Mouse Wheel direction

**Reference musi byt v Gui i ve spoustecim projektu (Shell)**

MVVM

This solution doesn't play very well with the MVVM design pattern though as the application logic ends up in the code-behind of the view instead of in the view model where it belongs. Handling events in MVVM is typically done using commands and interaction triggers and I have written a blog post about this if you want the full story: [https://blog.magnusmontin.net/2013/06/30/handling-events-in-an-mvvm-wpf-application/ [Jump](https://blog.magnusmontin.net/2013/06/30/handling-events-in-an-mvvm-wpf-application/)](https://blog.magnusmontin.net/2013/06/30/handling-events-in-an-mvvm-wpf-application/).

If you are using a MVVM Library such as Prism or MvvmLight, you could easily pass the MouseWheelEventArgs as a parameter to your command using a System.Windows.Interactivity.TriggerAction. Here is how to do it using Prism's InvokeCommandAction in three easy steps:

1. Install the Prism.Wpf package using NuGet (Tools->NuGet Package Manager->Package Manager Console):[https://www.nuget.org/packages/Prism.Wpf/ [Jump](https://www.nuget.org/packages/Prism.Wpf/)](https://www.nuget.org/packages/Prism.Wpf/)

2. Define a DelegateCommand<MouseWheelEventArgs> property in your view model class and an action to be invoked when the command is being executed:

**public** **class** ViewModel

{

**public** IEnumerable<**int**> Items { **get**; } = Enumerable.Range(0, 100);

**private** ICommand \_mouseWheelCommand;

**public** ICommand MouseWheelCommand => \_mouseWheelCommand = \_mouseWheelCommand ?? **new**DelegateCommand<MouseWheelEventArgs>(e => MouseWheelCommandExecute(e));

**private** **void** MouseWheelCommandExecute(MouseWheelEventArgs e)

    {

**if** (e.Delta > 0)

        {

            //scrolled up...

        }

**else** **if** (e.Delta < 0)

        {

            //scrolled down...

        }

    }

}

3. Use the InvokeCommandAction in your view:

<**ListBox** ItemsSource="{Binding Items}"

            xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"

            xmlns:cmd="[http://prismlibrary.com/ [Jump](http://prismlibrary.com/)](http://prismlibrary.com/)">

    <**i:Interaction.Triggers**>

        <**i:EventTrigger** EventName="PreviewMouseWheel">

            <**cmd:InvokeCommandAction** Command="{Binding MouseWheelCommand}" />

        </**i:EventTrigger**>

    </**i:Interaction.Triggers**>

</**ListBox**>

That's basically all you need to do. There is one issue with this approach though; the view model class has a dependency upon the view-related System.Windows.Input.MouseWheelEventArgs type and this kind of breaks the important MVVM principle of separation between view models and views. If you have defined your view model classes in a separate class library, this also means that you will have to add a reference to PresentationCore.dll from this project.

You could get around this by creating your own custom non-view related class that contains just a Delta property:

**public** **class** MyMouseWheelInfo

{

**public** MyMouseWheelInfo(**int** delta)

    {

        Delta = delta;

    }

**public** **int** Delta { **get**; }

}

You then specify this custom type as the type argument of DelegateCommand property in the view model:

**private** ICommand \_mouseWheelCommand;

**public** ICommand MouseWheelCommand => \_mouseWheelCommand = \_mouseWheelCommand ?? **new** DelegateCommand<MyMouseWheelInfo>(e => MouseWheelCommandExecute(e));

Finally you will also need to create a custom System.Windows.Interactivity.TriggerAction that creates an instance of the custom MyMouseWheelInfo class and passes it to the command when the MouseWheel event is being raised:

**namespace** Mm.MvvmMouseWheelExample.Desktop

{

**public** **class** MouseWheelInvokeCommandAction : InvokeCommandAction

    {

**protected** **override** **void** Invoke(**object** parameter)

        {

            MouseWheelEventArgs mouseWheelEventArgs = parameter **as** MouseWheelEventArgs;

**if** (mouseWheelEventArgs != **null**)

**base**.Invoke(**new** MyMouseWheelInfo(mouseWheelEventArgs.Delta));

**else**

**base**.Invoke(parameter);

        }

    }

}

<**ListBox** ItemsSource="{Binding Items}"

            xmlns:i="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"

            xmlns:local="clr-namespace:Mm.MvvmMouseWheelExample.Desktop">

    <**i:Interaction.Triggers**>

        <**i:EventTrigger** EventName="PreviewMouseWheel">

            <**local:MouseWheelInvokeCommandAction** Command="{Binding MouseWheelCommand}" />

        </**i:EventTrigger**>

    </**i:Interaction.Triggers**>

</**ListBox**>

With these changes in place, the view model can now easily handle the MouseWheel event without being coupled to the view in any way.

The MyMouseWheelInfo class should be defined in the view model project and the MouseWheelInvokeCommandAction should be created in the project where the view resides.