

I de følgende opgaver skal vi i gang med mere reel databinding. Og vi skal til at begynde med selv lave objektet som der databindes til. Og i den sidste ende skulle dette gerne ende ud i et simpelt datalag. Der fungere helt separat fra brugerfladen.

I WPF kan vi som sagt databinde de fleste properties til andre properties. Og herunder også properties til vores egne objekter.

Start ud med at lave et nyt WPF projekt og kald det TwoWayTest.

I projektet skal du tilføje endnu en class. Denne kalder du "Person" Indsæt koden herunder til din nye class:

```
public class Person
{
    public int ID;
    public string Fornavn { get; set; }
    public string Efternavn { get; set; }
    public int Formue { get; set; }

    public Person(int ID, string Fornavn, string Efternavn, int Formue)
    {
        this.ID = ID;
        this.Fornavn = Fornavn;
        this.Efternavn = Efternavn;
        this.Formue = Formue;
    }
}
```

Herefter skal du i gang med brugerfladen. Tilføj 3 tekstbokse og en knap, som vist her.

Så skal vi have lavet lidt om i xaml koden, herunder er vist hvad der skal til for at tilføje helt grundlæggende databinding, til de 3 tekstbokse:

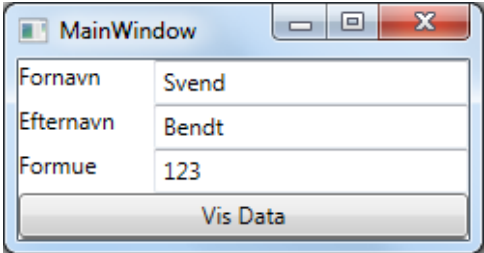
```
<Grid VerticalAlignment="Stretch" HorizontalAlignment="Stretch">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="70" />
        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="27" />
        <RowDefinition Height="27" />
        <RowDefinition Height="27" />
        <RowDefinition Height="32*" />
    </Grid.RowDefinitions>

    <TextBox Height="23" Text="{Binding Fornavn}" HorizontalAlignment="Stretch" Margin="10,2"
        Name="tbFornavn" Grid.Row="0" Grid.Column="1" VerticalAlignment="Stretch" />
    <TextBox Height="23" Text="{Binding Efternavn}" HorizontalAlignment="Stretch" Margin="10,2"
        Name="tbEfterNavn" VerticalAlignment="Stretch" Grid.Column="1" Grid.Row="1" />
    <TextBox Height="23" Text="{Binding Formue}" HorizontalAlignment="Stretch" Margin="10,2"
        Name="tbFormue" VerticalAlignment="Stretch" Grid.Row="2" Grid.Column="1"/>

    <TextBlock Grid.Row="0" Height="23" HorizontalAlignment="Left" Name="textBlock1" Text="Fornavn"
        VerticalAlignment="Center" />
    <TextBlock Grid.Row="1" Height="23" HorizontalAlignment="Left" Name="textBlock2" Text="Efternavn"
        VerticalAlignment="Center" />
    <TextBlock Grid.Row="2" Height="23" HorizontalAlignment="Left" Name="textBlock3" Text="Formue"
        VerticalAlignment="Center" />

    <Button Content="Vis data" Height="23" HorizontalAlignment="Stretch" Margin="10,2" Name="btnVisData"
        VerticalAlignment="Stretch" Grid.Row="3" Grid.ColumnSpan="2" />
</Grid>
```



I Code behind filen skal du tilføje følgende kode:

```
public partial class MainWindow : Window
{
    Person person = new Person(0 "Svend", "Bendt", 100);
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = person;
    }
}
```

Prøv at køre projektet. Som du kan se, kommer de data du har indtastet i Person objektet ud i de rette tekstbokse. Og da tekstboksene understøtter TwoWay Databinding, vil de ændringer du skriver i tekstboksen, automatisk blive sendt tilbage til objektet.

For at se dette i aktion. Skal du tilføje følgende koden til **"Vis data" knappens click event**:

```
string PersonData = person.Fornavn +
    " " +
    person.Efternavn +
    " har en formue på " +
    person.Formue +
    " Kr.";

MessageBox.Show(PersonData);
```

Hvis du starter projektet igen og ændrer noget i en af tekstboksene. Vil du ved et tryk på "Vis data" knappen, se at informationerne i Person objektet også er ændret.

Dette er jo som forventet. Men hvad nu hvis data i objektet bliver ændret, uden om vores brugerinterface? Lad os sige at data i objektet bliver hentet fra en database. Og at disse data er blevet opdateret, siden vi startede vores program op. Hvad sker der så??

For at efterprøve dette, skal du tilføje en "Update" knap til din brugerflade under "Vis data" knappen. Og i click eventet skal du tilføje koden herunder:

```
private void btnUpdate_Click(object sender, RoutedEventArgs e)
{
    person.Formue++;
}
```

Prøv nu at starte projektet igen og tryk på update knappen et par gange. Som du kan se sker der ikke meget i brugerfladen, men trykker du på vis data knappen er sagen en anden. Person objektet blevet opdateret!

Dette er jo ikke optimalt. Da vi gerne skulle have vores brugerflade til at opdatere sig selv, når vores datakilde opdateres.

For at få denne form for funktionalitet, skal vi implementere dependency properties i vores Person objekt. Og umiddelbart er det en hel del kode der skal til:

```
using System.ComponentModel; // Tilføjes i toppen af din C# fil!

(1) class Person : INotifyPropertyChanged
{
    public int ID;
    public string Fornavn { get; set; }
    public string Efternavn { get; set; }

(2)     private int _formue;
    public int Formue
    {
        get
        {
            return _formue;
        }
        set
        {
            _formue = value;
(3)         OnPropertyChanged("Formue");
        }
    }

    public Person(int ID, string Fornavn, string Efternavn, int Formue)
    {
        this.ID = ID;
        this.Fornavn = Fornavn;
        this.Efternavn = Efternavn;
        this.Formue = Formue;
    }

(4)     public event PropertyChangedEventHandler PropertyChanged;

(5)     private void OnPropertyChanged(string PropertyNavn)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(PropertyNavn));
        }
    }
}
```

- (1) Først og fremmest har jeg tilføjet implementeringen af INotifyPropertyChanged Interfacet. Dette er en del af System.ComponentModel namespace. Derfor tilføjes af using statementen øverst.
- (2) Herefter har jeg tilføjet alt koden til "Formue" propertyen. Dette er nødvendigt, da vi nu skal lave andet, end overføre værdien til den bagvedliggende variabel.
- (3) Som du kan se, er der nu også tilføjet et kald, til OnPropertyChanged Metoden. Som er en hjælpe funktion jeg har lavet længere nede.
- (4) Dette er det underliggende event, vi skal fyre hver gang vi har lavet ændringer til de enkelte properties.
- (5) Dette er en simpel hjælpefunktion, vi benytter til at fyre eventet. Først kigger vi efter om der er nogle abonnenter på eventet. Og hvis dette er tilfældet, fyrrer vi eventet af, med en PropertyChangedEventArgs indeholdende navnet på det event, der er ændret.

Så en dependency property er grundlæggende en helt almindelig property, der gør brug af et underliggende event. Hvilket gør det muligt for abonnenter at se når der er sket ændringer i de enkelte properties.

Prøv nu at tilføje tilsvarende getters og setters til Fornavn og Efternavn og kørså programmet og se om ikke informationerne opdateres efter hensigten, når du trykker "Update" knappen.

Hvis du har afprøvet din kode og tingene virker efter hensigten, kan vi gå videre med et lidt mere avanceret eksempel på hvorledes vi også kan databinde større mængder data.

Start med at tilføje følgende kode til dit program:

```
ObservableCollection<Person> Personer = new ObservableCollection<Person>();

Person person = new Person("Svend", "Bendt", 123);
public MainWindow()
{
    InitializeComponent();

    Personer.Add(person);
    Personer.Add(new Person(0, "Bein", "Stagge", -987654321));
    Personer.Add(new Person(1, "Turt", "Khorsen", 0));
    Personer.Add(new Person(2, "Gill", "Bates", int.MaxValue));

    this.DataContext = Personer;
}
```

Her laver vi en Liste af typen Person, som så selvfølgelig skal hedde Personer. Og vi tilføjer et par personer mere til listen hvis økonomi vi ønsker at holde øje med...

Derudover har vi ændret `this.DataContext` så den nu peger på den nye datasamling `Personer`. Hvis du kører dit program igen, vil du se at der ikke er ændret ret meget. Dette skyldes at `TextBoksene` kun kan vise et enkelt element af gangen. Og det viser derfor kun det første element i vores liste.

Hvis vi vil have flere data frem, er vi nødt til at vise dem via et af de paneler i toolboksen, der kan håndtere flere data (gentagende data)

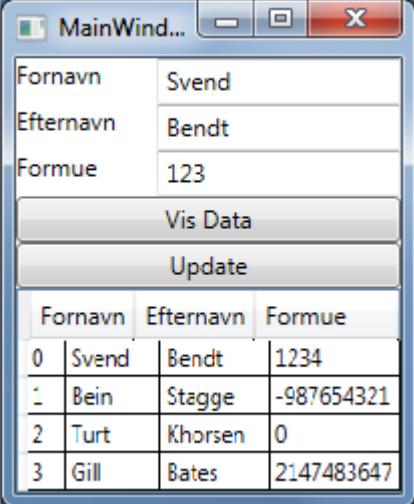
Start med at lægge et `DataGrid` ind nedenunder knapperne og sætte de to properties på datagriddet, som vist herunder:

```
AutoGenerateColumns="True" ItemsSource="{Binding}"
```

`AutoGenerate Columns`, indikerer ganske enkelt at vores `DataGrid` selv skal finde ud hvor mange Kolonner der skal være. Og den vælger derudover en overskrift til hver kolonne, ud fra de enkelte property navne. Og `ItemSource` har du jo benyttet før, den indikere at vi skal hente vores indhold via `DataBinding`.

Start dit program igen og se om ikke resultatet ser ud som her?

Altså er det med relativ små midler muligt at lave en fuld databinding via objekter og et `DataGrid` i WPF.



	Fornavn	Efternavn	Formue
0	Svend	Bendt	1234
1	Bein	Stagge	-987654321
2	Turt	Khorsen	0
3	Gill	Bates	2147483647

Og hvis du prøvet at trykke på `Update` Knappen, vil du se at vores notify implementering stadig virker som forventet. Og at det selvfølgelig er muligt at ændre i `DataGrid`'et og så se ændringerne slå igennem i tekstboksene også. (Hvis du vil undgå at brugeren kan ændre i dit datagrid, kan du sætte `IsReadOnly="True"` propertyen på det.

Men hvad med de tilfælde hvor vi ikke vil have data i et `DataGrid`? Hvis vi i stedet vil organisere vores data selv, eksempelvis via et `ListBox`, eller et `ListView`?

Kan vi bare benytte disse elementer ved at trække dem ud fra toolboksen og så sætte `ItemsSource="{Binding}"`?!...

Ja, og nej... For både et `ListView` og en `ListBoks`, ved hvordan man gentager indhold flere gange. Men den aner ikke hvorledes den skal vise dine data, da det er et Objekt du har lavet. Havde det været en liste af strings, ville det ikke have været noget problem. Men en liste af typen `Person`, ved den ikke hvordan skal repræsenteres. Altså skal vi selv til at lave en "`DataTemplate`". Hvilket er emnet i næste opgave