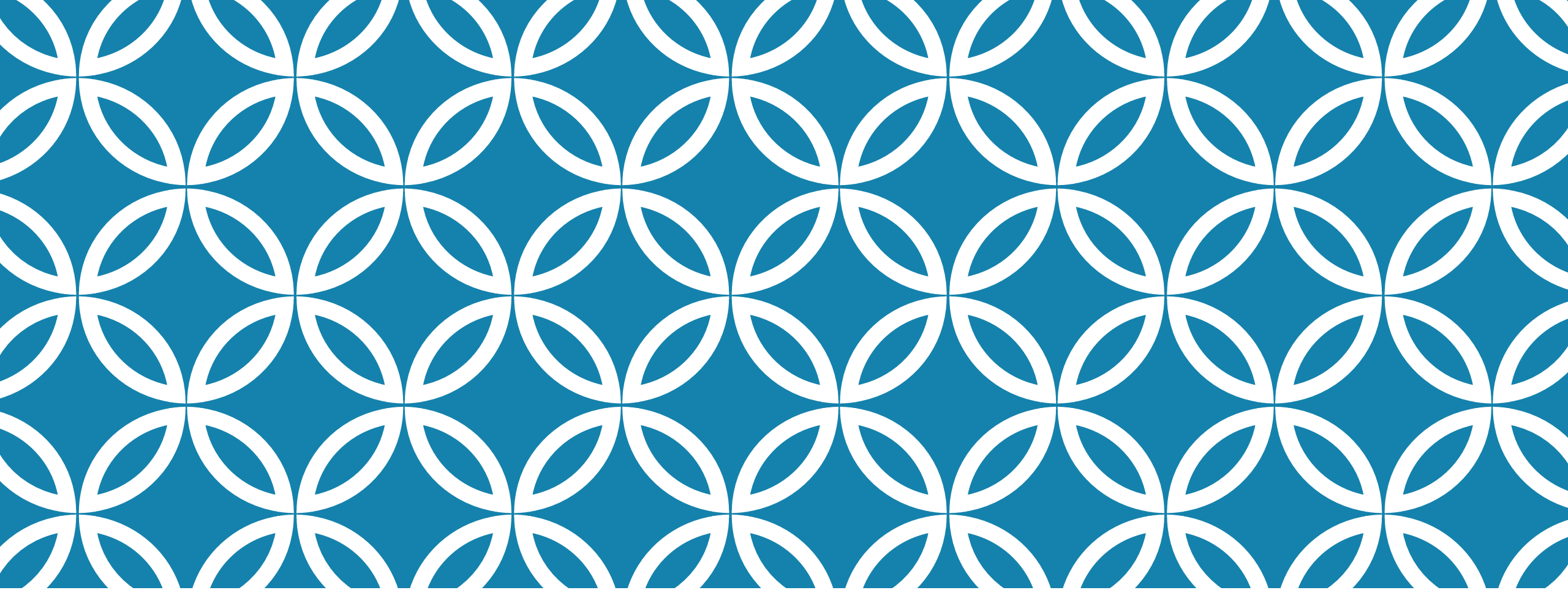


WINDOWS PRESENTATION FOUNDATION LEKTION 3

Mahmud Al Hakim
mahmud@alhakim.se
www.alhakim.se

AGENDA

Introduktion till Databindning (Data Binding)

Element Binding

Data Context

Stringformat

Object Binding

Gränssnittet INotifyPropertyChanged (INPC)

ObservableCollection

DATA BINDING

"Windows Presentation Foundation (WPF) data binding provides a simple and consistent way for applications to present and interact with data. Elements can be bound to data from a variety of data sources in the form of common language runtime (CLR) objects and XML. ContentControls such as Button and ItemsControls such as ListBox and ListView have built-in functionality to enable flexible styling of single data items or collections of data items. Sort, filter, and group views can be generated on top of the data."

Källa

https://msdn.microsoft.com/library/ms752347%28v=vs.100%29.aspx

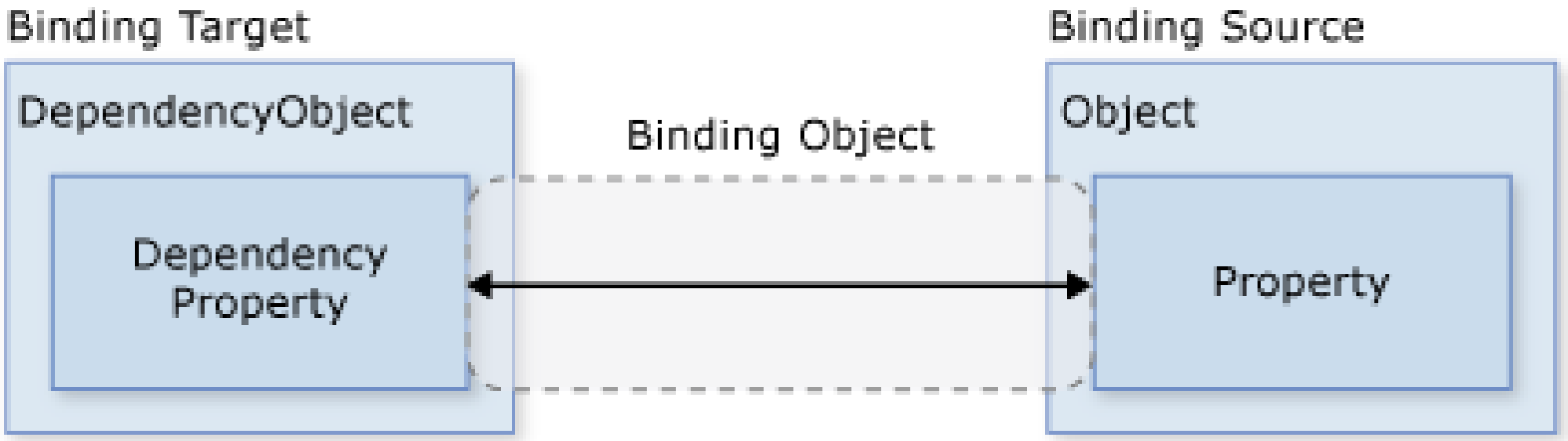
VAD ÄR DATA BINDING

"Data binding is the process that establishes a connection between the application UI and business logic. If the binding has the correct settings and the data provides the proper notifications, then, when the data changes its value, the elements that are bound to the data reflect changes automatically. Data binding can also mean that if an outer representation of the data in an element changes, then the underlying data can be automatically updated to reflect the change. For example, if the user edits the value in a TextBox element, the underlying data value is automatically updated to reflect that change."

Källa

https://msdn.microsoft.com/en-us/library/ms752347%28v=vs.100%29.aspx#what_is_data_binding

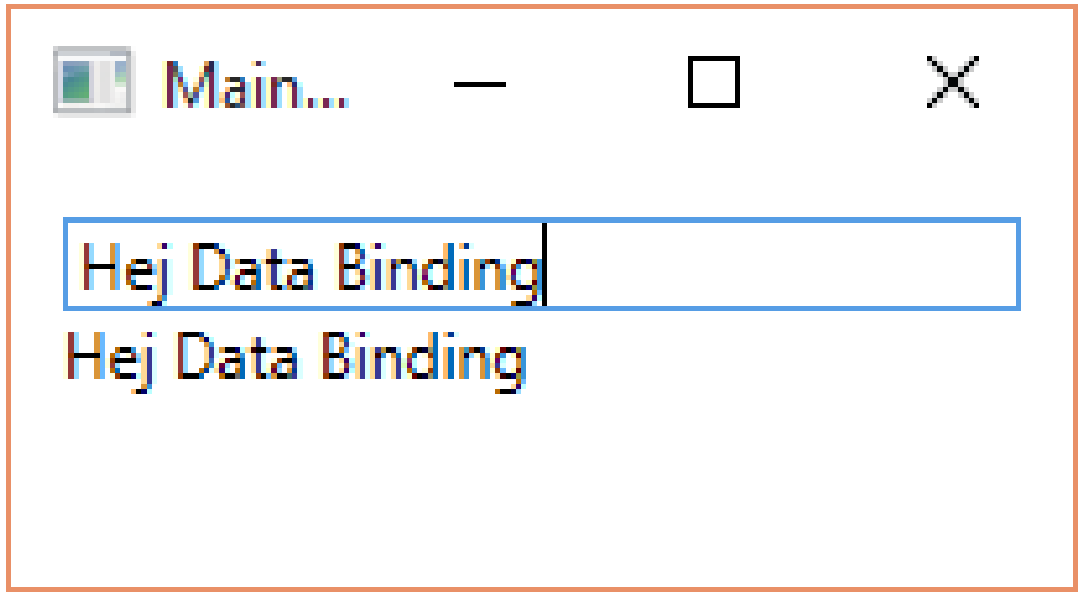
DATA BINDING MODEL



Bildkälla:
https://msdn.microsoft.com/library/ms752347%28v=vs.100%29.aspx#basic_data_binding_concepts

ELEMENT BINDING

```
<StackPanel Margin="10">
<TextBox Name="data" />
<TextBlock
Text="{Binding Path=Text, ElementName=data}" />
</StackPanel>
```



Tips
Att skriva ett värde med klamrar kallas {Markup Extension}

KLASSEN BINDING

```
<TextBlock Text="{Binding Path=Text, ElementName=data}" />
```

Binding [from metadata]

```
11  namespace System.Windows.Data
12  {
13      //
14      // Summary:
15      //     Provides high-level access to the definition of a binding, which connects the
16      //     properties of binding target objects (typically, WPF elements), and any data
17      //     source (for example, a database, an XML file, or any object that contains data).
18      public class Binding : BindingBase
19      {
```

EGENSKAPEN PATH

Tips!
Path=
kan utelämnas om källan är den första egenskapen som skrivs direkt efter Binding

```
<TextBlock Text="{Binding Path=Text, ElementName=data}" />
```

```
169         //     Gets or sets the path to the binding source property.
170         //
171         // Returns:
172         //     The path to the binding source. The default is null.
173         public PropertyPath Path { get; set; }
```

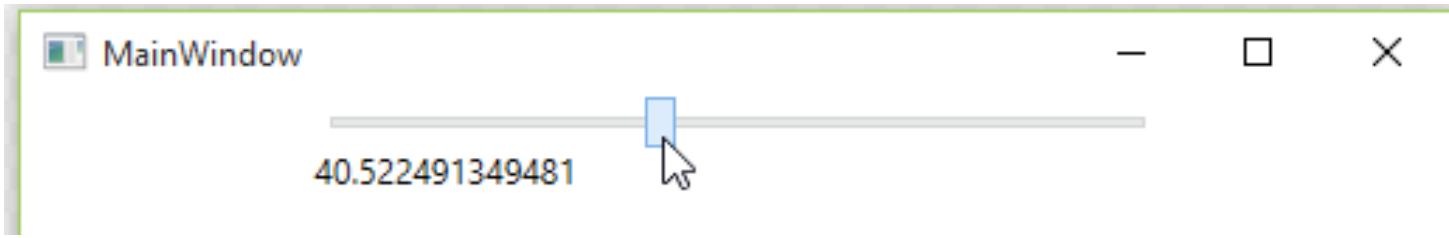
EGENSKAPEN ELEMENTNAME

```
<TextBlock Text="{Binding Path=Text, ElementName=data}" />
```

Binding [from metada

```
100     //
101     // Summary:
102     //     Gets or sets the name of the element to use as the binding source object.
103     //
104     // Returns:
105     //     The value of the Name property or x:Name Directive of the element of interest.
106     //     You can refer to elements in code only if they are registered to the appropriate
107     //     System.Windows.NameScope through RegisterName. For more information, see WPF
108     //     XAML Namescopes.The default is null.
109     [DefaultValue(null)]
110     public string ElementName { get; set; }
```

ELEMENT BINDING – EXEMPEL 2




```
<StackPanel>
<Slider Name="mySlider"
        Minimum="0"
        Maximum="100"
        Width="300" />
<TextBlock Width="300"
Text="{Binding Value, ElementName=mySlider}" />
</StackPanel>
```

ELEMENT BINDING – EXEMPEL 3

```
<StackPanel Orientation="Horizontal" Height="25">
  <Label Target="{Binding ElementName=namn}">
    _Namn:
  </Label>
  <TextBox x:Name="namn" Width="200"/>
  <Label Target="{Binding ElementName=epost}">
    _Epost:
  </Label>
  <TextBox x:Name="epost" Width="200"/>
</StackPanel>
```

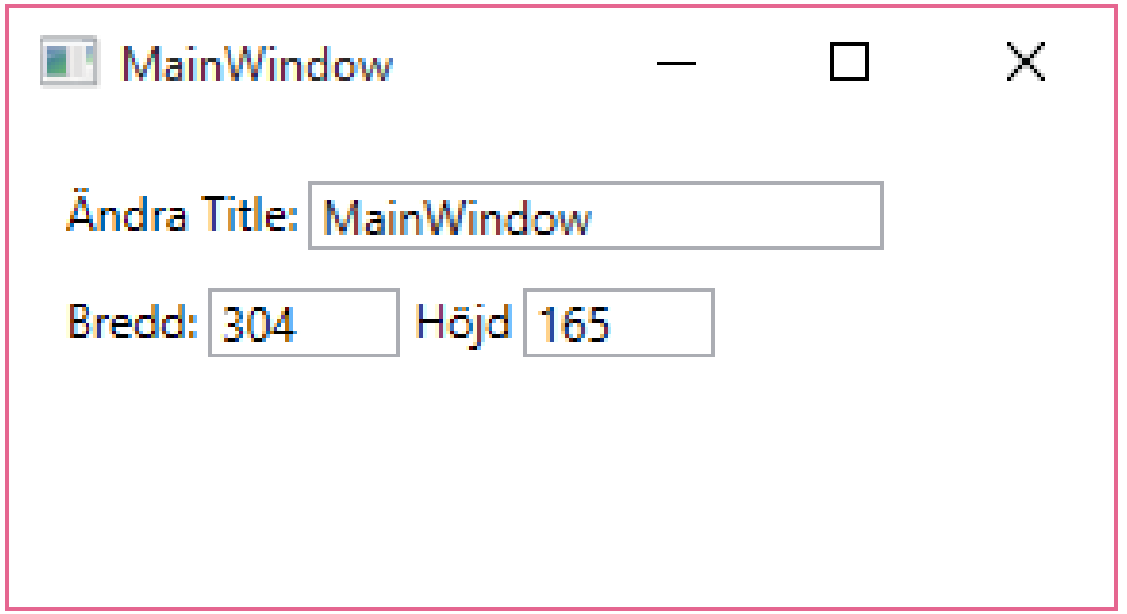
EGENSKAPEN DATACONTEXT

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

        }
    }
```

Gets or sets the data context for an element
when it participates in data binding.
Returns: The object to use as data context.

ÖVNING



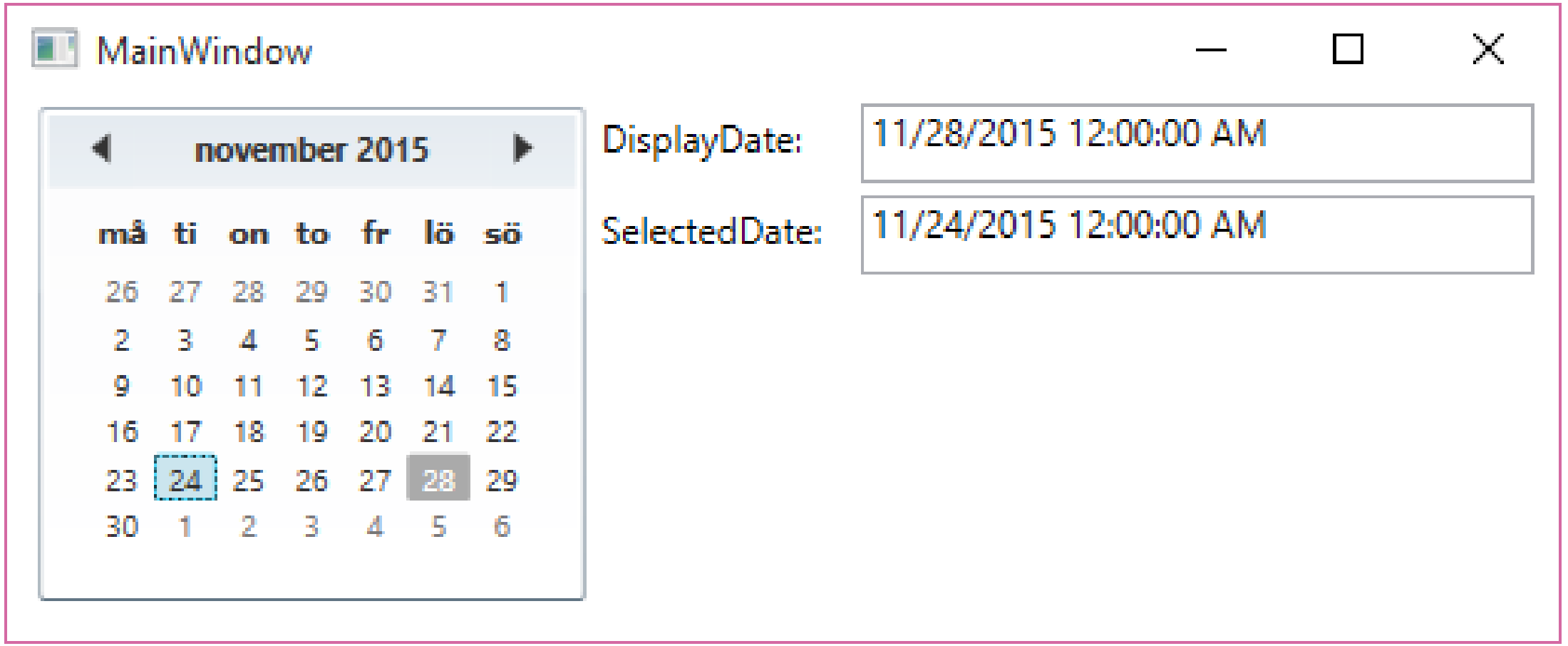

```
<StackPanel Margin="15">
 <WrapPanel>
   <TextBlock Text="Ändra Title: " />
   <TextBox Text="{Binding Title}" Width="150" />
 </WrapPanel>
  <WrapPanel Margin="0,10,0,0">
    <TextBlock Text="Bredd: " />
    <TextBox Text="{Binding Width}" Width="50" />
    <TextBlock Text=" Höjd " />
    <TextBox Text="{Binding Height}" Width="50" />
  </WrapPanel>
</StackPanel>
```

EGENSKAPEN UPDATESOURCETRIGGER

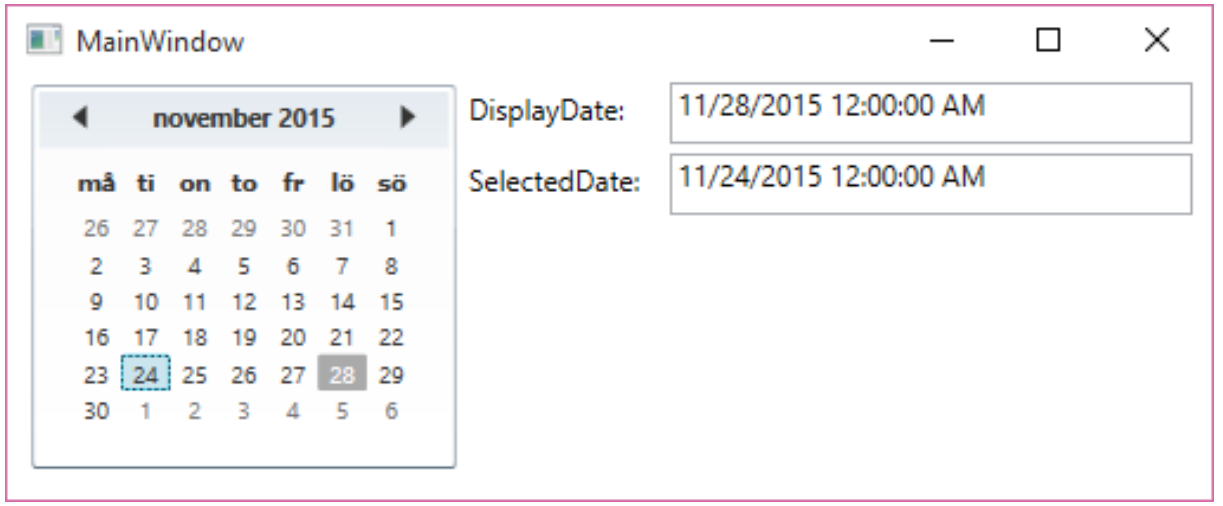
```
<WrapPanel>
<TextBlock Text="Ändra Title: " />
<TextBox Text="{Binding Title,
UpdateSourceTrigger=PropertyChanged}" Width="150" />
</WrapPanel>
```

Gets or sets a value that determines the timing of binding source updates.

ÖVNING – SKAPA EN KALENDER
BINDA TVÅ TEXTFÄLT TILL DISPLAYDATE OCH SELECTEDDATE



FACIT




```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30" />
        <RowDefinition Height="30" />
    <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="200" />
    </Grid.ColumnDefinitions>
</Grid>
```

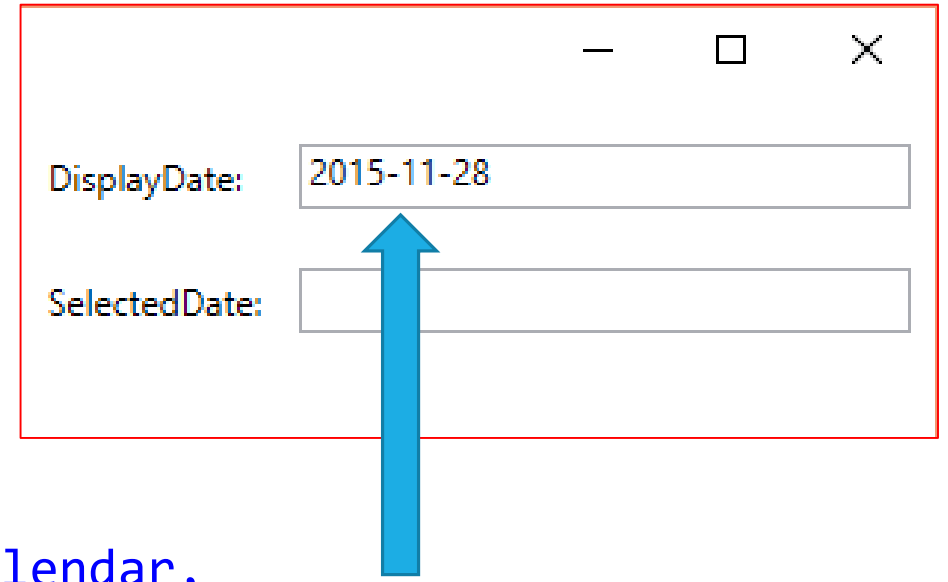
FACIT FORTS...

```
<Calendar x:Name="theCalendar"  Width="180" Height="170"
Margin="10,0,0,0" VerticalAlignment="Top" Grid.RowSpan="3" />

<Label Content="DisplayDate:" Grid.Column="1" Grid.Row="0" />
<TextBox x:Name="displayDateTextBox"
Text="{Binding DisplayDate, ElementName=theCalendar, UpdateSourceTrigger=PropertyChanged}"
Margin="8,2" Grid.Column="2" Grid.Row="0" />

<Label Content="SelectedDate:" Grid.Column="1" Grid.Row="1" />
<TextBox x:Name="selectedDateTextBox"
Text="{Binding SelectedDate, ElementName=theCalendar, UpdateSourceTrigger=PropertyChanged}"
Margin="8,2" Grid.Column="2" Grid.Row="1" />
```

STRINGFORMAT



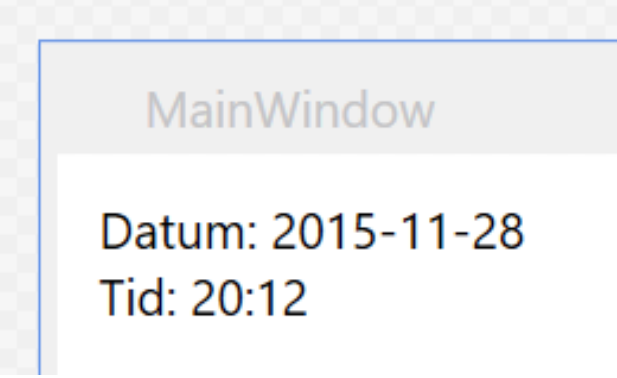

```
<TextBox x:Name="displayDateTextBox"
Text="{Binding DisplayDate, ElementName=theCalendar,
UpdateSourceTrigger=PropertyChanged, StringFormat=\{0:yyyy-MM-dd\} }"
Margin="8,2" Grid.Column="2" Grid.Row="0" />
```

Tips!

DateTime XAML Syntax

https://msdn.microsoft.com/library/dd631811(v=vs.100).aspx#format_strings_for_datetime_xaml_syntax

STRINGFORMAT – EXEMPEL 2



OBS! Vi behöver lägga till namnrymden System

```
xmlns:system="clr-namespace:System;assembly=mscorlib"
```

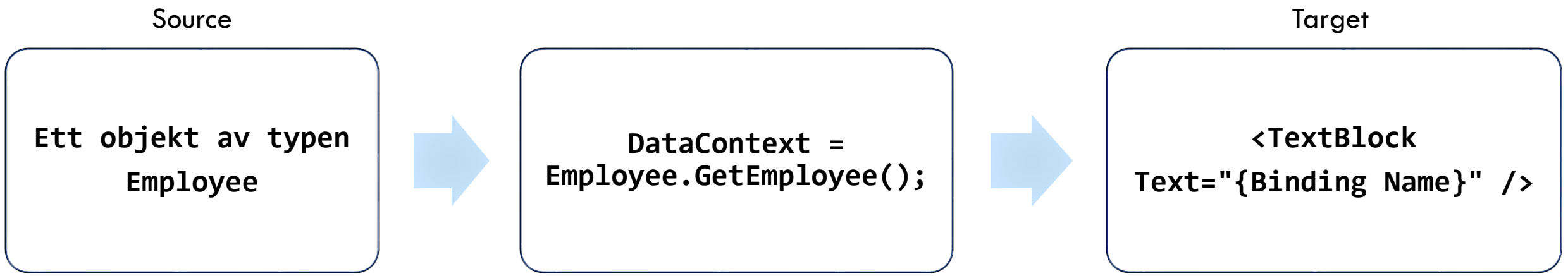
```
<StackPanel Margin="10">

<TextBlock Text="{Binding Source={x:Static system:DateTime.Now},
StringFormat=Datum: {0:yyyy-MM-dd}}" />


<TextBlock Text="{Binding Source={x:Static system:DateTime.Now},
StringFormat=Tid: {0:HH:mm}}" />

</StackPanel>
```

OBJECT BINDING

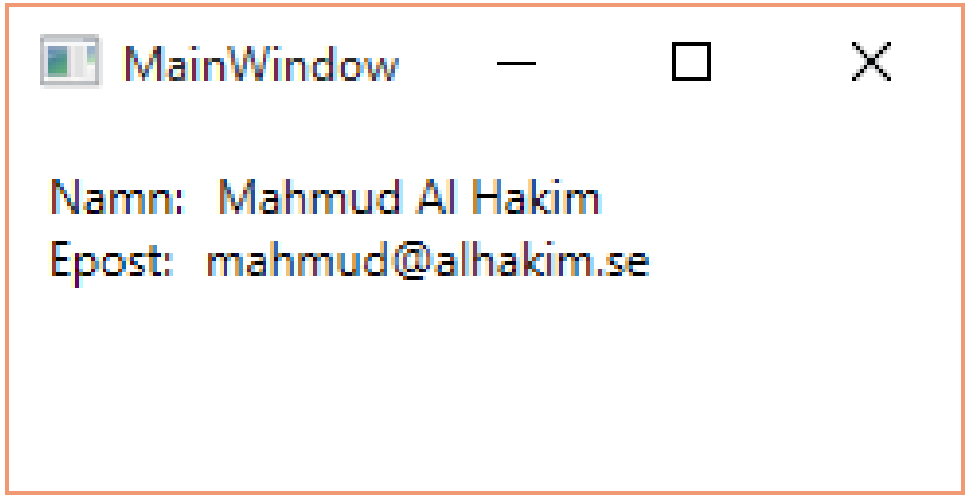


SOURCE – KLASSEN EMPLOYEE

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

        public static Employee GetEmployee() {
            var emp = new Employee() {
                Name  = "Mahmud Al Hakim",
                Email = "mahmud@alhakim.se"
            };
            return emp;
        }
}
```

TARGET – UI




```
<StackPanel Name="Display" Margin="10">
 <StackPanel Orientation="Horizontal">
    <TextBlock Text="Namn: " />
    <TextBlock Margin="5,0,0,0"  Text="{Binding Name}" />
 </StackPanel>
 <StackPanel Orientation="Horizontal">
    <TextBlock Text="Epost: " />
    <TextBlock Margin="5,0,0,0"  Text="{Binding Email}" />
 </StackPanel>
</StackPanel>
```

DATACONTEXT I CODE BEHIND

```
public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            DataContext = Employee.GetEmployee();
        }
    }
```

INPC

Gränssnittet uppdates inte automatiskt vid förändringar i bakomliggande objekt!

Detta är bra för …

- Att få bättre prestanda på applikationen.
- Att manuellt kunna notifiera flera kontroller som är beroende av en aktuell uppdatering.

Hur uppdaterar vi gränssnittet då?

1. Implementera gränssnittet `INotifyPropertyChanged` (INPC).
2. Initiera händelsen `PropertyChangedEventHandler`

EN KLASS SOM IMPLEMENTERAR INPC

```
using System.ComponentModel;
public class BindableBase : INotifyPropertyChanged  {
        public event PropertyChangedEventHandler PropertyChanged;

        public void NotifyPropertyChanged(string propertyName) {
            if (PropertyChanged != null)  {
                PropertyChanged.Invoke(this,
                    new PropertyChangedEventArgs(propertyName));
            }
        }
    }
```

PERSON ÄRVER FRÅN BINDABLEBASE

```
public class Person : BindableBase    {
    //public string Name { get; set; }
    private string _name;
    public string Name {
            get { return _name;  }
            set {
                _name = value;
                NotifyPropertyChanged("Name");
            }
     }
}
```

OBS! Auto-egenskaper funkar inte.

Vi behöver notifiera PropertyChanged

ENKELT UI FÖR ATT TESTA

```
<StackPanel>
<TextBlock  HorizontalAlignment="Center" FontSize="30"
            Text="{Binding Name}" />

<Button     Content="Ändra" Width="100"
            Click="Button_Click"/>
</StackPanel>
```

CODE BEHIND

```
public partial class MainWindow : Window {
    Person p = new Person();
    public MainWindow() {
        InitializeComponent();
        p = new Person() { Name = "Mahmud" };
        DataContext = p;
    }

    private void Button_Click(object sender, RoutedEventArgs e){
        p.Name = "Kalle";
    }
}
```

Testa att ta bort
NotifyPropertyChanged("Name");
från klassen Person
Vad händer?

OBSERVABLECOLLECTION

Klassen ObservableCollection är en lista som implementerar gränssnittet INotifyPropertyChanged.

ObservableCollection representerar en dynamisk samling med automatiska notifieringar när element läggs till, tas bort eller uppdateras!

Finns i namnrymden System.Collections.ObjectModel

OBSERVABLECOLLECTION - EXEMPEL

```
public partial class MainWindow : Window {
        private ObservableCollection<Person> personer;
        public MainWindow() {
            InitializeComponent();
            personer = new ObservableCollection<Person>()
            {  new Person(){Name="Mahmud Al Hakim"} };
            DataContext = personer;
        }
        private void btn_Click(object sender, RoutedEventArgs e){
            personer.Add(new Person() { Name = txtName.Text });
            txtName.Text = string.Empty;
        }
    }
```

OBSERVABLECOLLECTION - UI


```
<Grid>
<Grid.ColumnDefinitions> <ColumnDefinition/>
<ColumnDefinition Width="*"/> </Grid.ColumnDefinitions>
<StackPanel Margin="5">
    <TextBlock x:Name="lblName" Text="Namn:"></TextBlock>
    <TextBox x:Name="txtName"></TextBox>
    <Button Width="100" Height="20" Margin="5" Click="btn_Click" Content="Lägg till"></Button>
</StackPanel>
<ListView Margin="5" Grid.Column="1" ItemsSource="{Binding}">
    <ListView.View>
      <GridView><GridViewColumn Header="Lista över personer..."  DisplayMemberBinding="{Binding Name}"/>
     </GridView>
   </ListView.View>
</ListView>   </Grid>
```

ÖVNING

Utveckla föregående exempel.

Skapa ett E-post-fält.

Lägg till en kolumn som visar epost.

Lägg till en knapp för borttagning av en markerad person.