역할을 Service 객체가 옆에서 도와주는 것이다. ViewModel과 View의 의존성을 논하는 것이 MVVM이지, Service 객체는 이런 역할을 할 수 있다. 그래서 이런 Service 객체를 접근성이 좋도록 MVVM에서 있으면 유용한 것이 IoC Container다.
- MVVM을 잘 지키려면 어떻게 해야할까?
  - Vincent: Xunit, Nunit 등의 Unit Test Framework 를 통해 ViewModel을 Test 가능하도록 개발한다. Prism의 경우 Visual Studio 확장을 설치하면 Template에 'Prism Full App' 같은 것을 이용하면 도움을 받을 수 있다.

DEVEXPRESS WPF MVVM FRAMEWORK

DEVEXPRESS WPF MVVM FRAMEWORK

- DevExpress는 해외 .NET 3rd Party 개발 업체로 국내에서 인지도가 높다.
- DevExpress와 비슷한 업체는 GraphCity, Infragistics, SyncFusion, Telerik 등이 있다.
- 3rd Party 인 만큼 유료 .NET Windows Control을 개발하는 곳이지만, MVVM Framework는 MIT License로 풀려있고, Nuget에서 받아서 사용할 수 있다.
- MIT License: https://community.devexpress.com/blogs/wpf/archive/2014/06/11/free-devexpress-mvvm-framework-released.aspx
- Nuget: https://www.nuget.org/packages/DevExpressMvvm/

왜 DXMVVM을 사용할까?

- WPF MVVM Framework에는 가장 유명한 mvvmlight이 있고 이 밖에도 caliburn, mvvm toolkit 등이 있다.
- 여러 Framework 간 호불호가 갈리는 여러가지 이유가 있을테지만 dxmvvm 같은 경우 POCOViewModel Style로 생산성을 증가시킬 수 있어서 사용한다. 또한 mvvm framework들의 용어를 정립한 mvvmlight와 용어와 개념이 유사하여 접근성이 좋다.
- 이후 자료에서 모두 POCOViewModel Style로 예제를 소개함.
- POCOViewModel: https://docs.devexpress.com/WPF/17352/mvvm-framework/viewmodels/runtime-generated-poco-viewmodels

POCOVIEWMODEL

DXMVVM POCOVIEWMODEL

- ViewModel로 사용할 Class에 POCOViewModel Attribute를 붙이는 것으로 적용가능.
- https://docs.devexpress.com/WPF/17353/mvvm-framework/commands/delegate-commands#poco

```
[POCOViewModel]
public class ViewModel {
    public void Save(string fileName) {
        //...
    }
    public bool CanSave(string fileName) {
        return !string.IsNullOrEmpty(fileName);
    }
}
```

POCOVIEWMODEL-PROPERTY

DXMVVM POCOVIEWMODEL-PROPERTY(1)

- Dxmvvm에서 pocoviewmodel에서 INotifyPropertyChanged 인터페이스가 구현된 Property를 사용하려면 일반 property 와 똑같이 정의하되, 반드시 virtual 키워드를 붙여서 정의.Virtual 키워드를 붙이지 않으면 Notify가 발생하지 않는 일반 Property.
- Property가 변경됨에 따라 Callback Method로 동작을 제어하고 싶은 경우 아래와 같이 적용 가능하다.
  - 값이 적용되기 전 Callback Method: 접두사로 On, 접미사로 Changing
  - 값이 적용된 후 Callback Method: 접두사로 On, 접미사로 Changed
- 위 방법으로 정의된 Callback Method의 Parameter로 들어오는 것이 변경되는 property 값이다. Changing Method의 경우 현재 Property Value, Changed Method의 경우 바뀔 Property Value.

DXMVVM POCOVIEWMODEL-PROPERTY(2)

- 정의가 끝나면 아래 그림같이 되며, 필요한 Callback Method만 정의해도 된다.
- Callback Method Naming이 마음에 들지 않는다면 Custom하게 바꿀 수 있고, dxmvvm 공식 문서를 참고하면 된다.
- https://docs.devexpress.com/WPF/17352/mvvm-framework/viewmodels/runtime-generated-poco-viewmodels#bindableproperties

```
public class LoginViewModel {
    public virtual string UserName { get; set; }
    protected void OnUserNameChanged(string oldValue) {
        //...
    }
    protected void OnUserNameChanging(string newValue) {
        //...
    }
}
```

POCOVIEWMODEL-COMMAND

DXMVVM POCOVIEWMODEL-COMMAND(1)

- ViewModel에 속해 있는 Method는 기본적으로 모두 Command로 간주하게 됨.
- ICommand의 CanExecute기능을 이용하려면 아래 사진 처럼 Command Method명과 일치하는 함수에 Can 접두사를 붙이고 Return Type을 bool로 지정하면 해당 Command에 대해 CanExecute Method로 동작.
- CanExecute Method가 필요없다면 굳이 정의하지 않아도 무방함.

DXMVVM POCOVIEWMODEL-COMMAND(2)

- ViewModel에 속해 있는 Method중 Command로 만들기를 원치 않는 Method가 있다면 Method에 Command Attribute를 적용하여 isCommand를 false로 준다.
- Runtime에서 Command로 만들어진 Method는 XAML에서 Method명 + 접미사로 Command를 붙혀서 Binding 할 수 있다. 예를 들어 아래 그림의 Save Method는 [Command(isCommand: false)] 가 붙지 않는 메서드이므로 Runtime에서 SaveCommand 메서드로 변경되므로 XAML에서 SaveComand로 바인딩 해야한다.

```
public class LoginViewModel {
    [Command(isCommand: false)]
    public void SaveCore() {
        //...
    }
}
```

```
[POCOViewModel]
public class ViewModel {
    public void Save(string fileName) {
        //...
    }
    public bool CanSave(string fileName) {
        return !string.IsNullOrEmpty(fileName);
    }
}
```

POCOVIEWMODEL-EVENTTOCOMMAND

DXMVVM POCOVIEWMODEL-EVENTTOCOMMAND

- WPF의 Control에 Event가 발생했을 경우 ViewModel의 Command가 동작하게 만들고 싶다면 EventToCommand를 사용하면 된다. Behavior 방식도 가능하지만 이번 세션에서는 시간 상의 이유로 다루지 않는다.
- XAML에서 EventToCommand를 사용하고 싶은 Control에서 아래 그림과 같이 코딩한다.
- 혹시 내가 사용하고 싶은 Control의 Event가 어떤 것들이 있는지 모를 때는 Visual Studio에서 XAML에 원하는 컨트롤을 적은 후 속성-이벤트에서 이벤트 목록을 확인한다.
- https://docs.devexpress.com/WPF/DevExpress.Mvvm.UI.EventToCommand

```
<UserControl>
    <dxmvvm:Interaction.Behaviors>
        <dxmvvm:EventToCommand EventName="Loaded" Command="{Binding InitializeCommand}"/>
    </dxmvvm:Interaction.Behaviors>
</UserControl>
```

DXMVVM POCOVIEWMODEL-KEYTOCOMMAND

- WPF KeyGesture에 따라 특정 Control에서 Command를 발생시키고 싶은 경우 KeyToCommand를 이용하며, 역시 접미사로 Command가 붙어 있은 POCOViewModel의 Method를 Binding한다.
- https://docs.devexpress.com/WPF/DevExpress.Mvvm.UI.KeyToCommand

MESSENGER

DXMVVM POCOVIEWMODEL-MESSENGER(1)

- Mvvmlight에도 messenger라는 이름으로 있는 이 기능은 ViewModel간 Event Aggregator 방식으로 데이터 교환할 때 사용한다.
- Event Aggregator는 메시지를 보낼 때 구독되고 있는 모든 Callback Method에게 Broadcasting 되는 방식이고 특정 ViewModel에게 전송할 수 없기 때문에 원시자료형을 사용하여 Messenger를 사용하면 안된다. 따라서 매번 Type을 만들어서 Message를 구분해야 한다. Dxmvvm 예제에서는 원시자료형을 사용하고 있지만 실제로 그렇게 하면 안된다.
- https://docs.devexpress.com/WPF/17474/mvvm-framework/messenger#default-messenger

DXMVVM POCOVIEWMODEL-MESSENGER(2)

- Dxmvvm의 링크된 샘플대로 진행하면 된다. App 생성자에서 Default Messenger를 생성한다.

```
public class App : Application {
    public App() {
        Messenger.Default = new Messenger(isMultiThreadSafe: true, actionReferenceType:
ActionReferenceType.WeakReference);
    }
}
```

- Rergister로 구독할 메시지의 Type을 등록하고, Send로 Message를 보내면 된다.

```
public static void Register<TMessage>(this IMessenger messenger, object recipient, Action<TMessage> action);
public static void Send<TMessage>(this IMessenger messenger, TMessage message);
```

DXMVVM POCOVIEWMODEL-MESSENGER(3)

- Recipient의 생성자에서 string과 MyMessage Type을 구독하는 Callback Method를 만든다.
- OnMessage1은 string Type을 구독한다.
- OnMessage2는 MyMessage Type을 구독한다.
- SendMessages Method를 호출하면 "test" Type의 string이 전달되고 new MyMessage() 객체가 전달된다.
- 전달했던 순서가 있기 때문에 OnMessage1과 OnMessage2는 순차적으로 발생하지만, 동기방식이 아니기 때문에 SendMessage Method는 기다리지 않고 메시지만 전송한다.
- OnMessage1과 OnMessage2가 처리된다.

```
public class MyMessage {
    //...
}
public class Recipient {
    public Recipient() {
        // Receives messages of the `string` type.
        Messenger.Default.Register<string>(this, OnMessage1);
        // Receives messages of a custom `MyMessage` type.
        Messenger.Default.Register<MyMessage>(this, OnMessage2);
    }
    void SendMessages() {
        // Sends messages of the `string` type.
        Messenger.Default.Send("test");
        // Sends messages of a custom `MyMessage` type.
        Messenger.Default.Send(new MyMessage());
    }
    void OnMessage1(string message) {
        //...
    }
    void OnMessage2(MyMessage message) {
        //...
    }
}
```

감사합니다