



HÖGSKOLAN I BORÅS

# Windows Presentation Foundation (WPF) 1

## Troelsen Kapitel 24

Grundläggande applikationsutveckling med C#

# Agenda

- WPF vs. Windows Forms
- Arvshierarkin i WPF
- XML och XAML
- Anatomien av en WPF applikation
- Property-Element syntax
- Attached Properties
- Markup Extensions
- WPF applikationer i Visual Studio
- App (.xaml, .xaml.cs, .g.cs)
- MainWindow (.xaml, .xaml.cs, .g.cs)
- Application events
- Window events
- Mouse events
- Keyboard events

# Windows Forms

- .NET 1.0 släpptes med biblioteket **Windows Forms** (System.Windows.Forms.dll) för att bygga grafiska användargränssnitt (Graphical User Interfaces - GUIs), tillsammans med biblioteket **GDI+** (System.Drawing.dll) för att skapa 2D grafik.
- Om man ville skapa 3D grafik, spela upp media (video) och manipulera PDF filer, var man tvungen att använda ytterligare bibliotek.

Desired Functionality	Technology
Building windows with controls	Windows Forms
2D graphics support	GDI+ (System.Drawing.dll)
3D graphics support	DirectX APIs
Support for streaming video	Windows Media Player APIs
Support for flow-style documents	Programmatic manipulation of PDF files

# Windows Presentation Foundation (WPF)

- I .NET 3.0 släppte Microsoft det modernare GUI biblioteket **Windows Presentation Foundation (WPF)**, där all funktionalitet samlades i ett och samma bibliotek med en gemensam objektmodell, dvs man behövde inte längre lära sig flera olika API:n (Application Programming Interface) för att skapa grafiska användargränssnitt, 2D och 3D grafik, strömma video och skapa PDF-dokument.
- I WPF **separeras applikationens visuella design från programmeringslogiken** som driver den, via det **XML-baserade språket XAML (*eXtensible Application Markup Language*)**.
- **All rendering i WPF görs med DirectX** (utan att behöva koda direkt mot DirectX API:t), till skillnad mot Windows Forms som använder GDI (och om man vill skapa avancerad 2D och 3D grafik är man tvungen att koda direkt mot DirectX API:t).

Desired Functionality	Technology
Building forms with controls	WPF
2D graphics support	WPF
3D graphics support	WPF
Support for streaming video	WPF
Support for flow-style documents	WPF

**OBS! I nuvarande version av .NET Framework (4.8) och .NET Core (3.1) kan WPF och Windows Forms applikationer endast exekveras på Windows OS eftersom anrop till Windows-specifika lågnivå-bibliotek görs under huven. Därför behöver man Visual Studio på Windows när man utvecklar WPF och WinForms applikationer. Det finns dock andra ramverk man kan använda, t.ex. Avalonia (open source): <https://avaloniaui.net>**

# Önskvärda egenskaper som finns i WPF

- **Layout managers** för att organisera grafiska komponenter på ett flexibelt sätt.
- En **avancerad databindingsmotor** för att binda data till grafiska komponenter.
- **Inbyggd “style” motor** som gör det möjligt att använda **CSS**-liknande funktionalitet.
- **Vektor-baserad grafik** som automatiskt anpassar sig efter skärmens upplösning.
- Support för **2D och 3D grafik, animering**, samt uppspelning av **audio och video**.
- Ett **PDF-liknande API** - XML Paper Specification (XPS).
- Support för **integration med legacy GUI modeller** (t.ex. Windows Forms).

# WPF assemblies

- WPF är huvudsakligen uppbyggd med typer som finns i assemeblies **PresentationCore.dll**, **PresentationFramework.dll**, **System.Xaml.dll** och **WindowsBase.dll** (som automatiskt refereras till av Visual Studio för WPF projekt).

Assembly	Meaning in Life	
PresentationCore.dll	This assembly defines numerous namespaces that constitute the foundation of the WPF GUI layer. For example, this assembly contains support for the WPF Ink API, animation primitives, and numerous graphical rendering types.	WPF Foundation (Ink API, animations, rendering types)
PresentationFramework.dll	This assembly contains a majority of the WPF controls, the Application and Window classes, support for interactive 2D graphics and numerous types used in data binding.	WPF Framework (Controls, Application, Window, 2D graphics, data binding)
System.Xaml.dll	This assembly provides namespaces that allow you to program against a XAML document at runtime. By and large, this library is useful only if you are authoring WPF support tools or need absolute control over XAML at runtime.	Programmatic XAML support
WindowsBase.dll	This assembly defines types that constitute the infrastructure of the WPF API, including those representing WPF threading types, security types, various type converters, and support for <i>dependency properties</i> and <i>routed events</i> (described in Chapter 27).	WPF API Infrastructure (threading, security, converters, dependency properties, routed events)



# WPF namespaces

WPF typerna finns  
huvudsakligen i namespaces:

- **System.Windows**
- **System.Windows.Controls**
- **System.Windows.Data**
- **System.Windows.Documents**
- **System.Windows.Ink**
- **System.Windows.Markup**
- **System.Windows.Media**
- **System.Windows.Navigation**
- **System.Windows.Shapes**

Namespace	Meaning in Life	
System.Windows	This is the root namespace of WPF. Here, you will find core classes (such as Application and Window) that are required by any WPF desktop project.	Core Classes (Application, Window)
System.Windows.Controls	This contains all of the expected WPF widgets, including types to build menu systems, tool tips, and numerous layout managers.	WPF Controls
System.Windows.Data	This contains types to work with the WPF data-binding engine, as well as support for data-binding templates.	Data Binding
System.Windows.Documents	This contains types to work with the documents API, which allows you to integrate PDF-style functionality into your WPF applications, via the XML Paper Specification (XPS) protocol.	XPS Documents
System.Windows.Ink	This provides support for the Ink API, which allows you to capture input from a stylus or mouse, respond to input gestures, and so forth. This is useful for Tablet PC programming; however, any WPF can make use of this API.	Ink API (stylus, gestures, etc)
System.Windows.Markup	This namespace defines a number of types that allow XAML markup (and the equivalent binary format, BAML) to be parsed and processed programmatically.	Programmatic XAML
System.Windows.Media	This is the root namespace to several media-centric namespaces. Within these namespaces you will find types to work with animations, 3D rendering, text rendering, and other multimedia primitives.	3D & Text rendering, Animations, etc
System.Windows.Navigation	This namespace provides types to account for the navigation logic employed by XAML browser applications (XBAPs) as well as standard desktop applications that require a navigational page model.	Navigational support
System.Windows.Shapes	This defines classes that allow you to render interactive 2D graphics that automatically respond to mouse input.	2D graphics support

# System.Windows.Application

- **System.Windows.Application** representerar en **WPF applikation**, och innehåller en referens till applikationen i property **Current**.
- Klassen innehåller en statisk metod **Run()** som används för att starta applikationen.
- Propertyn **StartupUri** används för att specificera ett **URI (Uniform Resource Identifier)** för vilket fönster (**Window**) om skall visas då applikationen startar.
- Applikationens *huvudfönster* kan sättas via klassens **MainWindow** property, där fönstret är en subclass till **System.Windows.Window**.
- Alla applikationens fönster sparas i property **Windows** (av typ **WindowCollection**).
- Property **Properties** kan användas för att lagra objekt som är globalt tillgängliga i hela applikationen.
- **Application** klassen innehåller *events*, t.ex. **Startup** och **Exit**, som kan användas för att implementera logik som skall köras när applikationen startar samt stängs.

Property	Meaning in Life
Current	This static property allows you to gain access to the running Application object from anywhere in your code. This can be helpful when a window or dialog box needs to gain access to the Application object that created it, typically to access application-wide variables and functionality.
MainWindow	This property allows you to programmatically get or set the main window of the application.
Properties	This property allows you to establish and obtain data that is accessible throughout all aspects of a WPF application (windows, dialog boxes, etc.).
StartupUri	This property gets or sets a URI that specifies a window or page to open automatically when the application starts.
Windows	This property returns a WindowCollection type, which provides access to each window created from the thread that created the Application object. This can be helpful when you want to iterate over each open window of an application and alter its state (such as minimizing all windows).



# System.Windows.Application

- Varje WPF applikation måste ärvas från **Application**, samt definiera programmets **Main()** metod som skapar en instans av klassen.
- En händelsehanterare för **Startup** händelsen kan skapas för att t.ex. hantera växlar (*command line arguments*) och skapa applikationens huvudfönster (**Window**).
- En händelsehanterare för **Exit** händelsen kan skapas för att t.ex. spara användarpreferenser
- **Main()** metoden måste markeras med “attributet” **[STAThread]** (som ser till att legacy COM objekt som eventuellt används av applikationen är trådsäkra).
- Alla fönster som skapas i applikationen sparas i propertyn **Windows**, som t.ex. kan användas för att iterera genom och minimera alla fönster.

```
// Define the global application object for this WPF program.
class App : Application {
    [STAThread]
    static void Main(string[] args) {
        // Create the application object.
        App app = new App();

        // Register the Startup/Exit events.
        app.Startup += (s, e) => { /* Start up the app */ };
        app.Exit += (s, e) => { /* Exit the app */ };

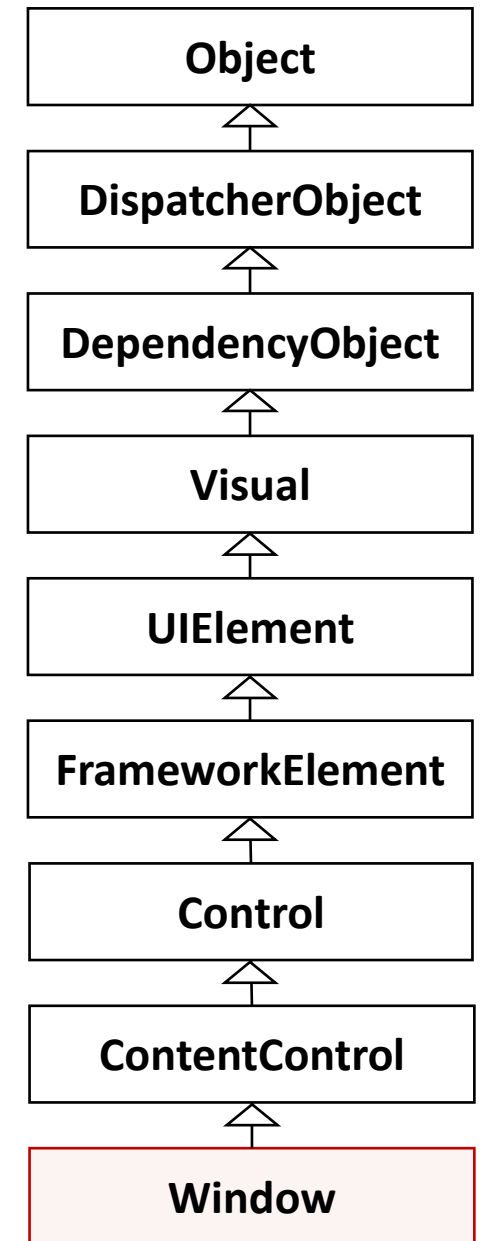
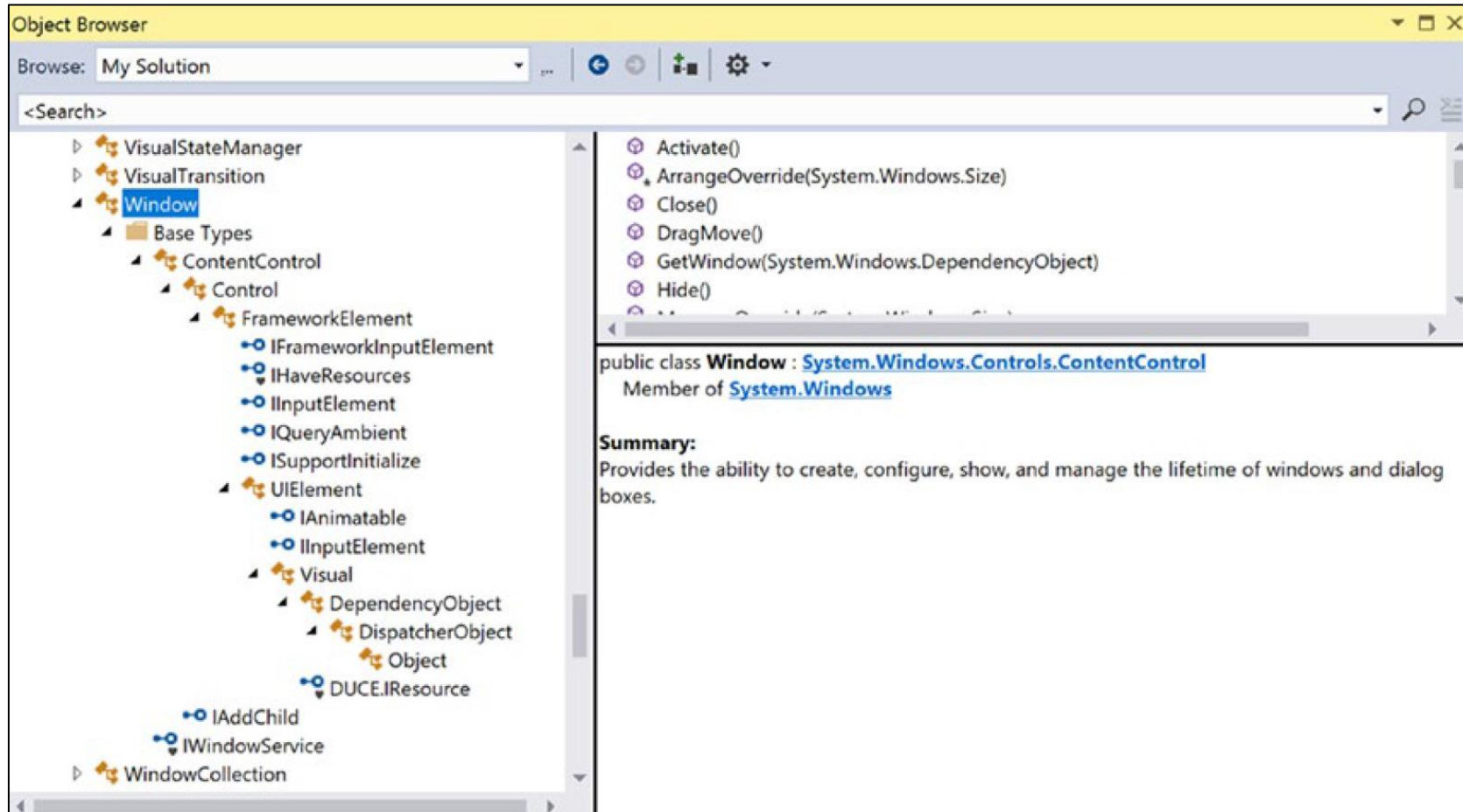
        app.Run();
    }
}
```

*Main() metoden genereras automatiskt av Visual Studio och är inte direkt tillgänglig för programmeraren som default.*

```
static void MinimizeAllWindows() {
    foreach (Window wnd in Application.Current.Windows)
        wnd.WindowState = WindowState.Minimized;
}
```

# System.Windows.Window

- Klassen **System.Windows.Window** representerar ett fönster, som ägs av **Application**, och ärver funktionalitet från superklasser.

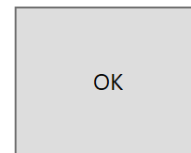


# System.Windows.ContentControl

- **ContentControl** förser subclasser med funktionaliteten att innehålla ett enda visuellt element (*content*), dvs det som visas visuellt inom den aktuella klassen, via **Content** propertyn.
- WPFs *content* modell gör det enkelt att konfigurera hur ett visuellt element skall se ut. Exempelvis om man vill tilldela en text som skall visas på en knapp kan man helt enkelt tilldela en sträng till knappens **Content** property, antingen programmatiskt eller via XAML.

```
// Setting the buttons Content value programmatically  
myButton.Content="OK";
```

```
<!-- Setting the Content value in the opening element -->  
<Button Height="80" Width="100" Content="OK"/>
```



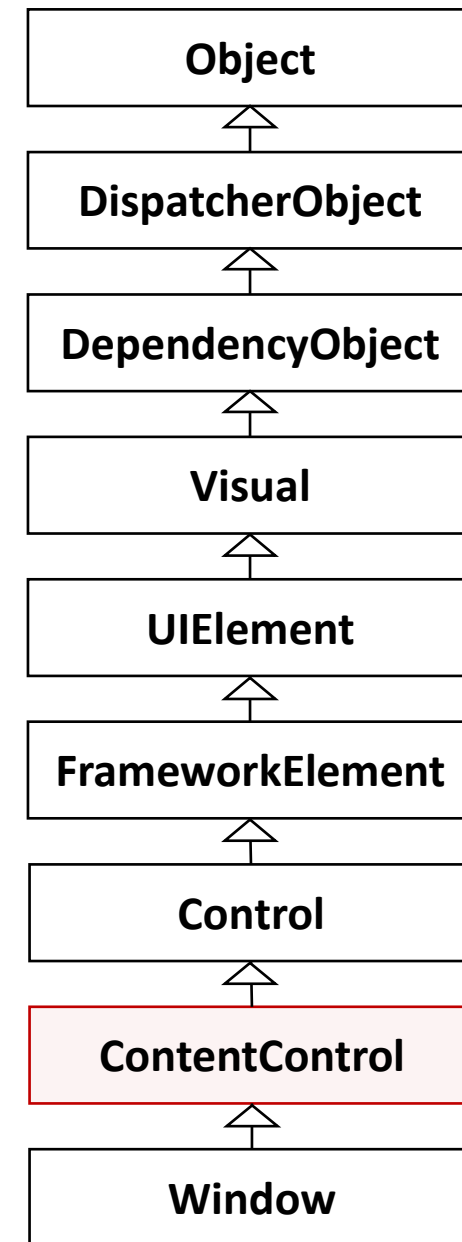
- Dock kan det som tilldelas **Content** vara nästan vad som helst. Exempelvis kan man tilldela ett värde som är sammansatt av flera delar, t.ex. en **Label** och en **Elips** (inom en **StackPanel**).

```
<!-- Implicitly setting the Content property with complex data -->  
<Button Height="80" Width="100">
```

```
  <StackPanel>  
    <Ellipse Fill="Red" Width="25" Height="25"/>  
    <Label Content="OK!" />  
  </StackPanel>
```

```
</Button>
```

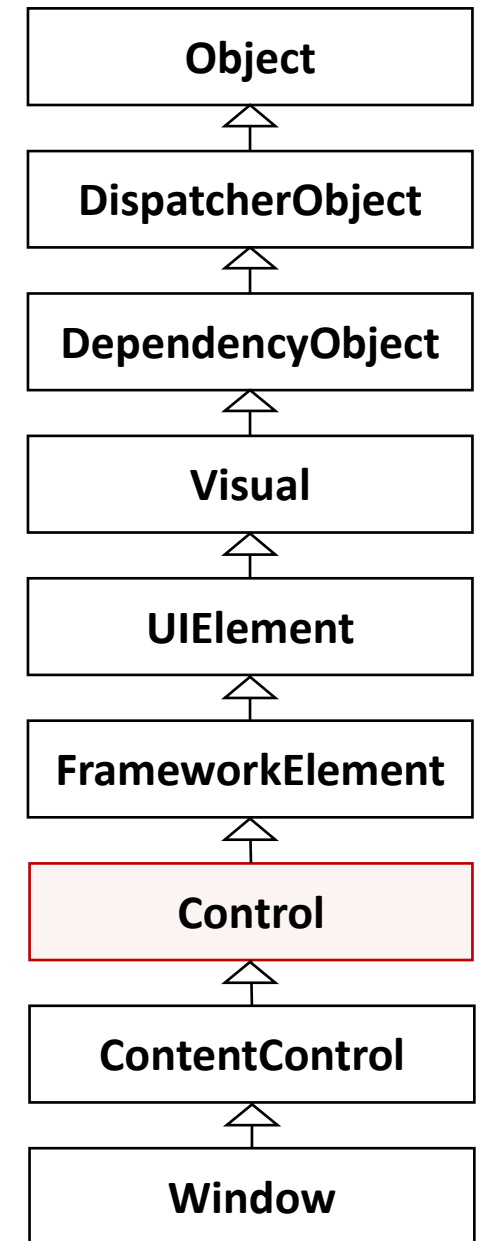
} content



# System.Windows.Control

- Till skillnad mot **ContentControl**, har alla WPF *kontroller* klassen **Control** som superklass.
- **Control** tillhandahåller funktionalitet för att t.ex. bestämma kontrollens storlek, genomskinlighet, tab-ordning, för- & bakgrundsfärg, osv. samt support för *templating services*, dvs **CSS (Cascading Style Sheets)**-liknande funktionalitet.

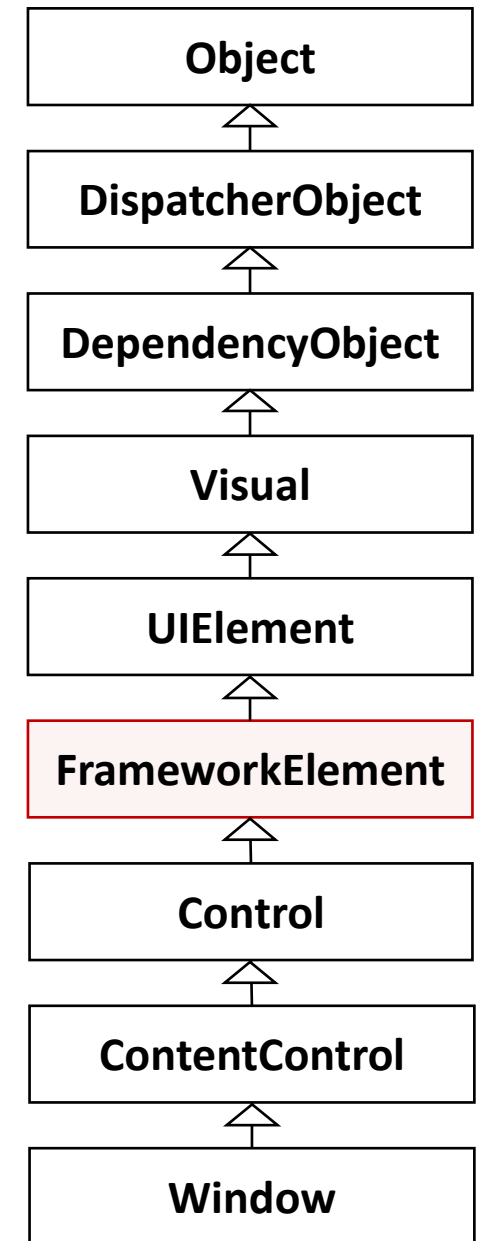
	Members	Meaning in Life
För- & bakgrundsfärg	→ <b>Background, Foreground</b> , BorderBrush, BorderThickness, Padding, HorizontalContentAlignment, VerticalContentAlignment  FontFamily, FontSize, FontStretch, FontWeight	These properties allow you to set basic settings regarding how the control will be rendered and positioned.  These properties control various font-centric settings.
Tab-ordning	→ <b>IsTabStop, TabIndex</b>	These properties are used to establish tab order among controls on a window.
	MouseDownClick, PreviewMouseDownClick	These events handle the act of double-clicking a widget.
Templating (CSS)	→ <b>Template</b>	This property allows you to get and set the control's template, which can be used to change the rendering output of the widget.



# System.Windows.FrameworkElement

- **FrameworkElement** innehåller bl. a. support för
  - **Storyboarding** (används vid animering).
  - **Data binding** (används för att koppla data till GUI element).
  - Att namnge en medlem (via property **Name**).
  - Access till en visuell typs resurser (**resources**).
  - Beräkna **dimensionerna** för en visuell typ.

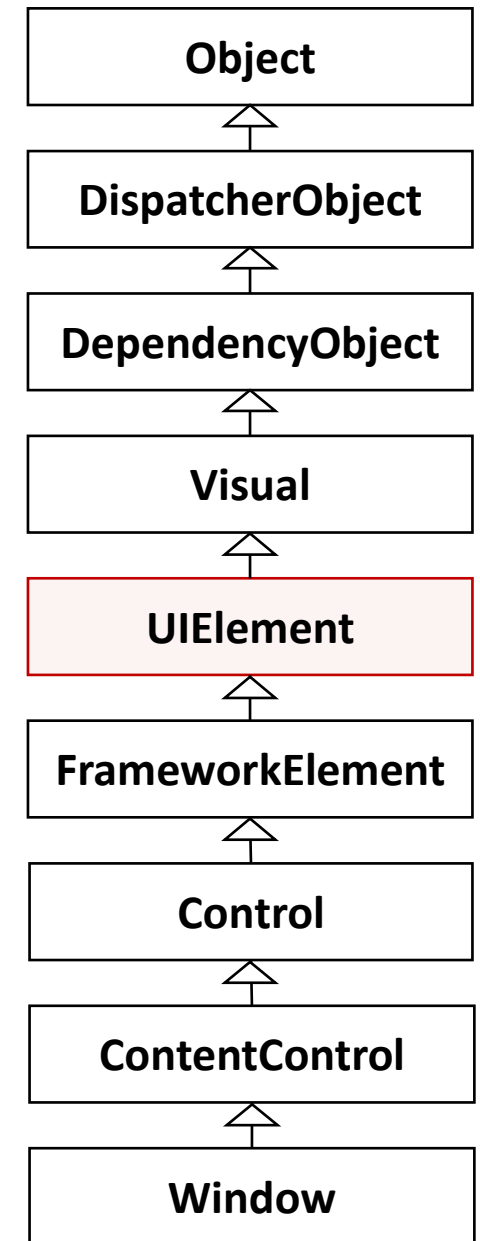
	Members	Meaning in Life
Dimensioner	ActualHeight, ActualWidth, MaxHeight, MaxWidth, MinHeight, MinWidth, Height, Width	These properties control the size of the derived type.
	ContextMenu	Gets or sets the pop-up menu associated with the derived type.
	Cursor	Gets or sets the mouse cursor associated with the derived type.
	HorizontalAlignment, VerticalAlignment	Gets or sets how the type is positioned within a container (such as a panel or list box).
Namnet på typen i kod	Name	Allows to you assign a name to the type in order to access its functionality in a code file.
Access till resurser	Resources	Provides access to any resources defined by the type (see Chapter 29 for an examination of the WPF resource system).
	ToolTip	Gets or sets the tool tip associated with the derived type.



# System.Windows.UIElement

- **UIElement** tillhandahåller många händelser (*events*) för subklasser, t.ex. för att erhålla *fokus* och hantera **mus- och tangetbordshändelser**, samt properties för att t.ex. **gömma** och **visa** en visuell komponent.

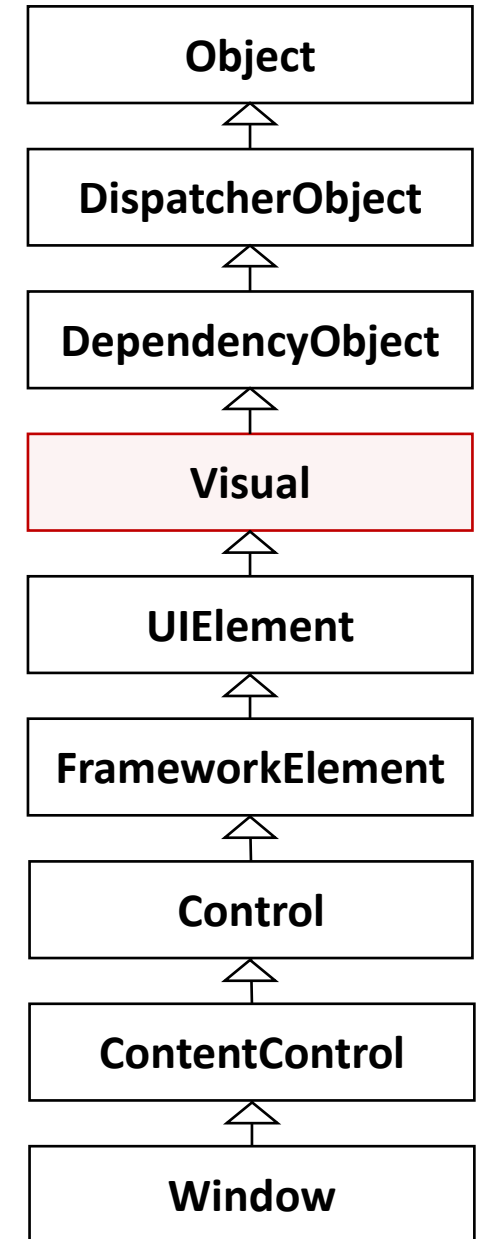
	Members	Meaning in Life
Fokus	Focusable, IsFocused	These properties allow you to set focus on a given derived type.
Aktivering	IsEnabled	This property allows you to control whether a given derived type is enabled or disabled.
Mushändelser	IsMouseDirectlyOver, IsMouseOver	These properties provide a simple way to perform hit-testing logic.
Synlighet	IsVisible, Visibility	These properties allow you to work with the visibility setting of a derived type.
	RenderTransform	This property allows you to establish a transformation that will be used to render the derived type.





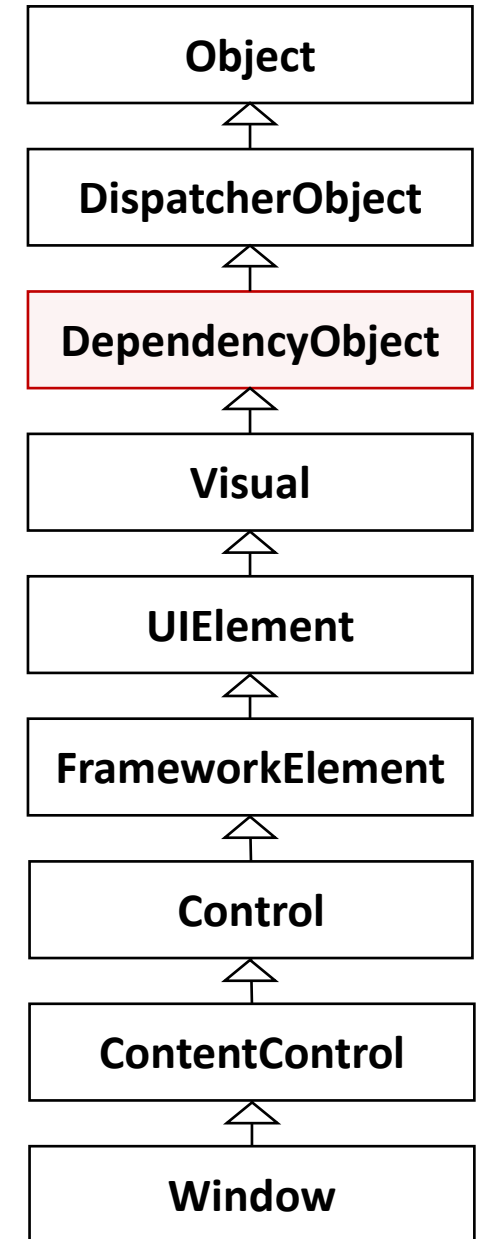
# System.Windows.Visual

- Klassen **Visual** implementerar grundläggande support för att **rendera** de visuella komponenterna, vilket inkluderar **hit testing**, **transformationer** mellan olika koordinatsystem och **bounding box** beräkningar.
- **Visual** kommunicerar med det underliggande *DirectX* subsystemet för att rendera grafiken på skärmen.



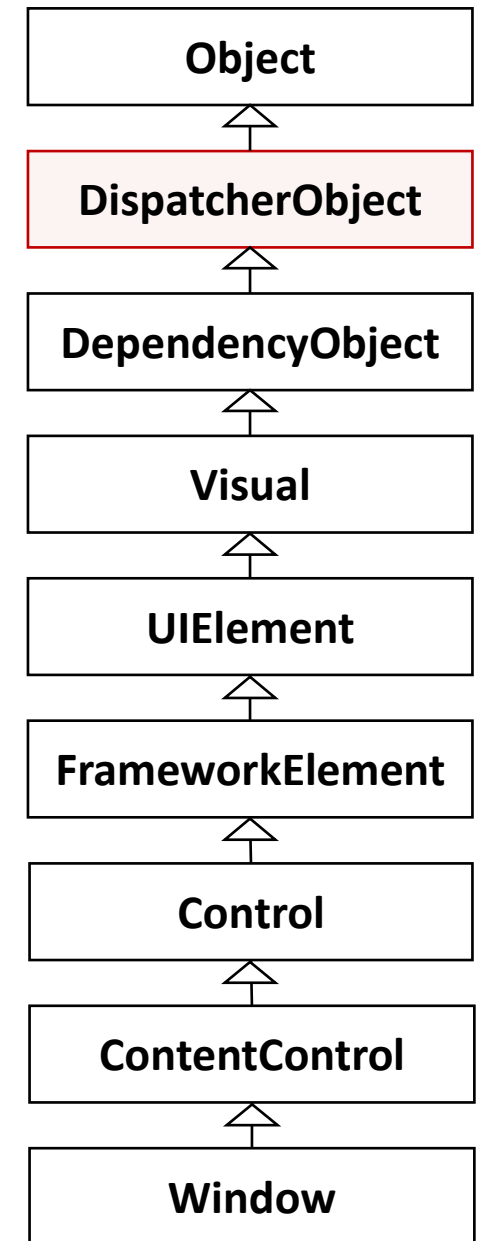
# System.Windows.DependencyObject

- **DependencyObject** implementerar support för *dependency properties*, dvs speciella WPF properties som innehåller kod för att möjliggöra WPF teknologier såsom *styles*, *data binding* och *animering*.
- Alla subklasser till **DependencyObject** har support för *dependency properties*.



# System.Windows.DispatcherObject

- **DispatcherObject** innehåller en property av intresse, **Dispatcher**, som returnerar det associerade **System.Windows.Threading.Dispatcher** objektet.
- **Dispatcher** klassen utgör en *entry point* till WPF applikationens **event kö**, dvs en kö som tar emot alla events för applikationen såsom mus- och tangentbordshändelser.
- **DispatcherObject** innehåller även konstruktioner för att hantera **trådning**.
- **OBS!** Man kan **INTE** accessa en GUI komponent på någon annan tråd än **GUI tråden**!
- Genom att använda **DispatcherObject** kan man **byta kontext** till **GUI tråden** via **Dispatcher** propertyn.



# XAML

- Professionella WPF applikationer använder ofta dedikerade verktyg (t.ex. ***Expression Blend*** och **Visual Studios visuella designer**) för att designa grafiska gränssnitt och automatgenerera tillhörande **XAML**.
- Dock är det viktigt att lära sig **XAML** innan man börjar använda verktygen.
- I kursboken (s.976-977) hänvisar Andrew Troelsen till ett hjälpverktyg ***kaxaml*** ([www.github.com/skimedec/kaxaml](http://www.github.com/skimedec/kaxaml)) för att lära sig **XAML**, dock kommer vi att koda **XAML** direkt i Visual Studio.

# XML (ej i kursboken)

- **XAML** (eXtensible Application Markup Language) är ett **XML** (eXtensible Markup Language) baserat *märkspråk* (*markup language*).
- Ett *märkspråk* är ett språk som använder *taggar* (*tags*) för att definiera *element* (elements) i ett dokument (fil). **Markup dokument** är text-baserade och innehåller vanliga ord istället för en speciell syntax som i ett programmeringsspråk.
- **HTML** (*Hyper-Text Markup Language*) är ett exempel på *märkspråk* för att skapa webbsidor, där varje webbsida definieras med **HTML taggar**, t.ex. **<head>**, **<body>** och **<h1>**.
- De flesta *elementen* börjar med en **start-tag** och slutar med en **slut-tag** med innehållet (*elementets innehåll*) mellan taggarna, t.ex. **<h1>Hello World</h1>**. **Slut-taggen** ser precis likadan ut som **start-taggen**, fast med ett "slash" tecken "/" efter mindre än tecknet "<".

```
<element>  
</element>
```

```
<head>  
  <body>  
    <h1>Hello World!</h1>  
  </body>  
</head>
```

```
<head>  
  <body>  
    <h1>Hello World!</h1>  
  </body>  
</head>
```

# XML (ej i kursboken)

- **XML** används för att lagra **strukturerad data**, istället för att formatera information på en webbsida.
- Medan **HTML** dokument använder fördefinierade taggar, använder **XML** skräddarsydda taggar t.ex. en **XML** fil som lagrar information om bilar.
- **XML** kallas "e**X**tensible **M**arkup **L**anguage" eftersom skräddarsydda taggar kan användas för att supporta många olika element.
- Ett program kan sedan parse **XML** filen för att extrahera datan (elementens innehåll) som finns mellan taggarna, t.ex. kan programmet leta efter tag-paren **<car>** och **</car>** för att hämta ut information (**regno** och **make**) om varje bil.

```
<cars>
  <car>
    <regno>ABC123</regno>
    <make>Volvo</make>
  </car>
  <car>
    <regno>XYZ789</regno>
    <make>Saab</make>
  </car>
</cars>
```



# XML (ej i kursboken)

- XML är en W3C (*World Wide Web Consortium*) standard som beskriver olika regler för hur XML dokument är uppbyggda, t.ex. via XML 1.0 specifikationen (<https://www.w3.org/XML>).
- Ett XML dokument (<https://en.wikipedia.org/wiki/XML>) innehåller bland annat:
  - Markup och Content.
  - Taggar.
  - Element.
  - Attribut.
  - XML deklarationer.
  - Kommentarer.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is a car. -->
<car regno="ABC123">
    Volvo
</car>
```

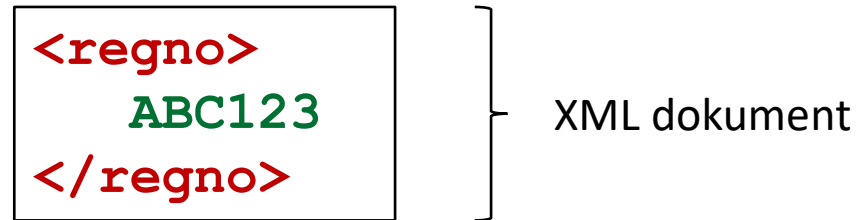
Start-tag för  
element Car.

Slut-tag för  
element Car.

All text inom <> är markup.

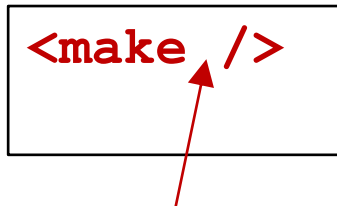
# XML Markup och Content (ej i kursboken)

- I XML dokument kategoriseras tecken som antingen **markup** eller **content**.
  - Strängar som innehåller **markup** börjar med “<” tecknet och avslutas med “>” tecknet.
  - Strängar som inte innehåller markup är **content** (innehåll, data).



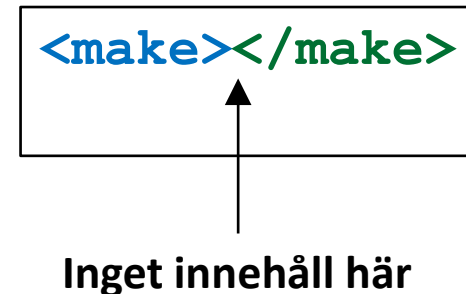
# XML Tagg (ej i kursboken)

- En **tagg** börjar med ett "<" tecken och avslutas med ett ">" tecken.
- Det finns tre olika typer av taggar:
  - start-tag, t.ex. **<make>**
  - slut-tag, t.ex. **</make>**
  - "tomt-element" tagg, t.ex. **<make />**
- Lägg märke till att "tomma-element" taggen **<make />** blir samma sak som ett **start-tag/slut-tag** par som inte innehåller någon data **<make></make>**.



A rectangular box containing the XML tag `<make />`. The text is red. A red arrow points from the bottom left towards the space between the slash and the closing angle bracket.

**OBS! "tomt-element" taggen har ett blanksteg innan slash tecknet "/"**

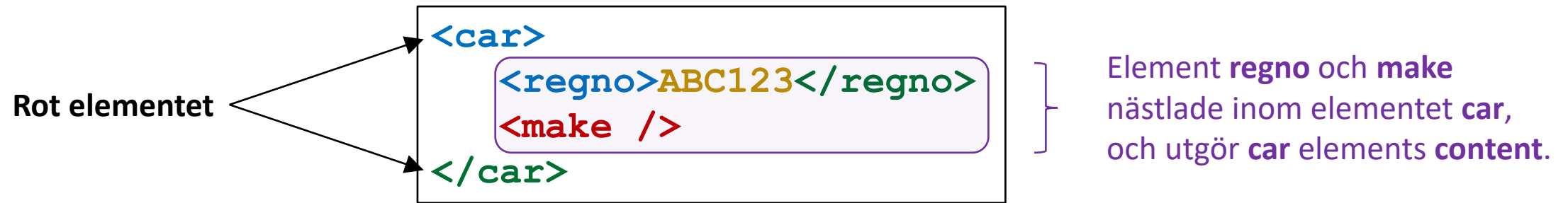


A rectangular box containing the XML tag pair `<make></make>`. The text is blue for the start tag and green for the end tag. A black arrow points from the text 'Inget innehåll här' below to the space between the two tags.

Inget innehåll här

# XML Element (ej i kursboken)

- Ett **element** definieras antingen:
  - Med en **start-tag** och en **slut-tag**.
  - Som en **“tomt-element” tagg**.
- Alla tecken mellan taggarna utgör elementets innehåll (**content**), och kan, i sin tur, bestå av **markup** (med **nästlade element**) eller **content**.
- Nästlade element kallas **barn element** (*child elements*).
- Det första (yttersta) elementet i ett dokument kallas **rot elementet**.



# XML Attribut (ej i kursboken)

- Ett *attribut* består av ett **namn-värde** par som antingen finns inuti en **start-tag** eller en **“tomt-element” tag**.
- Ett attribut kan endast innehålla **ett värde**.
- Ett attribut med ett visst **namn** kan endast **förekomma en gång inom ett och samma element**.
- Om ett element innehåller flera attribut brukar attributen särskiljas med ett kommatecken, semikolon eller ett **blanksteg** (om själva **värdet** innehåller blanksteg **anges värdet inom citattecken**).

```
<car id="12" regno="ABC 123">  
  <make name="Volvo" />  
  <model>S80</model>  
</car>
```

# XML Deklaration (ej i kursboken)

- XML dokument kan **börja med** en **XML deklARATION** som innehåller **metadata** om XML dokumentet, t.ex. vilken XML version (t.ex. 1.0) och teckenkodning (t.ex. UTF-8) som används i dokumentet.
- En **XML deklARATION** anges inom den speciella taggen **<? ?>**.

```
<?xml version="1.0" encoding="UTF-8"?>
<car id="1" regno="ABC123">
  <make="Volvo" />
  <model>S80</model>
</car>
```



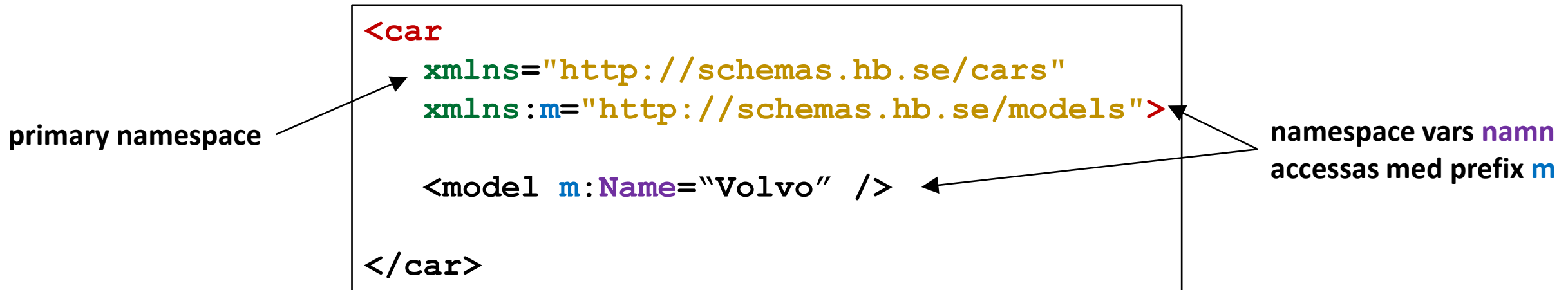
# XML Kommentar (ej i kursboken)

- En **kommentar** i XML anges mellan taggarna `<!--` och `-->`.
- Kommentarer kan täcka ett eller flera rader, och kan innesluta andra XML element (som då blir bortkommenterade).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is a car. -->
<car id="1" regno="ABC123">
    <make="Volvo" />
    <model>S80</model>
</car>
```

# XML namespaces (ej i kursboken)

- Syftet med **XML namespaces** är samma som för namespaces i C#, dvs för att logiskt organisera namn (t.ex. XML attributnamn) och för att undvika “namnkrockar”.
- En XML namespace importeras i XML dokumentets **första start-tag** med attributnamnet **xmlns** efterföljt av ett **kolon** och ett godtyckligt **prefix**, samt därefter ett “=” tecken och **namnet på aktuell namespace** (som “liknar” en webbadress).
- **Prefixet** används sedan framför varje **attribut namn** som ingår i aktuell namespace.
- **Första namespace** som importeras kallas dokumentets **primary namespace**, och har **inget prefix**, varför **namn** som ingår i detta namespace accessas utan **prefix**.



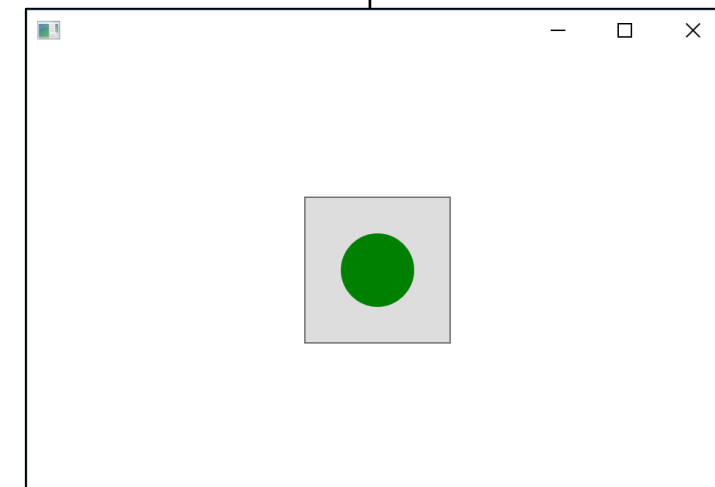
# XAML och XML

- **XAML** (uttalas "Zammel") är ett XML-baserat språk som definierar taggar och attribut specifikt för att **beskriva visuella element i WPF XAML dokument**, t.ex. ett fönster **<Window>** som innehåller **namespace deklARATIONER** och ett rutnät **<Grid>**, som i sin tur innehåller en knapp **<Button>**, som i sin tur innehåller en ellips **<Ellipse>**.
- **Kommentarer** i XAML anges mellan taggarna **<!--** och **-->**.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

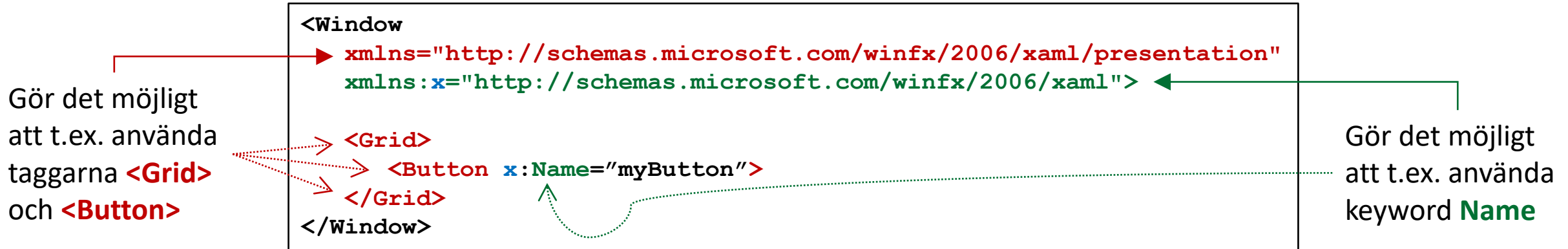
  <Grid>
    <!-- A button with custom content -->
    <Button Height="100" Width="100">
      <Ellipse Fill="Green" Height="50" Width="50"/>
    </Button>
  </Grid>

</Window>
```



# XAML XML namespaces och XAML keywords

- Rot-elementet för ett WPF XAML dokument (t.ex. ett **<Window>**, **<Page>** eller **<Application>** element) importerar nästan alltid följande två XML namespaces:



- Den **första namespace** (som också är **primary namespace**) mappar till ett flertal WPF .NET namespaces som importerar i aktuell \*.xaml fil (**System.Windows**, **System.Windows.Controls**, **System.Windows.Data**, **System.Windows.Ink**, **System.Windows.Media**, **System.Windows.Navigation**, osv.). Denna mappningen är hårdkodad i assemblies **WindowsBase.dll**, **PresentationCore.dll** och **PresentationFramework.dll**.
- Den **andra namespace** importerar XAML-specifika “**keywords**” (nyckelord) samt WPF .NET namespace **System.Windows.Markup**.

# Import av egna *namespaces*

- Om man har skapat egna typer och vill använda dem i en XAML fil, importeras dessa också med en XAML namespace, där två “tokens” (**clr-namespace** och **assembly**) används för att ange typens **namespace** respektive **assembly**.
- Exempelvis, om man har definierat klassen **MyCustomControl** i namespace **MyControls** i assembly **MyControls.dll**, så kan typen importeras i XAML dokumentet med syntaxen.

```
xmlns:myCtrls="clr-namespace:MyControls;assembly=MyControls"
```

- Därefter kan typen kommas åt via **prefixet** som angavs i ovanstående XML namespace direktiv.

```
<Window x:Class="WpfApp.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:myCtrls="clr-namespace:MyControls;assembly=MyControls"
  Title="MainWindow" Height="350" Width="525">

  <Grid>
    <myCtrls:MyCustomControl />
  </Grid>

</Window>
```

Applikationens **huvudnamespace** importeras som default med prefix **local**.

```
xmlns:local="clr-namespace:WpfApp"
```

Ett **<Window>** element har ofta följande som talar om för XAML processor att den skall skippa allt med prefix **d** (används bl.a. för att tala om att något endast skall gälla **design time**).

```
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
```

- Om typen finns i samma assembly som XAML filen tillhör, kan man skippa “token” **assembly**.

# Vanliga XAML *keywords* (nyckelord)

Förekommer i alla XAML-filer och är viktigast att lära sig först!

XAML Keyword	Meaning in Life
x:Class	Specifies the code-behind class for an element (namespace.class).
x:Array	Represents a .NET array type in XAML.
x:ClassModifier	Allows you to define the visibility of the C# class (internal or public) denoted by the Class keyword.
x:FieldModifier	Allows you to define the visibility of a type member (internal, public, private, or protected) for any named subelement of the root (e.g., a <Button> within a <Window> element). A <i>named element</i> is defined using the Name XAML keyword.
x:Key	Allows you to establish a key value for a XAML item that will be placed into a dictionary element.
x:Name	Allows you to specify the generated C# name of a given XAML element.
x:Null	Represents a null reference.
x:Static	Allows you to make reference to a static member of a type.
x:Type	The XAML equivalent of the C# typeof operator (it will yield a System.Type based on the supplied name).
x:TypeArguments	Allows you to establish an element as a generic type with a specific type parameter (e.g., List<int> vs. List<bool>).



# XAML filer (.xaml) och code-behind filer (.xaml.cs)

- Varje XAML dokument (.xaml) har en tillhörande **“code-behind”** fil (.xaml.cs):
  - **XAML dokumentet** beskriver strukturen på det grafiska interfacet.
  - **Code-behind filen** innehåller tillhörande kod (logik), t.ex. **händelsehanterare**.
- Code-behind filen har samma namn som XAML filen, fast med filändelsen .xaml.cs
- Keyword **Class** används i XAML filen för att koppla **<Window>** elementet till tillhörande **klass** i code-behind filen med syntaxen **Class=“namespace.klass”**.

## XAML fil (MainWindow.xaml)

```
<Window x:Class="WpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Grid>
        <Button x:Name="myButton" Click="myButton_Click" />
    </Grid>

</Window>
```

*Observera att klassen MainWindow är definierad som partial, dvs definitionen av klassen är utspridd i flera .cs filer.*

## Code-behind fil (MainWindow.xaml.cs)

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

        private void myButton_Click(object sender, RoutedEventArgs e)
        {
            Environment.Exit(0);
        }
    }
}
```

# Automatgenererade filer (.g.cs,.i.g.cs och .baml)

- Varje XAML dokument (.xaml) har också 3 tillhörande automatgenererade filer som har samma namn som XAML filen, fast med filändelserna **.g.cs**, **.g.i.cs** och **.baml**.
  - **XAML dokumentet** definierar grafiska typer och medlemmar för det grafiska interfacet.
  - De automatgenererade filerna **.g.cs** och **.g.i.cs** innehåller tillhörande definitioner i kod.
  - Den automatgenererade filen **.baml** är en binärrepresentation av XAML filen.
- Keyword **Class** används återigen i XAML filen för att koppla **<Window>** elementet till tillhörande **klass** i **.g.cs** och **.g.i.cs** filerna med syntaxen **Class="namespace.class"**.
- Ett GUI element namnges med keyword **Name**, och får samma namn i **.g.cs** filen.

## XAML fil (MainWindow.xaml)

```
<Window x:Class="WpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Grid>
        <Button x:Name="myButton" Click="myButton_Click" />
    </Grid>

</Window>
```

Man använder detta variabelnamnet i kod för att komma åt det grafiska elementet

## Code-behind fil (MainWindow.g.cs)

```
namespace WpfApp
{
    public partial class MainWindow : Window
    {
        public void InitializeComponent() { }

        void Connect(int connectionId, object target) { }

        internal Button myButton;
    }
}
```

Definitonen av MainWindow fortsätter här (partial).

# Automatgenererade filer (.g.cs,.i.g.cs och .baml)

- Visual Studio använder **partiella klasser** för att separera kod filer som innehåller:
  - Kod som programmeraren skriver (.xaml.cs).
  - Automatgenererad "boiler plate" kod (.g.cs, i.g.cs)
- Man använder keyword **Name** för att namnge ett visuellt element i XAML dokumentet (t.ex. **myButton**) så att man kan referera till elementet i sin kod.

Code-behind fil (**MainWindow.xaml.cs**)

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

        private void myButton_Click(object sender, RoutedEventArgs e)
        {
            myButton.Content = "Hello World!";
        }
    }
}
```

XAML fil (**MainWindow.xaml**)

*Ett element namnges i XAML.*

```
<Window x:Class="WpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Grid>
        <Button x:Name="myButton" Click="myButton_Click" />
    </Grid>

</Window>
```

*Det namngivna elementet deklareras i den automatgenererade filen.*

Code-behind fil (**MainWindow.g.cs**)

```
namespace WpfApp
{
    public partial class MainWindow : Window
    {
        public void InitializeComponent() { }

        void Connect(int connectionId, object target) { }

        internal Button myButton;
    }
}
```

*Det namngivna elementet används i programmerarens code behind fil (.xaml.cs).*

# Kontroll av synligheten för typer och medlemmar

- Keyword **x:ClassModifier** och **x:FieldModifier** kan användas i XAML för att bestämma synligheten för klasser respektive attribut i tillhörande automatgenererade filer (.g.cs och .g.i.cs).
- Som default är typer **public**, medan medlemmar är **internal**.

## XAML fil (MainWindow.xaml)

```
<!-- This class will now be declared internal in the *.g.cs file -->
<Window x:Class="MyWPFApp.MainWindow" x:ClassModifier="internal"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <!-- This button will be public in the *.g.i.cs file -->
  <Button x:Name="myButton" x:FieldModifier="public" Content="OK"/>
</Window>
```

## Code-behind fil (MainWindow.g.cs)

```
namespace WpfApp
{
    internal partial class MainWindow : Window
    {
        public void InitializeComponent() { }

        void Connect(int connectionId, object target) { }


        public Button myButton;
    }
}
```

*Man kan alltså bestämma synligheten på de automatgenererade klasserna, attributen, osv i koden via dessa XAML Keywords.*

# XAML element, XAML attribut och typkonverterare

- När man har skapat sitt **rot-element**, t.ex. `<Window>`, och har importerat nödvändiga ***XML namespaces***, anges ett **barn-element** för rot-elementet (dvs dess ***content***).
- Barn-elementet är ofta en ***layout manager***, men anta att det är en knapp `<Button>`.
- ***XAML element*** mappar till **.NET klasser** (eller ***structar***) inom en given .NET namespace.
- ***XAML attribut*** inom start-taggen för ett XAML element mappar till **.NET attribut, properties eller events** för aktuell typ.
- Exempelvis mappar elementet `<Button>` till .NET typen `System.Windows.Controls.Button` och attributen i start-taggen `<Button Height="50" Width="100" Content="OK!">` mappar till properties `Height`, `Width` och `Content` i typen `System.Windows.Controls.Button`.
- Lägga märke till att **strängar** tilldelas till attributen, vilket egentligen är **fel datatyp** för t.ex. `Height` som är av datatyp `int`. Detta möjliggörs via ett antal **typkonverterar-klasser** som automatiskt transformerar enkla text-värden till rätt underliggande typ.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Grid>
    <Button Height="50" Width="100" Content="OK!" />
  </Grid>
</ Window>
```



## Motsvarande kod

```
Button myBtn = new Button();
myBtn.Height = 50;
myBtn.Width = 100;
myBtn.Content = "OK!";
```

# XAML *Property-Element* syntax

- Ibland måste man tilldela ett värde till ett XAML attribut som inte kan uttryckas som en simpel sträng.
- Antal att man vill tilldela en ny bakgrundsfärg till en knapp (Button), vilket görs med en pensel (**Brush**) i .NET.
- I kod hade det sett ut enligt nedan till vänster, men hur kan man göra detta i XAML?
- I XAML finns en speciell syntax, **property-element syntax**, som kan användas när man behöver tilldela ett komplext objekt, vilket visas nedan till höger (inom **<Button>** elementet har ett **<Button.Background>** *scope* definierats, och i denna, en **<LinearGradientBrush>**).

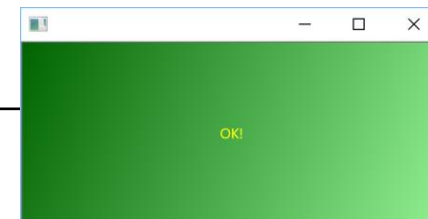
## Kod

```
public void MakeAButton()
{
    Button myBtn = new Button();
    myBtn.Content = "OK!";
    myBtn.Foreground = new SolidColorBrush(Colors.Yellow);

    // A fancy brush for the background.
    LinearGradientBrush fancyBrush =
    new LinearGradientBrush(Colors.DarkGreen, Colors.LightGreen, 45);
    myBtn.Background = fancyBrush;
}
```

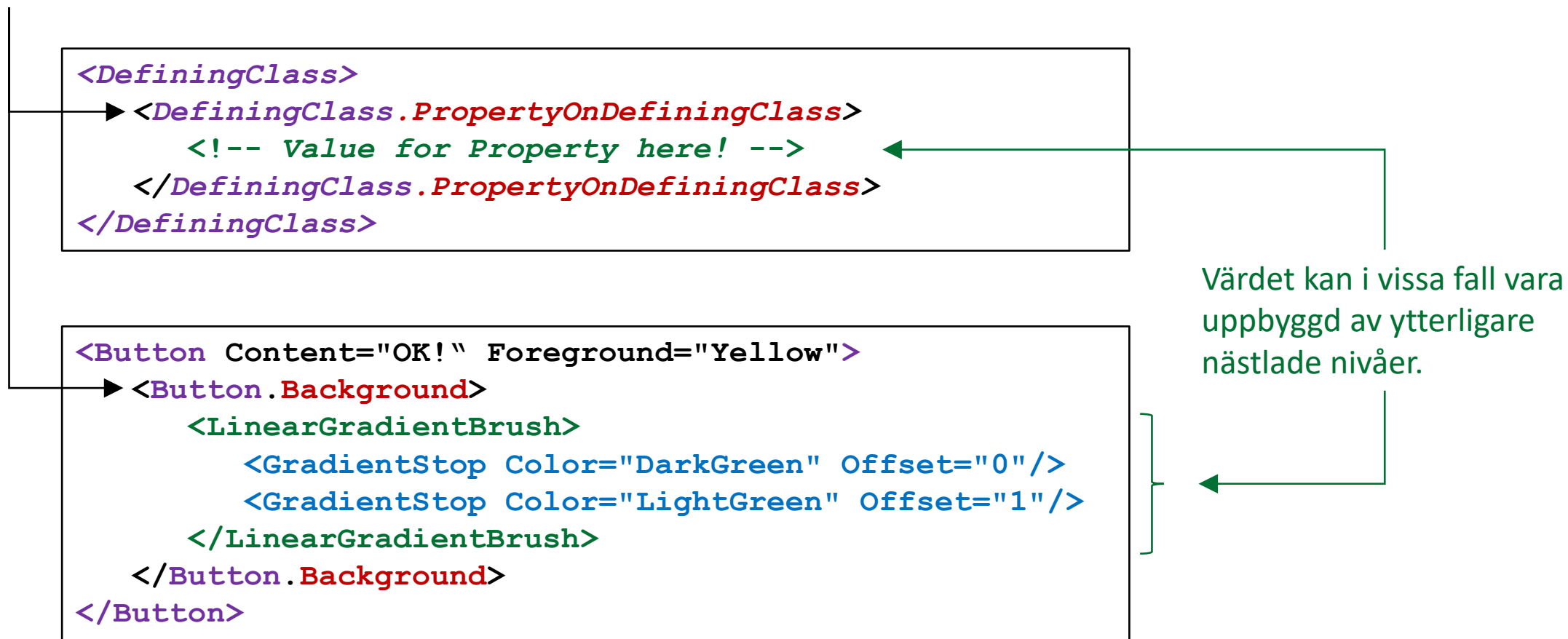
## XAML med property-element syntax

```
<Button Content="OK!" Foreground="Yellow">
    <Button.Background>
        <LinearGradientBrush>
            <GradientStop Color="DarkGreen" Offset="0"/>
            <GradientStop Color="LightGreen" Offset="1"/>
        </LinearGradientBrush>
    </Button.Background>
</Button>
```



# XAML *Property-Element* syntax

- Ett godtyckligt XAML attribut kan tilldelas ett komplext objekt med ***property-element*** syntax enligt mönstret:





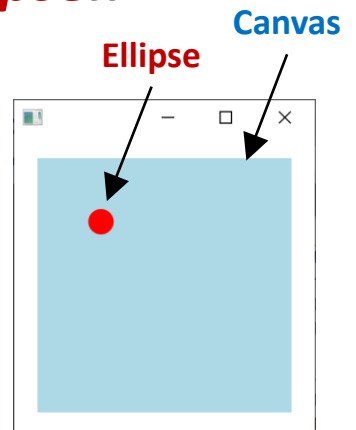
# XAML Attached Properties

- Det finns ytterligare en speciell XAML syntax som kallas en **attached property**, som innebär att ett **barn-element** kan sätta värdet för en **property** som egentligen tillhör **förälder-elementet**. Syntaxen ser ut enligt nedan:

```
<ParentElement>  
    <ChildElement ParentElement.PropertyOnParent = "Value">  
</ParentElement>
```

- **Attached properties** fungerar inte för alla properties av en godtycklig förälder, utan används oftast för att positionera GUI element i en WPF **layout manager** (t.ex. en **Grid** eller **Canvas**). Nedan finns ett exempel med en **Canvas** (en **layout manager**) som innehåller en **Ellipse**, där **Ellipsen** informerar sin förälder (**Canvas**) var **Ellipsen** skall positioneras via **attached properties** **Canvas.Top** och **Canvas.Left**.

```
<Window  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">  
    <Canvas Height="200" Width="200" Background="Red">  
        <Ellipse Canvas.Top="40" Canvas.Left="40" Height="20" Width="20" Fill="DarkBlue"/>  
    </Canvas>  
</ Window >
```





# XAML Markup Extensions

- XAML attribut tilldelas oftast ett värde via en **sträng** eller **property-element** syntax.
- Man kan också sätta ett värde med hjälp av **markup extensions**, som innebär att **värdet erhålls från en dedikerad extern klass**, vilket är fördelaktigt om värdet behöver beräknas.
- En **markup extension** representeras av en klass som ärver från **MarkupExtension**, men oftast behöver man inte definiera egna **markup extensions**.
- XAML keywords såsom **x:Array**, **x:Null**, **x:Static** och **x:Type** är egentligen **markup extensions** “under huven”.
- En **markup extension** anges mellan krullparanteser **{ }** enligt syntaxen:

```
<Element PropertyToSet = "{MarkupExtension}"/>
```

# XAML Markup Extensions

- Exempelvis kan man importera typerna i .NET namespace **System** (som finns i assembly **mscorlib.dll**) och sedan använda **markup extensions** för att t.ex. läsa statiska properties i typerna med hjälp av keyword **x:Static** samt hämta det fullständiga namnet på en typ med hjälp av keyword **x>Type**.

Importera typerna i namespace **System** (från filen **mscorlib.dll**) och använd prefix **CorLib**.

Använd **x:Static** i en **markup extension** för att erhålla OS versionen och antalet processorer från **Environment** typens statiska properties **OSVersion** respektive **ProcessorCount**.

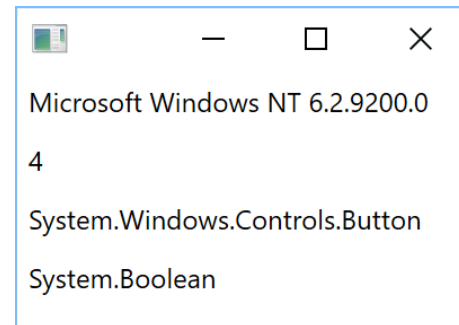
Använd **x>Type** i en **markup extension** för att hämta det fullständiga namnet på typerna **Button** och **Boolean** (ekvivalent med **typeof(Boolean)** i kod).

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:CorLib="clr-namespace:System;assembly=mscorlib">
  <StackPanel>

    <!-- The Static markup extension lets us obtain a value
    from a static member of a class -->
    <Label Content = "{x:Static CorLib:Environment.OSVersion}" />
    <Label Content = "{x:Static CorLib:Environment.ProcessorCount}" />

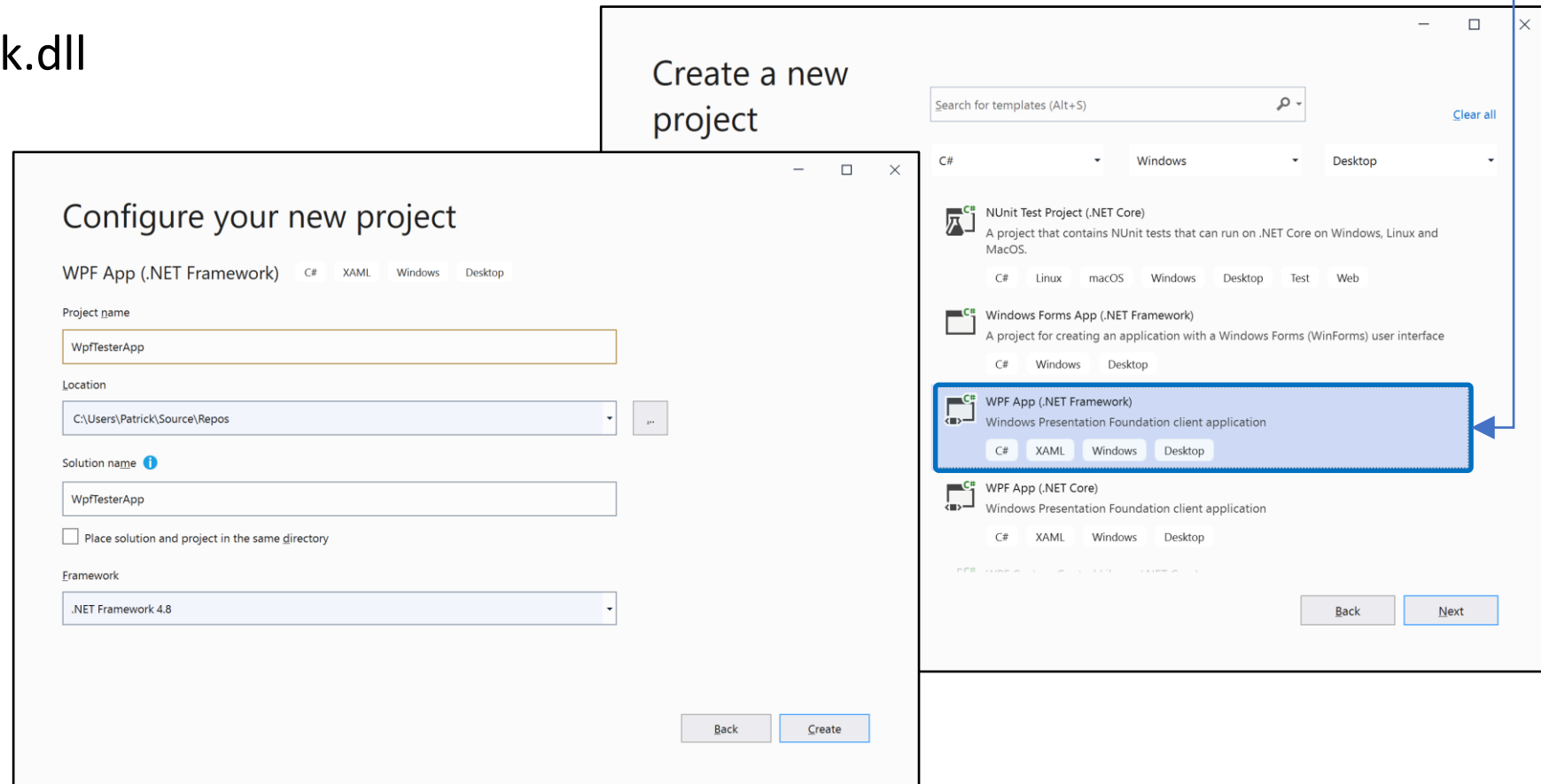
    <!-- The Type markup extension is a XAML
    version of the C# typeof operator -->
    <Label Content = "{x>Type Button}" />
    <Label Content = "{x>Type CorLib:Boolean}" />

  </StackPanel>
</Window>
```



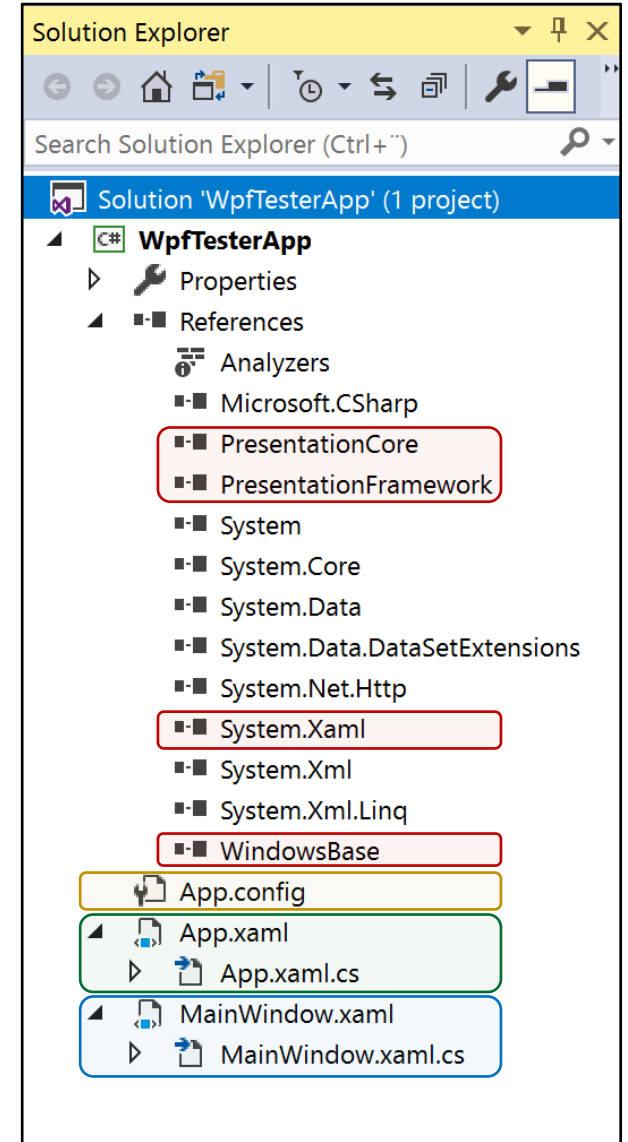
# Skapa WPF applikationer i Visual Studio (VS)

- Projekttypen för WPF applikationer är **WPF App (.NET Framework)**.
- Visual Studio sätter automatiskt referenser till WPF assemblies:
  - PresentationCore.dll
  - PresentationFramework.dll
  - System.Xaml.dll
  - WindowsBase.dll
- Visual Studio skapar en XAML applikation ...
  - App.xaml
  - App.xaml.cs
- ... och huvudfönstret:
  - MainWindow.xaml
  - MainWindow.xaml.cs



# Skapa WPF applikationer i Visual Studio (VS)

- VS sätter automatiskt referenser till **WPF assemblies**:
  - PresentationCore.dll
  - PresentationFramework.dll
  - System.Xaml.dll
  - WindowsBase.dll
- Visual Studio skapar även en **XAML applikation** ...
  - App.xaml
  - App.xaml.cs
- ... och huvudfönstret **MainWindow** ...
  - MainWindow.xaml
  - MainWindow.xaml.cs
- ... samt applikationens **konfigureringsfil**:
  - App.config (en XML fil)



# Skapa WPF applikationer i Visual Studio (VS)

- Som default ser applikationens filer ut enligt:

```
<Application x:Class="WpfTesterApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WpfTesterApp"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

- **<Application>** elementet i XAML filen motsvarar superklassen **Application** i code-behind filen.
- **Namespacen** och **klassen** i XAML filen motsvarar **namespacen** och **klassen** i code-behind filen.
- **Namespaces** för **primary namespace**, XAML nyckelorden (prefix **x**) och **applikationen** (prefix **local**) importeras i **<Application>** elementet.
- Ett element **<Application.Resources>** (utan innehåll) för applikationens resurser finns i XAML filen.
- Attributet **StartupUri** anger **namnet på huvudfönstrets XAML fil**.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace WpfTesterApp
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {

    }
}
```

# Skapa WPF applikationer i Visual Studio (VS)

- Som default ser huvudfönstrets filer ut enligt:

```
<Window x:Class="WpfTesterApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfTesterApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

- **<Window>** elementet i XAML filen motsvarar superklassen **Window** i code-behind filen.
- **Namespacen** och **klassen** i XAML filen motsvarar **namespacen** och **klassen** i code-behind filen.
- **Namespaces** för bl.a. **primary namespace**, XAML nyckelorden (prefix **x**) och **applikationen** (prefix **local**) importeras i **<Application>** elementet.
- Ett element **<Grid>** (en **layout manager** för ett rutnät) finns i XAML filen.
- Ett antal **attribut** för **huvudfönstret** tilldelas värden i XAML filen.
- **Huvudfönstrets konstruktor** anropar metoden **InitializeComponent()** i code-behind filen.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfTesterApp
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

# Skapa WPF applikationer i Visual Studio (VS)

- Som default ser konfigurationsfilen ut enligt:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
</configuration>
```

- **<XML>** elementet anger **XML versionen** och **teckenkodningsformatet**.
- **<configuration>** elementet innehåller ett **<startup>** element.
- **<startup>** element innehåller ett **<supportedRuntime>** element.
- **<supportedRuntime>** elementet anger:
  - **versionen (major version) på .NET ramverket (4.0).**
  - **typen på ramverket (.NETFramework) och den exakta versionen (4.8).**



Toolboxen

WpfTesterApp - Microsoft Visual Studio

File Edit View Project Build Debug Team Design Format Tools Architecture Test Analyze Window Help

Huvudmeny & Toolbar

Document Outline

Server Explorer Test Explorer Data Sources

MainWindow.xaml | MainWindow.xaml.cs

XAML

Code-behind

Grafisk designyta / Kodeditor  
(Shift + F7 / F7)

Som default kan man byta mellan XAML filen och Code Behind filen med <Shift> + <F7> respektive <F7>.

Document Outline

Solution Explorer

WpfTesterApp

Properties References App.config App.xaml App.xaml.cs MainWindow.xaml MainWindow.xaml.cs

Solution Explorer

Properties

Name <No Name> Type Grid

Arrange by: Category

Common

Cursor

DataContext

IsEnabled

Events

Properties

XAML editor

```
1 <Window x:Class="WpfTesterApp.MainWindow"
2 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6 xmlns:local="clr-namespace:WpfTesterApp"
7 mc:Ignorable="d"
8 Title="MainWindow" Height="450" Width="800">
```

Utskrifter

Felmeddelanden

Felmeddelanden / utskrifter

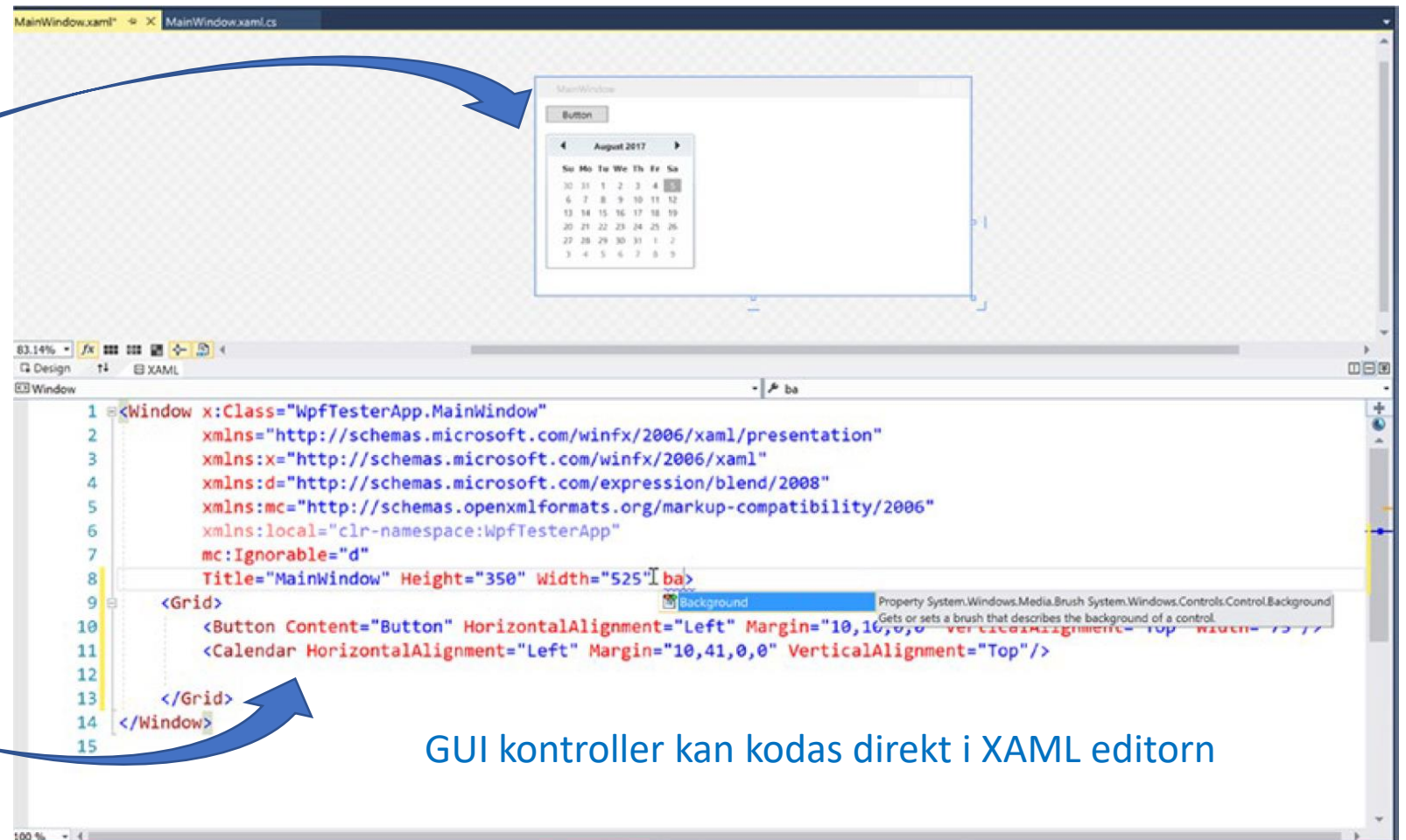
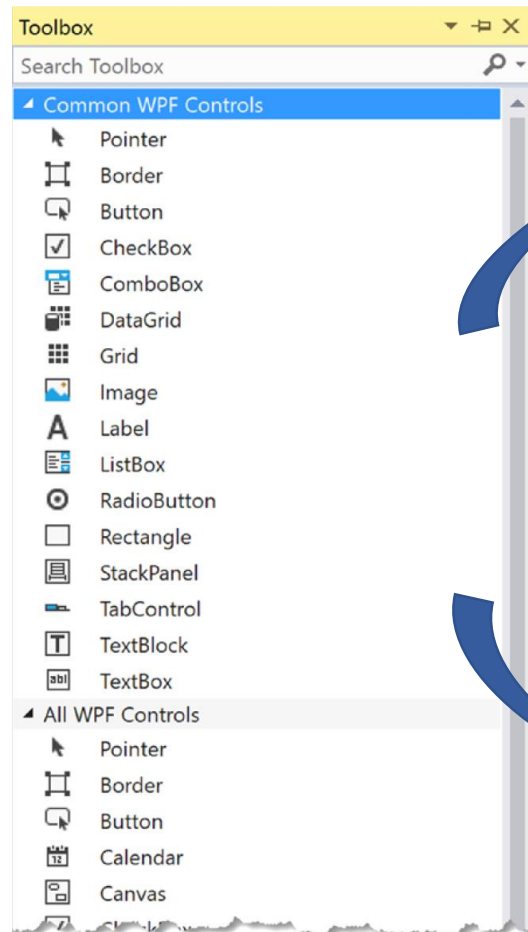
Error List ... Output



# Toolboxen och XAML designern/editorn

- Toolboxen innehåller GUI kontroller som kan *drag-and-drop*as till XAML designern/editorn.

View → Toolbox (om ej synlig)

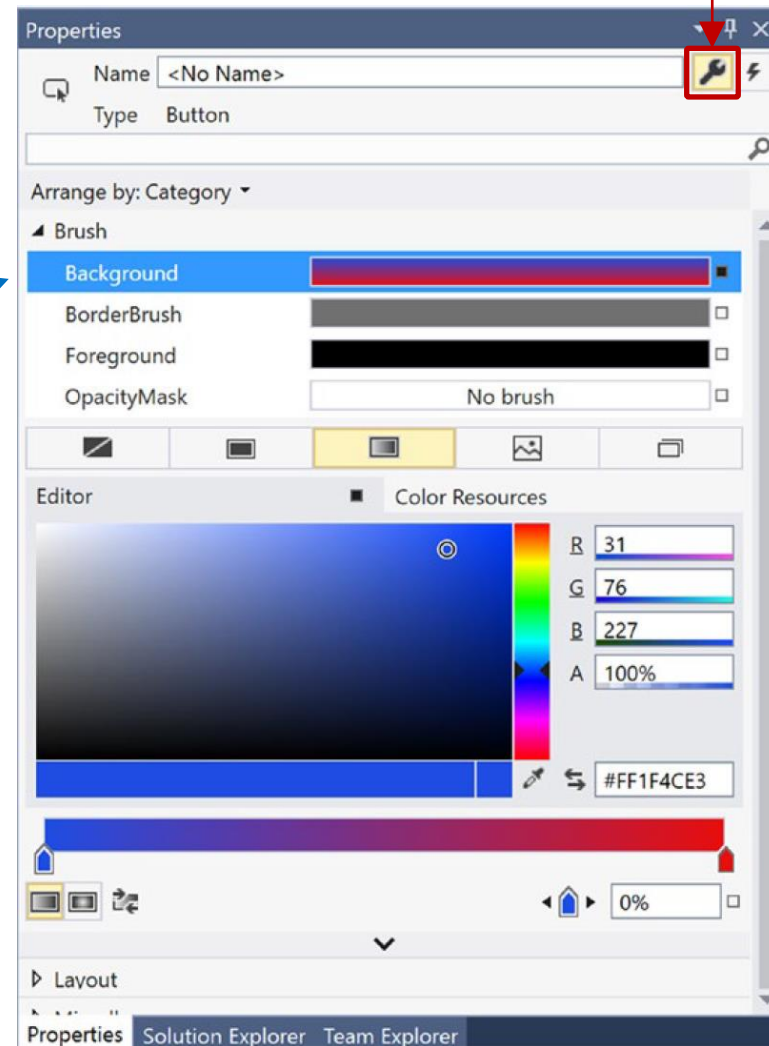


GUI kontroller kan kodas direkt i XAML editorn

# Properties editorn

- När man markerar GUI element i XAML designer eller XAML editor, kan man använda **properties editorn** för att manipulera properties för elementet.
- Property editorn är (default) längst ner till höger i VS, men man måste även välja "skiftnyckel" symbolen.
- Många properties har avancerade editors, t.ex. bakgrundsfärgen (**Background**) för en knapp (**Button**), som kan manipuleras med den integrerade pensel (**Brush**) editorn.
- Ändringarna återspeglas direkt i XAML koden.

```
<Button Content="Button" HorizontalAlignment="Left"
        Margin="10,10,0,0" VerticalAlignment="Top" Width="75">
  <Button.Background>
    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="#FFE90E0E" Offset="1"/>
      <GradientStop Color="#FF1F4CE3"/>
    </LinearGradientBrush>
  </Button.Background>
</Button>
```



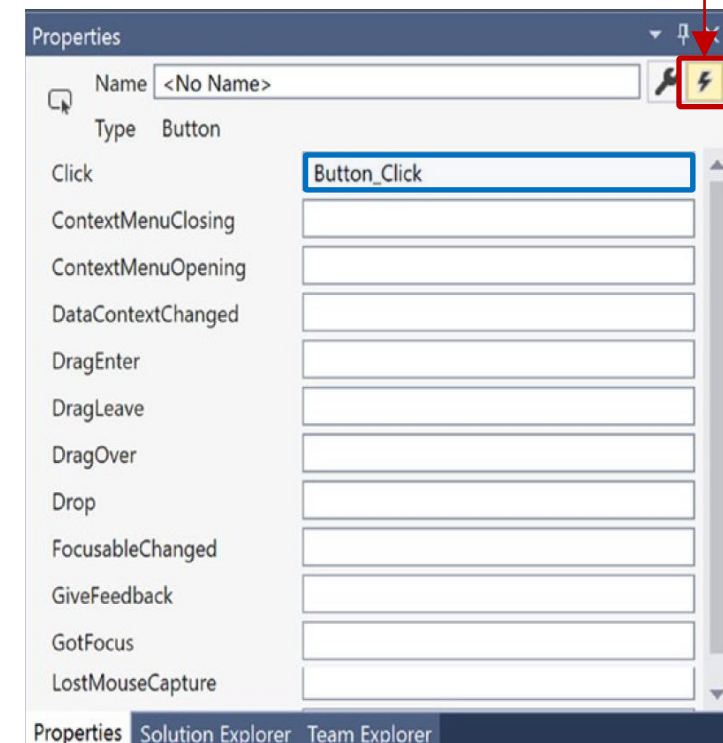
# Events editorn

- Om man istället väljer “blix” symbolen kan man editera *events* i **event editorn**.
- Exempelvis, om en knapp (**Button**) är markerad i XAML designer eller XAML editorn, kan man skapa en händelsehanterare för en knapptryckning genom att **dubbel-klicka** i “**Click**” fältet, vilket automatiskt skapar en händelsehanterare i **code-behind** filen med ett namn enligt mönstret:

*NameOfControl\_NameOfEvent*

- Om man inte har döpt GUI element, genererar VS ett namn som börjar med elementets typ med ett eventuellt löpnummer “**Button\_Click**”. Om knappen t.ex. hade döpts till “**myButton**” med XAML attributet **x:Name=“myButton”** hade istället händelsehanteraren fått namnet “**myButton\_Click**”. Det går även att dubbel-klicka på knappen i XAML designern för att generera **Click**-hanteraren. Man kan också skriva in ett godtyckligt namn på händelsehanteraren i **event editorn**.
- Den automatgenererade händelsehanteraren i **code-behind** filen ser ut enligt nedan, där man kan koda logiken för att hantera händelsen:

```
private void Button_Click(object sender, RoutedEventArgs e) {  
    MessageBox.Show("You clicked the button!");  
}
```

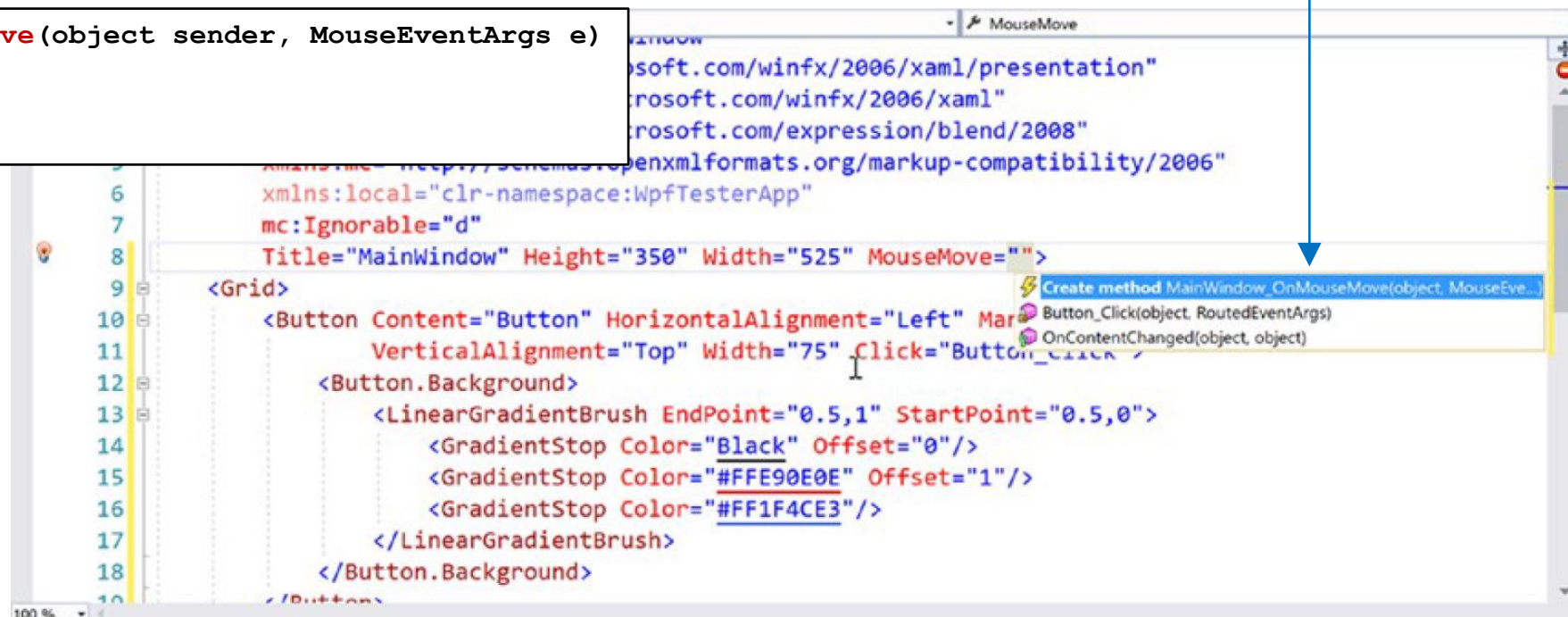


# Events i XAML editorn

- Man kan också hantera händelser i XAML editorn.
- Exempelvis, för att skapa en händelsehanterare för musförflyttningar i huvudfönstret, kan texten "**MouseMove**" efterföljt av ett "=" tecken skrivas in i **<Window>** elementet.
- Då kommer VS att **lista alla existerande kompatibla händelsehanterare** i **code-behind** filen, tillsammans med ett **Create** val. Om man t.ex. väljer **Create** valet kommer VS att skapa nedanstående händelsehanterare i **code-behind** filen.

```
private void MainWindow_MouseMove(object sender, MouseEventArgs e)
{
}

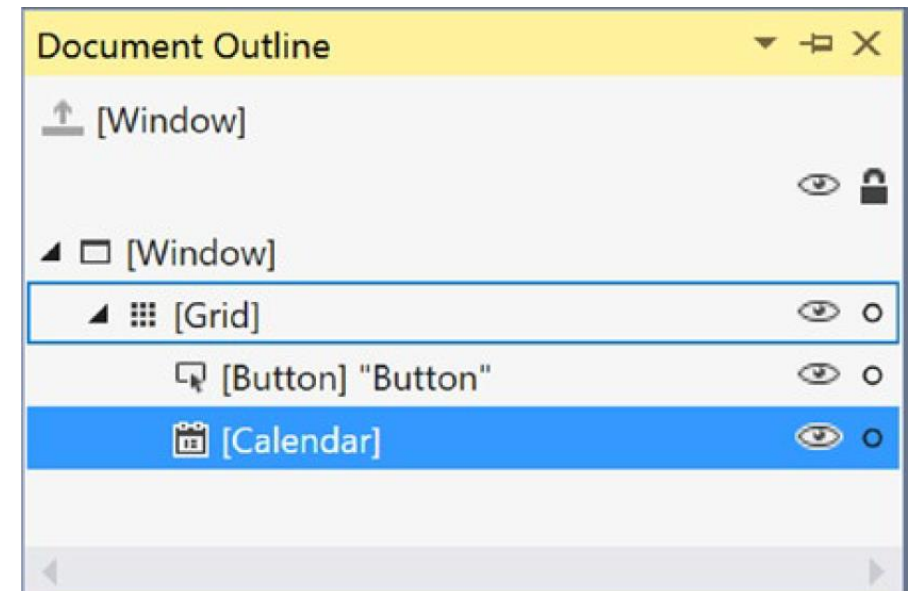
```





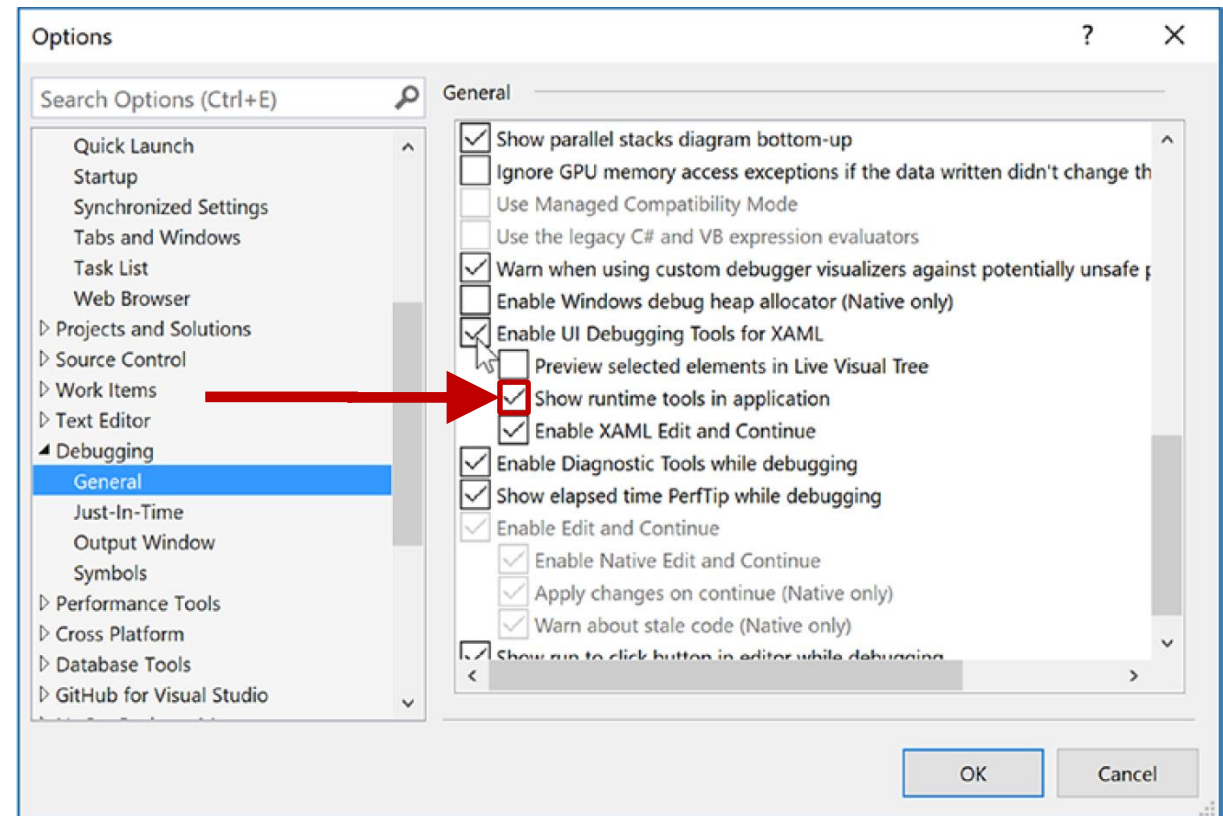
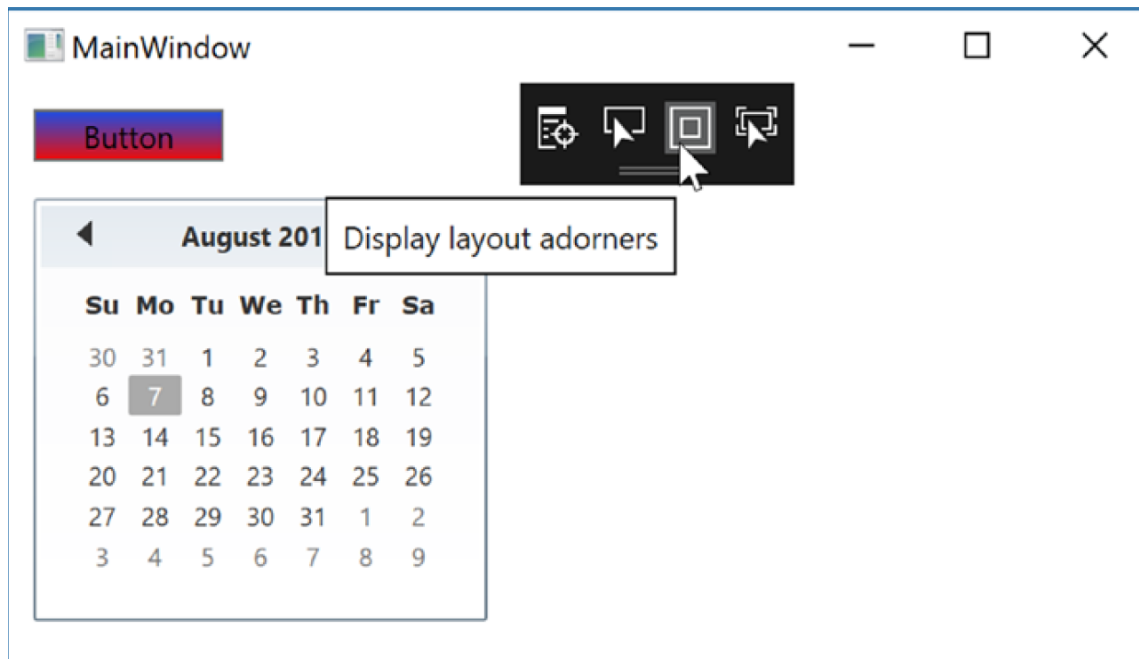
# Document Outline fönstret

- När man har många nästlade element i XAML designern/editorn kan **Document Outline** fönstret vara till hjälp för att visualisera alla element och välja element som skall manipuleras.
- **Document Outline** fönstret visas genom att välja **View → Other Windows → Document Outline** i huvudmenyn (och visas till vänster i VS som default).
- När man selekterar XAML designern, kommer **Document Outline** fönstret att visa alla nästlade element.
- **Document Outline** fönstret kan även användas för att temporärt dölja eller låsa element (dvs göra dem icke-modifierbara *design-time*), samt t.ex. för att gruppera element i nya **layout managers**.



# XAML debuggern

- När man debuggar en WPF applikation, visas som default XAML debuggern (vilket kan vara irriterande).
- Denna kan stängas av via huvudmenyn **Tool → Options → Debugging → General**.



# App.xaml

- **App.xaml** innehåller XAML för WPF programmets **applikationsklass** (som ärver från **Application**) och har en tillhörande **code-behind** fil **App.xaml.cs**.
- I applikationselementets start-tag **<Application>** anges först **applikationsklassen** med XAML nyckelordet **x:Class**. Därefter importeras de vanliga XAML namespaces, inklusive applikationens namespace med prefixet **local**. Slutligen anger XAML attributet **StartupUri** applikationens **huvudfönster** (fönstret som skall visas då applikationen startar).
- **Application** klassen har ett antal events, t.ex. **Startup** och **Exit** som kan utnyttjas för att implementera logik för vad som skall hända när applikationen startar respektive stängs. Om events skapas i XAML filen för dessa, genereras motsvarande händelsehanterare i **code-behind** filen **App.xaml.cs**.

## App.xaml

```
<Application x:Class="WpfTesterApp.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:WpfTesterApp"
  StartupUri="MainWindow.xaml" Startup="App_OnStartup" Exit="App_OnExit">

  <Application.Resources>
  </Application.Resources>

</Application>
```

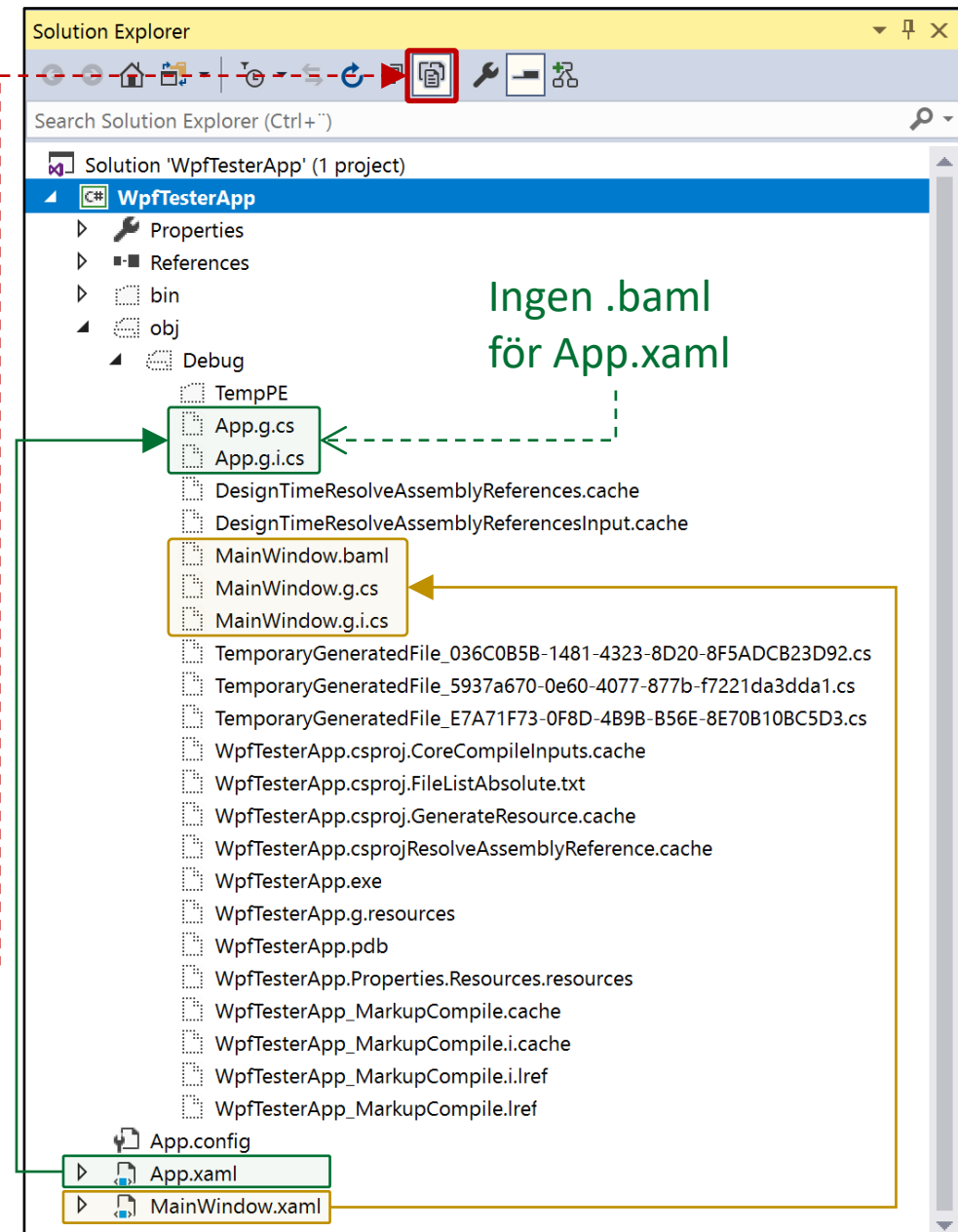
## App.xaml.cs

```
public partial class App : Application
{
    private void App_OnStartup(object sender, StartupEventArgs e)
    {
    }

    private void App_OnExit(object sender, ExitEventArgs e)
    {
    }
}
```

# Autogenererade filer

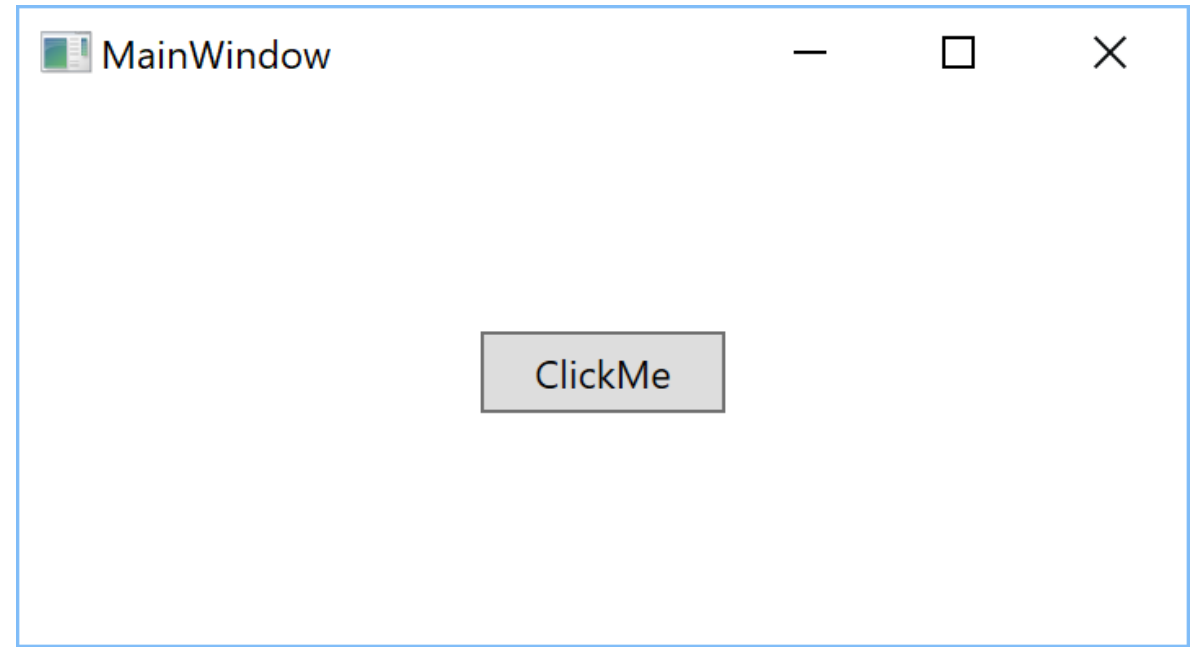
- När ett projekt kompileras skapas, för varje XAML fil (.xaml), tre autogenererade filer med samma namn som XAML filen, fast med filändelserna:
  - \*.g.cs (där “g” står för *autogenererad*)
  - \*.g.i.cs (där *i* står för *IntelliSense*)
  - \*.baml (*Binary Application Markup Language*)
- Filerna tillhör inte själva VS projektet, utan är byggartefakter som sparas i foldern **\obj\Debug** vid ett “debug” bygge (eller **\obj\Release** vid ett “release” bygge).
- Filerna kan ses genom att markera **projekt noden** i **Solution Explorer** och sedan klicka på **Show All Files** knappen.
- \*.g.cs och \*.g.i.cs filerna är oftast **identiska**, där den intressanta filen är \*.g.cs.
- \*.baml filen är \*.xaml filen i **binär-form**.





# Autogenererade filer

- Anta att en WPF applikation **App** har skapats i namespace **WpfTesterApp** med ett huvudfönster **MainWindow** som innehåller en knapp **Button** med namnet **myButton** och en händelsehanterare **myButton\_Click**.
- Vad innehåller filerna nedan?
  - App.xaml
  - App.xaml.cs
  - App.g.cs
  - MainWindow.xaml
  - MainWindow.xaml.cs
  - MainWindow.g.cs



# App.xaml, App.xaml.cs, App.g.cs

X:Class bestämmer namnet på applikationens **namespace** och **klass**.

## App.g.cs

```
using System;
using System.Diagnostics;
using System.Windows;
/*
 * Många System.Windows.* importer här
 */
using WpfTesterApp;
```

```
namespace WpfTesterApp {
    public partial class App : System.Windows.Application {
```

```
        public void InitializeComponent() {
            #line 5 "..\..\App.xaml"
            this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
            #line default
            #line hidden
        }
```

```
        public static void Main() {
            WpfTesterApp.App app = new WpfTesterApp.App();
            app.InitializeComponent();
            app.Run();
        }
    }
```

**Main()** metoden skapas implicit i App.g.cs (default) i WPF.

Metoden **InitializeComponent()** initialiserar **applikationsklassen**.

**Main()** metoden (entry point) skapar en instans av **applikationsklassen**, anropar **InitializeComponent()** och därefter **Run()** för att starta applikationen.

## App.xaml

```
<Application x:Class="WpfTesterApp.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:WpfTesterApp"
    StartupUri="MainWindow.xaml">

    <Application.Resources>
    </Application.Resources>

</Application>
```

**StartupUri** bestämmer **huvudfönstret**, dvs vilket fönster som skall visas när applikationen startar.

**Applikationsklassen** är **partial** i alla .cs filer

## App.xaml.cs

```
using System.Windows;

namespace WpfTesterApp
{
    public partial class App : Application
    {
    }
}
```

# MainWindow (.xaml, .xaml.cs och .g.cs)

## MainWindow.g.cs

```
namespace WpfTesterApp {
    public partial class MainWindow : System.Windows.Window, System.Windows.Markup.IComponentConnector {

        #line 10 "..\..\MainWindow.xaml"
        internal System.Windows.Controls.Button myButton;
        #line default
        #line hidden

        private bool _contentLoaded;

        public void InitializeComponent() {
            if (_contentLoaded) { return; }
            _contentLoaded = true;
            System.Uri resourceLocator = new System.Uri("/WpfTesterApp;component/mainwindow.xaml", System.UriKind.Relative);
            #line 1 "..\..\MainWindow.xaml"
            System.Windows.Application.LoadComponent(this, resourceLocator);
            #line default
            #line hidden
        }

        void System.Windows.Markup.IComponentConnector.Connect(int connectionId, object target) {
            switch (connectionId) {
                case 1:
                    this.myButton = ((System.Windows.Controls.Button)(target));
                    #line 10 "..\..\MainWindow.xaml"
                    this.myButton.Click += new System.Windows.RoutedEventHandler(this.myButton_Click);
                    #line default
                    #line hidden
                    return;
            }
            this._contentLoaded = true;
        }
    }
}
```

X:Name bestämmer knappens namn.

InitializeComponent() initialiserar fönsterklassen och alla dess element.

Connect() kopplar myButton knappens Click händelse till Click händelsehanteraren.

X:Class bestämmer namnet på fönstrets namespace och klass.

## MainWindow.xaml

```
<Window x:Class="WpfTesterApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfTesterApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Button x:Name="myButton" Content="ClickMe" Click="myButton_Click"/>
</Window>
```

## MainWindow.xaml.cs

```
/* Många using här. */
namespace WpfTesterApp
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void myButton_Click(object sender, RoutedEventArgs e)
        {
        }
    }
}
```

Konstruktorn anropar InitializeComponent().

Fönsterklassen är partial i alla .cs filer

# Sammanfattning: Filer för applikationen (Application)

- **App.xaml**

- **x:Class** i **<Application>** taggen bestämmer namnet på **applikationens namespace och klass**.
- XAML attributet **StartupUri** bestämmer **huvudfönstret** (som visas när applikationen startar).

- **App.xaml.cs**

- Innehåller **kod för XAML klassen** (code-behind), t.ex. **händelsehanterare**.

- **App.g.cs**

- Innehåller metoden **InitializeComponent()** - **sätter huvudfönstret** som **StartupUri** refererar till.
- Innehåller applikationens **Main()** metod (entry point) som:
  - Skapar en **instans av applikationsklassen**.
  - **Anropar InitializeComponent()**.
  - **Anopar Run()** för att starta applikationen.

# Sammanfattning: Filer för fönster (Window)

- **MainWindow.xaml**

- **x:Class** i **<Window>** taggen bestämmer namnet på **fönstrets namespace och klass**.
- **x>Name** bestämmer **namnet på element (klasser)** t.ex. för en knapp (**Button**).
- Events, t.ex. **Click**, bestämmer namnet på tillhörande **händelsehanterare**.
- Övriga XAML attribut sätter egenskaper för respektive element (klass).

- **MainWindow.xaml.cs**

- Innehåller **kod för XAML klassen** (code-behind), t.ex. **händelsehanterare**.
- Konstruktorn anropar **InitializeComponent()**.

- **MainWindow.g.cs**

- Innehåller **attribut för varje klass** (som tillhör respektive element i .xaml filen).
- Innehåller metoden **InitializeComponent()** som:
  - Skapar en **instans av varje klass** (som tillhör respektive element i .xaml filen).
  - Laddar alla initialvärden för alla element (klasser) från **BAML** filen som är inbäddad i assemblyn.
- Innehåller metoden **Connect()** som:
  - **Kopplar varje händelse till respektive händelsehanterare** (som anges i .xaml filen).

# Sammanfattning: \*.baml filer

- Innehåller all **XAML (.xaml)** i **binärform** och sparas som en **inbäddad resurs** i **assemblyn**.
- Alla **initialvärden** för **element** (klasser) finns lagrade i BAML filen.
- När applikationen startar och ett fönster laddas via ett anrop till **InitializeComponent()** i **.g.cs** filen, används **initialvärden i BAML filen för att initialisera respektive element (klass)**

MainWindow.g.cs

Detta sätter en referens till den inbäddade BAML filen

```
public void InitializeComponent()
{
    if (_contentLoaded) { return; }
    _contentLoaded = true;
    System.Uri resourceLocator = new System.Uri("/WpfTesterApp;component/mainwindow.xaml", System.UriKind.Relative);
    #line 1 "..\\..\\MainWindow.xaml"
    System.Windows.Application.LoadComponent(this, resourceLocator);
    #line default
    #line hidden
}
```

Detta laddar den inbäddade BAML filen och initialiserar alla element (klasser), t.ex. myButton.Height = 25.

# Application.Startup och Application.Current.Properties

- Man kommer inte åt **Main()\*** metoden i ett WPF program eftersom metoden definieras i den automatgenererade **.g.cs** filen. För att kunna läsa in växlar (*command line arguments*) kan man istället använda den statiska **Environment.GetCommandLineArgs()** metoden.
- Dock är växlarerna även tillgängliga via **Application** klassens **Startup** event, där de kan läsas från händelsehanterarens **StartupEventArgs** parameter, via **Args** propriety.
- Om man behöver lagra globala applikationsinställningar som kan accessas överallt i applikationen, kan dessa lagras i **Application** klassens **Properties** mapp.
- **Properties** mappen lagrar objekt av typ **Object**, dvs kan lagra objekt av alla typer, dock måste en **explicit typomvandling** göras när objekten läses från mappen.

```
private void App_OnStartup(object sender, StartupEventArgs e)
{
    Application.Current.Properties["GodMode"] = false;
    // Check the incoming command-line arguments and see if they
    // specified a flag for /GODMODE.
    foreach (string arg in e.Args)
    {
        if (arg.Equals("/godmode", StringComparison.OrdinalIgnoreCase))
        {
            Application.Current.Properties["GodMode"] = true;
            break;
        }
    }
}
```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // Did user enable /godmode?
    if ((bool)Application.Current.Properties["GodMode"])
    {
        MessageBox.Show("Cheater!");
    }
}
```

\* Det går att skapa en egen **Main()** metod explicit, men kräver lite extra arbete, se t.ex.:  
[https://www.infosysblogs.com/microsoft/2008/09/how\\_to\\_write\\_custom\\_main\\_metho.html](https://www.infosysblogs.com/microsoft/2008/09/how_to_write_custom_main_metho.html)  
<https://blog.magnusmontin.net/2020/01/31/custom-entry-point-wpf-net-core>  
<https://stackoverflow.com/questions/6156550/replacing-the-wpf-entry-point>



# Window.Closing och Window.Closed

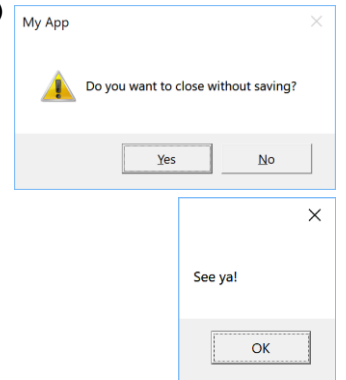
- När ett fönster stängs triggas två events; **Closing** och därefter **Closed**.
- Händelsehanteraren för **Closing** har en parameter av typ **CancelEventArgs** som innehåller **Cancel** propertyn.
  - Om man sätter **Cancel** till **false** kommer event **Closed** att anropas och fönstret stängs ner.
  - Om man sätter **Cancel** till **true** kommer **INTE** event **Closed** att anropas och fönstret stängs **INTE** ner.
- Händelsehanteraren för huvudfönstrets **Closing** event är t.ex. ett bra ställe att fråga användaren om han/hon verkligen vill stänga ner applikationen eller först spara sitt arbete.

```
public MainWindow()  
{  
    InitializeComponent();  
    this.Closing += MainWindow_Closing;  
    this.Closed += MainWindow_OnClosed;  
}
```

**MessageBox.Show()** skapar ett popup fönster med ett meddelande till användaren och får tillbaka användarens val (t.ex. "Yes" eller "No") som returvärde

När huvudfönstret stängs avslutas applikationen

```
private void MainWindow_Closing(object sender, System.ComponentModel.CancelEventArgs e)  
{  
    // See if the user really wants to shut down this window.  
    string msg = "Do you want to close without saving?";  
    MessageBoxResult result = MessageBox.Show(msg, "My App", MessageBoxButton.YesNo,  
                                                MessageBoxImage.Warning)  
    if (result == MessageBoxResult.No)  
    {  
        // If user doesn't want to close, cancel closure.  
        e.Cancel = true;  
    }  
}  
  
private void MainWindow_Closed(object sender, EventArgs e) {  
    MessageBox.Show("See ya!");  
}
```



# Mushändelser (*mouse events*)

- WPF APIt innehåller ett antal **mus-relaterade events** som man kan skapa **händelsehanterare** för. Basklassen **UIElement** definierar bl.a. mushändelserna **MouseMove**, **MouseUp**, **MouseDown**, **MouseEnter** och **MouseLeave**.
- Exempelvis kan man hantera **händelsen (event) MouseMove**, som använder **delegate System.Windows.Input.MouseEventHandler**, vars första parameter är av typ **object** och andra parameter av typ **System.Windows.Input.MouseEventArgs**.
- Klassen **MouseEventArgs** innehåller ett antal medlemmar med information om musens nuvarande egenskaper:
- XButton1** och **XButton2** utgör “*extended mouse buttons*” såsom (“*next*” och “*previous*” knapparna som finns på vissa möss för att navigera frammåt/bakåt i en webbläsare).
- GetPosition()** metoden returnerar musens **X** och **Y** koordinater (returneras som en **Point struct**) relativt till ett UI-element på fönstret (eller fönstret själv). För att erhålla koordinaterna relativt till fönstret kan man skicka in **this** som parameter till **GetPosition()** metoden. Exempelvis kan man registrera en **händelsehanterare** för **MouseMove** i fönstrets konstruktor.
- Nedanstående **händelsehanterare** för **MouseMove** händelsen visar musens koordinater (relativt fönstret) i fönstrets titel.

```
public class MouseEventArgs : EventArgs
{
    public Point GetPosition(IInputElement relativeTo);
    public MouseButtonState LeftButton { get; }
    public MouseButtonState MiddleButton { get; }
    public MouseDevice MouseDevice { get; }
    public MouseButtonState RightButton { get; }
    public StylusDevice StylusDevice { get; }
    public MouseButtonState XButton1 { get; }
    public MouseButtonState XButton2 { get; }
}
```

```
public MainWindow(string windowTitle, int height, int width)
{
    this.MouseMove += MainWindow_MouseMove;
}
```

```
private void MainWindow_MouseMove(object sender, System.Windows.Input.MouseEventArgs e)
{
    // Set the title of the window to the current (x,y) of the mouse.
    this.Title = e.GetPosition(this).ToString();
}
```



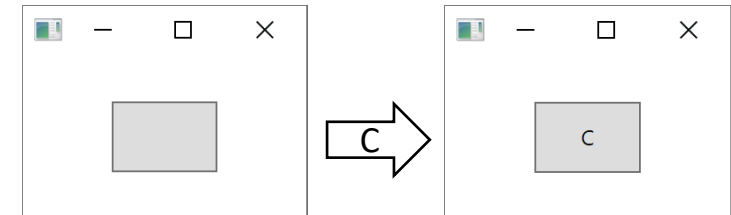
# Tangentbordshändelser (*keyboard events*)

- **UIElement** definierar också ett antal **tangentbords-relaterade events** som man kan skapa händelsehanterare för. Exempelvis, för att hantera händelserna för när en tangent “trycks ner” respektive “släpps upp” kan events **KeyDown** och **KeyUp** hanteras.
- Både **KeyDown** och **KeyUp** använder **delegate System.Windows.Input.KeyEventHandler**, vars första parameter är av typ **object** och vars andra parameter är av typ **KeyEventArgs**. Klassen **KeyEventArgs** definierar ett antal medlemmar med information om tangenters tillstånd.
- Anta att det finns ett fönster med namn **MainWindow**. För att hantera händelsen **KeyDown** på fönstret, kan en **händelsehanterare** registreras i fönstrets konstruktor.
- Nedanstående **händelsehanterare** ändrar texten på en knapp med namn **myButton** till aktuell tangent som trycks ner på tangentbordet.

```
public class KeyEventArgs : KeyboardEventArgs
{
    public bool IsDown { get; }
    public bool IsRepeat { get; }
    public bool IsToggled { get; }
    public bool IsUp { get; }
    public Key Key { get; }
    public KeyStates KeyStates { get; }
    public Key SystemKey { get; }
}
```

```
public MainWindow(string windowTitle, int height, int width)
{
    this.KeyDown += MainWindow_KeyDown;
}
```

```
private void MainWindow_KeyDown(object sender, System.Windows.Input.KeyEventArgs e)
{
    // Display key press on the button.
    myButton.Content = e.Key.ToString();
}
```



# Microsofts WPF dokumentation

- Använd Microsofts dokumentation för detaljerade beskrivningar om WPF.

<https://docs.microsoft.com/en-us/dotnet/framework/wpf/index>

# Sammanfattning Windows Presentation Foundation (WPF) 1

## Windows Presentation Foundation (WPF) vs. Windows Forms (WinForms)

- **WinForms** är ett **äldre GUI bibliotek** där **olika API:n** används för att skapa grafiska användargränssnitt, 2D/3D grafik och strömma video.
- **WPF** är ett **modernt GUI bibliotek** som har en **gemensam objektmodell** där **samma API** används för att skapa grafiska användargränssnitt, 2D/3D grafik och strömma video.
- I **WPF** är applikationens **visuella design** (XAML) **separerad från programmeringslogiken** (C#).
- **WPF och WinForms** applikationer **kan endast köras på Windows OS** för tillfället (.NET Framework 4.8, .NET Core 3.1).
- **Huvudsakliga assemblyn för WPF:** PresentationCore, PresentationFramework, System.Xaml, WindowsBase.
- **Huvudsakliga namespaces för WPF:** System.Windows.\*

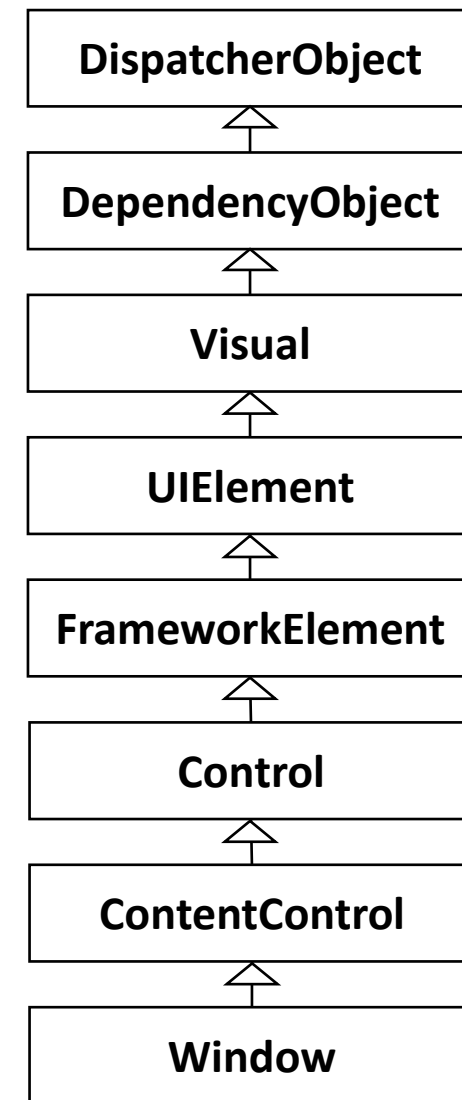
## Application och Window

- En **WPF applikation** representeras av klassen **System.Windows.Application**.
- Varje **WPF** applikation har ett antal **fönster** (**System.Windows.Window**) där **ett fönster utgör huvudfönstret**.
- När **huvudfönstret stängs, avslutas WPF applikationen**.

# Sammanfattning Windows Presentation Foundation (WPF) 1

## Arvshierarkin i WPF

- **DispatcherObject** innehåller bl.a. **propertyn Dispatcher** som används för att **byta kontext till GUI tråden** (ett **GUI element** kan endast **accessas från GUI tråden**).
- **DependencyObject** innehåller funktionalitet för **dependency properties** som **möjliggör WPF teknologier** såsom **styles, databindning och animering**.
- **Visual** kommunicerar med det underliggande **DirectX subsystemet** och innehåller bl.a. support för att **rendera och transformera ett element**.
- **UIElement** innehåller bl.a. funktionalitet för att **gömma och visa ett element**, samt support för ett antal **mus- och tangentbordshändelser**.
- **FrameworkElement** innehåller bl.a. funktionalitet för att beräkna ett elements **dimensioner** och **propertyn Name** för att **namnge ett element**, samt support för bl.a. **storyboarding** (animering), **databindning** och **WPFs resurssystem**.
- **Control** innehåller bl.a. funktionalitet för en kontrolls **storlek, färg och tab-ordning**, samt support för bl.a. **templating services** (CSS).
- **ContentControl** innehåller **Content propertyn**, dvs det **visuella elementets innehåll**.
- **Window** representerar ett fönster, som ägs av **Application**, och ärver funktionalitet från sina superklasser.



# Sammanfattning Windows Presentation Foundation (WPF) 1

## XML och XAML

- **XAML** (eXtensible Application Markup Language) är ett **XML-baserat märkspråk**.
- Alla XML-baserade språk kan innehålla **element, attribut, XML deklARATIONER, namespaces och kommentarer**.
- Ett **märkspråk använder taggar** (tags) för att **definiera element** (elements) i ett **dokument** (fil), där dokumentet är **text-baserad** och innehåller **vanliga ord** istället för en speciell syntax som i ett programmeringsspråk.
- **XAML** definierar taggar (element) och attribut för att **beskriva visuella element i WPF applikationer**.



# Sammanfattning Windows Presentation Foundation (WPF) 1

## XAML

- En **WPF applikation** representeras i XAML med taggen (elementet) **<Application>**.
- Ett **fönster** representeras i XAML med taggen (elementet) **<Window>**.
  - Varje **WPF applikation** har åtminstone ett fönster (**Window**), dvs **huvudfönstret**.
- Andra **element**, t.ex. **<Button>**, används för att **bygga upp GUI** inom **<Window>** elementet.
- Varje **XAML element** kan ha noll till flera **XAML attribut**, t.ex. **<Button Height="10">**.
- Längst upp i både **<Application>** och **<Window>** elementen, anges åtminstone alltid **attributen**:
  - **x:Class="..."** som anger **namespacen** och **klassen** för motsvarande **Application/Window**.
  - **xmlns="..."** (utan prefix), dvs **primary namespace** som importerar **WPFs grundläggande assemblyn**.
  - **xmlns:x="..."** (prefix **x**), dvs en **namespace** som importerar **XAML-specifika keywords (nyckelord)**.
  - **xmlns:local="clr-namespace:..."** (prefix **local**), importerar alla **typer** i **applikationens huvudnamespace**.
- **<Window>** element har ofta följande (talar om för XAML processorn att skippa allt med prefix **d**):
  - **xmlns:d="http://schemas.microsoft.com/expression/blend/2008"**
  - **xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"**
  - **mc:Ignorable="d"**

# Sammanfattning Windows Presentation Foundation (WPF) 1

## XAML

- Varje **XAML element** representerar en **typ**.
- Varje **XAML attribut** (förutom XAML nyckelord) representerar en **medlem i en typ**.
- XAML attribut kan användas för att **namnge en händelsehanterare** för en händelse (event) i elementet.
- **XAML keywords (nyckelord)**
  - XAML innehåller ett antal **nyckelord** (reserverade ord).
  - **x:Name** används för att **namnge ett element** så att den kan **accessas via en variabel med samma namn i code-behind filen** (.xaml.cs).
  - **x:Class="..."** anger **namespace** och **klass** för motsvarande **Application/Window**.
  - **x:ClassModifier="..."** anger **synligheten** för motsvarande **klass** (XAML element).
  - **x:FieldModifier="..."** anger **synligheten** för motsvarande **medlem** (XAML attribut).
  - Andra vanliga nyckelord är **x:Array**, **x:Key**, **x:Null**, **x:Static**, **x:Type** och **x:TypeArguments**.

# Sammanfattning Windows Presentation Foundation (WPF) 1

## XAML

- XAML element ärver **propertyn Content** (från klassen **ContentControl**), där **Content** propertyn kan **tilldelas ett värde**:
  - Via ett **XAML attribut**, t.ex. `<Button content="Yo!" />`
  - Mellan elementets **start- och slut-tag**, t.ex. `<Button>Yo!</Button>`
  - Med **Property-Element syntax**.
- **Typkonverterar-klasser** omvandlar automatiskt **strängar** i enkla **tilldelningar** av **Content** propertyn till korrekt underliggande typ.
- För att tilldela **komplex (nästlad) Content** till ett XAML element används **Property-Element** syntax:

```
<DefiningClass>  
  <DefiningClass.PropertyOnDefiningClass>  
    <!-- Value for Property here! -->  
  </DefiningClass.PropertyOnDefiningClass>  
</DefiningClass>
```
- Ett **barn-element** kan **tilldela ett värde till ett förälder-elements property** via **Attached Properties** syntax:

```
<ParentElement>  
  <ChildElement ParentElement.PropertyOnParent = "Value">  
</ParentElement>
```
- **Markup Extensions**
  - En **Markup Extension** är en **klass som ärver från MarkupExtension**, där man oftast inte behöver definiera egna Markup Extensions.
  - **Keywords** (nyckelord) såsom **x:Array**, **x:Null**, **x:Static** och **x:Type** är egentligen **Markup Extensions** "under huven".
  - **Markup Extensions** används för att **tilldela ett värde till ett XAML attribut**, där **värdet erhålls från en annan klass**, enligt syntaxen:

```
<Element PropertyToSet = "{MarkupExtension}"/>
```

# Sammanfattning Windows Presentation Foundation (WPF) 1

## Code-Behind

- Varje **XAML fil** (\*.xaml) har en **motsvarande code-behind fil** (\*.xaml.cs) med samma namn, fast med filändelsen **.xaml.cs** (C# kod).
- **Code-behind filen** har samma **namespace och klass** som anges med nyckelordet **x:Class** i XAML filen.
- Ett **namngivet element** (x:Name="myName") i XAML filen kan **accessas som en variabel** med samma namn "myName" i **code-behind filen**.
- **Code-behind filen innehåller händelsehanteraren** som finns namngiven för en händelse (event) i ett XAML element.
- **Klassen i code-behind filen är definierad som en partiell klass.**

## Automatgenererade filer

- **För varje XAML fil** \*.xaml **automatgenereras tre filer** (\*.g.cs, \*.g.i.cs och .baml) utöver code-behind filen \*.xaml.cs.
- I filerna finns **resten av den partiella klassen** (från \*.xaml.cs filen) definierad, i samma namespace.
- Filerna **\*.g.cs och \*.g.i.cs är identiska**, där **\*.g.i.cs används av Visual Studios Intellisense**, och **innehåller "boiler plate" kod** för motsvarande \*.xaml.cs fil.
- Filen **\*.baml innehåller all XAML (.xaml) i binärform** och sparas som en **inbäddad resurs i assemblyn**.
- Filerna tillhör inte själva VS projektet, utan är **byggartefakter** som sparas i foldern:
  - \obj\Debug (debug bygge)
  - \obj\Release (release bygge).

# Sammanfattning Windows Presentation Foundation (WPF) 1

## App (.xaml, .xaml.cs, .g.cs, .baml)

- **App.xaml** innehåller **XAML** för **Application (App)** klassen i **<Application>** elementet.
  - **x:Class** i **<Application>** taggen bestämmer **namnet** på applikationens namespace och klass.
  - XAML attributet **StartupUri** bestämmer **huvudfönstret** (som visas när applikationen startar).
- **App.xaml.cs** är **code-behind** filen för **Application** klassen och innehåller bl.a. kod för Application klassens händelsehanterare.
- **App.g.cs** är en **automatgenererad** fil som bl.a.:
  - Innehåller metoden **InitializeComponent()** som **sätter huvudfönstret** som **StartupUri** refererar till.
  - **Innehåller applikationens Main() metod** (entry point) som:
    - Skapar en **instans** av applikationsklassen.
    - Anropar **InitializeComponent()**.
    - Anopar **Run()** för att starta applikationen.
- **App.baml** är en **automatgenererad** fil:
  - Innehåller all **XAML (.xaml)** för App klassen i **binärform** och sparas som en **inbäddad resurs** i **assemblyn**.
  - Alla **initialvärden** för **element** (klasser) finns lagrade i **BAML** filen.
  - När **applikationen startar** och applikationen laddas via ett anrop till **InitializeComponent()** i .g.cs filen, används **initialvärden** i **BAML** filen för att **initialisera respektive element** (klass instans).

# Sammanfattning Windows Presentation Foundation (WPF) 1

## MainWindow (.xaml, .xaml.cs, .g.cs)

- **MainWindow.xaml** innehåller **XAML** för WPF applikationens **huvudfönster** (**MainWindow** klassen) i **<Window>** elementet
  - **x:Class** i **<Window>** taggen bestämmer **namnet på fönstrets namespace och klass**.
  - **x>Name** bestämmer **namnet på element** (klasser) t.ex. för en knapp (Button).
  - **Events**, t.ex. Click, bestämmer **namnet på tillhörande händelsehanterare**.
  - Övriga **XAML attribut** sätter **egenskaper för respektive element** (klass).
- **MainWindow.xaml.cs** är **code-behind** filen för **MainWindow** klassen och innehåller bl.a. kod för alla händelsehanterare.
  - Konstruktorn **MainWindow()** anropar **InitializeComponent()**.
- **MainWindow.g.cs** är en **automatgenererad** fil som bl.a.:
  - Innehåller **attribut för varje klass** (som tillhör respektive element i .xaml filen).
  - Innehåller metoden **InitializeComponent()** som:
    - **Skapar en instans av varje klass** (som tillhör respektive element i .xaml filen).
    - **Laddar alla initialvärden för alla element** (klasser) **från BAML filen** som är inbäddad i assemblyn.
  - Innehåller metoden **Connect()** som:
    - **Kopplar varje händelse till respektive händelsehanterare** (som anges i .xaml filen).
- **MainWindow.baml** är en **automatgenererad** fil:
  - Innehåller all **XAML (.xaml)** för huvudfönstret i **binärform** och sparas som en **inbäddad resurs i assemblyn**.
  - Alla **initialvärden för element** (klasser) finns **lagrade i BAML filen**.
  - När **applikationen startar** och ett fönster laddas via ett anrop till **InitializeComponent()** i .g.cs filen, används **initialvärden i BAML filen** för att **initialisera respektive element** (klass instans).

# Sammanfattning Windows Presentation Foundation (WPF) 1

## WPF applikationer i Visual Studio

- **Projekttypen** i Visual Studio för en WPF applikation är **WPF App (.NET Framework)**.
- Visual Studio sätter automatiskt **referenser till** assemblies **PresentationCore**, **PresentationFramework**, **System.Xaml** och **WindowsBase**.
- Visual Studio **skapar filerna** för applikationen (**App.xaml**, **App.xaml.cs**) och huvudfönstret (**MainWindow.xaml**, **MainWindow.xaml.cs**) samt applikationens konfigureringsfil (**App.config**).
- Visual studio innehåller:
  - En **Toolbox** där **visuella element** kan *drag-and-droppas* till den **Grafiska Designytan**.
  - En **Property Editor** där XAML **attributs värden** kan **editeras**.
    - Visual Studio innehåller många **Grafiska Property Editors**, t.ex. för att skapa en komplex pensel (Brush).
  - En **Event editor** där ett XAML elements **händelser** (events) kan **kopplas till en händelsehanterare**.
  - En **XAML editor** där XAML element och XAML attribut kan definieras.
  - En **Document Outline** som visar **trädstrukturen** av element i XAML dokumentet.
    - Man kan välja att **dölja eller visa element** (*design time*).
    - Man kan **flytta element** i trädstrukturen.
  - En **XAML debugger** (som går att stänga av).



# Sammanfattning Windows Presentation Foundation (WPF) 1

## Application events

- **Application** klassen innehåller bl.a.:
  - Händelserna **Startup** och **Exit**, som kan hanteras för att implementera **logik för när applikationen startas/avslutas**.
  - En property **Current.Properties**, där **global information** för applikationen kan sparas.

## Window events

- **Window** klassen innehåller bl.a.:
  - Händelserna **Closing** och **Closed**, som kan hanteras för att implementera **logik för när fönstret stängs**.

## Mouse events

- XAML element ärver bl.a. musehändelserna **MouseMove**, **MouseUp**, **MouseDown**, **MouseEnter** och **MouseLeave** från **UIElement**.
- **MouseEventArgs** innehåller ett antal medlemmar med information om **musens nuvarande egenskaper**, t.ex. kan **muspekarens koordinater** hämtas med metoden **GetPosition()**.

## Keyboard events

- XAML element ärver bl.a. tangentbordshändelserna **KeyDown** och **KeyUp** från **UIElement**.
- **KeyEventArgs** innehåller ett antal medlemmar med information om **tangenters nuvarande tillstånd**, t.ex. kan information om **aktuell tangent** fås från propertyn **Key**.