ТЕХНОЛОГІЯ WINDOWS PRESENTATION FOUNDATION

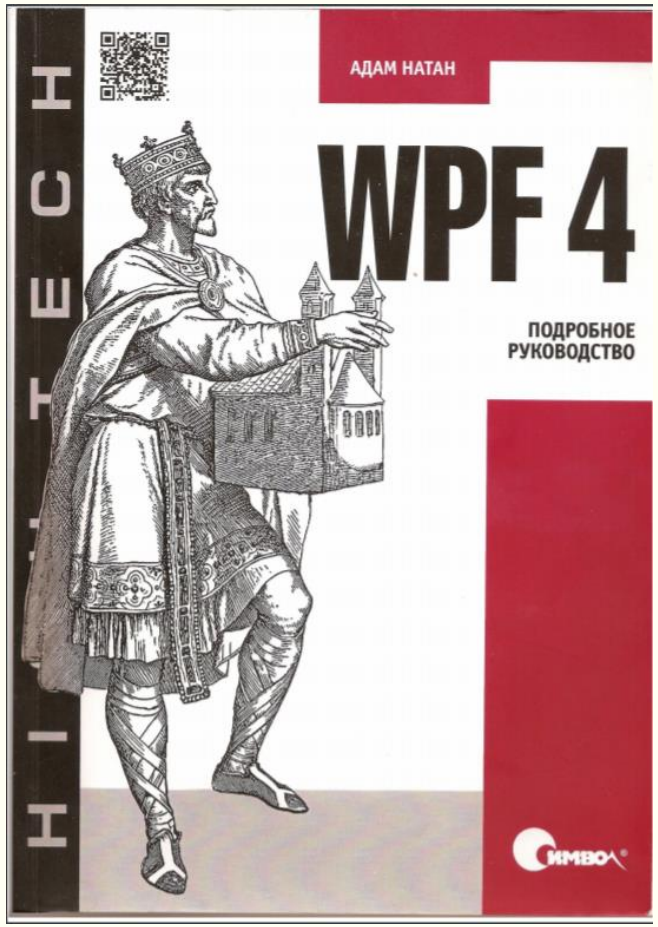
Лекція 02
Інструментальні засоби візуального програмування
ЧДБК, 2020

Питання лекції

- Декларативна мова програмування XAML.
- Технологія Windows Presentation Foundation.
- Компонування інтерфейсу за допомогою контейнерних елементів.
- Основні елементи управління XAML.
- Взаємодія з додатком за допомогою подій.

Література


В. В. Подбельский
Язык декларативного программирования XAML
ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2018


АДАМ НАТАН
WPF 4
ПОДРОБНОЕ РУКОВОДСТВО
СИМВОЛ

ДЕКЛАРАТИВНА МОВА ПРОГРАМУВАННЯ XAML

Питання 2.1.

Extensible Application Markup Language

- XAML (вимовляється “zammle”) – розроблена на базі Extensible Markup Language (XML) мова розмітки для дизайну інтерфейсу користувача.
  - Майже завжди використовується в поєднанні з імперативною мовою, на зразок C#, VB.NET, C++ тощо.
- Парсер XAML компілює XAML-код у двійковий формат – Binary Application Markup Language (BAML), який використовується в процесі виконання додатку.
  - BAML значно зменшує XAML-код, що дозволяє усунути проблеми продуктивності.
  - BAML також токенізований, тому повторювані посилання можна заміняти набагато коротшими токенами.
- На відміну від WinForms, XAML не вимагає від дизайнера створювати C# представлення інтерфейсу користувача.
  - XAML спроектовано самодостатнім для визначення вигляду інтерфейсу.
- Разом з прив’язуванням даних (data binding), маршрутизованими подіями (routed events) та прикріпленими властивостями (attached properties) додатки на основі XAML можуть використовувати шаблони проектування, які дозволяють повне відокремлення дизайну користувацького інтерфейсу (XAML) від прошарку логи представлення (presentation layer logic, C#).
  - Базою є архітектуриний шаблон Model-View-ViewModel (MVVM).
  - Проте можливе використання інших шаблонів проектування також.

Декларативне та імперативне програмування

- При ***декларативному програмуванні*** первинний код записується так, щоб виразити бажаний результат з незначним або відсутнім ухилом в реалізацію.
  - Фреймворк займатиметься парсингом декларативного коду та обробкою “heavy lifting”, потрібного для інстанціювання об’єктів та встановлення властивостей об’єктів, визначених у XAML-елементах.
- ***Імперативне програмування*** – противага декларативному програмуванню.
  - Якщо розглядати декларативне програмування як оголошування того, що бажано отримати, імперативне програмування розглядається як написання коду, що представляє інструкції того, ЯК досягти бажаного результату.
- Декларативний стиль оголошення XAML Window не вимагає інстанціювання екземплярів класів.
  - Замість цього виражаються потрібні елементи управління (controls) у вигляді XML-елементів.
  - Також елемент Button містить вкладений елемент типу TextBlock, у якому вкладено рядкове значення, що відображатиметься як напис на кнопці.
  - Парсер XAML займається створенням відповідних об’єктів (Window, Grid, Button, TextBlock).
  - Також визначає вкладені елементи та вирішує, яку властивість вкладений об’єкт буде populate.

Декларативна розмітка WPF-вікна

- Елементи XAML пов’язуються з класом, у якому парсер під час виконання створить екземпляр заданого типу.
  - Атрибути цих елементів напряму пов’язані з властивостями об’єкта, який представляє елемент.

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Name="btnOK">
            <TextBlock Name="ButtonText" FontSize="100">
                Click Me!
            </TextBlock>
        </Button>
    </Grid>
</Window>
```

Імперативне створення інтерфейсу

```
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 WpfApplication1
{
    //Equivalent to the MainWindow element
    public class CSMainWindow : Window
    {
        private Grid _ContentGrid;
        private Button _BtnClickOk;
        private TextBlock _ButtonText;
```

```
        public CSMainWindow()
        {
            //Set the window properties
            SetValue(TitleProperty, "CSMainWindow");
            this.Height = 350;
            this.Width = 525;

            //Instantiate each object (element) in the Window
            _ButtonText = new TextBlock();
            _ButtonText.Text = "Click Me!";
            _ButtonText.FontSize = 100.0;

            _BtnClickOk = new Button();
            _BtnClickOk.Content = _ButtonText;

            _ContentGrid = new Grid();
            _ContentGrid.Children.Add(_BtnClickOk);

            this.Content = _ContentGrid;

        }
    }
}
```

Визначення XAML

- XAML – досить проста декларативна мова програмування загального призначення, що призначена для конструювання та ініціалізації об’єктів.
  - Це діалект XML з додаванням ряду правил, що відносяться до елементів, атрибутів та їх відображення на об’єкти, їх властивості та значення власивостей.
- Оскільки XAML є механізмом для використання різноманітних API каркаса .NET, спроби порівняння з HTML, SVG (Scalable Vector Graphics) та іншими предметно-орієнтованими мовами та форматами некоректні.
  - XAML містить правила інтерпретації XML синтаксичними аналізаторами та компіляторами, а також ряд ключових слів, проте сам не визначає ніяких суттєвих елементів.

Специфікації XAML і словників XAML

- Детальні специфікації XAML та 2 словників:
  - XAML Object Mapping Specification 2006 (MS-XAML): http://go.microsoft.com/fwlink/?LinkId=130721
  - WPF XAML Vocabulary Specification 2006 (MS-WPFXV): http://go.microsoft.com/fwlink/?LinkId=130722
  - Silverlight XAML Vocabulary Specification 2008 (MS-SLXV): *http://go.microsoft.com/fwlink/?LinkId—130707*
- WPF і XAML можуть використовуватись незалежно один від одного.

Функціональність XAML, яка не доступна з процедурного коду

- Створення повного набору шаблонів. У процедурному коді можна створювати шаблони за допомогою класу FrameworkElementFactory, проте виразні можливості цього підходу обмежені.
- Використання конструкції x:Shared = “False”, яка змушує WPF повертати новий екземпляр при кожному зверненні до елементу зі словника ресурсів.
- Відкладене створення об’єктів всередині словника ресурсів.
  - Важливо для оптимізації продуктивності та доступне тільки за допомогою скомпільованого XAML-коду.

Елементи й атрибути

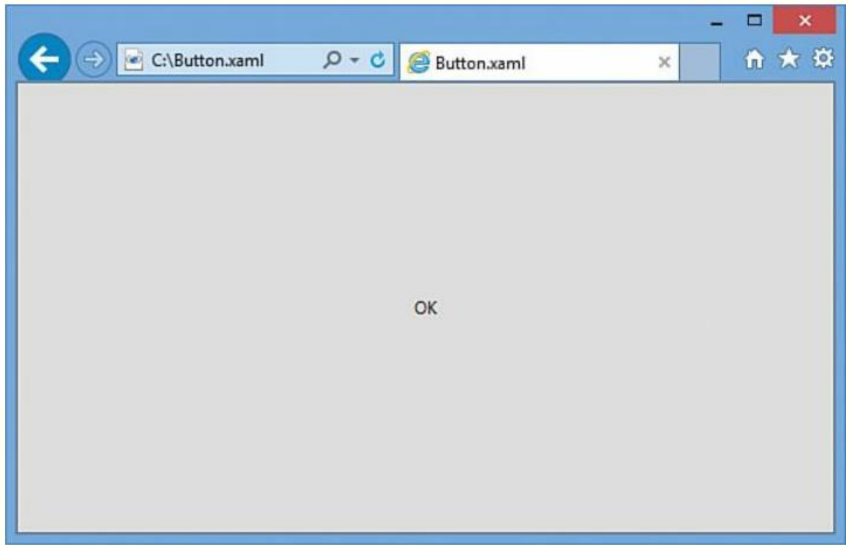
- У специфікації XAML визначені правила відображення просторів імен, типів, властивостей та подій .NET на простори імен, елементи та атрибути XML.
- Оголошення XML-елементу в XAML-коді (називають ***об'єктним елементом***) еквівалентно створенню екземпляра відповідного класу .NET за допомогою конструктора за замовчуванням.
  - Задання атрибута об'єктного елемента еквівалентне заданню однойменної властивості (називаються атрибутами властивостей) або підключенню обробника однойменної події (атрибути подій).

XAML:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK"/>
```

C#:

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Content = "OK";
```



Порядок обробки властивостей та подій

- Під час виконання обробники подій приєднуються до встановлення властивостей об'єктів, оголошених у XAML-коді
  - Крім властивості Name, яка задається відразу після конструювання
  - Тому при генерації подій можна не думати про порядок запису атрибутів у XAML.
- Установлення кількох властивостей та приєднання кількох обробників подій зазвичай виконується в порядку задання властивостей та подій в об'єктному елементі.
  - На практиці цей порядок не важливий

XAML:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK" Click="button_Click"/>
```

C#:

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Click += new System.Windows.RoutedEventHandler(button_Click);
b.Content = "OK";
```

Простори імен

- Відображення на http://schemas.microsoft.com/winfx/2006/xaml/ presentation та інші простори імен WPF жорстко зашите у збірках WPF, точніше в кількох екземплярах атрибуту XmlnsDefinitionAttribute.
  - URL-адреса в домені schemas.microsoft.com не відповідає реальній веб-сторінці.
  - У кореневому об'єктном елементі XAML-файлу потрібно вказувати принаймні один простір імен XML; він використовується для кваліфікації власне елемента та його потомків.
  - Можна оголошувати додаткові простори імен XML за умови, що для кожного простору імен задано унікальний префікс, який буде супроводжувати всі ідентифікатори з цього простору.
  - Часто вказується другий простір імен з префіксом x (xmlns:x):

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

Неявні простори імен .NET

- WPF відображає наступні простори імен .NET на простір імен XML, що відповідає WPF (http://schemas.microsoft.com/winfx/2006/xaml/presentation):
  - System.Windows
  - System.Windows.Automation
  - System.Windows.Controls
  - System.Windows.Controls.Primitives
  - System.Windows.Data
  - System.Windows.Documents
  - System.Windows.Forms.Integration
  - System.Windows.Ink
  - System.Windows.Input
  - System.Windows.Media
  - System.Windows.Media.Animation
  - System.Windows.Media.Effects
  - System.Windows.Media.Imaging
  - System.Windows.Media.Media3D
  - System.Windows.Media.TextFormatting
  - System.Windows.Navigation
  - System.Windows.Shapes
  - System.Windows.Shell

- Використання простору імен WPF XML як основного, а простору імен XAML як додаткового з префіксом x, є просто угодою, аналогічною до починання C#-коду директивою using System;
  - http://schemas.microsoft.com/winfx/2006/xaml/presentation http://schemas.microsoft.com/winfx/2006/xaml
- Можна записати XAMLфайл так – смисл не зміниться:

```
<WpfNamespace:Button
    xmlns:WpfNamespace="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
Content="OK"/>
```

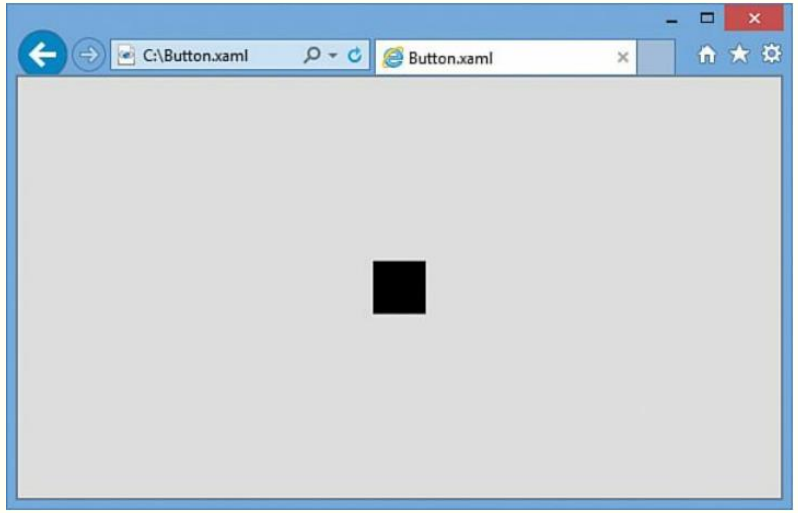
У процесі розвитку утворилось кілька просторів імен WPF XML

- Історично склалось, що на найважливіші типи WPF, визначені в просторі імен System.Windows та вкладених у нього, відображається понад один простір імен XML.
  - У WPF 3.0 підтримувався простір імен http://schemas.microsoft.com/ winfx/2006/xaml/presentation, а у WPF 3.5 було визначено новий простір імен XML - http://schemas.microsoft.com/netfx/2007/xaml/ presentation, який відображається на ті ж типи.
  - У WPF 4 визначено інший простір імен XML, який відображається на ті ж типи: http://schemas.microsoft.com/netfx/2009/xaml/presentation
- Краще не звертати увагу на такий різнобій, а притримуватись початкового простору імен (2006), оскільки він застосовний до будь-якої версії WPF.

Елементи властивостей

- Одна з найбільш потужних особливостей WPF – розвинений механізм композиції.
  - Наприклад, у просту кнопку можна помістити довільний вміст, а не тільки текст!
  - Властивість Content об'єкта Button – об'єкт типу System.Object;
  - зокрема, це може бути об'єкт Rectangle.

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
System.Windows.Shapes.Rectangle r = new System.Windows.Shapes.Rectangle();
r.Width = 40;
r.Height = 40;
r.Fill = System.Windows.Media.Brushes.Black;
b.Content = r; // Делаем квадрат содержимым Button
```



- XAML пропонує альтернативний синтаксис для установки складених властивостей — елементи властивостей.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Button.Content>
        <Rectangle Height="40" Width="40" Fill="Black"/>
    </Button.Content>
</Button>
```

Синтаксис елементів властивостей можна використовувати і для простих значень властивостей

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK" Background="White"/>
```

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Button.Content>
        OK
    </Button.Content>
    <Button.Background>
        White
    </Button.Background>
</Button>
```

Конвертери типів

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Content = "OK";
b.Background = System.Windows.Media.Brushes.White;
```

- Як рядок White може бути еквівалентим статичному полю System.Windows.Media.Brushes.White?
  - Компілятор або аналізатор XAML повинен знайти ***конвертер типу.***
  - WPF надає конвертери типів для багатьох типів даних, які часто використовуються: Brush, Color, FontWeight, Point і т.д.
  - Ці класи породжені від TypeConverter (BrushConverter, ColorConverter і т.д.)
  - Можна написати власний конвертер для довільного типу даних.

- Якби не існувало конвертера типу Brush, то в XAML-коді для пришивання значення для властивості Background довелось би застосувати синтаксис елементів властивостей

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK">
    <Button.Background>
        <SolidColorBrush Color="White"/>
    </Button.Background>
</Button>
```

Конвертери типів

- Проте і це можливо тільки тому, що конвертер типу для Color вміє інтерпретувати рядок "White". Інакше довелось би писати так:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK">
    <Button.Background>
        <SolidColorBrush>
            <SolidColorBrush.Color>
                <Color A="255" R="255" G="255" B="255"/>
            </SolidColorBrush.Color>
        </SolidColorBrush>
    </Button.Background>
</Button>
```

  - А це можливо лише завдяки наявності конвертера типу, який вміє перетворити рядок "255" у значення типу Byte, очікуване властивостями A, R, G і B типу Color.

Використання конвертерів типів у процедурному коді

- У процедурному коді механізм перетворення типів при конвертуванні не застосовується.

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Content = "OK";
b.Background = (Brush)System.ComponentModel.TypeDescriptor.GetConverter(
typeof(Brush)).ConvertFromInvariantString("White");
```

  - На відміну від попереднього C#-коду, друкарська помилка в слові “White” не призведе до помилки компіляції, проте викличе виняток під час виконання, як і в випадку XAML.
  - Visual Studio на етапі компіляції XAML-коду попереджає про поимилки такого роду.

Розширення розмітки. Клас розширення розмітки

- Розширення розмітки також дозволяють покращити виразність мови XAML.
  - Обидва механізми можуть інтерпретувати рядкові атрибути під час виконання (крім деяких вбудованих розширень) та створювати об'єкти, що відповідають рядкам.
- Крім стандартних конвертерів типів у дистрибутиві WPF є кілька вбудованих розширень розмітки.
  - На відміну від конвертерів типів, для розширень розмітки в XAML передбачено явний логічний синтаксис.
  - Крім того, розширення розмітки дозволяють обійти потенційні обмеження, характерні для існуючих конвертерів типів.

Розширення розмітки. Клас розширення розмітки

- Перший ідентифікатор у кожному значенні в фігурних дужках – назва класу розширення розмітки, який повинен успадковуватись від класу MarkupExtension.
  - За угодою, імена таких класів закінчуються словом Extension, проте в XAML його можна пропускати.
  - Тут NullExtension (записано у вигляді x:Null) та StaticExtension (записано у вигляді x:Static) - класи з простору імен System.Windows.Markup, тому для їх пошуку необхідно вказувати префікс **x.**
- Якщо розширення розмітки підтримує такий синтаксис, йому можна передавати параметри, розділені комою.
  - Позиційні параметри (наприклад, SystemParameters.IconHeight) розглядаються як рядкові аргументи для відповідного конструктора класу розширення.
  - Іменовані параметри (в даному прикладі Path і RelativeSource) дозволяють встановлювати в конструйованому об'єкті розширення розмітки властивості з відповідними іменами.
- Значенням такої властивості може бути ще одне розширення розмітки (задається за допомогою вкладених фігурних дужок) або літерал, який можна піддати звичайною процедурою конвертації типів.

Еквівалентний код

- Оскільки розширення розмітки - це просто класи з конструкторами за умовчанням, то їх можна використовувати в елементах властивостей.

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Button.Background>
    <x:Null/>
  </Button.Background>
  <Button.Height>
    <x:Static Member="SystemParameters.IconHeight"/>
  </Button.Height>
  <Button.Content>
    <Binding Path="Height">
      <Binding.RelativeSource>
        <RelativeSource Mode="Self"/>
      </Binding.RelativeSource>
    </Binding>
  </Button.Content>
</Button>
```

Розширення розмітки та процедурний код

- Еквівалентний процедурний код

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
// Установити Background:
b.Background = null;
// Установити Height:
b.Height = System.Windows.SystemParameters.IconHeight;
// Установити Content:
System.Windows.Data.Binding binding = new System.Windows.Data.Binding();
binding.Path = new System.Windows.PropertyPath("Height");
binding.RelativeSource = System.Windows.Data.RelativeSource.Self;
b.SetBinding(System.Windows.Controls.Button.ContentProperty, binding);
```

  - Проте цей код працює інакше, ніж компілятор або аналізатор XAML, який передбачає, що будь-яке розширення розмітки встановлює потрібні значення під час виконання (викликаючи метод ProvideValue).
  - Еквівалентний процедурний код часто виявляється складним і іноді вимагає знання контексту, відомого тільки аналізатору (наприклад, як вирішувати префікс простору імен, який може зустрічатися в властивості Member розширення StaticExtension).
  - На щастя, працювати з розширеннями розмітки таким чином в процедурному коді необов'язково!

Дочерні об’єктні елементи. Властивість Content

- Об'єктний елемент може мати нащадків трьох різних типів:
  - значення властивості вмісту,
  - елементи колекції
  - значення, тип якого може бути перетворений в тип об'єктного елемента.
- У більшості класів WPF є властивість (задається за допомогою атрибута), значенням якої є вміст даного XML-елемента.
  - Воно називається ***властивістю вмісту*** і в дійсності представляє собою просто зручний спосіб зробити XAML-представлення більш компактним.
- Для властивості Content кнопки Button є спеціальна угода, тому опис
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Content="OK"/>
- можна представити так:
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    OK
    </Button >

Дочерні об’єктні елементи. Властивість Content

- Складений вміст Button, наприклад

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Button.Content>
        <Rectangle Height ="40" Width="40" Fill="Black"/>
    </Button.Content>
</Button >
```

- можна переписати так:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Rectangle Height ="40" Width="40" Fill="Black"/>
</Button >
```

- Ніде не вимагається, щоб властивість вмісту називалась саме Content.
  - У класах ComboBox, ListBox і TabControl (всі з простору імен System.Windows.Controls) властивість вмісту названо Items.

Елементи колекцій. Списки

```
<ListBox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <ListBox.Items>
    <ListBoxItem Content="Item 1"/>
    <ListBoxItem Content="Item 2"/>
  </ListBox.Items>
</ListBox>
```

- Цей XAML-код еквівалентний такому коду на C#:

```
System.Windows.Controls.ListBox listbox =
                        new System.Windows.Controls.ListBox();
System.Windows.Controls.ListBoxItem item1 =
                        new System.Windows.Controls.ListBoxItem();
System.Windows.Controls.ListBoxItem item2 =
                        new System.Windows.Controls.ListBoxItem();
item1.Content = "Item 1";
item2.Content = "Item 2";
listbox.Items.Add(item1);
listbox.Items.Add(item2);
```

Розширення розмітки. Клас розширення розмітки

- Оскільки Items - властивість вмісту для ListBox, то XAML-код можна ще скоротити:

```
<ListBox
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <ListBoxItem Content="Item 1"/>
  <ListBoxItem Content="Item 2"/>
</ListBox>
```

  - Цей код працює тому, що властивість Items класу ListBox автоматично ініціалізується порожньою колекцією.
  - Якби вона ініціалізувалась значенням null (і, на відміну від доступної тільки для читання властивості Items класу ListBox, допускала читання і запис), то довелося б помістити всі елементи всередину явно заданого елемента XAML, який створює екземпляр колекції.
  - Умовний OtherListBox:

```
<OtherListBox>
  <OtherListBox.Items>
    <ItemCollection>
      <ListBoxItem Content="Item 1"/>
      <ListBoxItem Content="Item 2"/>
    </ItemCollection>
  </OtherListBox.Items>
</OtherListBox>
```

Словники

- Колекція System.Windows.ResourceDictionary реалізує інтерфейс System.Collections.IDictionary, тому підтримує додавання, видалення і перелічення пар ключ / значення в процедурному коді, як будь-яка хеш-таблиця.
  - Наступний XAML-код додає в словник ResourceDictionary два кольори Color:

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Color x:Key="1" A="255" R="255" G="255" B="255"/>
  <Color x:Key="2" A="0" R="0" G="0" B="0"/>
</ResourceDictionary>
```

  - Тут використовується ключове слово XAML Key (певне в додатковому просторі імен XML), яке обробляється спеціальним чином і дозволяє пов'язати з кожним значенням Color якийсь ключ.
  - В самому типі Color властивість Key не визначена.

Словники

- Колекція System.Windows.ResourceDictionary реалізує інтерфейс System.Collections.IDictionary, тому підтримує додавання, видалення і перелічення пар ключ / значення в процедурному коді, як будь-яка хеш-таблиця.
  - Наступний XAML-код додає в словник ResourceDictionary два кольори Color:

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Color x:Key="1" A="255" R="255" G="255" B="255"/>
  <Color x:Key="2" A="0" R="0" G="0" B="0"/>
</ResourceDictionary>
```

  - Тут використовується ключове слово XAML Key (певне в додатковому просторі імен XML), яке обробляється спеціальним чином і дозволяє пов'язати з кожним значенням Color якийсь ключ.
  - В самому типі Color властивість Key не визначена.

Еквівалентний код на C#

```
System.Windows.ResourceDictionary d = new System.Windows.ResourceDictionary();
System.Windows.Media.Color color1 = new System.Windows.Media.Color();
System.Windows.Media.Color color2 = new System.Windows.Media.Color();
color1.A = 255; color1.R = 255; color1.G = 255; color1.B = 255;
color2.A = 0;   color2.R = 0;   color2.G = 0;   color2.B = 0;
d.Add("1", color1);
d.Add("2", color2);
```

- Аналізатор WPF XAML завжди підтримував тільки колекції IList та IDictionary, проте фунціональність аналізатора XAML2009 дещо розширена.
  - Спочатку він перевіряє інтерфейси IList та Idictionary, потім – ICollection<T> і IDictionary<K,V>, а далі – наявність методів Add() та GetEnumerator().

Ще про перетворення типів

- Нащадком об’єктного елемента може бути звичайний текст:
  - <SolidColorBrush>White</SolidColorBrush>
- Цей запис еквівалентний такому:
  - <SolidColorBrush Color="White" />
- навіть незважаючи на те, що Color не описана як властивість вмісту.
  - Тут перший фрагмент працює тому, що існує конвертер типу, який може перетворювати такі рядки, як “White”, “white” чи “#FFFFFF” в об’єкт типу SolidColorBrush.
- Оскільки XAML призначений для роботи з системою типів .NET, то його можна використовувати практично з будь-яким об'єктом .NET, у тому числі визначеним вами.
  - При цьому зовсім неважливо, чи належать ці об'єкти користувацькому інтерфейсу.
  - Проте класи необхідно визначати з урахуванням можливості використання в декларативному коді.
  - Наприклад, якщо в класі немає ні конструктора за умовчанням, ні корисних відкритих властивостей, то ним не можна буде безпосередньо скористатися в XAML (якщо не XAML2009).
  - Програмні інтерфейси WPF ретельно продумані з тим, щоб вони відповідали декларативній моделі XAML (звичайних принципів розробки для .NET недостатньо).

Ще про перетворення типів

- Збірки WPF відмічені атрибутом XmlnsDefinitionAttribute, який відображає простори імен .NET всередині них на простір імен XML у XAML-файле.
  - Що робити зі збірками, які розроблялись без орієнтації на XAML і не містять цього атрибута?
  - Типи з них все-одно можна використовувати, додавши спеціальну директиву, що описує простір імен XML.
  - Наприклад, звичайний код на C#, у якому використовуються класи .NET зі збірки mscorlib.dll:

```
System.Collections.Hashtable h = new System.Collections.Hashtable();
h.Add("key1", 7);
h.Add("key2", 23);
```

- У XAML його можна представити так:

```
<collections:Hashtable
    xmlns:collections="clr-namespace:System.Collections;assembly=mscorlib"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <sys:Int32 x:Key="key1">7</sys:Int32>
    <sys:Int32 x:Key="key2">23</sys:Int32>
</collections:Hashtable>
```

Коментарі до коду

- Директива clr-namespace дозволяє використати простір імен .NET безпосередньо в XAML.
  - Специфікація збірки напркінці необхідна лише тоді, коли потрібні типи не знаходяться в тій же збірці, де зберігається скомпільований XAML-код.
  - Зазвичай достатньо простої назви збірки (як у випадку mscorlib), проте можна використовувати і канонічне представлення, яке підтримується методом System.Reflection.Assembly.Load() та включає додаткову інформацію, наприклад, номер версії і/або маркер відкритого ключа.
- Важливі моменти щодо інтеграції не тільки з системою типів .NET, але й з конкретними типами .NET Framework:
  - Дочерні елементи можна додавати в батьківську хеш-таблицю Hashtable за допомогою стандартного синтаксису XAML x:Key, оскольки Hashtable, як і інші класи колекцій в .NET Framework, реалізує інтерфейс Dictionary.
  - Тип System.Int32 можна використовувати так просто, оскільки вже існує конвертер типу, що вміє перетворювати рядок у ціле число. Це пояснюється тим, що конвертери типів, що підтримує XAML, всього лиш підкласи класу System.ComponentModel.TypeConverter, що існує з версії .NET Framework 1.0. Це ой же механізм перетворення типів, який використовується в Windows Forms.

ДЯКУЮ ЗА УВАГУ!

Наступне питання: Основи модульного тестування C#-коду