### 소스코드 정리 ###

1. [TemplatePart(Name = ElementItemsHost, Type = typeof(Panel))]

특정 컨트롤(A)이 TemplatePart 에 있는 특정 컨트롤(B)을 ControlTemplate 에 포함하고 있음을 의미하고 있다. 즉, 제작하고자 하는 특정컨트롤 A 의 구성이 B 컨트롤을 포함하고 있다라는 뜻이다. 이름은 PART\_ 로 시작한다.

1. [StyleTypedProperty(Property = nameof(SelectionRectangleStyle), StyleTargetType = typeof(Rectangle))]

스타일 타입의 속성을 구분해준다.??? 이거 테스트 하거나 실험을 해봐야 한다.

1. DependencyProperty.Register(nameof(Scale), typeof(double), typeof(NodifyEditor), new FrameworkPropertyMetadata(BoxValue.Double1, FrameworkPropertyMetadataOptions.BindsTwoWayByDefault, OnScaleChanged, ConstrainScaleToRange));

프로퍼티 이름, 프로퍼티 타입, 프러퍼티 소유자, 메타데이터 정의.( FrameworkPropertyMetadata)

여기서 메타데이터 정의는 프로퍼티 기본값, 프로퍼티 속성특징을 나타낸다. 여기서는 바인딩은 기본적으로 TwoWay 라고 명시하고 있다.

(<https://docs.microsoft.com/ko-kr/dotnet/api/system.windows.frameworkpropertymetadataoptions?view=net-5.0>)

프로퍼티가 변할때의 처리 함수, 프로퍼티 값을 체크할 때 잘못된 값이 들어올 때 강제적으로 해당 값으로 정할 수 있도록 하는 함수.

1. DataTemplae

데이터 오브젝트의 시각화를 할 수 있도록 한다.

1. Public static readonly DependencyProperty

* Scale, MinScale, MaxScale, Offset (이건 좀더 살펴봐야함.) 등은 어느정도 이해함
* BringIntoViewAnimationDuration 은 이해가 안됨.

#### 소스 분석 중 생각 나는 것 정리 ####

* Connector 는 Control 객체로부터 상속받았다 여기서 사용자 정의 이벤트를 만들어 주기 위해서 ConnectorEventArgs 클래스를 생성해주고 여기서 생성한 OnDisconnect 에서 사용한다.??? 즉, Connector 에서 생성한(또는 다른 클래스와 협력을 통해서 생성한) 이벤트를 만들어 주고, 이것을 NodifyEditor 에서 AddHandler 를 통해서 집어 넣어준다. 왜 AddHandler 를 썼는지 정확히 이유는 좀더 밝혀봐야 할 것 같다.
* Connector 에도 nodifyEditor 객체가 있음??? 무엇인가 anchor 와 관련있는듯한데 좀더 살펴봐야 겠다.
* 01/23
* NodifyEditor 는 MultiSelector(->Selector->ItemControl) 에서 상속을 받았다. 즉, ItemsControl 에서 상속을 받았다. ItemsControl 은 item 과 이것을 시각적으로 데이터와 연결해서 그려주는 ItemContainer 에 대해서 알아야 할 필요가 있는데,

이것은 아래 두함수에서 해당 ItemContainer 를 재구성하여 넣었다. ItemContainer 는 상속해서(public class ItemContainer : ContentControl) 구현한 객체이다.

|  |
| --- |
| protected override DependencyObject GetContainerForItemOverride()  protected override bool IsItemItsOwnContainerOverride(object item) |

그리고,

[TemplatePart(Name = ElementItemsHost, Type = typeof(Panel))]

와

protected const string ElementItemsHost = "PART\_ItemsHost";

와

NodifyEditor.xaml 에서의

<local:NodifyCanvas x:Name="PART\_ItemsHost" IsItemsHost="True" /> 을 통해서 UI 를 구성하고 있다.

NodifyEditor 의 ItemsSource 는 단순히 리스트에 불과하다. 하지만 여기에 들어간 정보들은 추후 <DataTemplate>을 통해서 데이터를 <Node> 로 표현하는 구조를 가졌다. 즉, custom 컨트롤 밖에서 Node 를 바인딩을 통해서 설정할 수 있도록 만들었다.

서비스쪽 코드에서(NodifyEditor 를 사용하는 코드)

public INodifyCommand CreateConnectionCommand { get; } 와

컨스트럭트에서 아래와 같이 선언하고 이것을 delegate 로 연결한다. 바인딩 되는 순간 해당 컨스트럭터가 실행이되고 delegae 함수가 생성된다.

아래 코드에서는 delegate 로 두개의 함수를 연결 시켜주었다. ICommand 에서 Execute 와 CanExecute 에 대응시키기 위함이다.

|  |
| --- |
| CreateConnectionCommand = new DelegateCommand<(object Source, object Target)>(target => CreateConnection((ConnectorViewModel)target.Source, (ConnectorViewModel)target.Target), target => CanCreateConnection((ConnectorViewModel)target.Source, target.Target as ConnectorViewModel)); |

Mouse 이벤트 처리

1. Connector.cs

* **OnMouseLeftButtonUp** 커넥터에 대한 마우스왼쪽 눌렀다가 up 하기, 이때 **OnConnectorDragCompleted**() 함수 실행. 이 함수에서

**PendingConnectionEventArgs** 이벤트 발생 **RaiseEvent(args);** 이때 발생시키는 이벤트는 **PendingConnectionCompletedEvent** 이다.

이 이벤트의 등록은

|  |
| --- |
| public static readonly RoutedEvent PendingConnectionCompletedEvent = EventManager.RegisterRoutedEvent(nameof(PendingConnectionCompleted), RoutingStrategy.Bubble, typeof(PendingConnectionEventHandler), typeof(Connector)); |

Connector.cs 에서 작성되었다.

이때 발생하는 **PendingConnectionCompletedEvent** 이벤트의 처리함수를 찾아보면, PendingConnection.cs 와 NodifyEditor.cs 두곳에서 발견된다.

-PendingConnection

Editor.AddHandler(Connector.PendingConnectionCompletedEvent, new PendingConnectionEventHandler(OnPendingConnectionCompleted), true);

-NodifyEditor.

AddHandler(Connector.PendingConnectionCompletedEvent, new PendingConnectionEventHandler(OnConnectionCompleted));

* **OnConnectorDragCompleted 분석**
* **Connector 를 마우스 왼쪽 버튼을 누른 상태서 움직였을 때,**

OnMouseMove 함수가 호출되고 여기서 Thumb(NodeInput 또는 NodeOutput 에서의 동그란 버튼) 의 위치가 구해지고 이 값이 OnConnectorDrag(Thumb의 위치값 Point) 함수로 넘어간다. 그리고 PendingConnectionEventArgs 를 통해 이벤트 정보를 기입하고 이벤트를 발생시킨다.( RaiseEvent(args) ) 이때, 이 이벤트(PendingConnectionDragEvent) 의 이벤트 처리기는 Editor.AddHandler(Connector.PendingConnectionDragEvent, new PendingConnectionEventHandler(OnPendingConnectionDrag)) 함수에서 처리되고 이함수는 PendingConnection.cs 에서 OnApplyTemplate() 에서 선언되어 있다. 해당 이벤트 처리기는 아래와 같다.

protected virtual void OnPendingConnectionDrag(object sender, PendingConnectionEventArgs e)

* **/// <summary>**

**/// Gets the panel that holds all the <see cref="ItemContainer"/>s.**

**/// </summary>**

**protected internal Panel? ItemsHost { get; private set; }**

-**NodifyEditor ItemHost**

* **Connector.cs 에서는 이벤트 핸들러를 담을 수 있도록 관련 코드도 작성되어 있다. (사용하고 있지 않는듯하다.)**

사실 아래와 같은 코드는 외부에서 이벤트 처리함수를 적용시킬 수 있도록 만들어주는 역할을 하는데, 일단 Connector.cs 에서는 사용확인이 잘 안되고 있다. 해당 컨트롤을 독자적으로 쓰는 경우의 예시를 파악하지 못해서 그런것일 수 있다. **추후 이 내용은 파악해야한다.**

|  |
| --- |
| public event PendingConnectionEventHandler PendingConnectionCompleted  {  add => AddHandler(PendingConnectionCompletedEvent, value);  remove => RemoveHandler(PendingConnectionCompletedEvent, value);  } |

<nodify:NodifyEditor ItemsSource="{Binding Nodes}"

Connections="{Binding Connections}"

ConnectionCompletedCommand="{Binding ConnectionCompletedCommand}">

<nodify:NodifyEditor.ItemTemplate>

<DataTemplate>

<nodify:Node Header="{Binding Title}"

Input="{Binding Input}"

Output="{Binding Output}">

<nodify:Node.InputConnectorTemplate>

<DataTemplate>

<nodify:NodeInput Header="{Binding Title}"

Anchor="{Binding Anchor, Mode=OneWayToSource}"

IsConnected="{Binding IsConnected}" />

</DataTemplate>

</nodify:Node.InputConnectorTemplate>

<nodify:Node.OutputConnectorTemplate>

<DataTemplate>

<nodify:NodeOutput Header="{Binding Title}"

Anchor="{Binding Anchor, Mode=OneWayToSource}"

IsConnected="{Binding IsConnected}" />

</DataTemplate>

</nodify:Node.OutputConnectorTemplate>

</nodify:Node>

</DataTemplate>

</nodify:NodifyEditor.ItemTemplate>

<nodify:NodifyEditor.ConnectionTemplate>

<DataTemplate>

<nodify:Connection Source="{Binding Input.Anchor}"

Target="{Binding Output.Anchor}" />

</DataTemplate>

</nodify:NodifyEditor.ConnectionTemplate>

<nodify:NodifyEditor.ItemContainerStyle>

<Style TargetType="{x:Type nodify:ItemContainer}">

<Setter Property="Location"

Value="{Binding Location}" />

</Style>

</nodify:NodifyEditor.ItemContainerStyle>

</nodify:NodifyEditor>

/// <summary>

/// XAML 파일에서 이 사용자 지정 컨트롤을 사용하려면 1a 또는 1b단계를 수행한 다음 2단계를 수행하십시오.

///

/// 1a단계) 현재 프로젝트에 있는 XAML 파일에서 이 사용자 지정 컨트롤 사용.

/// 이 XmlNamespace 특성을 사용할 마크업 파일의 루트 요소에 이 특성을

/// 추가합니다.

///

/// xmlns:MyNamespace="clr-namespace:NodeInput\_Connector"

///

///

/// 1b단계) 다른 프로젝트에 있는 XAML 파일에서 이 사용자 지정 컨트롤 사용.

/// 이 XmlNamespace 특성을 사용할 마크업 파일의 루트 요소에 이 특성을

/// 추가합니다.

///

/// xmlns:MyNamespace="clr-namespace:NodeInput\_Connector;assembly=NodeInput\_Connector"

///

/// 또한 XAML 파일이 있는 프로젝트의 프로젝트 참조를 이 프로젝트에 추가하고

/// 다시 빌드하여 컴파일 오류를 방지해야 합니다.

///

/// 솔루션 탐색기에서 대상 프로젝트를 마우스 오른쪽 단추로 클릭하고

/// [참조 추가]->[프로젝트]를 차례로 클릭한 다음 이 프로젝트를 찾아서 선택합니다.

///

///

/// 2단계)

/// 계속 진행하여 XAML 파일에서 컨트롤을 사용합니다.

///

/// <MyNamespace:Connector/>

///

/// </summary>