Windows Presentation Foundation

Zbyszek Skowron

20 listopada 2006

Windows Presentation Foundation

Windows Presentation Foundation jest nową biblioteką Microsoftu do budowania interfejsów użytkownika. Jest przeznaczona na platformę .NET 3.0 i jest wbudowana w system Windows Vista. Jest także dostępna dla Windowsa XP.
Najważniejsze własności:

- Deklaratywne budowanie interfejsów przy pomocy języka XAML (Zammel).
- Dwa modele aplikacji: wolnostojąca i uruchamiana w przeglądarce.
- Wykorzystanie sprzętowej akceleracji grafiki.
- Wsparcie dla dwóch rodzajów dokumentów: Fixed i Flow.
- Konfigurowalność i rozszerzalność.

Możliwości

Themes	wsparcie dla skórek.
Styles	zmiana wyglądu kontrolek.
Control Templates	zmiana wewnętrznej struktury kontrolek.
Data Templates	szablony wizualizacji danych.
Data Binding	powiązania kontrolek z danymi.
Events	złożony system propagacji zdarzeń.
Commanding	powiązania zdarzenie→komenda→akcja.
Properties	zaawansowany system właściwości (z dziedziczeniem wartości itp.).

Dwa modele aplikacji

W WPF występują dwa modele aplikacji:

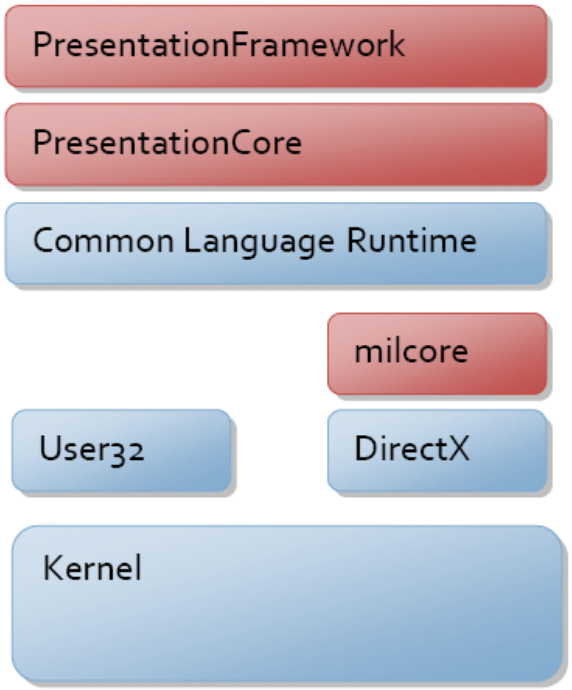
- zwykła aplikacja "okienkowa",
- XAML Browser Application - XBAP (exe-bap).

Aplikacja XBAP składa się ze stron napisanych w XAML'u, które są wyświetlane w oknie przeglądarki. Aplikacja XBAP jest:

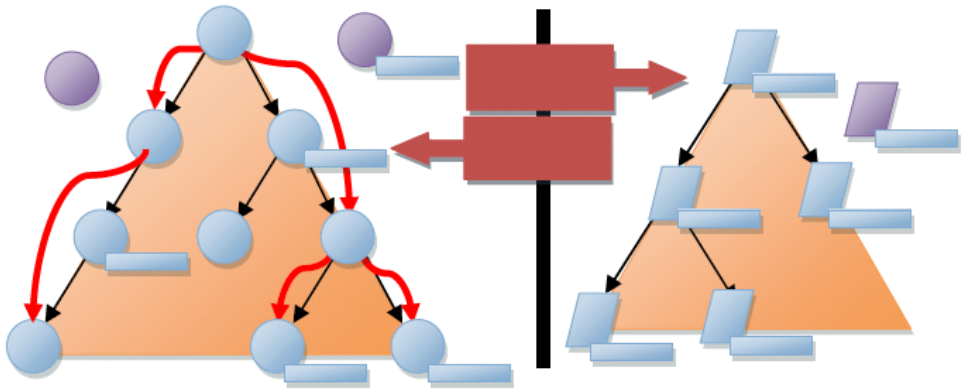
- zintegrowana z przeglądarką,
- wspiera nawigowanie przy pomocy przycisków Back/Forward,
- jest uruchamiana z ograniczonymi uprawnieniami.

Architektura WPF

Wszstkie funkcje graficzne WPF są wykonywane przy pomocy bibliotek DirectX.
Kod za to odpowiedzialny, milcore, to jedyna część WPF uruchamiana poza maszyną .NET.
milcore odpowiada też za niskopoziomowe zarządanie pamięcią i inne krytyczne dla prędkości wykonania elementy.



Drzewa obiektów w WPF



Kontrolki w WPF trzymane są w dwóch drzewach:

- logicznym,
- i wizualnym.

Drzewa te nie są izomorficzne.

Drzewo logiczne jest używane np. przy propagacji zdarzeń.

Drzewo wizualne jest używane do renderowania kontrolek.
Każda kontrolka tworzy pewną liczbę węzłów wizualnych.

Extensible Application Markup Language

W WPF cały wygląd interfejsu użytkownika może być zapisany deklaratywnie.
Pozwala to oddzielić warstwę prezentacji od zachowania.

Do opisu wyglądu aplikacji używany jest oparty na XML'u język XAML.

Elementy w plikach XAML tłumaczą się bezpośrednio na instancjacje obiektów.

Zwykle korzeniem pliku XAML jest element `<Window>`, `<Page>` lub `<Application>`.

Mapowanie XAML'a na drzewo obiektów

Mapowanie XAML'a:

- element $\rightarrow$ obiekt,
- atrybut $\rightarrow$ właściwości i komunikaty,
- przestrzeń nazw XML $\rightarrow$ przestrzeń nazw CLR,

XAML

```
<StackPanel>
  <Button Content="Click Me"/>
</StackPanel>
```

C#

```
StackPanel stackPanel =
     new StackPanel();
Button button = new Button();

button.Content = "Click Me";

stackPanel.Children.Add(button);
```

Ustawianie właściwości w XAML'u

Składnia atrybutowa

```
<Button Background="Blue" Content="This is a button"/>
```

Składnia elementowa

```
<Button>
  <Button.Background>
    <SolidColorBrush Color="Blue"/>
  </Button.Background>
  <Button.Content>
    This is a button
  </Button.Content>
</Button>
```

Jedyna równica polega na tym, że elementy bezpośrednio tworzą obiekty, a atrybuty rzutują swoją zawartość ze stringa na odpowiedni typ.

Wyrażenia wewnątrz atrybutów

Przy pomocy rzutowania ze stringa nie da się zrobić odwołania do istniejącego obiektu.
Z pomocą przychodzą wyrażenia, które są zapisywane w nawiasach klamrowych: {, }.

Przykład:

```
<Page.Resources>
  <Style TargetType="Border" x:Key="PageBackground">
    <Setter Property="Background" Value="Blue"/>
  </Style>
</Page.Resources>

<StackPanel>
  <Border Style="{StaticResource PageBackground}" />
</StackPanel>
```

Łączenie XAML'a z kodem

Aby powiązać element XAML'a z konkretną klasą należy w korzeniu dokumentu opisującego stronę, okno lub aplikację dodać atrybut `x:Class`:

XAML code-behind:

```
<Page xmlns="http://.../xaml/presentation"
      xmlns:x="http://.../xaml"
      x:Class="MyNamespace.MyPageCode">
  <Button Click="ClickHandler" >Click Me!</Button>
</Page>
namespace MyNamespace {
 public partial class MyPageCode {
  void ClickHandler(object sender, RoutedEventArgs e)
  {
     ((Button)e.Source).Background = Brushes.Red;
  } } }
```

Zdarzenia w WPF

Zdarzenia w WPF mogą pochodzić z klawiatury, myszy, od kontrolek itp.

Zdarzenia mają trzy własności:

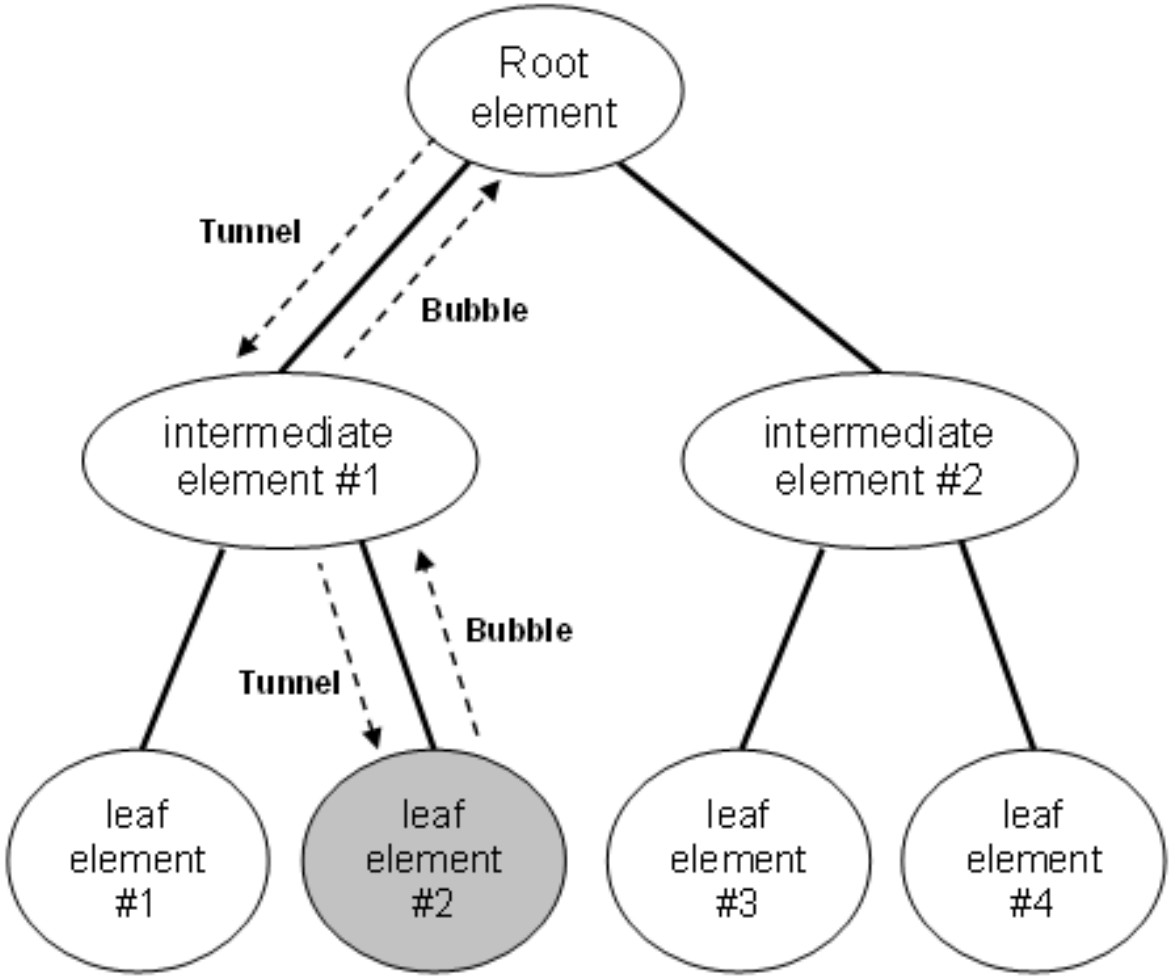
- source - obiekt który wywołał zdarzenie,
- target - obiekt do którego wysłano zdarzenie,
- eventArgs - parametry zdarzenia.

Zdarzenia mogą być propagowane na trzy sposoby:

- direct - bezpośredno do odbiorcy,
- tunneling - zdarzenie idzie od korzenia drzewa do odbiorcy,
- bubbling - zdarzenie idzie od odbiorcy w kierunku korzenia.

Propagacja zdarzeń

Zdarzenia propagowane są po drzewie logicznym, aż znajdą obiekt, który je obsłuży. Propagacja zdarzenia jest przerywana, poprzez ustawienie mu flagi Handled.



Propagacja zdarzeń

Większość zdarzeń jest faktycznie złożona z dwóch zdarzeń, kolejno propagowanych po drzewie logicznym.

Przykładowo, po naciśnięciu klawisza wywoływane są kolejno zdarzenia: PreviewKeyDown i KeyDown.

Zdarzenia Preview... są wywoływane najpierw, metodą tunnel. Zdarzenia właściwe są propagowane później metodą bubble.

Takie rozwiązanie jest bardzo elastyczne. Pozwala:

- filtrować zdarzenia schodzące do kontrolek-dzieci,
- obsługiwać zdarzenia dzieci przez ojców.

Przykład: zdarzenie obsługiwane przez ojca

```
<StackPanel Button.Click="CommonClickHandler">
  <Button Name="YesButton">Yes</Button>
  <Button Name="NoButton">No</Button>
</StackPanel>

private void CommonClickHandler(object sender, RoutedEventArgs e)
{
  FrameworkElement feSource = e.Source as FrameworkElement;
  switch (feSource.Name)
  {
    case "YesButton":
      // do something here ...
      break;
    case "NoButton":
      // do something ...
      break;
  }
  e.Handled = true;
}
```

Polecenia

Polecenia (Commands) dodają poziom abstrakcji pomiędzy zdarzeniami a akcjami.

Polecenia opisują pewną akcję, którą należy wykonać, np.:

- Copy,
- Paste,
- Open...

- Komenda może zostać zaaktywowana przez kontrolkę (np. MenuItem) lub programowo.
- Zaaktywowana komenda uruchamia zdarzenia PreviewExecuted i Executed, które normalnie są propagowane (domyślnie do kontrolki w ognisku).
- Inne kontrolki mogą odebrać te zdarzenia i wykonać akcję.

Przykład: Paste z klawiatury

```
<Window.InputBindings>
  <KeyBinding Key="P" Modifiers="Control"
      Command="ApplicationCommands.Paste" />
</Window.InputBindings>

<Window.CommandBindings>
  <CommandBinding Command="ApplicationCommands.Paste"
                  Executed="OnPaste"
                  CanExecute="IfCanPaste"/>
</Window.CommandBindings>
```

Standardowe polecenia: ApplicationCommands, NavigationCommands, MediaCommands, EditingCommands i ComponentCommands. Te klasy definiują komendy takie jak: Cut, BrowseBack, BrowseForward, Play, Stop...

Przykład: Paste z menu

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Paste" />
  </Menu>
  <TextBox />
</StackPanel>
```

Gdy kontrolka mogąca odebrać dane polecenia(tu TextBox) nie pozwala na to (CanExecute), to do kontrolki wysyłającej polecenie wysyłane jest zdarzenie CanExecuteChanged - pozwala to zaimplementować np. wyszarzanie nieaktywnych elementów menu.

Dispatcher

Aplikacje WPF od początku mają co najmniej dwa wątki:

- wątek interfejsu użytkownika,
- działający w tle wątek wyświetlający interfejs.

Do elementów UI można się odwoływać tylko z ich wątku.

Wszystkie zadania dla wątku UI są kolejkowane przez obiekt Dispatcher.

Jego zadaniem jest sortować żądania według priorytetu i wywoływać je po kolei.

Aby odwołać się do elementów UI z innego wątku należy wpisać się do kolejki. Służą do tego metody Invoke() i BeginInvoke().

Wywoływanie metody w wątku UI

```
okno.Dispatcher.BeginInvoke(DispatcherPriority.Normal,
        new MojDelegat(UaktualnijUI) );
```

Zasoby

Każdy element XAML'a może trzymać kolekcję zasobów.
Zasoby to dowolne dane, umieszczane w kolekcji Resources elementu.

Można się do nich odwoływać z wyrażeń w XAML'u lub programowo.

Zasoby są uporządkowane hierarchicznie: gdy dany element nie ma jakiegoś zasobu, to szuka u ojca, dziadka, itd.

Zasobami mogą być np. dane dla aplikacji, style, szablony zawartości, szablony danych...

Style

Jednym z możliwych zasobów są style:

Styl zmieniający tło na niebieskie

```
<Page.Resources>
  <Style x:Key="PageBackground">
    <Setter Property="Background" Value="Blue"/>
  </Style>
</Page.Resources>

<StackPanel>
  <Border Style="{StaticResource PageBackground}" />
</StackPanel>
```

Style domyślne

Poprzedni przykład definiował styl nazwany (PageBackground).

Style mogą też być domyślnie przypisywane elementom danego typu:

Styl dla ramek

```
<Page.Resources>
  <Style TargetType="Border">
    <Setter Property="Background" Value="Blue"/>
  </Style>
</Page.Resources>

<StackPanel>
  <Border/>  //aplikowany styl domyślny
</StackPanel>
```

Możliwości styli

Style mogą ustawiać właściwości i akcje

```
<Setter Property="Background" Value="Blue"/>
<EventSetter Event="Click" Handler="OnClick"/>
```

Takie style nie są zbyt elastyczne.

Bardziej zaawansowane możliwości dają szablony kontrolek.
Szablony kontrolek pozwalają zmienić nie tylko kolory i akcje.
Pozwalają zmienić cały wygląd kontrolki, razem z jej sturukturą wewnętrzną.

Szablony kontrolek

Zupełnie przerobiony przycisk

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Ellipse Fill="{TemplateBinding Background}"/>
    <ContentPresenter/>  //wkłada "zawartość" kontrolki
  </Grid>
  <ControlTemplate.Triggers>
    <Trigger Property="IsPressed" Value="true">
      <Setter TargetName="Border" Property="Background" Value="{
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
...
<Button Template="{StaticResource ...}" Content="Content" />
```

Szablony danych

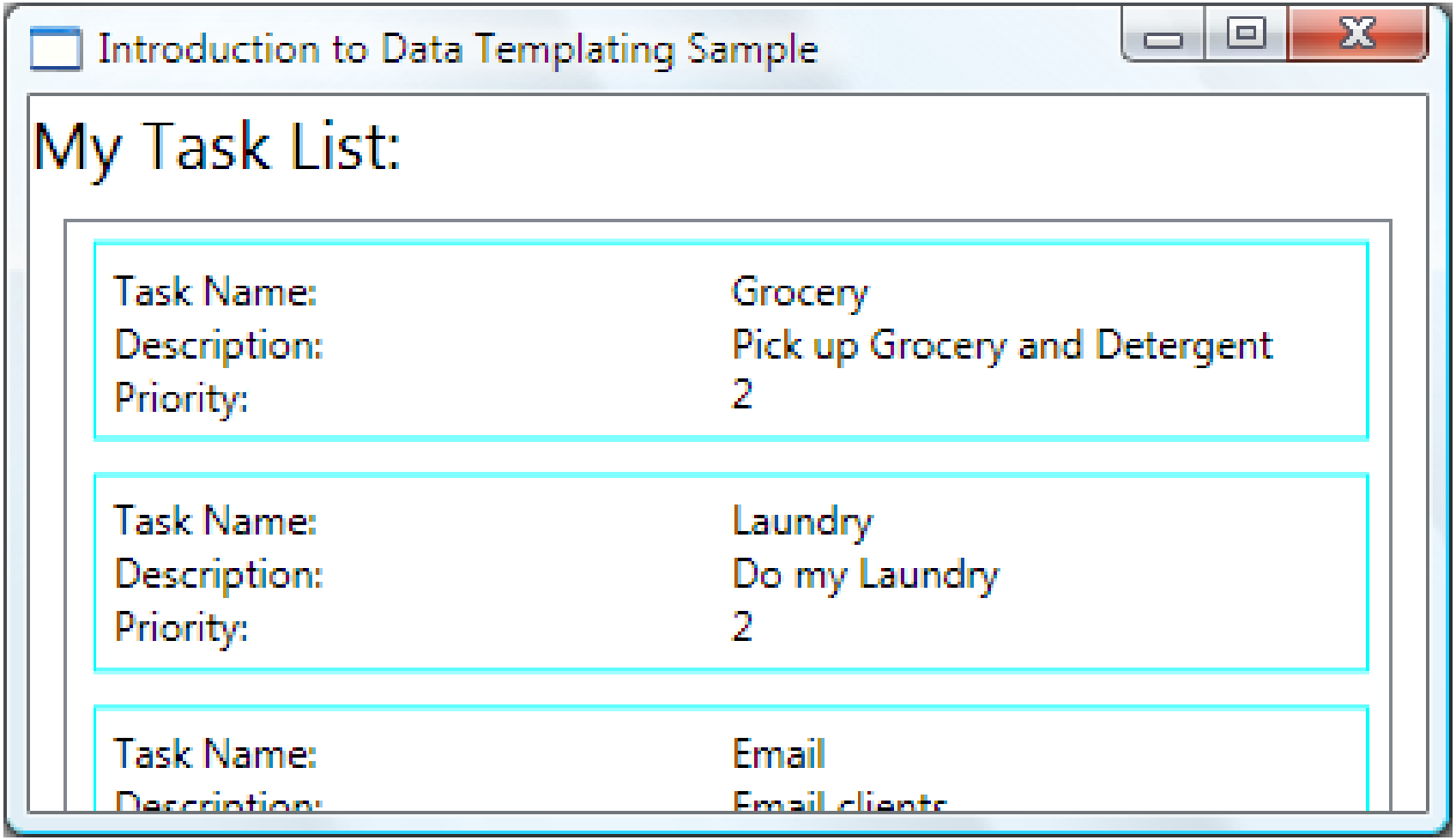
Szablony kontrolek dają ogromne możliwości: pozwalają zupełnie zmienić sposób, w jaki kontrolka się wyświetla.
Chciałoby się jeszcze zmienić sposób w jaki kontrolka wyświetla dane.
Do tego właśnie służą szablony danych.
Opisują one jak dane mają zostać przerobione na elementy wizualne.

String interpretowany jako rysunek

```
<DataTemplate DataType="{x:Type String}">
    <Border Margin="3">
      <Image Source="{Binding}"/>
    </Border>
</DataTemplate>
```

Teraz ListBox stringów zamiast tekstu będzie wyświetlał rysunki w ramce.

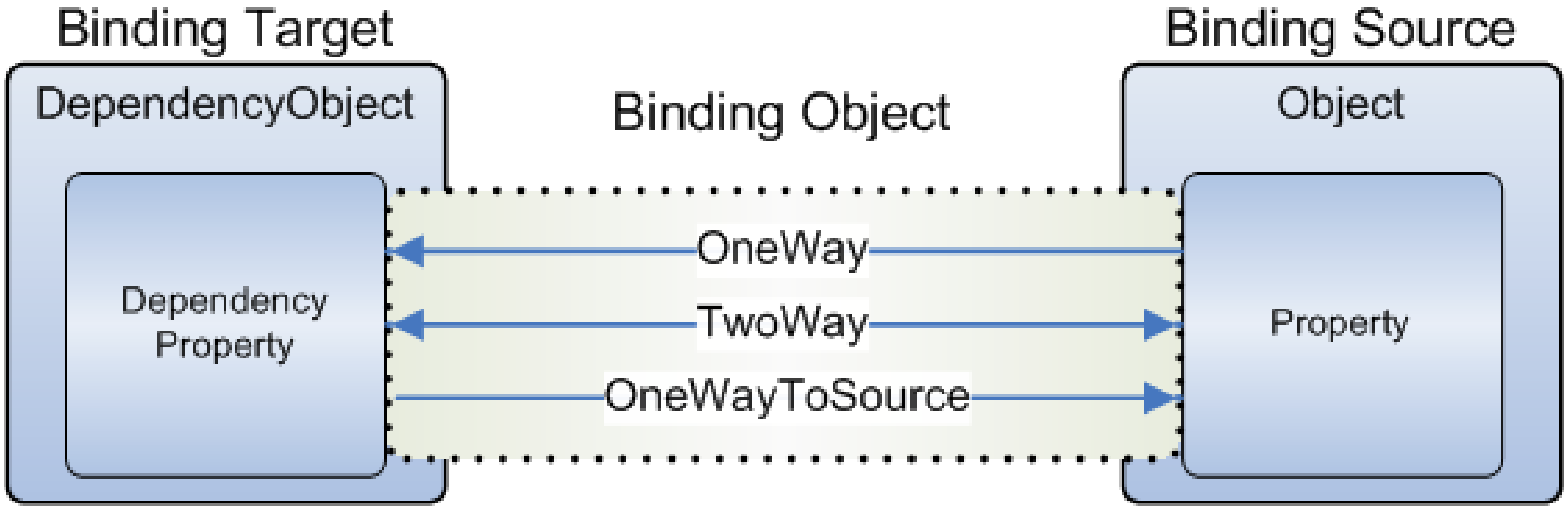
Przykład listy z zaaplikowanym szablonem danych



Dowiązania danych

Dane wyświetlane w kontrolkach mogą pochodzić z różnych źródeł:

- literalnie z atrybutu lub elementu Content,
- z zasobów,
- z właściwości obiektów.



Przykład dowiązania

Kolor przycisku brany z MyData.ColorName

```
<DockPanel.Resources>
    <c:MyData x:Key="myDataSource"/>
</DockPanel.Resources>
<Button Background="{Binding Source=
      {StaticResource myDataSource},
      Path=ColorName}" Content="Naciśnij" />
```

Każda kontrolka ma właściwość DataContext. Jest to domyślnie używane dowiązanie danych (gdy Binding nie ma parametru Source).

Lista wypełniana elementami z XmlDataProvidera

```
<Grid.Resources>
   <XmlDataProvider x:Key="xmlSource" XPath="Expenses">
     <x:XData>
       <Expenses xmlns="">
         <Person Name="Mike" e-mail="mikwake.com"/>
         <Person Name="Mary" e-mail="bellacut.com"/>
       </Expenses>
     </x:XData>
   </XmlDataProvider>

   <DataTemplate x:Key="dataTemp">
     <Hyperlink NavigateUri="{Binding XPath=e-mail}"
                Content="{Binding XPath=Name}" />
   </DataTemplate>
</Grid.Resources>
<ListBox ItemsSource="{Binding Source=
          {StaticResource xmlSource}, XPath=Person}"
         ItemTemplate="{StaticResource dataTemp}" />
```

Dokumenty w WPF

WPF obsługuje dwa rodzaje dokumentów:

- FixedDocument,
- i FlowDocument.

WPF, a więc i te dokumenty, obsługują czcionki OpenType.
Mają one m. in.:

- ligatury,
- alternatywne wersje liter.

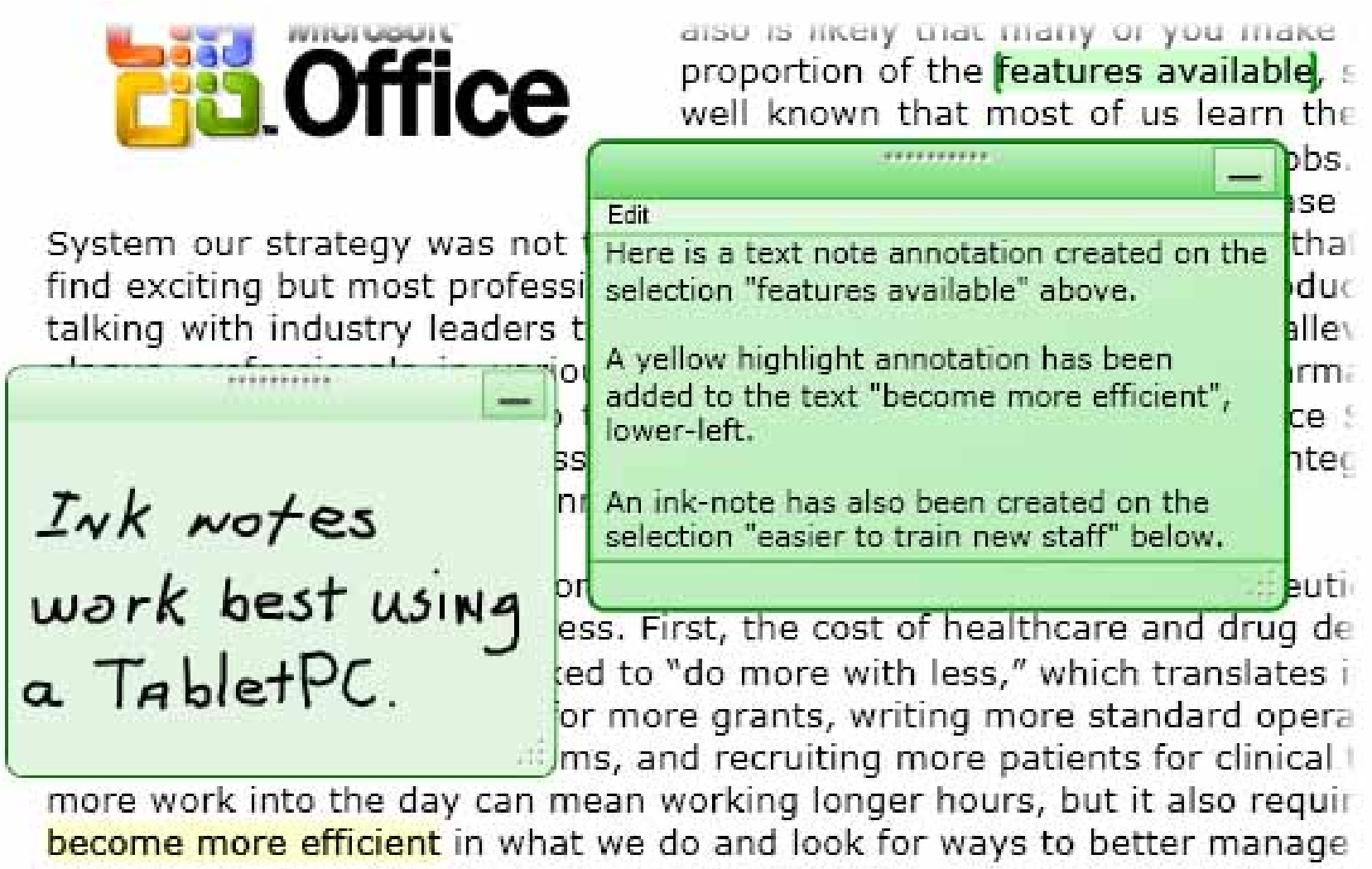
Fixed Document

FixedDocument jest dokumentem o ustalonym formatowaniu (jak np. PDF).
Formatem dla tych dokumentów jest XPSDocument (XAML Page Specification).
Jest to pakiet zawierający opis dokumentu w XAML'u i inne zasoby (np. rysunki), a całość spakowana jest ZIP'em.

Pliki w tym formacie można oglądać przy pomocy kontrolki DocumentViewer.

Kontrolka ta umożliwa dodawanie do dokumentu adnotacji i wyróżnień.

Ponadto systemie Windows Vista pliki do druku wysyłane są właśnie w formacie XPS.


Office
proportion of the features available,
well known that most of us learn the
System our strategy was not
find exciting but most professi
talking with industry leaders t
Edit
Here is a text note annotation created on the selection "features available" above.
A yellow highlight annotation has been added to the text "become more efficient", lower-left.
An ink-note has also been created on the selection "easier to train new staff" below.
Ink notes work best using a TabletPC.
ess. First, the cost of healthcare and drug de
ked to "do more with less," which translates i
or more grants, writing more standard opera
ms, and recruiting more patients for clinical
more work into the day can mean working longer hours, but it also requi
become more efficient in what we do and look for ways to better manage

Flow Document

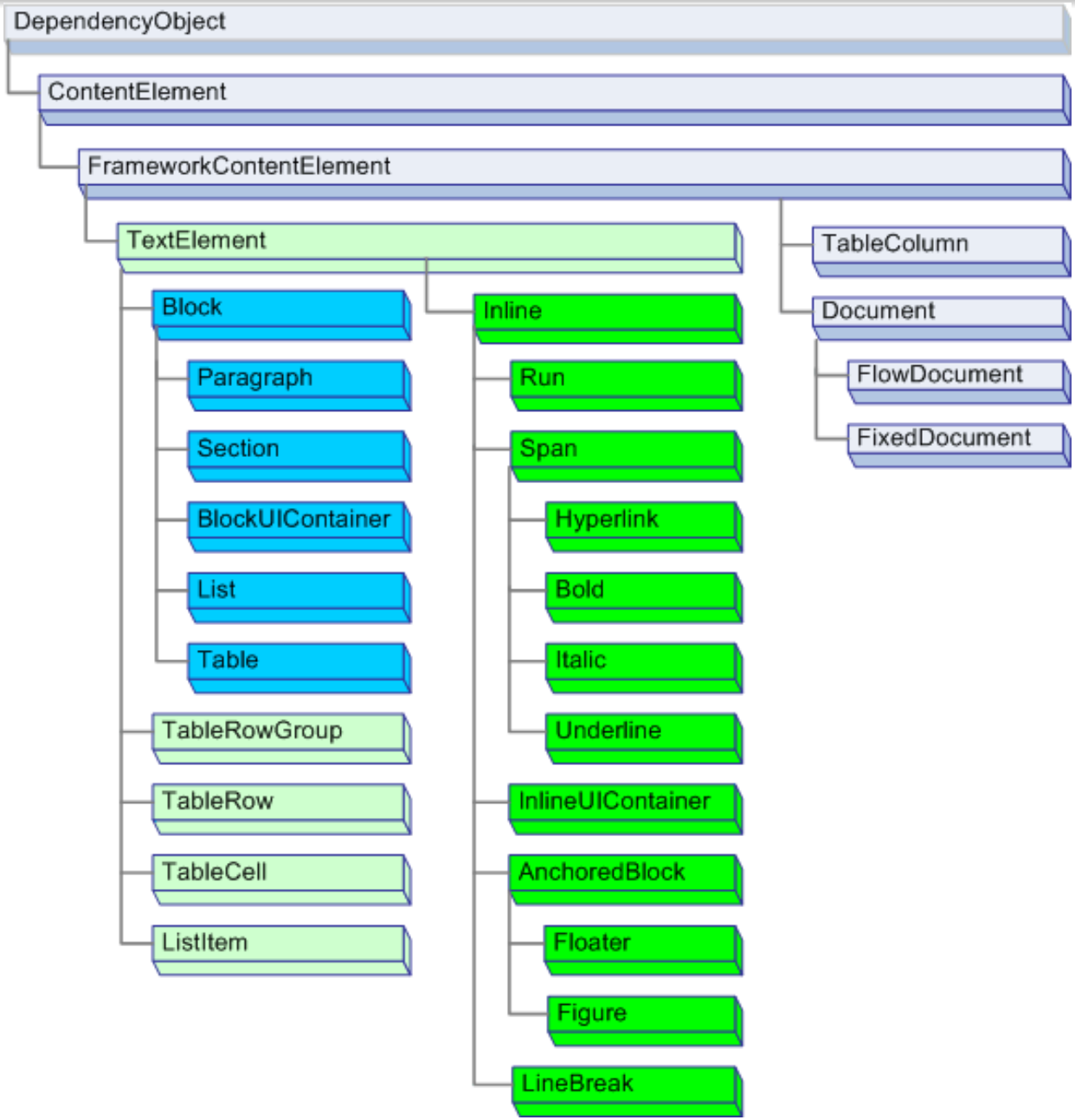
FlowDocument jest dokumentem o formatowaniu dostosowującym się do bieżących możliwości wyświetlania.

Jest to format podobny do HTML'a.

Pliki w tym formacie można oglądać przy pomocy kontrolki FlowDocumentReader.

Kontrolka ta posiada zaawansowane opcje składania tekstu:

- dzielenia na szpalty,
- łamania wyrazów,
- wstawiania obiektów "inline" i pływających,
- ...


DependencyObject
ContentElement
FrameworkContentElement
TextElement
Block
Paragraph
Section
BlockUIContainer
List
Table
TableRowGroup
TableRow
TableCell
ListItem
Inline
Run
Span
Hyperlink
Bold
Italic
Underline
InlineUIContainer
AnchoredBlock
Floater
Figure
LineBreak
TableColumn
Document
FlowDocument
FixedDocument

WPF Tools for Orcas

Microsoft udostępnił Community Technology Preview (CTP) narzędzi do towrzenia aplikacji na platformę .NET 3.0 dla Visual Studio.

Niestety są one bardzo niedorobione i z błędami:

- pojawiają się bzdurne błędy przy budowaniu,
- podobnie przy debuggowaniu,
- debuggowanie XAML'i praktycznie nie istnieje,

Cider - narzędzie do budowania UI:

- obsługuje niewiele opcji,
- nie obsługuje zdarzeń,
- ma chaotyczny interfejs,
- często nie jest w stanie poradzić sobie z prostymi XAML'ami,
- jest wolny.

Dokumentacja jest niedopracowana (mnóstwo drobnych błędów).

Co jeszcze WPF oferuje?

- Walidację danych w kontrolkach.
- Animację właściwości obiektów.
- Lokalizacje.
- Grafikę 3D.
- Rozszerzanie składni XAML'a.
- Zaawansowaną nawigację między stronami XAML'a (PageFunctions...).