Передача обработчика

xmlns:command="http://www.galasoft.ch/mvvmlight"

<i:Interaction.Triggers>

<i:EventTrigger EventName="Closing">

<command:EventToCommand Command="{Binding Closing}" PassEventArgsToCommand="True"></command:EventToCommand>

</i:EventTrigger>

</i:Interaction.Triggers>

Behaviors

As you are using MVVM, you can use a Behavior for this:

public class TabOnEnterBehavior : Behavior<TextBox>  
{  
  
 protected override void OnAttached()  
 {  
 AssociatedObject.PreviewKeyDown += AssociatedObject\_PreviewKeyDown;  
 }  
  
 private void AssociatedObject\_PreviewKeyDown(object sender, KeyEventArgs e)  
 {  
 if (e.Key == Key.Enter)  
 {  
 var request = new TraversalRequest(FocusNavigationDirection.Next);  
 request.Wrapped = true;  
 AssociatedObject.MoveFocus(request);  
 }  
 }  
  
 protected override void OnDetaching()  
 {  
 AssociatedObject.PreviewKeyDown -= AssociatedObject\_PreviewKeyDown;  
 }  
  
}

In your xaml:

<TextBox>  
 <i:Interaction.Behaviors>  
 <wpfTest:TabOnEnterBehavior />  
 </i:Interaction.Behaviors>  
</TextBox>

Передача данных через регистрацию

1) Register to a message in your ViewModel. I prefer to do this in its constructor. You also have to add a method to handler the callback from the messenger. Please note that "tokenId" use to make sure that you send the message to intend subscribers.

Messenger.Default.Register<double>(this, YourMethodHere, "tokenId");

private void YourMethodHere(double value)

{

// do your work here

}

2) Inside the View, you have to add an event to the button and just send the message through the messenger.

private void Button\_Click(object sender, RoutedEventArgs e)

{

Messenger.Default.Send<double>(this.ActualWidth, "tokenId");

}

3) You can Unregister the subscribtion on the ViewModel in its destructor to prevent mutiple message sending.

EventArgsConverterParameter

Преобразует координаты мышки на Grid-е

**Note:** [The code sample is available as fully executable WPF4.5 and WinStore 8.1 applications.](http://sdrv.ms/1diRfZ7)

**XAML**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <Window.Resources>      <ResourceDictionary>          <helpers:MouseButtonEventArgsToPointConverter x:Key="MouseButtonEventArgsToPointConverter" />      </ResourceDictionary>  </Window.Resources>    <Grid x:Name="LayoutRoot"        Background="#FF760000">        <i:Interaction.Triggers>          <i:EventTrigger EventName="MouseLeftButtonDown">              <mvvm:EventToCommand                  Command="{Binding ShowPositionCommand, Mode=OneWay}"                  EventArgsConverter="{StaticResource MouseButtonEventArgsToPointConverter}"                  EventArgsConverterParameter="{Binding ElementName=LayoutRoot}"                  PassEventArgsToCommand="True" />          </i:EventTrigger>      </i:Interaction.Triggers>        <TextBlock HorizontalAlignment="Center"                 Text="{Binding LastPosition}"                 VerticalAlignment="Center"                 Foreground="White"                 FontFamily="Segoe UI Light"                 FontSize="72"                 IsHitTestVisible="False" />  </Grid> |

**Converter**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class MouseButtonEventArgsToPointConverter : IEventArgsConverter  {      public object Convert(object value, object parameter)      {          var args = (MouseButtonEventArgs)value;          var element = (FrameworkElement)parameter;            var point = args.GetPosition(element);          return point;      }  } |

**ViewModel**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | public class MainViewModel : ViewModelBase  {      public const string LastPositionPropertyName = "LastPosition";        private string \_lastPosition = "Click somewhere";      private RelayCommand<Point> \_showPositionCommand;        public string LastPosition      {          get          {              return \_lastPosition;          }          set          {              Set(() => LastPosition, ref \_lastPosition, value);          }      }        public RelayCommand<Point> ShowPositionCommand      {          get          {              return \_showPositionCommand                      ?? (\_showPositionCommand = new RelayCommand<Point>(                          point =>                          {                              LastPosition = string.Format("{0:N1}, {1:N1}", point.X, point.Y);                          }));          }      }  } |