# WPF And MVVM

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf1**: anvende centrale metoder til at specificere og konstruere algoritmer […] * **1Pf2**: anvende centrale faciliteter i programmeringssproget til realisering af […] brugergrænseflader. |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Unistrukturel] Du kan identificere enkelte kendetegn ved **applikationstypen** GUI applikation – WPF, XAML, MVVM, UI-kontroller, databinding (inkl. interfacet INotifyPropertyChanged) * [Unistrukturel] Du kan identificere enkelte kendetegn ved **Arkitekturmønstre** – Model-View-ViewModel (MVVM) |
| **Din forberedelse** | Programmeringssporet:   * MVVM-arkitekturen:   + [Model–view–viewmodel](https://en.wikipedia.org/wiki/Model-view-viewmodel)   + [MVVM – Introduction](https://www.tutorialspoint.com/mvvm/mvvm_introduction.htm)   + [MVVM – Advantages](https://www.tutorialspoint.com/mvvm/mvvm_advantages.htm)   + WPF–MVVM Intro (video: 15:21; se planen i ItsLearning) * Databinding:   + [Introduction to WPF data binding](https://www.wpf-tutorial.com/data-binding/introduction/)   + [Hello, bound world!](https://www.wpf-tutorial.com/data-binding/hello-bound-world/) – the Binding syntax in XAML (Binding, Path, ElementName)   + [Using the DataContext](https://www.wpf-tutorial.com/data-binding/using-the-datacontext/)   + [Data binding via Code-behind](https://www.wpf-tutorial.com/data-binding/data-binding-via-code-behind/)   + [The UpdateSourceTrigger property](https://www.wpf-tutorial.com/data-binding/the-update-source-trigger-property/)   + [BindingMode Enum](https://docs.microsoft.com/en-us/dotnet/api/system.windows.data.bindingmode?view=netcore-3.1)   + [Explain **INotifyPropertyChanged** In WPF - MVVM](https://www.c-sharpcorner.com/article/explain-inotifypropertychanged-in-wpf-mvvm/)   + [Slider](https://www.wpf-tutorial.com/misc-controls/the-slider-control/)-kontrol   + Databind 1 – Intro (video: 8:21; se planen i ItsLearning)   + Databind 2 – XAML to XAML (video: 11:55; se planen i ItsLearning)   + Databind 3 – XAML to Object 1 (video: 15:07; se planen i ItsLearning)   + Databind 4 – XAML to Object 2 (video: 17:03; se planen i ItsLearning) |

Forrige uge udviklede du GUIs ved hjælp af XAML og kode i XAML-filens ’code-behind’-fil. Du fik det uden tvivl til at fungere fint, men oplevede nok også, at der var rigtig meget kode i ’code-behind’-filen med mange event-handlers. Man kan nemt forestille sig, at jo mere kompleks GUI’en er med mange UI-kontroller og indbyrdes sammenhæng mellem dem, desto større er sandsynligheden for, at ’code-behind’-filen bliver uoverskuelig og vanskelig at vedligeholde.

I denne uge skal du se på, hvordan man griber det an på en anden mere velovervejet og struktureret måde endda med det designmål at have mindst mulig kode i ’code-behind’-filen (helst ingen). Dette involverer fuld brug af arkitektur-mønstret Model-View-ViewModel (MVVM) og specielt med databinding, som er designet til at fungere tæt sammen i WPF. Øvelserne i dagens opgave gennemgår forskellige variationer af databinding. Formålet er at give dig et repertoire af muligheder, når du selv skal bruge databinding og MVVM i dine egne projekter fremover og samtidig sikre løs kobling mellem et View (i UI-laget) og en ViewModel (i applikationslaget).

# Dagens ord

WPF benytter sig af MVVM-arkitekturen, dvs. Model-View-ViewModel[[1]](#footnote-1). Du skal øve dig i to forskellige former for databinding, som begge er relevante ved udvikling af en grafisk brugergrænse­flade (GUI) i WPF. I WPF repræsenteres eller defineres et View via XAML-notationen. Den ene form for databinding sker internt i et XAML-View (XAML til XAML), og den anden mellem et XAML-View og et C# objekt uden for XAML, f.eks. et ViewModel-objekt. Ved hjælp af C# events (via INotifyPropertyChanged-interfacet) opnås en tæt databinding mellem UI-kontroller i et View og ViewModel-objekter i ViewModel-laget. C# events vil (sammen med Observer-mønstret) blive forklaret i de kommende uger. I opgaven behøver du bare at bruge event’et i INotifyPropertyChanged-interfacet og ikke at forstå dem til bunds.

# Øvelse 1: Terminologi

Del teamet op i mindre grupper og brug **Ordet rundt** til at reflektere over begreberne ”XAML”, ”Databinding”, ”Source”, ”Target”, ”Path”, ”ElementName” og ”DataContext”.  
Sørg for at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

**Benyt parprogrammering i alle følgende øvelser**.

**XAML** is a markup language used to create a GUI.

**Databinding** is binding the value a property of an object to the value of another property in another object. Either between two objects in XAML, or between an object in XAML and an object in the model. They can be oneway, where when the source property is updated the other one is updated automatically, but not the other way around. Bidirectional(twoday) binding is where if one property is updated, the other one is too. Reverse databinding, from the UI to the model.

**Source** is where the update is coming from that is reflected in the binded property.

**Target** is the property that is changed based on a change in the souce property.

**Path** is set equal to the property that you want to bind to, i.e. {Binding Path=property}, path is the default binding property though, so {Binding Property} is enough.

**ElementName** is another binding property which connects directly to another UI element, binding properties are separated by comma, so {Binding Path=Property, ElementName=UIElementName}

**DataContext** is the default data context for a set of controls, it’s default value is null and child controls inherits data context from it’s parent control. DataContext can be overwritten with the ElementName property.

# Øvelse 2: Databinding i XAML[[2]](#footnote-2)

Den første form for databinding sker mellem to eller flere UI-kontroller i XAML. Dette kan gøres på flere forskellige måder, hvor du i denne øvelse kun skal fokusere på en af dem, nemlig at anvende ElementName til at skabe en forbindelse fra en UI-kontrol til en anden.

Det kræver to trin at skabe en forbindelse mellem et Target-objekt og et Source-objekt i XAML:

1. Sørg for at Source-objektet har et navn; dette kan angives enten i ”x:Name”-attributten i XAML for Source-objektets XML-starttag eller i objektets Name-property i Visual Studio’s Properties-View.
2. Lad ElementName referere til Source-objektet fra Target-objektet via dette navn i en ”Binding”
3. Brug til sidst ”Path” til at referere til den property i source-objektet, du ønsker at databinde til.

Man kan angive en Binding både i C# kode og i XAML. I denne opgave skal du udelukkende anvende XAML til at definere Bindings, så vi helt undgår at angive kode i code-behind filen.

## Øvelse 2.1: Forberedelse

Udfør først følgende trin:

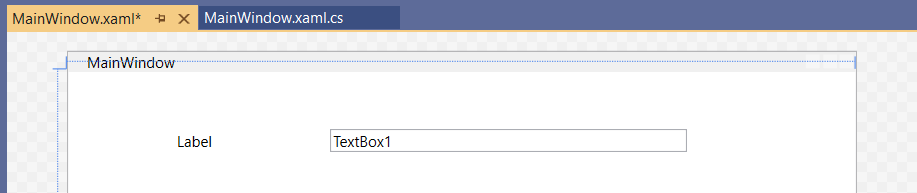
* Opret et helt nyt projekt med navnet ”WPFAndMVVM1”, hvor du vælger **WPF Application** som projektskabelon.
* Åbn MainWindow.xaml-filen, og orienter dig i editoren for MainWindow. Du vil se et split-vindue med det grafisk GUI-design øverst og den tilhørende XAML-fil nederst. Du kan rette enten i Design- eller XAML-vinduet, og rettelsen vil afspejles begge steder. Husk at XAML-filen er en XML-fil, og skal være korrekt XML med start-tags og afsluttende end-tags.
* Åbn Toolbox-vinduet (View->Toolbox), hvor du kan se alle WPF-kontroller, der kan indsættes i din GUI.

## Øvelse 2.2: Databinding mellem Label og TextBox i XAML

Du starter med at etablere en databinding mellem to simple UI-kontroller, nemlig en Label og en TextBox.

Udfør følgende:

* Indsæt øverst i dit MainWindow (enten i Design- eller XAML-vinduet) en Label-kontrol og til højre for den en TextBox-kontrol:



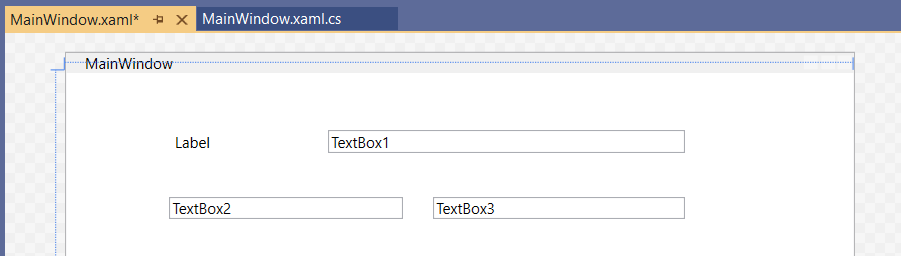
* Indholdet af din TextBox skal vises i din Label, så sørg for i XAML at oprette en databinding mellem Label og TextBox, der sikrer, at en ændring af værdien i TextBox afspejles i Label, dvs.:
  + Sørg for at Label-kontrollen entydigt kan referere til TextBox-kontrollen
  + Overvej hvilken property i hver kontrol skal indgå i denne databinding
  + Etablér din databinding mellem de to kontroller
  + Kør koden, og ret i din XAML, hvis det ikke virker

## Øvelse 2.3: Databinding mellem TextBox og TextBox i XAML

Du skal nu skabe en databinding mellem to TextBox-kontroller. Det bliver en anelse mere kompleks, da begge UI-kontroller’s indhold kan ændres og nu har mulighed for at påvirke hinanden. Du skal finjustere bindingens adfærd ved brug af UpdateSourceTrigger og binding modes.

Udfør følgende:

* Indsæt to TextBox-kontroller ved siden af hinanden (neden under de to kontroller fra øvelse 2.2):



* Lad den venstre TextBox (target) databinde til den højre TextBox (source), dvs. afgør hvilken property i target skal forbindes til hvilken property i source (Path).

Overvej nu:

* Hvad forventer du vil ske, når du ændrer tekstfeltet i henholdsvis venstre og højre TextBox?
* Kør programmet, og se efter om det fungerer, som du forventer.

### UpdateSourceTrigger

Du opdager måske, at mens du ændrer teksten i den højre TextBox (source), så ændrer teksten i den venstre TextBox (target) med det samme bogstav for bogstav, mens det omvendte ikke er tilfældet. Ændringen i venstre TextBox slår kun igennem i højre TextBox, når du fjerner fokus (dvs. du har klikket på noget andet i GUI’en) fra venstre TextBox.

Udfør følgende:

* Overvej hvordan du kan anvende UpdateSourceTrigger til at sørge for at en ændring i teksten for højre TextBox (target) øjeblikkeligt vises i den venstre TextBox (source).
* Implementér ændringen. Kør dit program, og ret indtil det virker korrekt.

### Binding modes

Grunden til, at de to TextBox-kontroller i det hele taget kan opdatere hinanden (uanset værdien af UpdateSourceTrigger), skyldes default-værdien af ”Binding mode” i bindingen mellem de to TextBox-kontroller. Det er vigtigt at forstå, hvordan de forskellige ”binding modes” fungerer for at opnå den fulde kontrol over interaktionen mellem target og source i en binding.

**OneTime**: Udfør følgende:

* Sæt binding mode for bindingen mellem de to TextBox-kontroller til ”OneTime”
* Overvej nu, hvad effekten vil være, når du opdaterer henholdsvis venstre og højre TextBox.
* Kør dit program, og afprøv om du havde ret.

**OneWay**: Udfør følgende:

* Sæt binding mode for bindingen mellem de to TextBox-kontroller til ”OneWay”
* Overvej nu, hvad effekten vil være, når du opdaterer henholdsvis venstre og højre TextBox.
* Kør dit program, og afprøv om du havde ret.

**OneWayToSource**: Udfør følgende:

* Sæt binding mode for bindingen mellem de to TextBox-kontroller til ”OneWayToSource”
* Overvej nu, hvad effekten vil være, når du opdaterer henholdsvis venstre og højre TextBox.
* Kør dit program, og afprøv om du havde ret.

**TwoWay**: Udfør følgende:

* Sæt binding mode for bindingen mellem de to TextBox-kontroller til ”TwoWay”
* Overvej nu, hvad effekten vil være, når du opdaterer henholdsvis venstre og højre TextBox.
* Kør dit program, og afprøv om du havde ret.

Til sidst:

* Hvad tror du, default binding mode er for en TextBox?

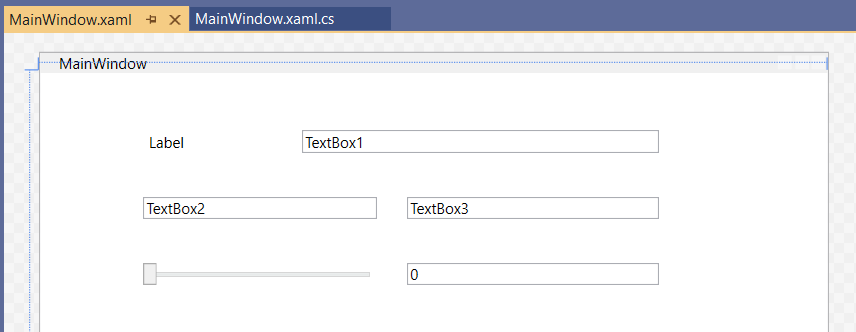
Where you define the binding is the target and the elementName.Path is the source, OneTime lets a change happen one time, OneWay is only from the source to the target, i.e. from the ElementName.Path to where the binding is defined, OneWayToSource is the same as OneWay, but the other way around, TwoWay is the default binding mode and a change in either the target or source property will be reflected in the other.

## Øvelse 2.4: Databinding mellem Slider og TextBox i XAML

Du skal nu benytte de principper, du har lært foroven, til at skabe en databinding mellem en Slider-kontrol og en TextBox-kontrol, som angivet forneden.

Udfør følgende:

* Under de to TextBox’e fra øvelse 2.3, indsæt en Slider-kontrol, og sørg for, at dens værdi minimum kan være 0 og maksimum 100.
* Til højre for Slider’en, indsæt en TextBox, hvis starttekst er ”0”:



* Opret nu en databinding mellem din TextBox (som target) og Slider (som source) og således, at en ændring i Slider-kontrollen **øjeblikkeligt** afspejles i TextBox-kontrollens tekst og tilsvarende en ændring i TextBox-kontrollens værdi (tal mellem 0 og 100) **øjeblikkeligt** afspejles i Slider-kontrollens position.
* Kør dit program, og ret i XAML, indtil den virker korrekt.

Udfør endvidere:

* Vend nu din databinding om, så Slider-kontrollen er bindingens source og TextBox-kontrollen dens target (dette kræver at du sletter din første binding).
* Konfigurer bindingen, så du forventer, den vil fungere præcist som før
* Kør dit program, og afprøv om du havde ret. Hvis ikke, så ret i XAML, indtil det virker korrekt.

Du har nu lært de vigtigste elementer i en databinding mellem to UI-kontroller i XAML, og hvordan de kan finjusteres. Sådanne XAML-til-XAML bindinger vil der ofte være i en GUI med mange UI-kontroller. Men hvad nu hvis du gerne vil lave en databinding fra en UI-kontrol i XAML til et eksisterende C#-objekt i din C#-kode? I WPF og MVVM vil dette vil typisk ske, hvis du ønsker, at en UI-kontrol i dit View skal læse data fra et objekt i dit ViewModel-lag.

# Øvelse 3: Databinding mellem XAML og et eksisterende C#-objekt

I denne øvelse skal du etablere en databinding mellem UI-kontroller i XAML og et eksisterende C#-objekt uden for XAML-delen (ofte et ViewModel-objekt). Nu kan du ikke anvende ElementName som før til at få forbindelse til en binding source, da dit C#-objekt slet ikke er defineret i XAML. I stedet for skal du anvende en DataContext til at skabe forbindelsen til source-objektet.

Det kræver to trin at skabe en forbindelse mellem et Target-objekt i XAML og et Source-objekt i C#:

1. Sæt DataContext for den relevant UI-kontrol til det Source C#-objekt (oftest en ViewModel) som indeholder de properties, du gerne vil databinde med.   
   DataContext kan med fordel sættes i code-behind (MainWindow.xaml.cs) **lige efter** kaldet til ”InitializeComponent()” i MainWindow-constructor’en.
2. For hver databinding til properties i DataContext’en:
   1. Benyt Path til at referere (fra Target-kontrollen i XAML) til den property i Source-objektet (sat i DataContext), du gerne vil databinde til.   
      **Husk at du kun kan databinde til en property i en source og aldrig et felt.**

## Øvelse 3.1: Forberedelse

Udfør følgende:

* Opret en ny C#-klasse med navnet ”MainViewModel”, og sørg for, at klassen er public.
* Indsæt to **public properties** ”MyLabelText” og ”MyTextBoxText” (begge af typen string) i din MainViewModel-klasse og initialisér begge properties med tekststrengen:
  + ”Text not set yet” (genopfrisk, hvordan man initialiserer en property)

Udfør dernæst:

* Tilret din eksisterende code-behind-fil ”MainWindow.xaml.cs”, så koden for MainWindow-klassen ser ud som følger:

public partial class MainWindow : Window

{

**MainViewModel mvm = new MainViewModel();**

public MainWindow()

{

InitializeComponent();

**DataContext = mvm;**

}

}

Dit View opretter hermed en instans af din MainViewModel-klassen, og sikrer, at DataContext for MainWindow (dvs. hovedvinduet) netop bliver sat til denne instans. Dette betyder, at du nu via databinding fra XAML har adgang til alle public properties i MainViewModel.

## Øvelse 3.1: Databinding mellem Label og MyLabelText-property (i MainViewModel)

Du skal nu etablere en databinding fra en Label til MainViewModel’s MyLabelText-property. Dvs. værdien af Label-kontrollen hentes fra MyLabelText.

Udfør følgende:

* Indsæt en ny Label-kontrol (under alle de andre kontroller fra øvelse 2)
* Med Path lav en databinding til MyLabelText
* Kør dit program, og afprøv, om det virker, som du forventer.
* Hvis ikke, så ret i din XAML, og prøv igen.

### Ændring af teksten i Label

Du vil nu gerne ændre teksten i MyLabelText-property for at se, hvilken effekt det har på Label-kontrollens tekst; den er jo bundet til MyLabelText.

Udfør følgende:

* Indsæt en Button-kontrol til venstre for din Label og sæt dens Content til teksten ”Update Label”
* Dobbelt-klik på Button for at oprette en button-click eventhandler
* I denne eventhandler da skriv følgende:

private void Button\_Click(object sender, RoutedEventArgs e)

{

mvm.MyLabelText = DateTime.Now.ToString();

}

Dvs. når du klikker på knappen ”Update Label” i GUI’en, så ændres MyLabelText i din MainViewModel til teksten for den lokale dato og tidspunkt lige nu (DateTime.Now). Denne værdi, dvs. DateTime.Now ændrer sig hvert sekund, så gentagne klik på knappen, vil hvert sekund give MyLabelText en ny værdi.

* Overvej nu, hvad der vil ske i din GUI, når du klikker på den nye knap.
* Kør dit program, og observér om Label-teksten ændres.
* Diskutér med din makker, hvorfor udfaldet blev som det blev, og hvad du kan gøre for at sikre, at Label-teksten bliver opdateret med den nye MyLabelText-værdi.

Overvej:

* Hvilken binding mode har en Label-kontrol som default? Kan det påvirke udfaldet?
* Prøv en anden binding mode, og se om det ændrer adfærden.

Labels default binding mode is OneWay, which means that the source updates the target and not vice versa. This isn’t the reason because the source, which is mvm.MyLabelText is updated.

## Øvelse 3.2: INotifyPropertyChanged

Når du anvender en XAML UI-kontrol som source (som du så på i øvelse 2), så databinder du til en UI-kontrol’s property, som i XAML er speciel-implementeret som en DependencyProperty. Faktisk nedarver både target og source properties fra DependencyProperty-klassen, som har indbygget kode til at notificere modparten, hvis der sker en ændring. Det er lidt avanceret, så det eneste du behøver at vide lige nu er:

* Når du databinder til en property i et eksisterende C#-objekt uden for XAML, så databinder du til en helt almindelig property (som du kender med en getter og setter) uden nogen form for kode, der kan notificere et target-objekt om en ændring i property’en.  
  **Bemærk: Denne notifikation skal du selv sørge for.**

Det er her interfacet INotifyPropertyChanged kommer i spil. Dette interface angiver et Event PropertyChanged, som skal kaldes, hvis der sker en ændring.

Udfør nu følgende:

* Lad MainViewModel-klassen nedarve fra interfacet INotifyPropertyChanged
* Implementér interfacet og supplér med OnPropertyChanged-hjælpemetoden, så klassen samlet ender med følgende kodetilføjelse:

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

PropertyChangedEventHandler propertyChanged = this.PropertyChanged;

if (propertyChanged != null)

{

propertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

Opbygning af ovenstående kode gennemgås i de følgende uger. Lige nu skal du bare sørge for at kalde OnPropertyChanged-metoden, hvis værdien af en property ændrer sig. Navnet på property’en angives i propertyName-parameteren.

Bemærk, at hvis du gerne vil have, at XAML får besked om, at en property, der er databindet til, har ændret sig, så skal du implementere INotifyPropertyChanged-interfacet for source-objektet OG notificere om ændringen ved at aktivere event’et (via OnPropertyChanged-metoden). **Dette er et krav.**

* Opdatér din MyLabelText-property, så den kalder OnPropertyChanged-metoden, hvis værdien ændrer sig (vink: det må være i set-delen). Når du angiver ”MyLabelText” som parameter til OnPropertyChanged, så vær absolut sikker på, at du har angivet det præcist som dette er defineret med både store og små bogstaver.
* Prøv nu igen, om dit program virker; gerne med forskellige binding modes.

Udfør endvidere:

* Opret nu tilsvarende en ny TextBox i GUI’en, og databind den til MyTextBoxText-property’en. Tilføj en knap ”Update TextBox”, som opdaterer indholdet af den nye tekstboks med en ny dato-tid som oven for. Afprøv om det virker. Tilret, hvis ikke.
* Hvis du ændrer tekstboksens indhold i GUI, hvordan kan du tjekke om MyTextBoxText-property’en faktisk også ændrer sig tilsvarende? By watching the property during debugging, it changes because TextBox is by default TwoWay

En sidste bemærkning omkring INotifyPropertyChanged er, at selvom dette interface har speciel betydning i forhold til XAML, begrænser det sig ikke til at databinde mellem XAML og et C#-objekt, men også mellem C#-objekter generelt uden tilknytning til en XAML eller en GUI. Men det ser vi på i senere opgaver.

1. I MVVM hører et View til UI-laget i Larman’s lagdelingsmodel, en ViewModel til applikationslaget og en Model til Domænelaget. [↑](#footnote-ref-1)
2. Bemærk i øvrigt, at XAML-notationen ikke begrænser sig til WPF, men også anvendes til at definere GUIs på andre platforme, f.eks. mobil-platforme via Xamarin, eller til Universal Windows Platform (UWP). Så principperne du lærer her, kan bruges i flere sammenhænge (med visse variationer). [↑](#footnote-ref-2)