Технологія Windows Presentation Foundation

Інструментальні засоби візуального програмування

Тема 02, ЧДБК-2019

План лекції

- Декларативна мова програмування XAML
- Технологія Windows Presentation Foundation

Декларативна мова програмування XAML

Extensible Application Markup Language

- XAML (pronounced “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.

Відокремлення User Interface Concerns

- На відміну від WinForms, XAML не вимагає від дизайнера створювати C# представлення інтерфейсу користувача.
  - XAML спроектовано самодостатнім для визначення вигляду інтерфейсу.
- Разом з data binding, routed events та attached properties додатки на основі XAML можуть використовувати шаблони проектування, які дозволяють повне відокремлення user interface design (XAML) та presentation layer logic (C#).
- Базою є шаблон проектування Model-View-ViewModel (MVVM).
  - Проте можливе використання інших шаблонів проектування також.

Декларативне та імперативне програмування

- При декларативному програмуванні вихідний код записується так, щоб виразити бажаний результат з little or no emphasis on the actual implementation.
  - Фреймворк займатиметься парсингом декларативного коду та обробкою "heavy lifting" required to instantiate objects and set properties on the these objects, which are defined in the XAML elements.
- *Імперативне програмування* - opposite of declarative programming.
  - Якщо розглядати декларативне програмування як оголошування того, що бажано отримати, імперативне програмування розглядається як написання коду, що представляє інструкції того, ЯК досягти бажаного результату.

Декларативна розмітка WPF-вікна

```
<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>
```

- 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.

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;

        }
    }
}
```

Декларативний та імперативний опис інтерфейсу

- 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 will populate.

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

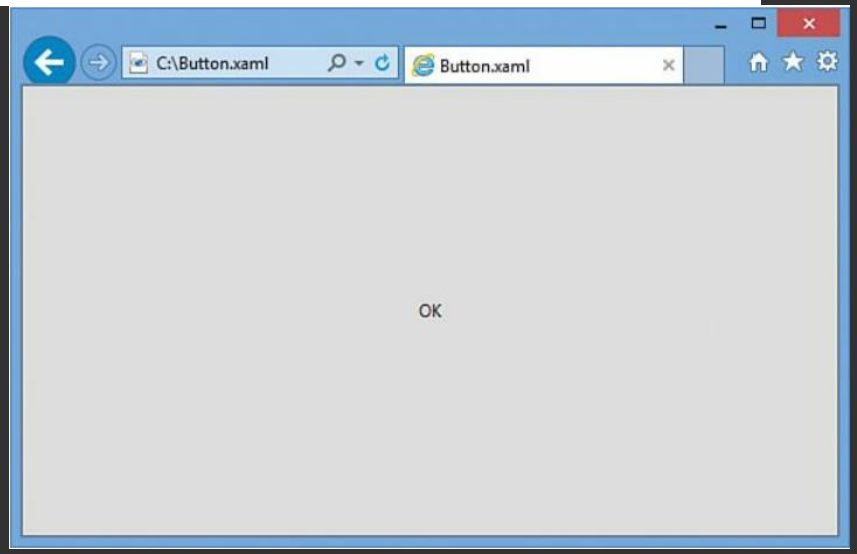
XAML:

```
<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";
```

- 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" 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";
```

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

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"/>
```

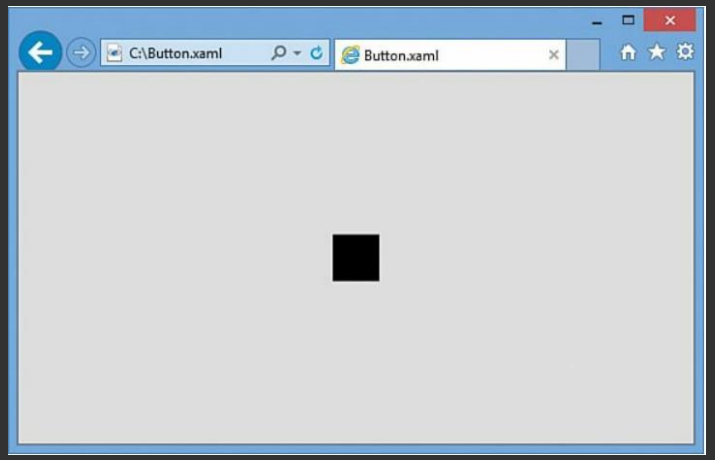
У процесі розвитку утворилось кілька просторів імен WPF XML

- 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

Но лучше не обращать внимания на этот разнобой, а придерживаться первоначального пространства имен http://schemas.microsoft.com/winfx/2006/xaml/presentation, поскольку оно применимо к любой версии WPF. (Но это вовсе не означает, что сам XAML-код будет работать со всеми версиями WPF; для этого необходимо пользоваться только возможностями, присутствующими в WPF 3.0) Отметим, что Silverlight также

Елементи властивостей

- Одна з найбільш потужних особливостей WPF – розвинений механізм композиції.
  - Наприклад, у просту кнопку можна помістити довільний вміст, а не тільки текст!




```
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
```

Властивість Content об'єкта Button – об'єкт типу System.Object;

зокрема, це може бути об'єкт Rectangle.

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 использовать и для простых значений свойств

```
<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>
```

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>
```

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");
```

В отличие от предыдущего кода на C#, опечатка в слове "White" не приведет к ошибке компиляции, но вызовет исключение во время выполнения, как и в случае XAML. (Хотя Visual Studio на этапе компиляции XAML-кода предупреждает об ошибках такого рода.)

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.

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.**

- Если расширение разметки поддерживает такой синтаксис, ему можно передавать параметры, разделенные запятой.
  - Позиционные параметры (например, SystemParameters.IconHeight) рассматриваются как строковые аргументы для соответствующего конструктора класса расширения.
  - Именованные параметры (в данном примере Path и RelativeSource) позволяют устанавливать в конструируемом объекте расширения разметки свойства с соответствующими именами.
- Значением такого свойства может быть еще одно расширение разметки (задается с помощью вложенных фигурных скобок) или литерал, который можно подвергнуть обычной процедуре конвертации типов.

Еквівалентний код

- Поскольку расширения разметки - это просто классы с конструкторами по умолчанию, то их можно использовать в элементах свойств.

```
<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();
// Установить Background:
b.Background = null;
// Установить Height:
b.Height = System.Windows.SystemParameters.IconHeight;
// Установить 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);
```

Однако этот код работает иначе, чем компилятор или анализатор XAML, который предполагает, что любое расширение разметки устанавливает нужные значение во время выполнения (вызывая метод ProvideValue).

Дочерние объектные элементы. Свойство Content

- Объектный элемент может иметь потомков трех разных типов:
  - значение свойства содержимого,
  - элементы коллекции
  - значение, тип которого может быть преобразован в тип объектного элемента.
- В большинстве классов WPF имеется свойство (задаваемое с помощью атрибута), значением которого является содержимое данного XML-элемента.
  - Оно называется *свойством содержимого* и в действительности представляет собой просто удобный способ сделать XAML-представление более компактным.
-

Дочерние объектные элементы. Свойство Content

- Для свойства Content кнопки Button имеется специальное соглашение, поэтому описание
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Content="OK"/>
- можно представить в следующем виде:
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"> OK </Button >

Дочерние объектные элементы. Свойство Content

- составное содержимое Button, например
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Button.Content>
    <Rectangle Height ="40" Width="40" Fill="Black"/>
    </Button.Content>
    </Button >
- можно переписать так:
  - <Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Rectangle Height ="40" Width="40" Fill="Black"/>
    </Button >
- Нигде не требуется, чтобы свойство содержимого называлось именно Content.
  - В классах ComboBox, ListBox и TabControl (все из пространства имен System.Windows.Controls) свойство содержимого названо Items.

Элементы коллекций. Списки

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

- Этот XAML-код эквивалентен 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);
```

- поскольку Items - свойство содержимого для ListBox, то XAML-код можно еще сократить:

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

- Этот код работает потому, что свойство Items класса ListBox автоматически инициализируется пустой коллекцией.
  - Если бы оно инициализировалось значением null (и, в отличие от доступного только для чтения свойства Items класса ListBox, допускало чтение и запись), то пришлось бы поместить все элементы внутрь явно заданного элемента XAML, который создает экземпляр коллекции

Умовний OtherListBox

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

Словари

- Коллекция System.Windows.ResourceDictionary реализует интерфейс System.Collections.IDictionary, а значит, поддерживает добавление, удаления и перечисление пар ключ/значение в процедурном коде, как любая хеш-таблица.
  - следующий XAML-код добавляет в словарь ResourceDictionary два цвета 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>
```

Здесь используется ключевое слово XAML Key (определенное в дополнительном пространстве имен XML), которое обрабатывается специальным образом и позволяет связать с каждым значением Color некий ключ.