===============================================

Documentation

Editor

패닝

패닝은 마우스 오른쪽 버튼을 누른 상태에서 마우스를 움직여 수행되며 DisablePanning 종속성 속성을 true로 설정하여 비활성화 할 수 있습니다.

참고 : Offset 종속성 속성을 설정하여 프로그래밍 방식으로 변경할 수 있습니다.

이동하는 동안 IsPanning 종속성 속성이 true로 설정되고 Viewport 및 AppliedTransform 종속성 속성이 업데이트됩니다.

자동 패닝도 기본적으로 활성화되며 DisableAutoPanning 종속성 속성을 true로 설정하여 비활성화 할 수 있습니다. 동작은 뷰포트 가장자리 근처에서 선택 또는 보류중인 연결을 선택하거나 끌 때 뷰포트를 이동하는 것입니다.

AutoPanSpeed ​​종속성 속성을 사용하여 자동 패닝 속도를 변경할 수 있으며, AutoPanEdgeDistance 종속성 속성을 사용하여 패닝을 트리거하는 가장자리에서 거리를 변경할 수 있습니다.

패닝은 선택 및 확대 / 축소와 함께 사용할 수도 있으며 자동 패닝은 선택 및 확대 / 축소와 함께 사용할 수 있으며 선택 항목 또는 보류중인 연결을 끌 때도 사용할 수 있습니다.

기본값 :

오프셋 : [0, 0]

DisablePanning : false

DisableAutoPanning : false

AutoPanSpeed ​​: 틱당 10 픽셀

AutoPanEdgeDistance : 15 픽셀

AutoPanningTickRate : 1 밀리 초

확대

확대 / 축소는 마우스 휠을 사용하거나 Ctrl +를 눌러 확대하거나 CTRL-를 눌러 축소하여 수행되며 DisableZooming 종속성 속성을 true로 설정하여 비활성화 할 수 있습니다.

참고 : Scale 종속성 속성을 MinScale과 MaxScale 사이의 값으로 설정하여 프로그래밍 방식으로 변경할 수 있습니다.

확대 / 축소하는 동안 Viewport 및 AppliedTransform 종속성 속성이 업데이트됩니다.

확대 / 축소는 패닝, 선택 항목 끌기 또는 보류중인 연결과 함께 사용할 수도 있습니다.

기본값 :

규모 : 1

MinScale : 0.1

최대 스케일 : 2

선택

항목 선택은 마우스 왼쪽 버튼을 누른 상태에서 마우스를 움직이면됩니다. 선택 작업이 진행 중이면 IsSelecting 종속성 속성이 true로 설정되고 SelectedArea 종속성 속성이 이동할 때마다 업데이트됩니다.

참고 : 컬렉션을 SelectedItems 종속성 속성에 바인딩하여 선택한 항목을 프로그래밍 방식으로 설정할 수도 있습니다.

실시간 선택이 활성화 된 경우 (EnableRealtimeSelection : true) 선택 사각형의 크기를 조정하는 동안 항목이 선택되고 선택 취소됩니다. 그렇지 않으면 SelectedArea에 포함 된 항목은 선택 작업이 완료된 후에 만 ​​선택됩니다.

ItemContainer를 선택하면 IsSelected 종속성 속성이 true로 설정됩니다.

선택을 시작할 때 보유한 ModifierKeys에 따라 다른 동작이 사용됩니다.

바꾸기-수정 자 키 없음 (기본 동작, 선택한 항목을 지우고 새 선택 시작)

추가-Shift 키 (현재 선택한 항목에 추가)

제거-Alt 키 (현재 선택한 항목에서 선택 항목 제거)

반전-제어 키 (선택한 항목을 제거하고 선택하지 않은 항목을 추가)

항목 선택은 이동 및 확대 / 축소와 함께 사용할 수도 있습니다.

기본값 :

EnableRealtimeSelection : true

스냅

선택 항목을 이동하면 GridCellSize 종속성 속성이 현재 선택한 항목을 스냅 할 위치를 결정하는 데 사용됩니다. 스냅은 가상 그리드가 아니라 선택한 항목의 위치를 ​​기준으로합니다.

선택한 항목이 처음 생성 될 때 그리드에 스냅되지 않거나 런타임에 GridCellSize가 변경된 경우 EnableSnappingCorrection 종속성 속성이 true 인 경우 선택 항목을 이동 한 후 최종 위치가 수정됩니다.

기본값 :

GridCellSize : 1

EnableSnappingCorrection : true

명령

다음 RoutedUICommands는 EditorCommands 클래스에 있습니다.

ZoomIn-CTRL + (뷰포트의 중심을 기준으로 확대)

ZoomOut-CTRL-(뷰포트의 중심을 기준으로 축소)

SelectAll-CTRL A (모든 항목 선택)

삭제-삭제 (ItemsSource가 바인딩되지 않은 경우 선택한 항목 삭제)

BringIntoView-뷰포트를 지정된 위치로 이동합니다. 기본값은 [0,0]입니다. (CommandParameter는 Point 또는 문자열 유형의 위치입니다.)

정렬-지정된 정렬 방법을 사용하여 선택한 항목을 정렬합니다. 기본값은 위쪽입니다. (CommandParameter는 정렬 또는 문자열 유형입니다. 가능한 정렬 : 위쪽, 왼쪽, 아래쪽, 오른쪽, 가운데, 가운데)

.net documentation

<https://docs.microsoft.com/ko-kr/dotnet/csharp/language-reference/operators/conditional-operator>

소스 코드 분석 (nodify.StateMachine)

===============================================

MainWindow.xaml 에서

<nodify:NodifyEditor x:Name="Editor"

ItemsSource="{Binding States}"

SelectedItem="{Binding SelectedState}"

SelectedItems="{Binding SelectedStates}"

Connections="{Binding Transitions}"

PendingConnection="{Binding PendingTransition}"

DisconnectConnectorCommand="{Binding DisconnectStateCommand}"

ConnectionCompletedCommand="{Binding CreateTransitionCommand}"

Grid.Column="1">

해당 코드의 바인딩은

<Window.DataContext>

<local:StateMachineViewModel />

</Window.DataContext> 에서 StateMachineViewModel 코드를 바인딩했다.

xmlns:local="clr-namespace:Nodify.StateMachine" 에서 보듯이 자신의 솔루션의 네임스페이스를 가지고 있고 ‘StateMachineViewModel.cs’ 를 바인딩 하는 것을 알 수 있다.

ItemsSource 에 바인딩 된 States 는

private NodifyObservableCollection<StateViewModel> \_states = new NodifyObservableCollection<StateViewModel>();

public NodifyObservableCollection<StateViewModel> States

{

get => \_states;

set => SetProperty(ref \_states, value);

}

와 같이 get/set 구문이 선언되어 있고 StateViewModel 타입의 NodifyObservableCollection 이다.

여기서 StateViewModel 의 경우는 Obserableobject 타입이다. 그리고 해당 객체는 Nodify.Shared.Obserableobject.cs 에서 구현되어 있다.(네임스페이스는 Nodify 임)

Guid Id , Point Location, Point Anchor, Size Size, string? Name, bool IsRenaming, bool IsActive, bool IsEditable 기본속성을 가지고 있다.

ObserableObject 에 대한 기억

|  |
| --- |
| ViewModels과 가장 관련있는 것이 궁금했습니다. INotifyPropertyChanged 인터페이스를 구현하거나 ObservableObject에서 파생되는 것이 더 낫습니다.  ObservableObject 클래스는 INotifyPropertyChanged을 구현하고 RaisePropertyChanged과 같은 지루한 코드를 수행합니다.  INotifyPropertyChanged은 PropertyChanged 이벤트를 구현해야합니다.  내 관점에서 보면 ObservableObject을 사용하는 것이 더 논리적 인 것처럼 보이지만 대부분의 튜토리얼에서는 ViewModel에 INotifyPropertyChanged 인터페이스를 구현합니다. |

위와 같이 실제로 해당 StateViewModel 의 경우 MVVM 모델을 쓰고 있고,

<https://m.blog.naver.com/PostView.nhn?blogId=seokcrew&logNo=221258186394&proxyReferer=https:%2F%2Fwww.google.com%2F>

|  |
| --- |
| MVVM Light Toolkit은 GalaSoft라는 곳(?)에서 만든 MVVM 아키텍쳐의 구현체다. Laurent Bugnion이라는 스위스 취리히에 사는 개발자가 만들었는데 현재는 Microsoft Azure 팀에서 근무중인듯 하다. MVVM은 Model-View-ViewModel의 View와 Model을 분리하기 위한 아키텍쳐 스타일로, Prism과 함께 WPF에서 가장 많이 쓰이는 패턴 중 하나다. WPF가 더이상 잘 쓰이진 않지만 ㅠㅠ. View와 Model이 분리된 구조로 개발을 진행하면 Model에 대한 테스트 그리고 디자이너간 협업 측면에서 여러 이점이 있다.  MVVM Light Toolkit을 이용하면 당연히 WPF 기반의 MVVM 애플리케이션을 빠르게 개발할 수 있다. 주요 클래스 개수도 많지 않기 때문에 어렵지 않게 적응이 가능하다. 설치는 Nuget Package Manager를 이용하는게 가장 간편한 방법이다.  MVVM Light Toolkit은 2개의 main DLL로 유지된다.  GalaSoft.MvvmLight.dll  GalaSoft.MvvmLight.Extras.dll  GalaSoft.MvvmLight.dll의 주요 클래스 목록이다.  ObservableObject - Base class for Model classes. It creates Model properties as observable.  ViewModelBase - Base class for ViewModel classes.  RelayCommand - A command. More about commands here.  WeakAction - Stores an System.Action without causing a hard reference to be created to the Action's owner. The owner can be garbage collected at any time.  WeakFunc - Stores an Func without causing a hard reference to be created to the Func's owner. The owner can be garbage collected at any time.  Messenger - Class for allowing objects to exchange messages.  MessageBase - Base class for all messages broadcasted by the Messenger.  DispatcherHelper - Helper class for dispatcher operations on the UI thread.  다음은 Galasoft.MvvmLight.Extras.dll의 주요 클래스 목록이다.  EventToCommand - Used for bind any event of FrameworkElement to ICommand.  SimpleIOC - An IOC container for register and resolve instances.  ObservableObject을 상속받아 모델을 구현하면 변경점이 바로 View에 반영되는데 필요한 기반 기능을 얻을 수 있다.  ViewModelBase를 상속받아 뷰-모델을 구현하면 View 도는 다른 ViewModel과 상호작용하는데 필요한 기반 기능을 얻을 수 있다.  Messenger는 의존 관계를 두지 않고 ViewModel과 ViewModel 간 상호작용할 수 있는 방법을 제공하며,  SimpleIOC라는 IOC 컨테이너를 통해 Service Locator 패턴을 쉽게 구현할 수 있는 방법을 제공한다.  void Register<TMessage>(object recipient, object token, bool receiveDerivedMessagesToo, Action<TMessage> action);  void Send<TMessage>(TMessage message, object token);  Messenger와 SimpleIOC는 모두 클래스 간 의존성을 줄여주기 위한 장치다.  ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);  SimpleIoc.Default.Register<MainViewModel>();  ServiceLocator.Current.GetInstance<MainViewModel>();  간단한 예제. 복잡하지 않고 개념잡기에 괜찮다. |

Nullable 에 대한 참고

<https://docs.microsoft.com/ko-kr/dotnet/csharp/language-reference/builtin-types/nullable-reference-types>

|  |
| --- |
| - finalize()  finalize()는 C#에서 '소멸자'를 의미합니다.  닷넷에서는 이미 객체를 마무리하기 위해 만들어 놓은 메서드가 있는데  이 메서드가 바로 'finalize()'라는 것입니다.  왜 소멸자를 안만들었을까요?  소멸자를 가지지 않은 언어들이 있을 수 있으니 안만든 것입니다.  소멸자가 없는 언어들은 닷넷에서는 finalize() 메서드를 direct로 집어넣습니다.  -nullabe  nullable 타입은 'NULL'이 가능한 타입을 말합니다.  즉, nullable 타입은 값 타입.. 그 중에서도 내장 타입에 존재합니다.  int?, double? 과 같이 물음표(?)를 붙여서 사용합니다.  그렇다면 '??'는 무엇일까요...  이번 글에서는 물음표도 함부로 못 붙이겠군요....허허...  ?? 연산자는 '기본 값'이라는 연산자입니다.  진짜 물음표를 뜻하는 것이 아니라, '타입에 대한 기본 값'을 의미하는 것입니다.  예를 들어서, 'int? n = k()?? 0;' 과 같은 코드가 있다고 해봅시다.  Sap() 이라는 함수를 호출 했는데  만약, Sap()의 리턴 값이 NULL이라면 기본 값을 0으로 주겠다는 뜻입니다. |

#### 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.