ItemsControl 공통 속성 및 기능

- ItemsControl 상속 구조
  - ItemsControl
    - ComboBox
    - ListBox
      - ListView
    - TreeView
    - TabControl
    - DataGrid
    - Menu / ContextMenu

- ItemsControl 기반 속성

속성	설명
ItemsSource	바인딩할 데이터 컬렉션
DisplayMemberPath	항목에서 표시할 속성명
SelectedValuePath	선택된 항목에서 가져올 속성명
SelectedValue	SelectedValuePath로 지정된 값
SelectedItem	현재 선택된 항목 객체
ItemTemplate	각 항목의 시각 표현을 지정 (DataTemplate)

  - ComboBox

속성	설명
IsEditable	텍스트 직접 입력 가능 여부
Text	선택되거나 입력된 텍스트
SelectedIndex, SelectedItem, SelectedValue, SelectedValuePath, DisplayMemberPath, ItemsSource	모두 사용 가능

```
<ComboBox
    ItemsSource="{Binding Countries}"
    DisplayMemberPath="Name"
    SelectedValuePath="Code"
    SelectedValue="{Binding SelectedCountryCode}"
    IsEditable="True"
    Text="{Binding EnteredText, Mode=TwoWay}" />
```

ViewModel 예제

```
public class Country {
    public string Name { get; set; }
    public string Code { get; set; }
}
public ObservableCollection<Country> Countries { get; } = new(...);
public string SelectedCountryCode { get; set; }
public string EnteredText { get; set; }
```

● ListBox

속성	설명
SelectionMode	Single, Multiple, Extended 가능
SelectedItems	다중 선택 시 사용 가능
SelectedIndex, SelectedItem, SelectedValue, SelectedValuePath, DisplayMemberPath, ItemsSource	모두 사용 가능

단일선택

```
<ListBox
    ItemsSource="{Binding Fruits}"
    DisplayMemberPath="Name"
    SelectedValuePath="Id"
    SelectedValue="{Binding SelectedFruitId}" />
```

단중선택

```
<ListBox SelectionMode="Multiple"
         ItemsSource="{Binding People}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <CheckBox Content="{Binding FullName}" />
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

● ListView (→ ListBox 상속)

속성	설명
View	GridView, TileView 등을 지정 가능
GridViewColumn	열 단위 정의 가능
Header, DisplayMemberBinding, CellTemplate	열 내용/머리글 설정
SelectedItem, SelectedItems, SelectedIndex, SelectedValue 등	ListBox 와 동일하게 동작

```
<ListView                                                    GridView 사용시
    ItemsSource="{Binding People}"
    SelectedItem="{Binding SelectedPerson}">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Name"
                            DisplayMemberBinding="{Binding Name}" />
            <GridViewColumn Header="Age"
                            DisplayMemberBinding="{Binding Age}" />
        </GridView>
    </ListView.View>
</ListView>

public class Person {                                        ViewModel 예제
    public string Name { get; set; }
    public int Age { get; set; }
}
public ObservableCollection<Person> People { get; } = new(...);
public Person SelectedPerson { get; set; }
```

- TreeView

  계층적(Hierarchical)데이터 구조에 적합

속성/기능	사용가능	설명
ItemsSource	사용가능	자식 노드 컬렉션
DisplayMemberPath	불가	대신 DataTemplate 사용
SelectedItem	사용가능	현재 선택된 항목
SelectedValue	불가	미지원 (직접 속성 바인딩 필요)
ItemTemplate	사용가능	항목 시각화
HierarchicalDataTemplate	사용가능	계층적 구조 표현

```
<TreeView ItemsSource="{Binding Categories}"
          SelectedItemChanged="TreeView_SelectedItemChanged">
    <TreeView.ItemTemplate>
        <HierarchicalDataTemplate ItemsSource="{Binding Children}">
            <TextBlock Text="{Binding Name}" />
        </HierarchicalDataTemplate>
    </TreeView.ItemTemplate>
</TreeView>
```

```
public class Category                                    ViewModel 예제
{
    public string Name { get; set; }
    public ObservableCollection<Category> Children { get; set; }
}

public ObservableCollection<Category> Categories { get; set; }
```

● TabControl

Tab 방식 UI, 각 항목은 TabItem, ItesSource 를 통해 바인딩 가능

속성/기능	사용가능	설명
ItemsSource	사용가능	탭 목록
DisplayMemberPath	사용가능	탭 제목 바인딩
SelectedItem	사용가능	선택된 탭 항목
SelectedContent	사용가능	현재 탭 내용
ContentTemplate	사용가능	본문 콘텐츠 템플릿

```
<TabControl ItemsSource="{Binding Pages}"
            DisplayMemberPath="Header"
            SelectedItem="{Binding SelectedPage}">
    <TabControl.ContentTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Content}" />
        </DataTemplate>
    </TabControl.ContentTemplate>
</TabControl>
```

```
public class TabPage                                     ViewModel 예제
{
    public string Header { get; set; }
    public string Content { get; set; }
}

public ObservableCollection<TabPage> Pages { get; set; }
public TabPage SelectedPage { get; set; }
```

● DataGrid

행/열 기반 고급 테이블 형태

속성/기능	사용가능	설명
ItemsSource	사용가능	탭 목록

속성/기능	사용가능	설명
DisplayMemberPath	사용불가	탭 제목 바인딩
Columns	사용가능	선택된 탭 항목
SelectedItem	사용가능	
AutoGenerateColumns	사용가능	현재 탭 내용
CellTemplate	사용가능	본문 콘텐츠 템플릿

```
<DataGrid ItemsSource="{Binding Employees}"
          SelectedItem="{Binding SelectedEmployee}"
          AutoGenerateColumns="False">
    <DataGrid.Columns>
        <DataGridTextColumn Header="이름" Binding="{Binding Name}" />
        <DataGridTextColumn Header="부서" Binding="{Binding Department}" />
    </DataGrid.Columns>
</DataGrid>
```

ViewModel 예제

```
public class Employee
{
    public string Name { get; set; }
    public string Department { get; set; }
}

public ObservableCollection<Employee> Employees { get; set; }
public Employee SelectedEmployee { get; set; }
```

● Menu / ContextMenu

메뉴 구조 표현 (계층 가능), MenuItem 을 포함하여 계층 구조 가능

속성/기능	사용가능	설명
ItemsSource	사용가능	메뉴 항목 목록
DisplayMemberPath	사용가능	메뉴 이름 설정
Command	사용가능	메뉴 클릭 명령
Icon	사용가능	이미지 또는 Path 표시
InputGestureText	사용가능	단축키 표시
HierarchicalDataTemplate	사용가능	자식 메뉴 표현

```
<Menu ItemsSource="{Binding MenuItems}">
    <Menu.ItemContainerStyle>
        <Style TargetType="MenuItem">
            <Setter Property="Header" Value="{Binding Header}" />
            <Setter Property="Command" Value="{Binding Command}" />
```

```
            </Style>
        </Menu.ItemContainerStyle>
    </Menu>

    <Button Content="Right-click me">
        <Button.ContextMenu>
            <ContextMenu ItemsSource="{Binding ContextMenuItems}">
                <ContextMenu.ItemContainerStyle>
                    <Style TargetType="MenuItem">
                        <Setter Property="Header" Value="{Binding Name}" />
                    </Style>
                </ContextMenu.ItemContainerStyle>
            </ContextMenu>
        </Button.ContextMenu>
    </Button>
```

전체 요약 비교표

컨트롤	계층 구조	Display-MemberPath	Items-Source	Selected-Item	Selected-Value	Template-사용	주 용도
ComboBox	✗	✓	✓	✓	✓	✓	선택 목록
ListBox	✗	✓	✓	✓	✓	✓	리스트
ListView	✗	✗ (GridView로)	✓	✓	✓	✓	Table 리스트
TreeView	✓	✗	✓	✓	✗	✓ (HDT)	트리 구조
TabControl	✗	✓	✓	✓	✓	✓	탭 인터페이스
DataGrid	✗	✗ (Binding 사용)	✓	✓	✓	✓	Data 테이블
Menu	✓	✓	✓	✗	✗	✓	메뉴 UI
ContextMenu	✓	✓	✓	✗	✗	✓	우클릭 메뉴

◈ ListView 는 DisplayMemberPath 는 무시되며,
대신 GridViewColumn.DisplayMemberBinding을 사용

◈ ComboBox는 Text, IsEditable 같은 고유 속성이 있으며, 드롭다운 인터페이스 특화

◈ ListBox와 ListView는 다중 선택(SelectionMode)이 가능하며 SelectedItems 속성을 제공

◈ SelectedItem 은 항상 "전체 개체"이며, SelectedValue 는 SelectedValuePath 에서
지정한 프로퍼티 값입니다.

주요 컨트롤별 사용 가능 여부

컨트롤	ItemTemplateSelector	GroupStyle	ItemContainerStyle
ComboBox	✅ 가능 (DropDown 항목에 적용)	❌ (그룹화 미지원)	✅ 가능 (ComboBoxItem)
ListBox	✅ 가능	✅ 가능	✅ 가능 (ListBoxItem)
ListView	✅ 가능	✅ 가능	✅ 가능 (ListViewItem)
TreeView	❌ (사용 불가) 대신 HierarchicalDataTemplateSelector)	❌ (그룹화 미지원)	✅ 가능 (TreeViewItem)
TabControl	✅ 가능 (탭 내용의 템플릿 선택)	(탭은 그룹화 개념 없	✅ 가능 (TabItem)
DataGrid	❌ (열별 템플릿이 있음)	✅ 가능 (GroupStyle 사용 가능)	✅ 가능 (DataGridRow)
Menu	✅ 가능 (MenuItem 마다 템플릿 선택 가능)	❌	✅ 가능 (MenuItem)
ContextMenu	✅ 가능	❌	✅ 가능

예제

```
<ListBox ItemsSource="{Binding Items}"
         ItemTemplateSelector="{StaticResource MyTemplateSelector}" />

<ListView ItemsSource="{Binding GroupedItems}">
    <ListView.GroupStyle>
        <GroupStyle>
            <GroupStyle.HeaderTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Name}" FontWeight="Bold"/>
                </DataTemplate>
            </GroupStyle.HeaderTemplate>
        </GroupStyle>
    </ListView.GroupStyle>
</ListView>

<TreeView ItemsSource="{Binding Nodes}">
    <TreeView.ItemTemplate>
        <HierarchicalDataTemplate ItemsSource="{Binding Children}">
            <TextBlock Text="{Binding Name}" />
        </HierarchicalDataTemplate>
    </TreeView.ItemTemplate>
</TreeView>
```

※ TreeView 는 ItemTemplateSelector 대신 HierarchicalDataTemplateSelector 를 별도로 구현해야 합니다

HeimdallrComboBox

```
<ctrls:HeimdallrComboBox
    Grid.Row="3" Width="150" Height="40" Background="BlanchedAlmond"
    CornerRadius="10" BorderThickness="3" FontSize="20" Foreground="Black"
    PathIcon="Account" PathIconFill="{StaticResource DarkViolet}" IconHeight="30"
    IconWidth="30" CheckedBackground="#FFBA55D3"
    MouseOverBackground="{StaticResource DarkCharcoal}"
    DropDownBackground="#DCD7C9" ItemsSource="{Binding ProductUnitOptions}"
    DisplayMemberPath="Value" SelectedValuePath="Key"
    SelectedValue="{Binding SelectedUnit, Mode=TwoWay}"/>
```

사용방법 ->

1 CornerRadius 사용자가 Border 의 CornerRadius 설정

2 PathIcon 아이콘 지정 (아이콘만 지정 가능)

3 PathIconFill 아이콘의 색상지정

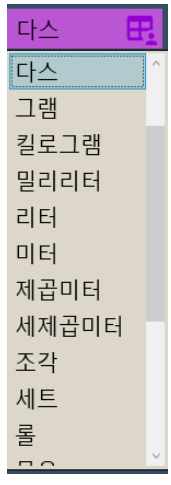
4 IconHeight 아이콘의 높이 지정

5 IconWidth 아이콘의 너비 지저

6 CheckedBackground 체크시 배경색상 지정

7 MouseOverBackground 마우스오버시 배경색상 지정

8 DropDownBackground 드롭다운시 배경색상 지정



9 ItemsSource="{Binding ProductUnitOptions}"

```
public ObservableCollection<KeyValuePair<ProductUnit, string>> ProductUnitOptions { get; }
```

ItemsSource 는 콤보박스에 표시할 항목들의 전체 컬렉션을 지정하는 속성입니다.

여기서는 ProductUnitOptions 라는 컬렉션(보통은 ObservableCollection<T>같은 형태)을 바인딩해서 콤보박스에 여러 항목을 띄우고 있습니다.

ProductUnitOptions 컬렉션 내부의 각 항목은 일반적으로
Key-Value 쌍(예: KeyValuePair<TKey, TValue>, 혹은 Dictionary<TKey, TValue>의 항목과 같은 구조)이라고 가정합니다.

ItemsSource 에 있는 각각의 아이템은 { Key, Value } 구조

DisplayMemberPath="Value"

콤보박스는 내부적으로 ItemsSource 컬렉션에 있는 각각의 개체(item)를 표시할 때, 보통 개체 자체를 ToString()으로 표시하거나, 직접 템플릿을 정의해야 합니다.

DisplayMemberPath는 "각 아이템의 어떤 프로퍼티 값을 화면에 텍스트로 보여줄지"를 지정합니다.

여기서 "Value"로 지정했으니, ProductUnitOptions 내 각 아이템의 Value 프로퍼티에 해당하는 값을 콤보박스 목록에서 보여준다는 뜻입니다.

예) 만약 아이템이 { Key = "kg", Value = "킬로그램" }이라면 콤보박스 목록에는 "킬로그램"이 표시됨.

"Value" 값은 특별한 키워드가 아니라 단지 클래스의 속성(Property) 이름입니다.

즉, DisplayMemberPath="DisplayName"이든 "Value"든, 이것은 전적으로 ItemsSource 에 바인딩된 개체의 속성 이름에 따라 달라집니다.

핵심 요약

이름	특별한 예약어인가요?	어디서 왔나요?
DisplayName	일반 속성 이름입니다.	여러분이 만든 클래스의 속성일 수 있음
Value	일반 속성 이름입니다.	예: KeyValuePair<TKey, TValue> 구조에서 나옴
Name, Title, Label 등	다 똑같이 일반 속성입니다.	의미만 다르고 역할은 동일합니다.

SelectedValuePath="Key"

콤보박스에서 아이템을 선택하면, 보통 선택된 항목 전체 개체가 SelectedItem 에 바인딩됩니다.

하지만 SelectedValuePath 를 지정하면, 선택된 아이템의 특정 프로퍼티 값만 SelectedValue 속성에 할당할 수 있습니다.

여기서는 "Key"를 지정했으므로, 선택된 아이템의 Key 프로퍼티가 SelectedValue에 바인딩됩니다.

예) 사용자가 "킬로그램" 항목을 선택하면, 내부적으로는 Key인 "kg"가 SelectedValue에 저장됨.

※ Key 외에 사용가능한 값의 조건

SelectedValuePath 에 지정할 수 있는 값은 ItemsSource 의 각 항목 개체의 공개(public) 속성 이름이어야 합니다

해당 속성은 문자열, 정수, 개체 등 어떤 타입이라도 가능합니다

```
public class ProductUnit                                    class
{
    public string Code { get; set; }            // 예: "kg"
    public string DisplayName { get; set; }     // 예: "킬로그램"
    public string Description { get; set; }     // 예: "중량 단위"
}
```

```
// ViewModel.cs
public ObservableCollection<ProductUnit> ProductUnitOptions { get; set; }
public string SelectedUnitCode { get; set; }

<ctrls:HeimdallrComboBox
    ItemsSource="{Binding ProductUnitOptions}"
    DisplayMemberPath="DisplayName"
    SelectedValuePath="Code"
    SelectedValue="{Binding SelectedUnitCode, Mode=TwoWay}" />
```

※ 익명형식은 불가

```
ItemsSource = new[] {
    new { Id = 1, Name = "사과" },
    new { Id = 2, Name = "바나나" }
};
```

익명 타입은 XAML에서 속성 경로(SelectedValuePath="Id" 등) 접근이 불가능합니다

이 경우에는 명시적인 클래스를 사용하는 것이 좋습니다

SelectedValue="{Binding SelectedUnit, Mode=TwoWay}"

SelectedValue 는 콤보박스에서 사용자가 선택한 항목의 SelectedValuePath 프로퍼티 값을 나타내는 속성입니다.

여기서는 SelectedValuePath="Key" 이므로, 실제로는 사용자가 선택한 항목의 Key 값이 됩니다.

이 값을 뷰모델(ViewModel)의 SelectedUnit 속성과 바인딩하고 있습니다.

Mode=TwoWay 이므로, 사용자 선택 시 뷰모델의 SelectedUnit 속성도 자동으로 업데이트되고, 뷰모델에서 SelectedUnit 값을 변경하면 콤보박스의 선택 항목도 자동으로 변경됩니다.

여기서 콤보박스 목록에는 Value (예: "킬로그램")를 보여주고

사용자가 선택한 항목의 Key (예: "kg")가 SelectedValue 를 통해 뷰모델의 SelectedUnit 에 바인딩(양방향)된다.

예 -> 다스는 열거형으로 아래와 같은 코드입니다.

```
[UnitInfo("다스", 12)]  // 12개 묶음
[Description("제품 단위를 묶음으로 나타냅니다. 일반적으로 여러 개의 제품이 함께 묶여 판매되는 경우에 사용됩니다.")]
Dozen,
```

DisplayMemberPath나 SelectedValuePath에 쓰는 값은 그 속성 이름 그대로를 정확기재

KeyValuePair vs Dictionary

항목	KeyValuePair<TKey, TValue>	Dictionary<TKey, TValue>
정체	단일 키-값 쌍	키-값 쌍의 컬렉션 (맵)
기능	데이터 구조 없음 (단일 데이터)	탐색, 추가, 삭제 기능 있음
주요 목적	키-값을 하나의 개체로 표현	키로 값을 빠르게 조회/저장
예제 타입	KeyValuePair<string, int>	Dictionary<string, int>
XAML 바인딩	ComboBox 의 ItemsSource 로 적합	직접 바인딩 어려움, 변환 필요
LINQ 사용	dictionary.ToList() 하면 반환되는 단위	

1 KeyValuePair<TKey, TValue>란?

단순히 하나의 키와 값의 짝을 표현하는 구조체입니다.

예를 들면:

```
var pair = new KeyValuePair<string, int>("나이", 30);
Console.WriteLine(pair.Key);   // "나이"
Console.WriteLine(pair.Value); // 30
```

주로 LINQ나 Dictionary 내부에서 foreach 순회 시 사용됩니다:

```
foreach (KeyValuePair<string, string> kvp in myDictionary)
{
    Console.WriteLine($"Key: {kvp.Key}, Value: {kvp.Value}");
}
```

2 Dictionary<TKey, TValue>란?

KeyValuePair들을 저장한 **해시 기반의 컬렉션(맵 구조)**입니다.

키를 통해 값을 매우 빠르게 조회할 수 있는 것이 장점입니다.

```
var dict = new Dictionary<string, string>();
dict["KR"] = "대한민국";
dict["JP"] = "일본";

Console.WriteLine(dict["KR"]); // "대한민국"
```

Dictionary는 내부적으로 KeyValuePair<TKey, TValue>들의 집합입니다.

빠른 조회, 중복 방지 필요시(조회 성능 최우선일때 적합)
알고리즘/검색/데이터 매핑 등에서는 Dictionary가 더 효율적

UI와 사람이 읽는 코드에서는 List<T> (직접 만든 클래스)가 더 가독성이 좋고
유지보수가 쉽습니다.

HeimdallrIconRadioButton

```
<ctrls:HeimdallrIconRadioButton Grid.Row="7" Content="Setting"
                                PathIcon="Setting" Background="Transparent"
                                CheckedForeground="Yellow" MouseOverForeground="Blue"
                                GroupName="MenuGroup" Width="120" Height="40"
                                IconFill="AliceBlue" />
```

사용방법 ->

1 Content="Setting" 문자열 (문자열만 지정 가능)
2 PathIcon="Setting" 아이콘 지정 (아이콘만 지정 가능)
3 CheckedForeground="Yellow" 체크시 아이콘 및 문자열 색상
4 MouseOverForeground="Blue" 마우스오버시 아이콘 및 문자열 색상
5 GroupName

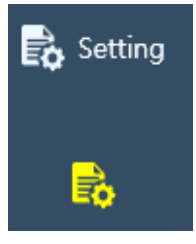
WPF에서 RadioButton 계열의 컨트롤들(예: RadioButton, 또는 ToggleButton 을 기반으로 한 커스텀 컨트롤들)에서 상호 배타적 선택 그룹을 만들기 위한 속성입니다

이 속성을 사용하면 동일한 그룹 이름을 가진 RadioButton 끼리 한 번에 하나만 선택될 수 있도록 연결됩니다.

※ 기본개념

- RadioButton은 한 번에 하나만 선택될 수 있는 옵션 집합을 제공하는 컨트롤입니다.
- XAML에서 RadioButton 여러 개를 사용하면, 기본적으로 같은 컨테이너 내에 있는 RadioButton들은 자동으로 상호 배타됩니다.
- 하지만 컨테이너가 다르거나, 명시적으로 그룹을 나누고 싶을 때 GroupName을 사용합니다.

```
<StackPanel>
  <RadioButton Content="Option A" GroupName="MenuGroup" />
  <RadioButton Content="Option B" GroupName="MenuGroup" />
  <RadioButton Content="Option C" GroupName="MenuGroup" />
</StackPanel>
```



위 세 개의 RadioButton 은 GroupName="MenuGroup"으로 지정되어 있어 한 번에 하나만 선택 가능합니다.

A를 선택하면 B와 C는 해제됨. 그룹이름이 없으면 간섭하지 않음

※ RadioButton, ToggleButton 만 GroupName 을 가질수 있음

HeimdallrNumericUpDown

```
<ctrls:HeimdallrNumericUpDown
    Grid.Row="9" Min="0" Max="1000" Step="10" Height="30" Width="100"
    Foreground="Black" FontSize="16"
    UpIconFill="#FF3DC2EC" DownIconFill="#FFD95F59"
    DownButtonIcon="ChevronDownEllipse" UpButtonIcon="ChevronUpEllipse"
    Value="{Binding SomeValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
```

사용방법 ->

1	Min	최소값 지정
2	Max	최대값 지정
3	Step	증감단위
4	UpIonFill	오른쪽 아이콘 색상 지정
5	DownIconFill	왼쪽 아이콘 색상지정
6	DownButtonIcon	왼쪽아이콘 지정
7	UpButtonIcon	오른족 아이콘 지정
8	Value	중앙 값 표기

※ Step 의 바인딩 + MVVM 자동 변환 방법

```
private ProductUnit _selectedUnit;
// ProductUnit 열거형 값입니다.
public ProductUnit SelectedUnit
{
    get => _selectedUnit;
    set
    {
        if (_selectedUnit != value)
        {
            _selectedUnit = value;
            OnPropertyChanged();
            // 단위 변경 시 Step도 자동 조정
            UpdateStep();
        }
    }
}
```

```
private double _step = 1;
public double Step
{
    get => _step;
    set
    {
        _step = value;
        OnPropertyChanged();
    }
}  바인딩 값
```

```
private void UpdateStep()
{
    switch (SelectedUnit)
    {
        case ProductUnit.Dozen:
            Step = 12;
            break;
        case ProductUnit.Kilogram:
            Step = 0.1;
            break;
        case ProductUnit.Gram:
            Step = 1;
            break;
        case ProductUnit.Each:
        case ProductUnit.Piece:
            Step = 1;
            break;
        default:
            Step = 1;
            break;
    }
}
            Step = 1;
            break;
        case ProductUnit.Each:
        case ProductUnit.Piece:
            Step = 1;
            break;
        default:
            Step = 1;
            break;
    }
}
```

xaml 에서 Step 속성을 바인딩

```
<ctrls:HeimdallrNumericUpDown
    Grid.Row="9"
    Min="0" Max="1000"
```

```
    Step="{Binding Step}" <!-- 여기 -->
    Height="30" Width="100"
    Foreground="Black" FontSize="16"
    UpIconFill="#FF3DC2EC" DownIconFill="#FFD95F59"
    DownButtonIcon="ChevronDownEllipse" UpButtonIcon="ChevronUpEllipse"
    Value="{Binding SomeValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
```

※ 단위 선택 ComboBox 예시 (선택 변경시 Step도 갱신됨)

```
<ComboBox ItemsSource="{Binding AvailableUnits}"
          SelectedItem="{Binding SelectedUnit}"
          DisplayMemberPath="Description" />
```

AvailableUnits 는 enum 목록이고, SelectedUnit 이 바뀔 때마다 Step 값도 UpdateStep()을 통해 자동 조정됩니다.

```
public IEnumerable<ProductUnit> AvailableUnits =>
    Enum.GetValues(typeof(ProductUnit)).Cast<ProductUnit>();
```

※ HeimdallrComboBox 와 HeimdallrNumericUpDown 의 협업(콜라보레이션)을 적용하여 증감숫자 변경방법

LoadingOverlay

```
<ctrls:LoadingOverlay
  Grid.Row="10" Foreground="#FFF4B7" IsLoading="True"
  IndicatorType="DoubleBounce" LoadingText="진행중..."
  VerticalAlignment="Center" HorizontalAlignment="Center" />
```

사용방법 ->

1	Foreground	Indicator(표시기) 색상 지정
2	IsLoading	IsLoading="{Binding IsLoading}" 기본값 false
3	IndicatorType	Indicator(표시기) 지정 (19개 선택가능)

Bar Blocks BouncingDot Cogs Cupertino Dashes DotCircle
DoubleBounce Ellipse Escalade FourDots Grid Piston Pulse
Ring Swirl ThreeDots Twist Wave

4	LoadingText	MVVM 으로 문자열 변경 및 진행률 표시 가능
5	Width, Height	지정시 일부 Type 표시기 깨짐주의
6	DownButtonIcon	왼쪽아이콘 지정
7	Backgroud	Transparent

LoadingText MVVM 버튼클릭시 처리기 진행률

```
private bool _isLoading;
private string _loadingMessage = "처리 중입니다";
private double _progress;
private int _remainingSeconds;
private CancellationTokenSource? _cts;

public MainViewModel()
{
  StartLoadingCommand = new RelayCommand(async () =>
    await StartLoadingAsync(100));
}

/// <summary>
/// 로딩 표시 여부입니다.
/// </summary>
public bool IsLoading
{
```

```
    get => _isLoading;
    set
    {
      if (_isLoading != value)
      {
        _isLoading = value;
        OnPropertyChanged();
      }
    }
  }

  /// <summary>
  /// 현재 진행률입니다 (0.0 ~ 1.0).
  /// </summary>
  public double Progress
  {
    get => _progress;
    set
    {
      if (Math.Abs(_progress - value) > 0.0001)
      {
        _progress = value;
        OnPropertyChanged();
      }
    }
  }

  /// <summary>
  /// 진행 상태 메시지입니다.
  /// </summary>
  public string LoadingMessage
  {
    get => _loadingMessage;
    private set
    {
      if (_loadingMessage != value)
      {
        _loadingMessage = value;
        OnPropertyChanged();
      }
```

```
  }
}

/// <summary>
/// 로딩 시작 명령입니다.
/// </summary>
public ICommand StartLoadingCommand { get; }

/// <summary>
/// 진행률과 남은 시간을 포함한 로딩 처리 예제입니다.
/// </summary>
public async Task StartLoadingAsync(int totalSeconds = 100)
{
  _cts?.Cancel();
  _cts = new CancellationTokenSource();

  IsLoading = true;
  Progress = 0.0;

  for (int i = 0; i <= totalSeconds; i++)
  {
    _remainingSeconds = totalSeconds - i;
    Progress = i / (double)totalSeconds;

    int percent = (int)(Progress * 100);
    LoadingMessage = $"처리 중입니다... {percent}% 진행됨 ({_remainingSeconds}초 남음)";

    try
    {
      await Task.Delay(1000, _cts.Token);
    }
    catch (TaskCanceledException)
    {
      break;
    }
  }

  LoadingMessage = "처리 완료";
  Progress = 1.0;
  IsLoading = false;
}
```

VSM (Visual State Manager)

컨트롤이 Focused, MouseOver, Pressed, Disabled, Loading 같은 상태에 따라 어떻게 보이고, 어떤 애니메이션을 적용할지 XAML로 정의하고 상태 변경 시 자동으로 전환되도록 도와주는 기능입니다.

● VSM의 주요 요소

[1] VisualStateGroup
상태들의 그룹.
예: CommonStates, FocusStates, LoadingStates

[2] VisualState
하나의 상태 정의.
이 상태에 들어가면 어떤 속성/애니메이션이 적용될지 정의.
예: Normal, MouseOver, Pressed, Visible, Hidden

[3] Storyboard
상태 전환 시 실행될 애니메이션.

[4] VisualStateManager.GoToState
코드에서 상태를 전환하는 함수.
예: GoToState(this, "Pressed", true)

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup x:Name="CommonStates">

    <VisualState x:Name="Normal"/>

    <VisualState x:Name="MouseOver">
      <Storyboard>
        <ColorAnimation Storyboard.TargetName="MyBorder"
          Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Color)"
          To="LightBlue"
          Duration="0:0:0.2"/>
      </Storyboard>
    </VisualState>
    <VisualState x:Name="Pressed">
      <Storyboard>
        <ColorAnimation Storyboard.TargetName="MyBorder"
```

```
            Storyboard.TargetProperty="(Border.Background).(SolidColorBrush.Color)"
            To="DarkBlue"
            Duration="0:0:0.2"/>
        </Storyboard>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
```

➡ 여기서 MouseOver 상태로 가면 MyBorder의 배경색이 LightBlue로 애니메이션.

● VSM의 장점
- XAML로 상태별 스타일과 애니메이션을 선언적으로 작성.
- 코드로 속성 변경/애니메이션 제어 필요 없음.
- VisualStateManager.GoToState로 간단히 상태 전환.
- 컨트롤 템플릿 제작 시 유지보수와 재사용성이 뛰어남.

● 코드에서 사용하는 방법

```
VisualStateManager.GoToState(this, "Pressed", true);
```

이 코드는 VSM이 정의된 컨트롤의 Pressed 상태로 전환하며 애니메이션을 실행합니다

● VSM 과 EventTrigger 차이

구분	VisualStateManager (VSM)	EventTrigger
용도	상태 전환 중심 (State Pattern)	특정 이벤트 발생 시 동작
구조	상태 그룹과 상태 이름 중심	이벤트(Click, Loaded 등) 중심
관리	상태 간 관계, 전환 선언	개별 이벤트 별도 정의
애니메이션 제어	상태 간 전환 애니메이션 내장	단순 이벤트 기반 애니메이션
유연성	상태 중심 설계에 최적	특정 동작 한정적 트리거

☑ VSM은 컨트롤의 상태 기반 전환에 적합하고
☑ EventTrigger는 단일 이벤트에 따른 단일 애니메이션에 적합합니다.