.NET og C#

WPF Data binding

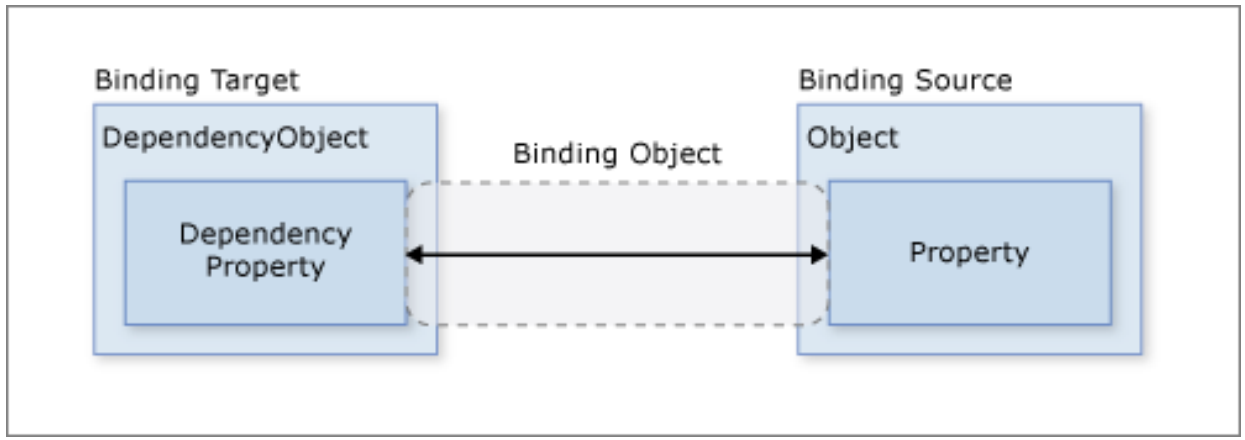
Indhold

DataBinding

- Simple binding
- DataObject
- DataContext
- DataContext i higher level structure
- OneWay/TwoWay/OneWayToSource (UpdateSourceTrigger=PropertyChanged)
- Lists
- Converters/Validators/…
- Binding i code

Data Binding

- DataBinding er en smart feature i WPF
- DataBinding en måde at binde et GUI-element, f.eks. en TextBox eller en ListBox, til en Data Source, sådan at gui elementer viser værdier fra Data Source
- Data Source kan være et hvilket som helst object, (som også kan indeholde andre objecter, hvis det ønskes). Så en slags container for data.
- I koden bruges ”DataContext” til at forbinde Data Source med en given UI komponent.


Binding Target
DependencyObject
Dependency Property
Binding Object
Binding Source
Object
Property

Eksempel: Binding til et object

- **Vi vil gerne vise et object i GUI:**

```
public class SomeDataClass {
    public string Author { get; set; }
    public string BookTitle { get; set; }
}
```

Binding :

1. Set DataContext for komponenterne

```
public partial class MainWindow : Window {
    public MainWindow() {
        InitializeComponent();
        TextForTextBox1 = "TextForTextBox1";
        TextForTextBox2 = "TextForTextBox2";
        textBox1.DataContext = this; // så bruger denne klasse som “base”
        textBox2.DataContext = this;
    }
    public string TextForTextBox1 { set; get; }
    public string TextForTextBox2 { set; get; }
}
```

2. Set binding i XAML

```
<TextBox x:Name="textBox1“ Text="{Binding Path=TextForTextBox1}"/>
<TextBox x:Name="textBox2“ Text="{Binding Path=TextForTextBox2}"/>
```

Binding :

- DataContext : Et object som indeholder den bundne data.
- Bruger ofte custom objects eller containers som data source – f.eks. en liste af data, **men kan være hvad som helst.**

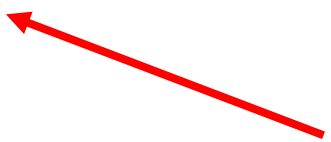
Lille opgave

Opgave 7.1

Binding : Notification

```
public partial class MainWindow : Window {
        public MainWindow() {
            InitializeComponent();

            TextForLabel = "From MainWindow Property : TextForLabel";
            labelForDataBinding.DataContext = this;
        }
        public string TextForLabel { set; get; }
        private void TextBox_TextChanged(object sender, TextChangedEventArgs e) {
            TextForLabel = ((TextBox)sender).Text;
        }
}
```



Fails to update label.

Binding : Notification

Fra Binding2_Problem demo project

- **Problem**: GUI (en Label her) bliver IKKE opdateret, når det underliggende data ændres.
- **Hvorfor:** Fordi ingen ved det sker!
- **Løsning:** Notify alle som lytter!

PropertyChanged

- **Hvis en ændring i DataSource skal vises i GUI, så skal DataSource implementere dette interface:**

  `INotifyPropertyChanged`

  `Som har en event`

  ```
  public event PropertyChangedEventHandler PropertyChanged;
  ```

- **Hvorfor**?: Fordi DataSource skal ”broadcaste” ændringerne, for at fortælle at GUI skal updateres med de nye værdier i denne DataSource. Det gøres ved at ”raise” et PropertyChanged Event.

Binding : Notification

(From Binding3_Solution)

```
public class MyDataContext : INotifyPropertyChanged {
    public MyDataContext() { Author = "Tom Clancy"; }

    private string author;
    public string Author {
        set {
            author = value;
            NotifyPropertyChanged("Author"); //her kaldes notify metoden
        }
        get { return author; }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(string propertyName) {
        if (PropertyChanged != null)  //er vist ikke nødvendigt
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Binding : Notification

```
public partial class MainWindow : Window {
    private MyDataContext dataContext;

    public MainWindow() {
        InitializeComponent();

        dataContext = new MyDataContext();
        labelForDataBinding.DataContext = dataContext;
    }

    private void TextBox_TextChanged(object sender, TextChangedEventArgs e) {
        if(dataContext!=null)
            dataContext.Author = ((TextBox)sender).Text;
            //kalder nu set metoden på Author, som raiser en PropertyChangedEvent
    }
}
```

Lille opgave

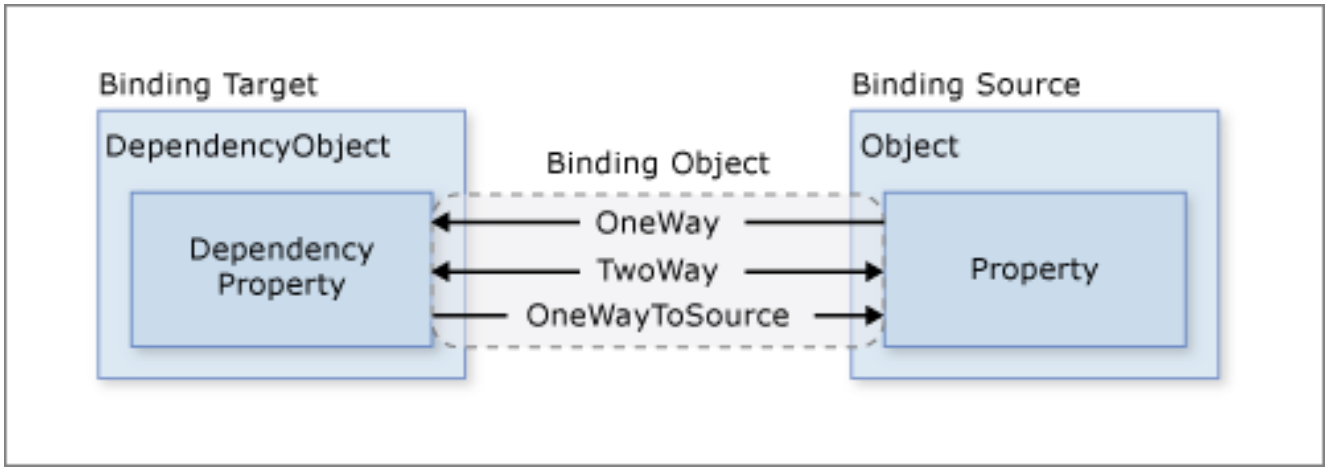
Opgave 7.2

Direction of binding (”Bindingsretning”)

- **En binding kan være:**
  - OneWay (Fra DataSource to GUI) (gui opdateres når datasource opdateres – ikke omvendt)
  - TwoWay (gui og datasources er i sync)
  - OneWayToSource (fra GUI til DataSource) (når ting i gui ændrer sig, så ændres data sources)

(se project Binding4_TwoWay : textBoxLower)

Direction of binding (”Bindingsretning”)



DataContext i Parent control

- Hvis data-binding er defineret for en gui component, **men hvis DataContext mangler**, så **leder komponenenten efter en DataContext i sine parents**, indtil den finder en DataContext.
- Så hvis flere gui-elementer skal bruge same DataContext, kan man **nøjes med at definere den på et top gui element**.
- Vist i project Binding4_TwoWay : MainWindow

Lille opgave

Opgave 7.3

Binding til en List

Binding to List

(Fra Binding5_List)

```
public partial class MainWindow : Window {
    private List<MyDataClass> data;

    public MainWindow() {
        InitializeComponent();

        data = new List<MyDataClass>() {
            new MyDataClass("Tom Clancy","Jack Ryan: Shadow Recruit"),
            new MyDataClass("Stephen King","Carrie"),
            new MyDataClass("Stephen King","The Shining"),
            new MyDataClass("Edgar Allan Poe","The Raven"),
            new MyDataClass("Edgar Allan Poe","The Murders in the Rue Morgue"),
        };

        listBox.ItemsSource = data;
        gridOuter.DataContext = data;
    }
}
```

Binding to List i xml

(Fra Binding5_List)

```
<ListBox x:Name="listBox" ItemsSource="{Binding}“></ListBox>
```

Binding to List

(Fra Binding5_List)

```
<ListBox x:Name="listBox" ItemsSource="{Binding}“></ListBox>

<ListBox x:Name="listBox" ItemsSource="{Binding}"
   DisplayMemberPath="ListBoxToString"></ListBox>
```

Binding to List

(Fra Binding5_List)

```
<ListBox x:Name="listBox" ItemsSource="{Binding}“></ListBox>

<ListBox x:Name="listBox" ItemsSource="{Binding}"
   DisplayMemberPath="ListBoxToString"></ListBox>

<ListBox x:Name="listBox" ItemsSource="{Binding}"
   DisplayMemberPath="ListBoxToString"
   IsSynchronizedWithCurrentItem="True"></ListBox>
//sørger for at vi kan vælge et “currentItem” og bruge dette
```

For IsSynchronizedWithCurrentItem så må collectionen – altså list data være set i en parent, f.eks. Grid, da nogle andre gui-children så skal have adgang til samme data

Binding to ObservableCollection (Fra Binding6_ObservableCollection)

```
private ObservableCollection<MyDataClass> data;

private void Button_Click(object sender, RoutedEventArgs e) {
    data.Add(new MyDataClass("Claude Shannon", "A Mathematical Theory of Communication"));
}
```

Forskellen på eksempel 5 og eksempel 6 er at i eksempel 6 er data defineret som ObservableCollection<MyDataClass>, mens det i eksempel 5 blot er en List<MyDataClass>, **som ikke giver besked til GUI når der sker ændringer i elementerne i listen.**

Databinding – andre muligheder

Set binding in code – en anden måde at gøre det på

Dette er hvad vi har gjort indtil nu – I XAML:

```
//Normal binding : Set a property in XAML
<TextBox x:Name="authorTextBox" Text="{Binding Path=Author}" ></TextBox>
<TextBox x:Name="bookTitleTextBox" Text="{Binding Path=BookTitle}"></TextBox>

// Normal binding : Set the DataContext
private void Button_Click(object sender, RoutedEventArgs e) {
            grid.DataContext = data;
}
```

Set binding in code (så ikke i XAML – et alternativ)

```
private void Button_Click(object sender, RoutedEventArgs e) {
    //grid.DataContext = data;

    Binding binding = new Binding();
    binding.Source = data; // or grid.DataContext = data;
    binding.Path = new PropertyPath("Author");
    binding.Mode = BindingMode.OneWay;
    authorTextBox.SetBinding(TextBox.TextProperty, binding);

    binding = new Binding();
    binding.Source = data; // or grid.DataContext = data;
    binding.Path = new PropertyPath("BookTitle");
    binding.Mode = BindingMode.OneWay;
    bookTitleTextBox.SetBinding(TextBox.TextProperty, binding);
}
```

Data templates

Eksempel

Mere advancerede emner findes også – vi vil ikke komme ind på disse:

- Converters
  - Man kan definere converters, som konvereterer data I DataSource objected til den type som kan vises I GUI – og den anden vej konvertere data fra et gui element til den type som skal bruges I data source.
  - Default converters finds: - f.eks. mellem int og string.
  - Hvis man skal konveretere mellem egne custom typer, så kan man lave sin egen Converter class.

- Validation
  - F.eks. Når en brugere kan editere nogle værdier i et gui input felt, så kan disse data valideres, før de bliver automatisk transferet til DataSource. F.eks. ved et dato intput felt eller andet

OPGAVER