.NET 3(4).x

WPF (Windows Presentation Foundation)

/*WinFS (*Windows File Storage Foundation*)*/

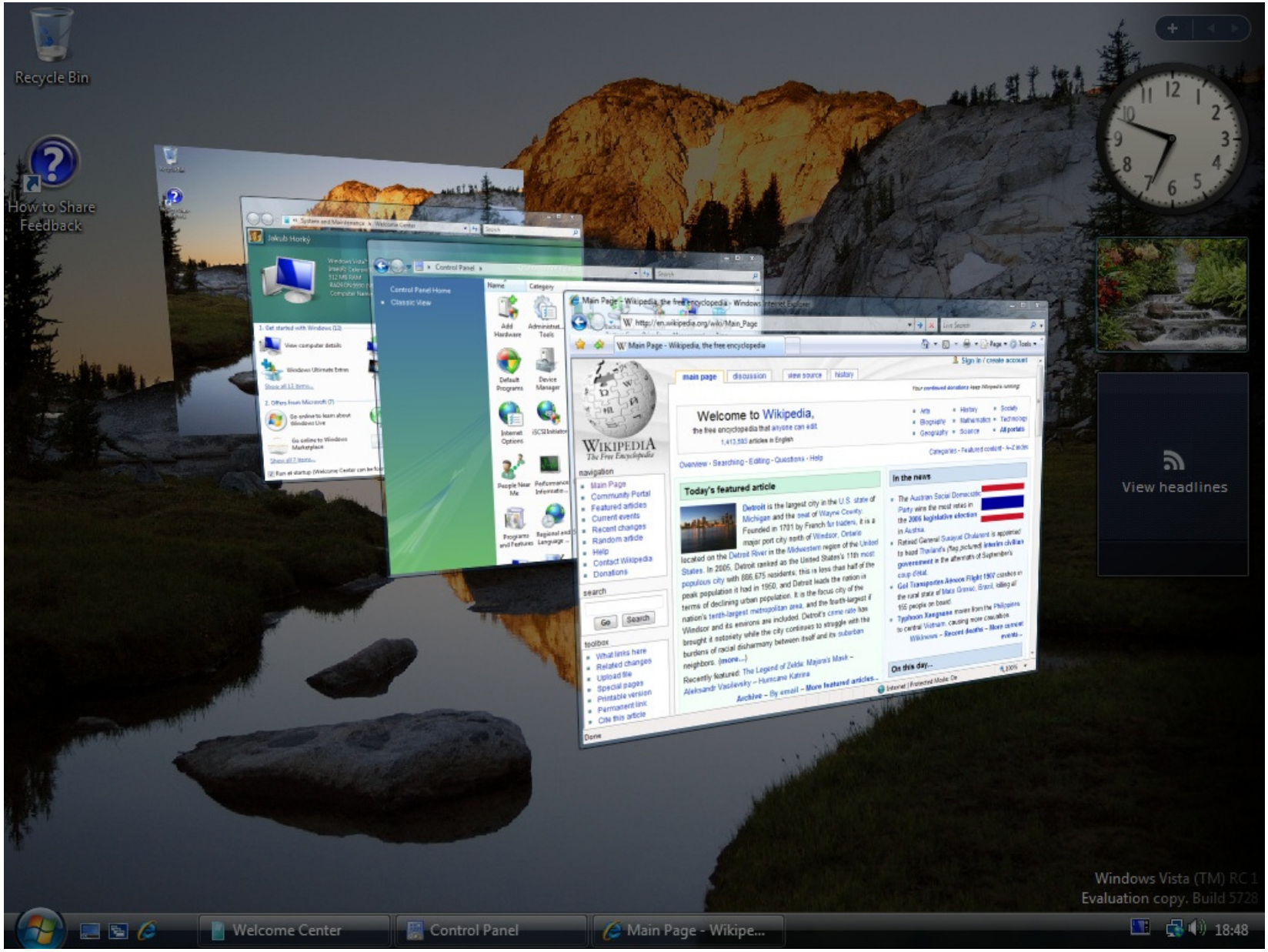
WCF (*Windows Communication Foundation*) ↔ Indigo

Windows Workflow Foundation

Windows CardSpace

Maja Štula

Ak. God. 2011/2012



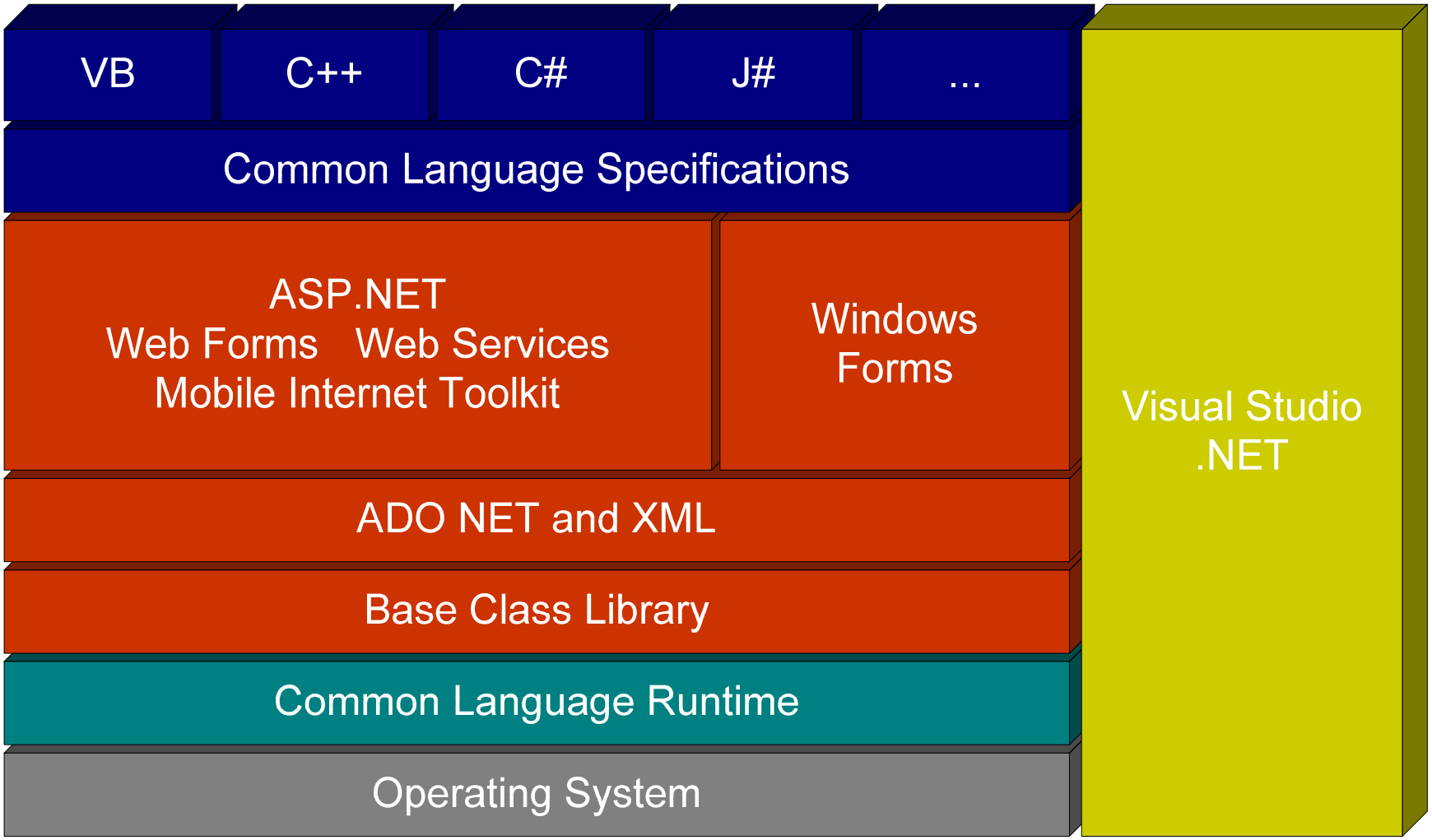
Windows Vista 3D look (Aero)

Windows Vista hardver

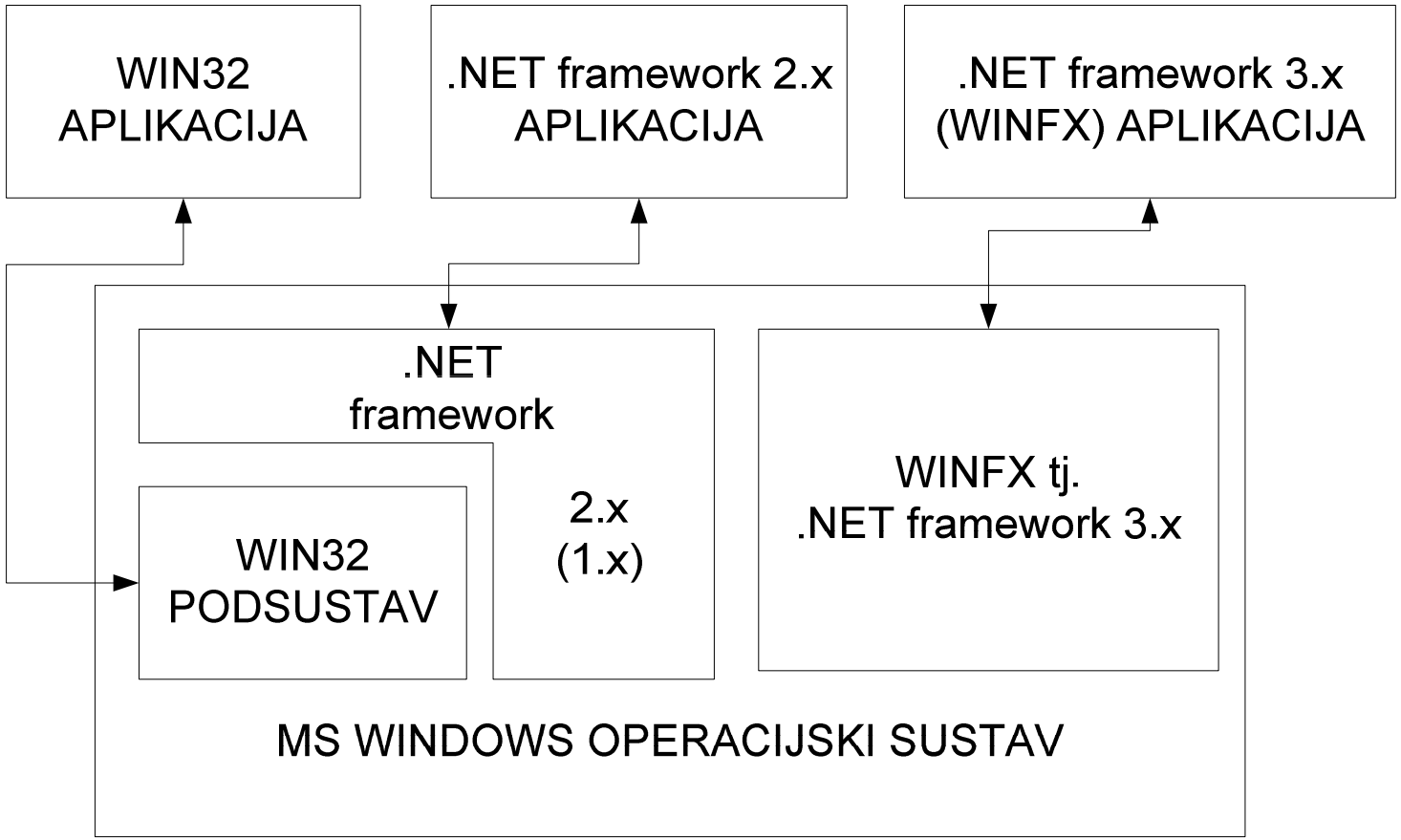
	Vista Capable	Vista Premium Ready
Processor	800 MHz	1.0 GHz
Memory	512 MB RAM	1 GB RAM
Graphics Card	DirectX 9 capable	DirectX 9 capable GPU with Hardware Pixel Shader v2.0 and WDDM Driver support
Graphics Memory	32 MB RAM	128 MB RAM up to 2,304,000 total pixels (e.g. 1920 × 1200) or 256 MB+ for greater resolutions [37]
HDD capacity	20 GB	40 GB
HDD free space	15 GB	15 GB
HDD type	Normal	Normal, but Hybrid flash memory/hard disk recommended
Other drives	DVD-ROM	DVD-ROM

DOTNET

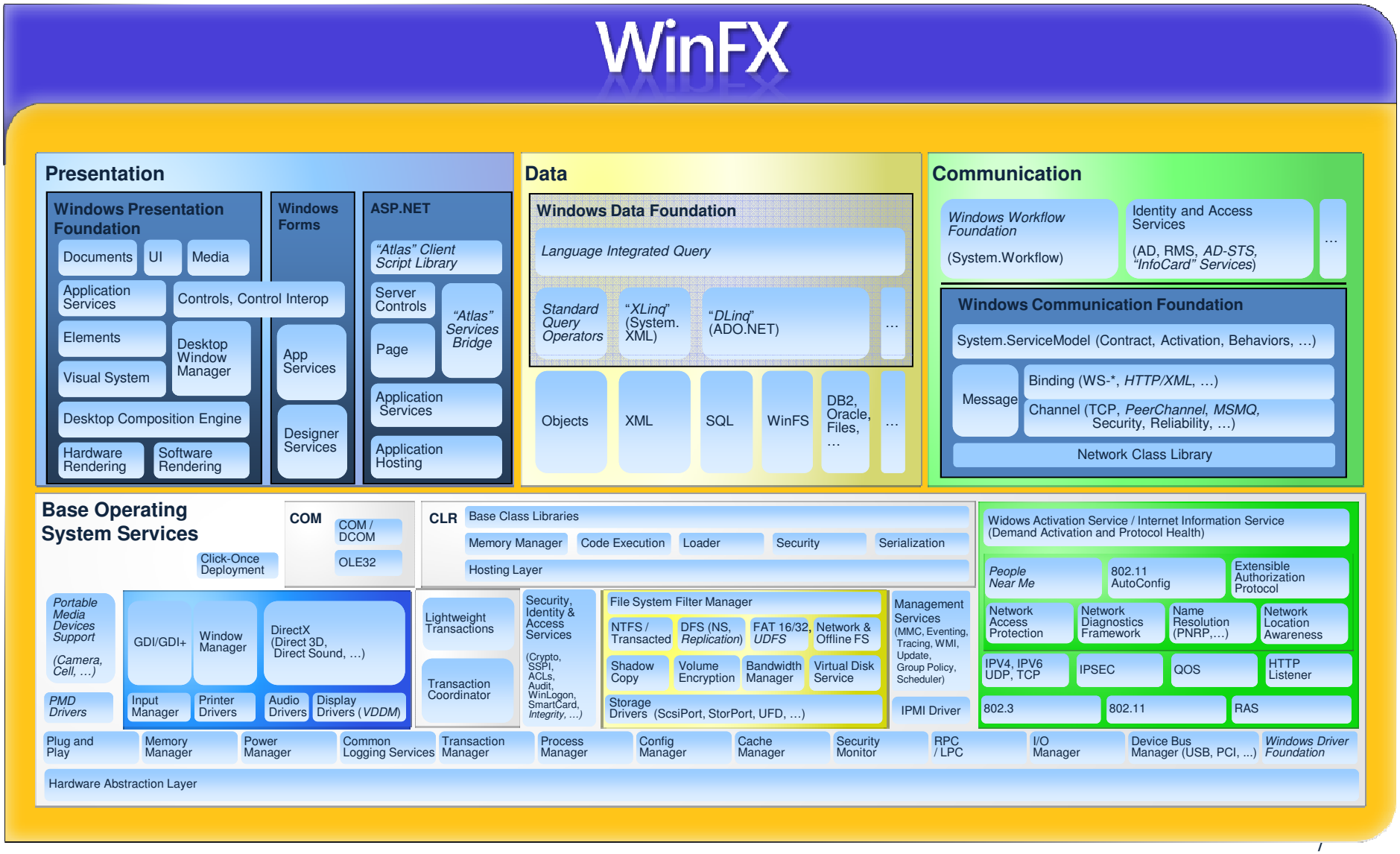
- Veliki dio klasa FCL-a (*Framework Class Library*) .NET 2.x su omotači WIN32 API funkcija. .NET 2.x framework se velikim dijelom oslanja na WIN32. Npr. forme su .NET sučelje prema klasičnim API pojmovima kao što su WndProc, HWND, MSG.
- Nova verzija .NET je verzija 3.x (najnovija 4.x.) se ne oslanja na WIN32 podsustav.


VB
C++
C#
J#
...
Common Language Specifications
ASP.NET
Web Forms Web Services
Mobile Internet Toolkit
Windows
Forms
ADO NET and XML
Base Class Library
Common Language Runtime
Operating System
Visual Studio
.NET

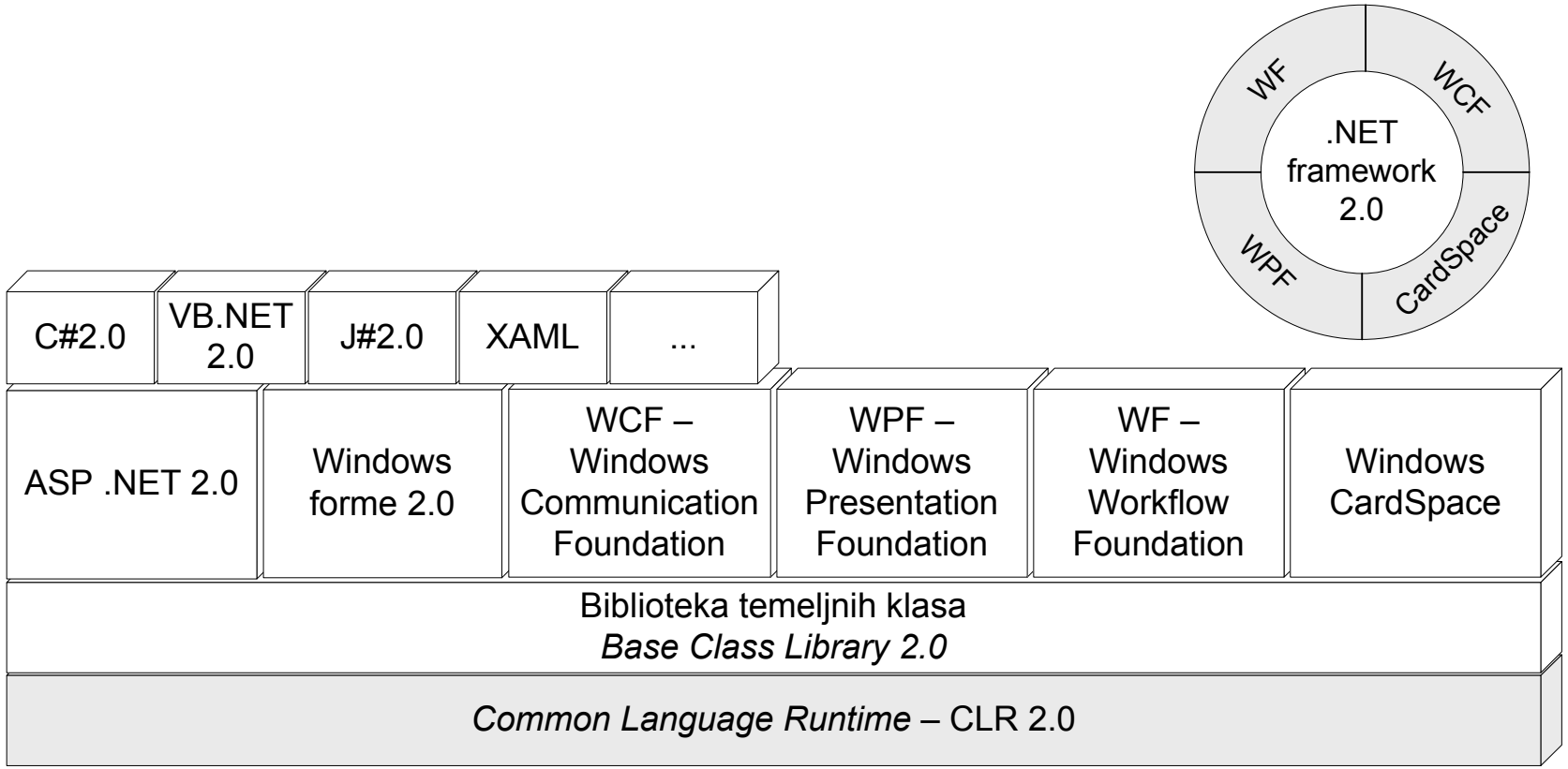
Podrška za različite programske modele



Windows Vista Arhitektura



Struktura .NET *frameworka* 3.x



.NET framework 3.x

- WPF je novi grafički podsustav .NET *frameworka 3.x* koji u sebi objedinjava funkcionalnosti postojećih grafički podsustava (WIN32, DirectX) i pruža jedinstveni programski model (kolekciju klasa) za razvoj grafičkog sučelja aplikacije.
- WCF modul predstavlja osnovu za razvoj distribuiranih aplikacija objedinjujući sve potrebne komunikacijske funkcionalnosti.

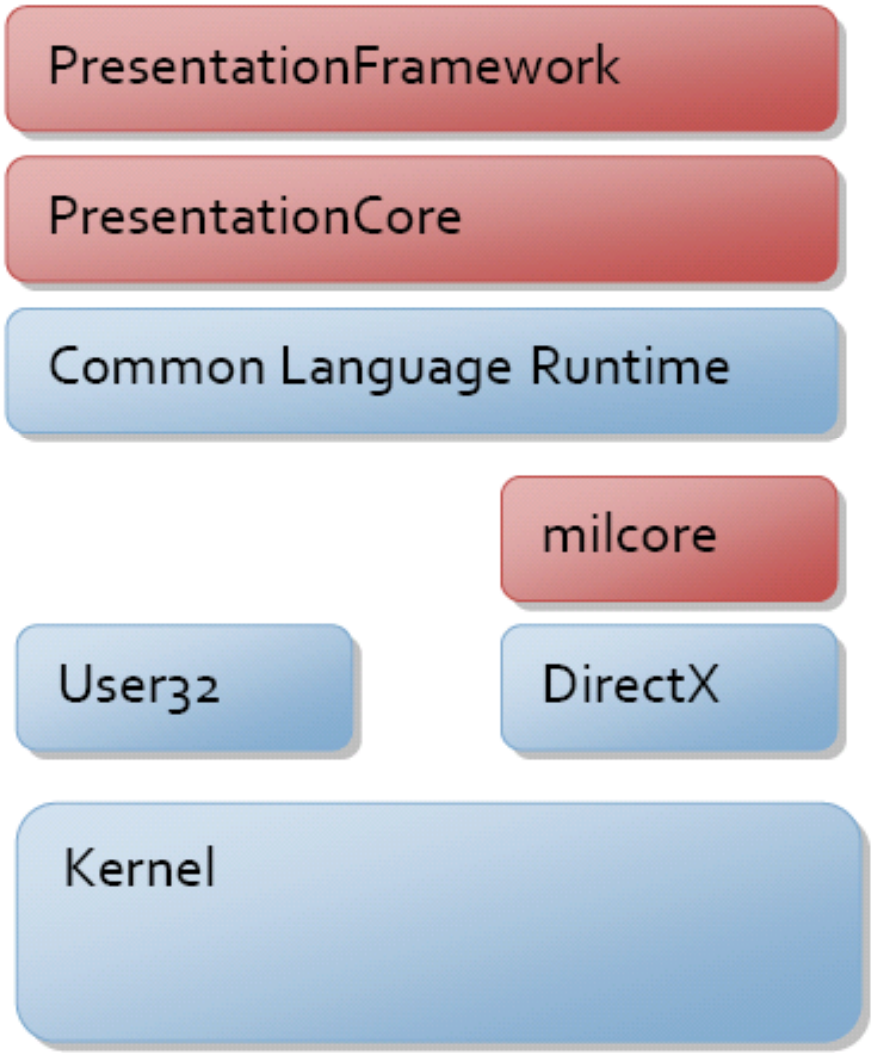
.NET framework 3.x

- WF dio .NET *frameworka 3.x* omogućava i olakšava razvoj aplikacija sa definiranim tokom radnje (*workflow-enabled applications*). Tok radnje aplikacije podrazumijeva niz događaja koji su međusobno povezani poput npr. ispunjavanja upitnika. Kada se uspješno popuni prva stranica upitnika korisnik može prijeći na sljedeću stranicu upitnika. Ukoliko korisnik nije popunio sve potrebne podatke na prvoj stranici upitnika aplikacija ga ponovo usmjerava na događaj ispunjavanja prve stranice upitnika prije no što može prijeći na događaj ispunjavanja sljedeće stranice upitnika.
- *CardSpace* komponenta upravlja digitalnim identitetima korisnika te pruža jedinstveno sučelje za odabir identiteta koji korisnik koristi u određenom računalnom postupku (npr. pristup web aplikaciji).

Windows Presentation Foundation (WPF)

- Windows Presentation Foundation (WPF), (Avalon) je grafički podsustav uključen u Vistu, a može se instalirati i na XP SP2 i Windows Server 2003 SP1 kao WinFX Runtime Components te kao alat (*managed-code programming suite*) koji proširuje .NET framework za Visual Studio 2005.
- Pruža konzistentni programski model za razvoj desktop aplikacija i aplikacija koje će se izvršavati u pretraživaču. Omogućava bogatije kontrole, dizajn i razvoj vizualnih aspekata Windows programa.
- Cilj je unificiranje aplikacijskih servisa kao što su grafičko sučelje, 2D i 3D grafika, vektorska grafika, rasterska grafika, animacije, povezivanje podataka (*data binding*), audio i video.

Windows Presentation Foundation



- Glavne komponente arhitekture su *PresentationFramework*, *PresentationCore* i *milcore* od kojih je jedino *milcore* napisana u neupravljanom kodu kako bi omogućila blisku integraciju s *DirectX*-om. Cijeli sustav prikaza u *WPF*-u se događa koristeći *DirectX*. Taj dio arhitekture je jako osjetljiv po pitanju performansi pa su odbačene brojne prednosti *CLR*-a kako bi se dobilo na brzini izvođenja.

Sigurnost

- WPF aplikacije dijelimo na dva osnovna tipa s obzirom na razinu sigurnosti:
  - *Full trust* - u ovu grupu spadaju samostalne aplikacije. One se instaliraju na korisničko računalo pomoću Microsoft Installer programa, te imaju potpuni pristup računalnim resursima kao i Microsoft Win32 aplikacije.
  - *Partial trust* - u ovu grupu spadaju pregledničke aplikacije, koje se prikazuju unutar preglednika koristeći *Internet Zone* dozvole. Partial trust aplikacije imaju ograničen pristup računalnim resursima.

Windows Presentation Foundation

- WPF se sastoji od mehanizma prikaza (*display engina*) i proširivog skupa upravljanih klasa.
- WPF uvodi novi markup jezik XAML (eXtensible Application Markup Language) koji je varijanta XML-a i koji omogućava korištenje deklarativnog XML modela za specificiranje korisničkog sučelja.
- To bi trebalo ubrzati razvoj korisničkog sučelja i olakšati suradnju između dizajnera i programera.
- I prikaz i funkcionalnost mogu se definirati XAML-om, ali također je moguće sve kodirati i u C# (ili nekom drugom .NET jeziku).
- XAML kôd će se kompajlirati u IL kôd isto kao što se i C# kôd kompajlira u IL kôd.

XAML (eXtensible Application Markup Language)

- XAML je izgrađen sa svrhom bržeg razvoja korisničkog sučelja i bolje suradnje dizajnera i programera, jer omogućava odvajanje definicije korisničkog sučelja od logike.
- XAML omogućava programeru da specificira hijerarhiju CLR objekata sa pripadajućim skupom svojstava pri čemu XAML tag odgovara instanci objekta, a XAML atribut odgovara svojstvima objekta.

  public class Window {public string Title }

XAML

- Razlikujemo 2 osnovna pristupa pisanju WPF aplikacija:
  - koristeći XAML, za definiranje korisničkog sučelja, i proceduralnog kôda za obradu događaja (*Event Handling*). Proceduralni dio aplikacije može biti napisan u C# ili Visual Basic.NET jeziku, te biti spremljen u *code-behind* datoteku (*Code-behind* datoteka sadrži implementaciju logike, tj. funkcionalnosti realizirane aplikacije.) ili biti ugrađen direktno u XAML.
  - koristeći samo proceduralni kôd – u ovom slučaju prikladan je bilo koji jezik koji podržava .NET *Common Language Specification* (CLS).

XAML

- Elementi se spremaju unutar .xaml izvorne datoteke koristeći XAML tagove.
- Svaki tag je povezan sa pripadajućom WPF klasom. Npr. element <Button> odgovara objektu klase Button.
- Sve što se može napisati deklarativno preko XAML elemenata, može se obraditi i proceduralno.
- XAML je varijanta XML-a. Prema tome, i u XAML-u vrijede pravila XML-a.

XAML

- Svaka datoteka može imati samo jedan root element.
- Root element može imati proizvoljan broj djece.
- Elementi se predstavljaju tagovima. Svaki tag koji je otvoren, mora biti i zatvoren.
- Tagovi su *case-sensitive*.
- XML, pa time i XAML, ne dopuštaju nepravilno ugnježđivanje.
- Svaki element može imati proizvoljan broj child elemenata.
- Vrijednosti atributa moraju biti zapisani u navodnicima.
- Razmaci se ne skraćuju. Kako je rečenica napisana, tako će biti i prikazana.
- Svaki element, između taga otvaranja i taga zatvaranja, može sadržavati neki drugi element, obični tekst, mješavinu jednog i drugog ili ništa.

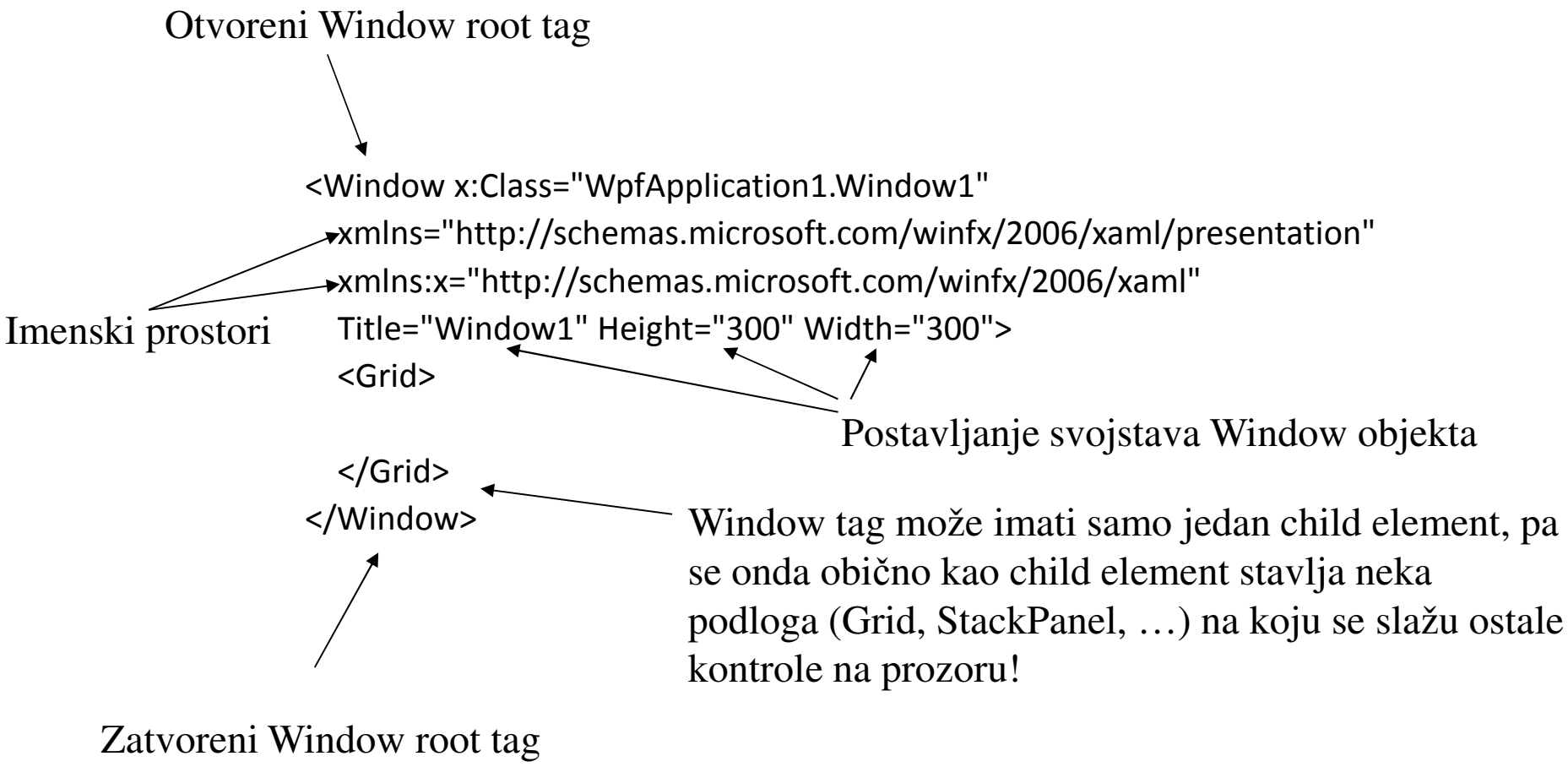
XAML

- XAML obično kao root element koristi <Window>, <Page>, <ResourceDictionary> ili <Application> tag.
- Elementi <Window> i <Application> moraju biti root elementi XAML datoteke.
- Unutar root taga nalaze se dvije deklaracije imenskog prostora.
- Atribut *xmlns* služi u iste svrhe kao i *using* direktive u C# jeziku.

XAML

- Prva deklaracija mapira sveukupni WPF imenski prostor kao osnovni. Druga deklaracija mapira odvojeni XAML imenski prostor u x: prefiks.
- Defaultni imenski prostor za WPF "http://schemas.microsoft.com/winfx/2006/xaml/presentation" odnosno za XAML http://schemas.microsoft.com/winfx/2006/xaml.
- Atribut x:Class sadrži ime klase iz proceduralnog kôda koja je povezana sa promatranom XAML datotekom.

Primjer XAML datoteke



XAML

- XAML root element može biti i podloga (podloge su kontrole koje služe isključivo za smještanje drugih kontrola) iz *System.Windows.Controls* imenskog prostora na kojoj će se stavljati ostali elementi (npr. klasa *Panel*).
- Panel
  - Canvas
  - DockPanel
  - Grid
  - StackPanel
  - WrapPanel
- Podloge nisu ograničene na postojanje kao root element.

Primjer XAML datoteke

```
<Window x:Class="WindowsApplication2.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="WindowsApplication2" Height="200" Width="300">
<StackPanel>
    <TextBox Name="MyTextBox" Width="200" Height="30"
    Margin="40" />
    <Button Click="ButtonClickHandler" Width="60“ Height="20">
                    Click me!
    </Button>
</StackPanel>
</Window>
```

Identifikatori XAML datoteke

- U XAML kôdu nije potrebno dodavati identifikator objektu (tj. XAML tagu) ukoliko se objekt koji predstavlja taj XAML tag neće pozivati negdje u kodu. Stoga je XAML kôd kraći nego proceduralni kôd.
- XAML tagovim se identifikator tj. ime objekta dodjeljuje preko atributa Name.
- Npr. sa prethodnog slajda tekst polje je imenovano (TextBox Name="MyTextBox") jer ćemo ga koristiti poslije u kodu. Ostali XAML elementi nemaju identifikatore jer se nigdje u proceduralnom kodu ne pozivaju. Npr. element botun nema identifikator.

Primjer *code-behind* datoteke za prethodnu XAML datoteku

```
void ButtonClickHandler(object sender, RoutedEventArgs e)
{
   MessageBox.Show(string.Format("You typed '{0}' in
  TextBox!", MyTextBox.Text), "XAML");
}
```

- Botun u XAML datoteci ima atribut Click postavljen na *event handler* tako da se u kodu može dodati *event handler* na događaj klik na botunu (Button Click="ButtonClickHandler"). Primjer

Umetnuti model

- XAML i *code-behind* datoteka mogu se spojiti u jednu datoteku korištenjem umetnutog modela (*inline model*).

```
<Window x:Class="IntroductionToXAML.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Introduction To XAML“
    Height="200" Width="300">
    <StackPanel>
   <TextBox Name="MyTextBox" Width="200" Height="30" Margin="40" />
 <Button Click="ButtonClickHandler" Width="60"  Height="20">Click me! </Button>
</StackPanel>
<x:Code>
 <![CDATA[
void ButtonClickHandler(object sender, RoutedEventArgs e)
{
MessageBox.Show(string.Format ("You typed '{0}' in TextBox!", MyTextBox.Text), "XAML");
}
]]>
 </x:Code>
</Window>
```

- Element <x:Code /> mora biti direktno dijete root elementa u XAML datoteci. Primjer

Aplikacijski objekt

- Svaka WPF aplikacija povezana je sa objektom *Application*.
- Objekt je dostupan svim stranicama aplikacije i postoji dok god je aplikacija aktivna.
- Klasa *Application* upravlja životnim ciklusom aplikacije (pokretanjem aplikacije, zaustavljanjem, …), prenošenjem parametara iz komandne linije, upravljanjem prozorima aplikacije i navigacijom unutar aplikacije.

Aplikacijski objekt

- U aplikaciji može biti kreirana samo jedna instanca objekta *Application*.
- Aplikacijski objekt se može kreirati korištenjem XAML i code-behind datoteke ili samo sa XAML-om ili korištenjem samo proceduralnog kôda.
- Aplikacijski objekt mora biti root XAML element.

Aplikacijski objekt

- Obično se unutar XAML *Application* elementa atribut *StartupUri* postavlja na ime početnog prozora definiranog XAML datotekom koja se navodi kao vrijednosti atributa.

```
<Application
x:Class="WpfApplication1.App"
xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   StartupUri="Window1.xaml">

   <Application.Resources>

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

Aplikacijski objekt

- Može se koristiti i drugačiji pristup.
- Unutar XAML datoteke može se definirati objekt *Application* te se njegov *Startup* događaj veže uz *event handler* definiran unutar *code-behind* datoteke.

```
//XAML datoteka
<Application x:Class="WpfApplication1.App"
        xmlns=http://schemas.microsoft.com/winfx/2006/xaml/
        presentation
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
        ml"
   Startup="otvori">
   <Application.Resources>
  </Application.Resources>
</Application>

//Code-behind datoteka
public partial class App : Application
  {
     public void otvori(object prvi, EventArgs drugi)
     {
        Window1 prozor = new Window1();
        prozor.Show();
     }
  }
```

Primjer

Aplikacijski objekt

```
using System;
using System.Windows;
namespace CSharp
{
    public class EntryPoint
      {
        // All WPF applications should execute on a single-threaded apartment (STA) thread
        [STAThread]
        public static void Main()
        {
        Application app = new Application();
        app.Run(new Window());
        }
      }
}
```

korištenjem samo proceduralnog kôda

Aplikacijski objekt i prozori aplikacije

- Temeljna klasa za kreiranje prozora .NET 3.x aplikacije je klasa System.Windows.Window i System.Windows. Navigation.NavigationWindow.
- Glavni prozor aplikacije obavezno mora biti klasa Window ili klasa izvedena iz klase Window.
- Međutim *StartupUri* atribut može poprimati pored Windows objekta i druge vrijednosti.

Type	Window	Application type
Window	Window	Standalone only
NavigationWindow	NavigationWindow	Standalone only
Page	NavigationWindow	Standalone/browser-hosted
UserControl	NavigationWindow	Standalone/browser-hosted
FlowDocument	NavigationWindow	Standalone/browser-hosted
PageFunction(T)	NavigationWindow	Standalone/browser-hosted

Aplikacijski objekt i prozori aplikacije

- Aplikacija tada automatski stvara prozor tipa Window ili NavigationWindow i učitava specificiranu datoteku.

```
<Application
    x:Class="startupuri_page.App"
    xmlns="http://schemas.microsoft
    .com/winfx/2006/xaml/presentat
    ion"
    xmlns:x="http://schemas.microso
    ft.com/winfx/2006/xaml"
   StartupUri="Page1.xaml">
   <Application.Resources>

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

Primjer

Klasa Page

- Klasa Page nije prozor i nije izvedena iz osnovne klase prozora Window stoga se kao i sve druge kontrole mora smještati na prozor.
- Međutim klasa Application, koja predstavlja aplikaciju, ukoliko se kao početni URI aplikacije u atributu StartupUri navede Page element umjesto Window elementa, sama instancira klasu prozora na koji se smješta stranica navedena u atributu StartupUri.

Klasa NavigationWindow

- Klasa NavigationWindow (System.Windows.Navigation) je izvedena iz klase Window te se može koristiti za glavni prozor aplikacije.
- Klasa se koristi kao navigator, sa implementiranom poviješću navigacije.

XAML elementi

- Svaki XAML element predstavlja .NET CLR klasu.
- Većina klasa je naslijeđena iz klasa System.Windows.UIElement, System.Windows.FrameworkElement, System.Windows.FrameworkContentElement i System.Windows.ContentElement.
- Postoje sljedeći osnovni tipovi XAML elemenata:
  1. Korijenski (Root) elementi: Obično su to Windows ili Page element. To su elementi koji se mogu koristiti isključivo kao korijenski elementi XAML datoteke.
  2. Podloge (Panel) elementi: StackPanel, DockPanel, Grid i Canvas služe u prvom redu da se na njima slažu drugi elementi.

XAML elementi

3. Kontrolni elementi: Ovi elementi sadrže različite tipove kontrola u XAML-u koje se mogu smjestiti na sučelje.
4. Geometrijski elementi: Omogućavaju crtanje oblika. (LineGeometry, EllipseGeometry, PathGeometry, LineSegment,….)
5. Dokument elementi: Ovi elementi se koriste za prezentaciju dokumenata. Umetnuti (*Inline*) i blok (*Block*) elementi čine dvije osnovne grupe dokument elemenata. (Inline elementi - Bold, LineBreak, Italic; Block elementi - Paragraph, List, Block, Figure, Table)

XAML atributi

- XAML atributi odgovaraju svojstvima .NET klase. Mogu se definirati eksplicitno

```
<Button Margin="5">Click me!</Button>
```

- ili mogu biti umetnuti u XAML element.

```
<Button><Button.Margin>5</Button.Margin>Click me!</Button>
```

Vlastite klase u XAML-u

- Moguće je koristiti vlastite klase u XAML.
- Postupak je sljedeći:
  - Klasa obavezno mora imati public konstruktor.
  - Klasa ne smije biti ugniježđena.
  - U XAML datoteku u kojoj će se koristiti nova klasa treba se dodati imenski prostor koji sadrži novu klasu.

Vlastite klase u XAML-u

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Input;
namespace vlastite_klase
{
   class Class1:  System.Windows.Controls.Button
   {
      public Class1()
      {       }
      protected override void OnMouseDown(MouseButtonEventArgs e)
      {
         MessageBox.Show("aaaaa");
         base.OnMouseDown(e);
      }
      public double visina
      {
         get { return base.Height; }
             set { base.Height = value; }
      }
      public double sirina
      {
         get { return base.Width; }
         set { base.Width = value; }
      }
   }
}
```

Vlastite klase u XAML-u

```
<Window x:Class="vlastite_klase.Window1“
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/present
    ation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    Title="vlastite_klase" Height="300" Width="300"
    xmlns:local="clr-namespace:vlastite_klase">
    <Grid>
      <local:Class1 sirina="100" visina="100">Moj
    Botun</local:Class1>
    </Grid>
</Window>
```

Primjer

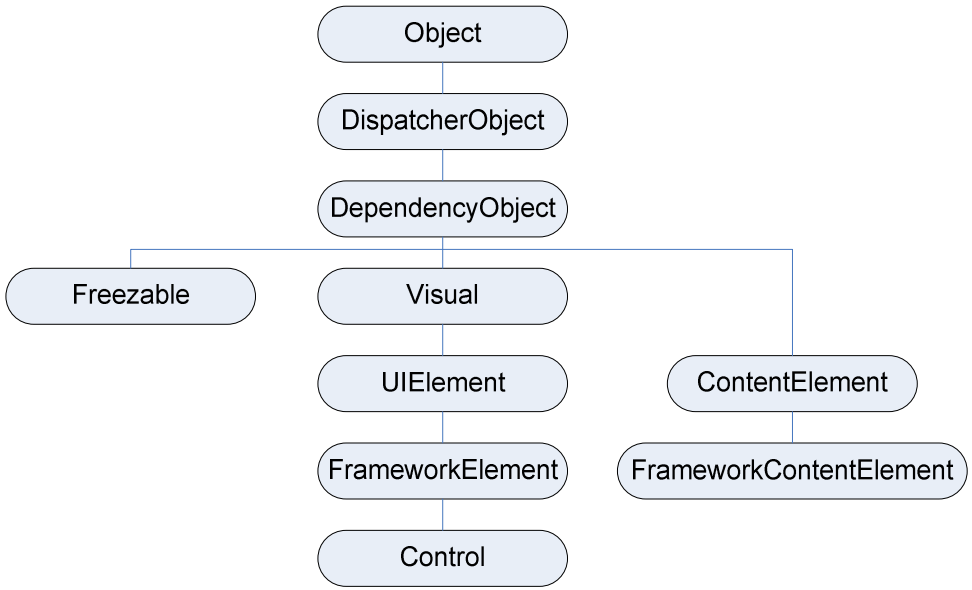
Vlastite klase u XAML-u

- Imenski prostor na prethodnom slajdu je nazvan xmlns:local, ali može se zvati i drugačije.
- Možete definirati niz različitih imenskih prostora koristeći različiti prefiks uz xmlns.
- Vrijednost imenskog prostora sadrži ime imenskog prostora (vlastite_klase) zajedno sa oznakom clr-namespace.
- Podržana je sljedeća sintaksa vrijednosti imenskog prostora:
  - clr-namespace: CLR (*common language runtime*) imenski prostor deklariran unutar .NET skupa (assembly). Vrijednost se odvaja karakterom (:)
  - assembly= .NET skup koji sadrži čitav ili samo dio referenciranog CLR imenskog prostora. Obično se navodi samo ime skupa bez puta do skupa. Put treba biti definiran kroz reference projekta. Vrijednost se odvaja karakterom (=).

Vlastite klase u XAML-u

- Ove dvije vrijednosti se odvajaju sa karakterom (;).
- Npr.: xmlns:custom="clr-namespace:SDKSample;assembly=SDKSampleLibrary"
- Ime .NET skupa se ne mora navoditi ukoliko se radi o istom skupu ili se može staviti prazna vrijednost.
- Npr.: xmlns:local="clr-namespace:vlastite_klase;assembly=“

.NET 3.x hijerarhija klasa



- Osnovna klasa svih objekata koji imaju grafičku reprezentaciju odnosno vizualnu reprezentaciju ili iscrtavanje je klasa Visual. To je temeljna WPF klasa za iscrtavanje.

.NET 3.x hijerarhija klasa

- Klasa UIElement je temeljna klasa za vizualne objekte koja pruža funkcionalnost prosljeđivanja događaja objektu, poravnavanje grafičkih objekata, fokus elemenata i sl.
- Temeljna klasa za vizualne objekte koji nemaju vlastito iscrtavanje već se smještaju na drugi objekt temeljen na klasi Visual za iscrtavanje, slična UIElement klasi je klasa ContentElement.
- Klasa FrameworkElement je temeljna klasa koja pruža podršku za stilove prozora, resurse, povezivanje sa izvorima podataka i mehanizme poput *tooltipova* i *context* izbornika.
- Klasa FrameworkContentElement je analogna FrameworkElement klasi, ali sa podrškom za sadržaj smješten unutar prozora tj. u klijentskom području prozora.

Glavne klase WPF-a

- Osnovni princip dizajn WPF klasa kojem se težilo jeste korištenje osobina (*properties*) što je više moguće (umjesto metoda i ključnih događaja (*events*)) da bi se kontroliralo ponašanja objekta.
- Potrebna je izmjena klasičnog pristupa osobinama gdje one označuju nefunkcionalno ponašanje kao boju i veličinu.
- Postojala je potreba da se omogući povezivanje osobina raznih objekata, tako da se osobine jednog objekta mogu mijenjati pomoću nekog drugog objekta.

Glavne klase WPF-a

- Ovdje dolazi klasa *DependencyObject*. Sustav svojstava postaje sustav zavisnih svojstava tek onda kad prati zavisnosti između pojedinih svojstava, te kad se pojavi promjena na nekom od svojstava automatski osvježava vrijednosti povezanih svojstava.
- Također, sustav svojstava je definiran na način da svaki objekt ne mora biti spremljen u cijelosti (tj. sva njegova svojstva) jer većinu svojstava ne iskoristimo te ona sadržavaju predodređenu vrijednost.
- Ovime se omogućuje da se mijenjanjem svojstava *parent* kontroli automatski mijenjaju svojstva (tj. osobine) svim *child* kontrolama koje ta kontrola posjeduje.

Zavisna svojstva

- Atribut nasljeđuje svojstvo elementa roditelja ukoliko element roditelj ima postavljen taj atribut i ukoliko je taj atribut sukladan atributu child elementa. Ako se postavi onda se *overrajda* vrijednost svojstva nasljeđena od elementa roditelja.

```
<Button Width="100" Height="40">
<Button.Background>Yellow
</Button.Background>
<TextBlock>
<TextBlock.Background>Red
</TextBlock.Background>
Ovo je neki tekst
</TextBlock>
</Button>
```

Primjer

Glavne klase WPF-a

- Svaki element koji se pojavljuje u *WPF-u* je, u srcu, *Visual*.
- Klasa *Visual* se može zamisliti kao objekt za crtanje, koji sadrži upute za crtanje tj. dodatne detalje kako bi se to crtanje trebalo izvršiti, te neke osnovne funkcionalnosti (kao *hit testing*).
- *Visual* klasa je dizajnirana na način da bude lagana i fleksibilna pa većina elemenata sučelja nije izložena za korištenje.
- Ova klasa se nalazi na granici između dvaju podsustava, upravljanog programskog sučelja i neupravljanog *milcore* podsustava.

Glavne klase WPF-a

- *WPF* prikaz se stvara prevođenjem elemenata prikaza u neupravljane strukture kojima upravlja *milcore.*
- Te se strukture zovu sastavni čvorovi (*composition nodes*), te tvore stablo prikaza (ili vizualizacijsko stablo), a dostupne su samo preko posebnog protokola kojima se prenose poruke.
- Elementi tipa *Visual* mogu stvoriti jedan, nijedan ili više sastavnih čvorova.
- Važno je napomenuti kako je cijelo stablo prikaza smješteno u *cache* memoriju. To omogućuje sustavu da jako brzo iscrtava prikaz na ekranu te sprječava pojavu aplikacija koje ne reagiraju.

Glavne klase WPF-a

- Kod *User32* i *GDI* sustava, svaka komponenta bi dobila granice u kojima treba biti iscrtana.
- Najčešće se te granice ne bi podudarale s granicama same komponente zbog preklapanja s drugim komponentama na ekranu.
- Ovakav sustav štedi memoriju u smislu da nikad dvije komponente ne utječu na boju pojedinog piksela pa nema niti potrebe ponovno iscrtavati druge komponente.

Glavne klase WPF-a

- *WPF* koristi „*painter's algorithm*“ model.
- Za razliku od prijašnjih modela prikaza, ovaj model iscrtava sve komponente na ekranu, počevši od zadnje prema prvoj.
- Ovakav model je sporiji, ali omogućuje stvaranje kompleksnih oblika na ekranu.
- Međutim, koristeći mogućnosti danas dostupnog hardvera, usporenje u prikazu se ne primjećuje.

Glavne klase WPF-a

- *UIElement* je srce tri glavna podsustava u *WPF-u* – *Layout*, *Input* i *Events.*
- *Layout* je glavni koncept *WPF-a*, i *UIElement* uvodi novi koncept koji izvodi raspoređivanje i iscrtavanje aplikacija na potpuno drugi način od dosadašnjeg.
- Taj koncept se izvodi kroz dva odvojena koraka, *measure* i *arrange*.
- Korak *measure* omogućuje elementu da odredi koliko mu stvarno prostora treba.
- Ovo pruža aplikacijama mogućnost da automatski odrede veličinu područja potrebnog za određeni element i dodjele onoliko prostora koliko mogu da daju s obzirom na veličinu aplikacije.

Glavne klase WPF-a

- Kada korak *measure* rezultira u promjeni željene veličine kontrole, korak *arrange* može (ako je potrebno) promijeniti način kako su kontrole raspoređene.
- Što se tiče ulaza koja proizlazi od signala određenog *device driver-a* prosljeđuje se kroz *kernel* i *User32* (slično kao u prijašnjim verzijama) prema *WPF-u* i korisničkoj aplikaciji.
- Na ovoj razini, taj signal se pretvara u *WPF* poruku koja je poslana otpremniku. Tu se ona može razložiti na više ključnih događaja. Ovi *event-i* mogu biti ''uhvaćeni'' u *UIElement* sloju i proslijeđeni do kontrola gdje programer može odlučiti što sa njima.

Glavne klase WPF-a

- *FrameworkElement* uvodi skup pravila i prilagodbi na podsustavima te uvodi neke nove podsustave.
- Primarno pravilo koje uvodi tiče se *layout* koncepta te upotrebom svojstava *HorizontalAligment*, *VerticalAligment*, *MinWidth*, *Margin* itd. sve komponente koje nasljeđuju *FrameworkElement* imaju dosljedno ponašanje unutar *layout* spremnika.
- Također, *FrameworkElement* osigurava pristup nekim funkcijama *API*-a koje se nalaze na nižim slojevima *WPF*-a. Primjer za to je metoda *BeginStoryboard* koja pruža direktan pristup upravljanju animacijama.

Glavne klase WPF-a

- Dva najvažnija koncepta kojima se bavi *FrameworkElement* su povezivanje s izvorima podataka i stilovi. Povezivanje s izvorima podataka je prilično intuitivno te je vrlo lako povezati svojstvo nekog elementa s podacima.
- Jedna od najinteresantnijih mogućnosti *WPF*-a je stvaranje predložaka prikaza podataka (*data template*). Umjesto da stvaramo sučelje koje prilagođavamo podacima, možemo na osnovu tipa podatka zadati način prikaza.
- Korištenjem stilova se određena svojstva vezuju za jednu ili više instanci elementa.

Glavne klase WPF-a

- Kontrola je element pomoću kojeg se može vršiti interakcija sa korisnikom.
- Klasa *Control* sadrži dodatna svojstva za izgled, ali najzanimljiviji je dio da sadrži podršku za predloške, što omogućuje da se zamjeni standardni izgled kontrole sa vlastitim izgledom.