Exploring Windows Presentation Foundation

With Practical Applications in .NET 5

Taurius Litvinavicius

Apress®

Exploring Windows Presentation Foundation

With Practical Applications in .NET 5

Taurius Litvinavicius

Apress®

Exploring Windows Presentation Foundation: With Practical Applications in .NET 5

Taurius Litvinavicius
Jonava, Lithuania

ISBN-13 (pbk): 978-1-4842-6636-6 ISBN-13 (electronic): 978-1-4842-6637-3
https://doi.org/10.1007/978-1-4842-6637-3

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Smriti Srivastava
Development Editor: Laura Berendson
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Pexels

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

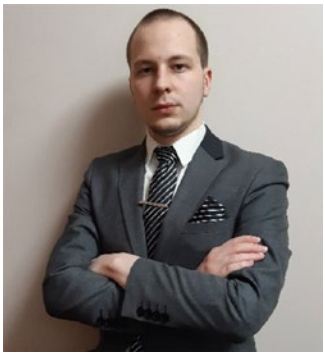
Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6636-6. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author



Taurius Litvinavicius is a businessman and technology expert based in Lithuania who has worked with various organizations in building and implementing projects in software development, sales, and other fields of business. He is responsible for technological improvements, development of new features, and general management. Taurius is also the director at the Conficiens solutio consulting agency where he supervises the development and maintenance of various projects and activities.

About the Technical Reviewer

Carsten Thomsen is a back-end developer primarily but working with smaller front-end bits as well. He has authored and reviewed a number of books, and created numerous Microsoft Learning courses, all to do with software development. He works as a freelancer/contractor in various countries in Europe, using Azure, Visual Studio, Azure DevOps, and GitHub as some of his tools. Being an exceptional troubleshooter, asking the right questions, including the less logical ones, in a most logical to least logical fashion, he also enjoys working with architecture, research, analysis, development, testing, and bug fixing. Carsten is a very good communicator with great mentoring and team lead skills and great skills researching and presenting new material.

Introduction

In this book, you will find lots of information about Windows Presentation Foundation (WPF) which will help you get started with it. Alongside the explanations of features, you will find use case examples and exercise assignments for you to practice what you have learned.

The first chapter will provide you with a basic introduction to the WPF. That will include handling button click event, window handling, accessing text box inputs, and a few more things. In the next chapter, you will see some generic events; some of them are related to the window, some to the mouse, and some to other things. The third chapter will cover various UI elements in WPF; it will also be useful to you for future reference. It is important to read and understand the first chapter, but in case you are in a hurry, you may skip the second and third chapters and only use them as reference later.

The fourth chapter will show you how to handle files in the WPF interface, and with that, it will provide some use cases to study. At this point, the examples will incorporate quite a few items from the previous chapters. Then in the fifth, you will see how the MVVM structure can be implemented – the explanation will be a practical one; this should help you understand MVVM quicker. The final chapter will cover the styling aspects of WPF, but with that, it will also show examples that incorporate most things that you can find in the previous chapters. Once again, if you are rushing to get started with WPF and do not have too much time, you may skip the MVVM chapter. But it would be a good idea to take a look at it later, as it is useful to understand what MVVM is and how it is implemented.

CHAPTER 1

Getting Started

Windows Presentation Foundation (WPF) has many features and many arrangements you can choose from, but there are a few crucial things that you have to know before going anywhere else. For any user interaction to be viable in WPF, you need to understand how to use the methods, and with that, you also need to understand how to set properties on the elements. In this chapter, we will begin with only a handful of them, and in Chapter 3, you will see many more to choose from, and they function in a very similar way. Before going further, you should also understand how to establish and then display or close windows and how to display quick and simple alert messages.

Button and Click Event

Probably the most important element in WPF is the button, and probably the most important event is the click event. Now, the main three things in WPF are elements (e.g., button), events, and names of the elements. The first thing you will learn about is a button, but for it to make sense, you will also need to look at something called text block and label.

T. Litvinavicius, *Exploring Windows Presentation Foundation*,
https://doi.org/10.1007/978-1-4842-6637-3_1

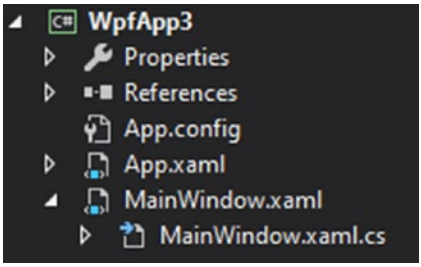


Figure 1-1. *File layout in the project*

To begin with, we will work with the MainWindow.xaml source file, which is where the XAML code for the window and its elements go (Figure 1-1). MainWindow.xaml.cs is what is called a code-behind file; that is where the C# code for that window goes. In the next section, you will learn how to add more windows and navigate between them.

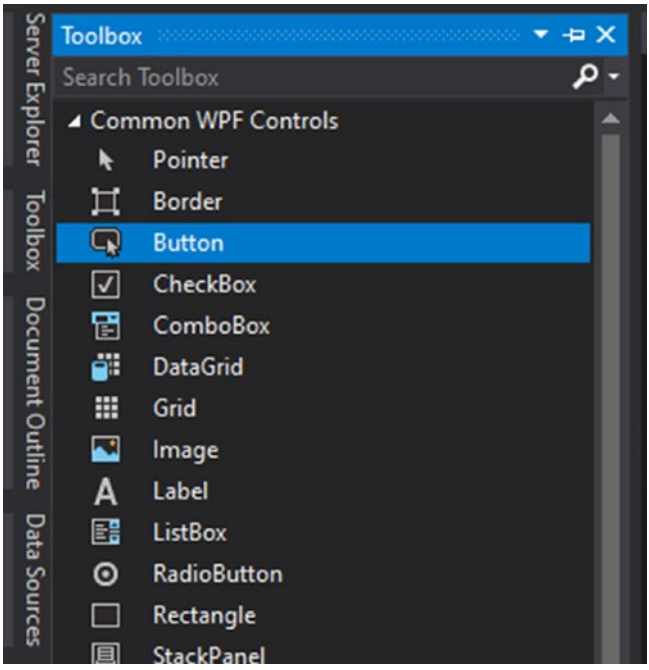


Figure 1-2. *Toolbox in design view editor*

Once you get into the MainWindow.xaml code, you will see a designer view. Although you can set various properties in XAML, it is best to drag and drop from the toolbox (Figure 1-2) and then move things around, expand them, and do other things in the designer. Once you drag and drop something onto the designer view, the XAML code for that element will be generated.

Figure 1-3. *Window view for the example*

This is what our example (see Figure 1-3) will look like (after the buttons are clicked).

Listing 1-1. XAML code for the MainWindow.xaml

```
<Window x:Class="WpfApp3.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/
        blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
        compatibility/2006"
```

```
        xmlns:local="clr-namespace:WpfApp3"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Button x:Name="bt1" Content="first option"
        HorizontalAlignment="Left" Margin="350,163,0,0"
        VerticalAlignment="Top" Width="101"/>
        <TextBlock x:Name="textblock1"
        HorizontalAlignment="Left" Margin="350,198,0,0"
        TextWrapping="Wrap" VerticalAlignment="Top"
        Width="101"/>

        <Button Click="Button_Click" Content="second option"
        HorizontalAlignment="Left" Margin="350,229,0,0"
        VerticalAlignment="Top" Width="101"/>
        <Label x:Name="label1" Content=""
        HorizontalAlignment="Left" Margin="350,254,0,0"
        VerticalAlignment="Top" Width="101"/>
    </Grid>
</Window>
```

This is the XAML code (Listing 1-1) for MainWindow.xaml; all the elements go into the Grid element. The first element in the grid is a Button. In it, we have a name property (you will see why we need it in the C# code); after that, we have Content, which is the text displayed in the button. After that, two very important properties are `HorizontalAlignment` and `VerticalAlignment`, which are important because they determine the alignment, and with that, you can use margins accordingly.

The next element is the TextBlock which is used to display text – the crucial part here is the name, as that will be the reference point in C#. With that, you can also see TextWrapping – if set, it will wrap the text onto a new line; if not set, the default value will be used (NoWrap) and the text will be displayed in a single line.

Another Button, similar to the first one, but with no name specified. Instead, we have the Click property which has a value of `Button_Click` – this will correspond with the event method name in C#. So, you will basically see two ways of declaring an event.

Finally, we have a label, which is an element very similar to a TextBlock. But the label has fewer options in terms of customization.

Listing 1-2. C# code for MainWindow.xaml.cs

```
public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            bt1.Click += Bt1_Click;
        }

        private void Bt1_Click(object sender, RoutedEventArgs e)
        {
            textblock1.Text = "button works 1";
        }

        private void Button_Click(object sender,
        RoutedEventArgs e)
        {
            label1.Content = "button works 2";
        }
    }
```

In the C# code (code-behind) (see Listing 1-2), you can first see the constructor method for the window class. In it, we declare the button click event for the first button. In the code, the first event is for the first button and the second for the second button. To set a display value for the TextBlock, you need to set the Text property, but for the label, you set the Content property.

Window and Page

In WPF there are two main view options – one is window, and the other is page. Another option may be tabs, or you may simply change visibility of grids and other containers, but that will be covered in the next chapters (Chapters 3 and 6).

Both window and page will have XAML part and C# part (code-behind). The main difference is how they are displayed and how the user navigates between them.

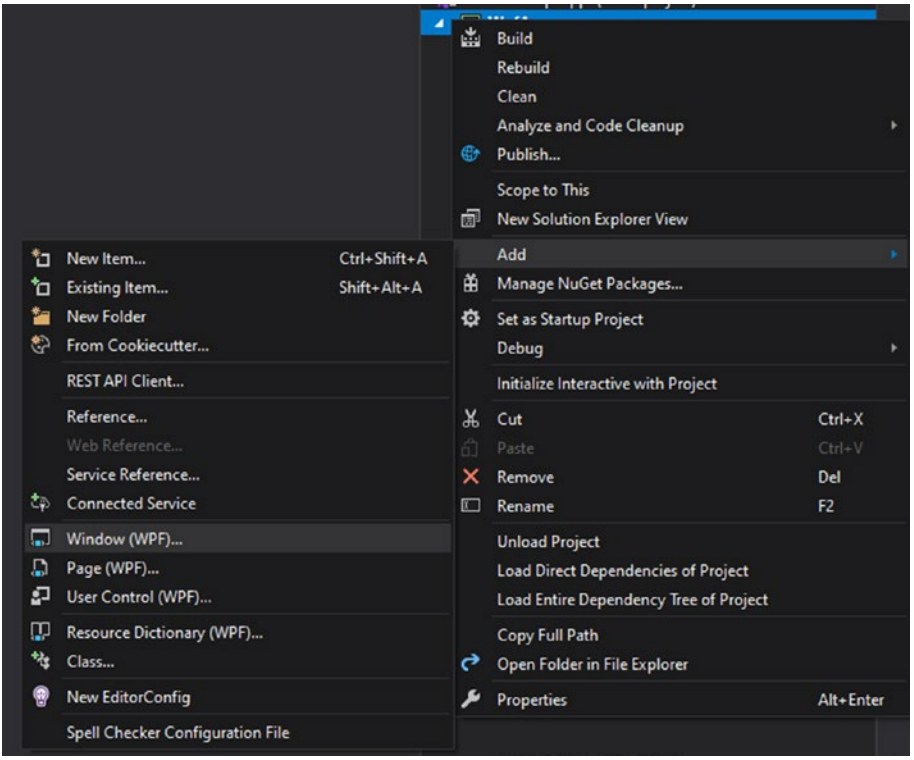


Figure 1-4. *Adding Window file in WPF project*

To create a new window, you need to right-click your project (or folder in which you will place the file) and then go to add and choose Window (WPF) (see Figure 1-4). Alternatively, you can find this option in "New Item".

Listing 1-3. XAML code in MainWindow.xaml

```
<Window x:Class="WpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/
        blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
        compatibility/2006"
        xmlns:local="clr-namespace:WpfApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>
        <Button x:Name="bt1" Content="window 1"
        HorizontalAlignment="Left" Margin="246,118,0,0"
        VerticalAlignment="Top" Width="75"/>

    </Grid>
</Window>
```

The XAML part of a window (See Listing 1-3) will contain all the markup for your elements and the window itself. By default, a window will contain a grid and that is where you should place your elements – you cannot do that in the window itself. In the default window, everything from x:Class to mc:Ignorable should not be modified; otherwise, it may break. You may change the title property (which appears at the top-left corner of a window), and you can change the height and width properties. There are many more things that you can do with a window, and some of them will be covered in the last chapter of this book.

Listing 1-4. MainWindow.xaml.cs contents

```
public MainWindow()
        {
            InitializeComponent();
            bt1.Click += Bt1_Click;

        }

        private void Bt1_Click(object sender, RoutedEventArgs e)
        {
            Window w1 = new Window1();
            w1.Show();
        }
```

By default, the application will open your main window (MainWindow), but later, you can close it and/or open more windows, and there are two ways of doing this. In this particular example, we have another window created, and it is named Window1. You can see that on button click event (see Listing 1-4), we construct and establish a variable for that new window. To display it, we execute the **Show** method. This will display the new window, but it will not close the existing one and will allow to interact with the existing window. An alternative to that is to open a window by using the `ShowDialog` method; this will freeze the existing window and will only allow the user to interact with the new one. To close the current window, you will need to use the **this.Close()** method or you can control by reference; in this case, it would be **w1.Close()**.

It is also important to understand the constructor method for the window. By default, you have the `InitializeComponent` method which initializes all the elements declared in the window. So, if you want to set properties (e.g., set text to text block), you need to do all that after `InitializeComponent` has occurred.

The Page is not as stand-alone as a Window; instead, you use pages to establish some navigation inside a Window.

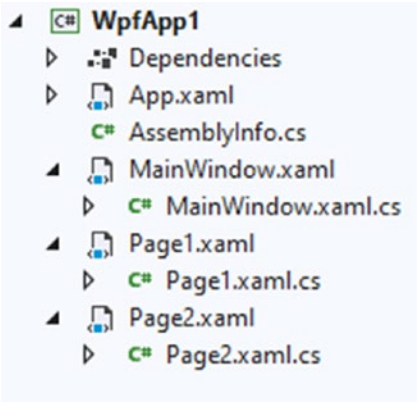


Figure 1-5. *File layout in the project*

In this example, we have added two new files (see Figure 1-5) – Pages.

Figure 1-6. *First state of the view (nothing clicked)*

Initially, you will see an empty window like this (Figure 1-6).

Figure 1-7. *Second state of the view ("To page 1" button clicked)*

After the button is clicked, a page will be displayed (see Figure 1-7). If the second button is clicked, the first page will be replaced with the second one.

Listing 1-5. MainWindow.xaml contents

```
<Grid>
        <Frame x:Name="fr"   Margin="0,0,0,66" NavigationUI
        Visibility="Hidden"/>
        <Button Content="To page 1" Click="Button_Click"
        HorizontalAlignment="Left" Margin="321,406,0,0"
        VerticalAlignment="Top" Height="28" Width="79"/>
        <Button Content="To page 2" Click="Button_Click_1"
        HorizontalAlignment="Left" Margin="400,406,0,0"
        VerticalAlignment="Top" Height="28" Width="79"/>
    </Grid>
```

In your window (see Listing 1-5) in which you want to display pages, you will need a Frame element – that is where a page is displayed. You need to set an appropriate size, establish a name, and then set `NavigationUIVisibility` to Hidden. The last one is not important from a functional perspective, but if not set, it would display a navigation bar, similar to what a browser might have.

Listing 1-6. Events in MainWindow.xaml.cs

```
private void Button_Click(object sender, RoutedEventArgs e)
        {
            fr.Content = new Page1();
        }

        private void Button_Click_1(object sender,
        RoutedEventArgs e)
        {
            fr.Content = new Page2();
        }
```

Navigating to a page is quite simple here (See Listing 1-6); you just need to set the Content property of the Frame element.

Listing 1-7. Constructor method for Page1 (Page1.xaml.cs)

```
public Page1()
        {
            InitializeComponent();
        }
```

A Page has a constructor (see Listing 1-7) just like a Window, and inside that, you will find the same arrangement with the InitializeComponent method.

Text Box

There are two basic interactive elements in WPF – the first one you already know, which is a button, and the second one is the text box. In this book, you will see more elements in the third chapter, but this will be your starting point in understanding the input elements.

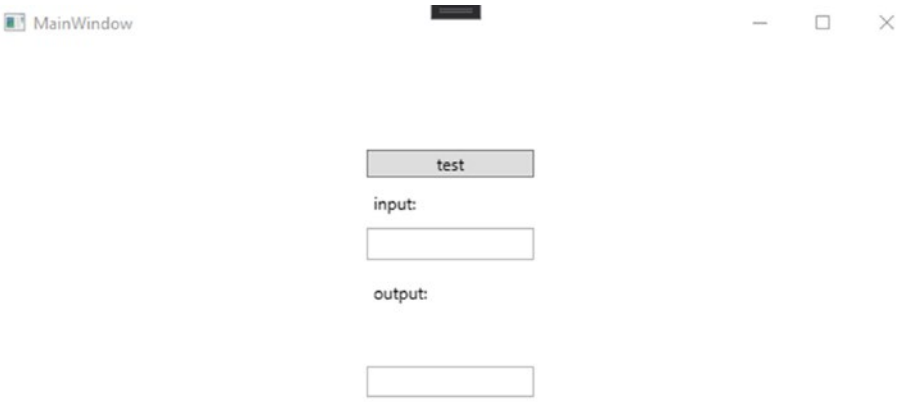


Figure 1-8. *Window view for the example*

Listing 1-8. MainWindow.xaml contents

```
<Grid>
        <Label Content="input:" HorizontalAlignment="Left"
        Margin="328,100,0,0" VerticalAlignment="Top"
        Width="106"/>
        <TextBox x:Name="tb1" HorizontalAlignment="Left"
        Height="23" Margin="328,131,0,0" TextWrapping="Wrap"
        VerticalAlignment="Top" Width="120"/>
        <Label Content="output:" HorizontalAlignment="Left"
        Margin="328,164,0,0" VerticalAlignment="Top"
        Width="106"/>
```

```
        <TextBox x:Name="output2" HorizontalAlignment="Left"
        Height="22" Margin="328,230,0,0" TextWrapping="Wrap"
        VerticalAlignment="Top" Width="120"/>
        <Label x:Name="output1" Content=""
        HorizontalAlignment="Left" Margin="328,199,0,0"
        VerticalAlignment="Top" Width="106"/>
        <Button Click="Button_Click" Content="test"
        HorizontalAlignment="Left" Margin="328,75,0,0"
        VerticalAlignment="Top" Width="120"/>
    </Grid>
```

In this case, we have six elements (see Figure 1-8 and Listing 1-8) in the XAML (the grid is in the window). The first label simply has some static text, the second element is a text box, and a text box needs a name for referencing in C#. After that, we have another static label and then we have another text box with a name. There is also another label with a name (it will be used from C#) and finally a button with an event. Notice, in this case, the event is declared in XAML; therefore, you will not see it declared in the C# code. The idea here is to enter something into the input box (tb1) and output the value in the label (output1) and text box (output2).

If you want to have multiple lines in your text box input, you need to set the `AcceptsReturn` property to `True`. Also, by default, the text box does not accept tab input; if you want that, you will need to set `AcceptsTab="True"`.

Listing 1-9. Button click event

```
private void Button_Click(object sender, RoutedEventArgs e)
        {
            output1.Content = tb1.Text + " 1";
            output2.Text = tb1.Text + " 2";
        }
```

This is the event on button click in the C# part (see Listing 1-9). Now, the output1 is a label and a label has the property Content, which is what you assign to display something. On the other hand, text box has the Text property, which will hold the input and can also be assigned to display something. The most important thing in any input element is to know which property holds the value – in this case, it is quite easy to find, but later, you will see some elements that are a bit more tricky.

Message Box

Message box is a great way to display basic notifications and alerts to the user; it is simple to use and can be a great safeguard on some occasions. Now, we will look at how it is implemented, but before that, you have to know about the possible arrangements of this feature. There are four default options in a message in terms of the buttons it may contain. Each button, when clicked, will close the box and return a result (`MessageBoxResult`). Later in this book, you will learn how to make your own custom message box (refer to Chapter 6).

Figure 1-9. *Simple message box*