WPF Desktop Application Workshop

# Summary

This workshop shows you how to create your own Desktop application in WPF (Windows Presentation Foundation).

You will learn how to read images of your computer and present them in any way you like.

# Content

* Create a Visual Studio Solution and Project
* Add references to libraries with ready to use functionality
* Design the layout of your user interface (UI)
* Add controls to your UI
* Interact with the user
* Read data of your computer
* Bind data to controls on your UI
* Optional: Customize your UI by setting various options, creating a better user experience

# Languages

* C# (you’re free to choose VB.NET instead)
* XAML

# Prerequisites

* Visual Studio Community

<https://www.visualstudio.com/products/visual-studio-community-vs>

* .NET Framework 4.0/4.5

Note: The .NET Framework is installed with Visual Studio

# Author

Mertsch - <https://github.com/Mertsch>

# Get ready for some coding

## Goal

Set up a Visual Studio project to work with.

## Instructions

Start Visual Studio and click "New project", select "WPF Application" and give it a name like "PictureViewer".

## Explanation

In Visual Studio a project usually represents a collection of code files which will then be compiled into executable code. A number of different project form a solution. In this case we will need only one project, which will automatically be put under a solution. The WPF Application represents a template which will pre-set a few things for you. For example the output is set to "Windows application" which will let you execute the result, as you know for any other Windows application. Also a few code files are already put into the project to get you started with the basics.

# What’s all this?

## Goal

Understanding what the project contains.

## Instructions

Look at all the different items under your project in the Solution Explorer.

## Explanation

Your project already contains a number of files which are more or less interesting for this workshop.

The first category Properties contains a file AssemblyInfo.cs which stores a number of information about your project. Feel free to look at them. It also contains files for resources (images, text and other) and a definition for possible settings you might want to safe later. You can ignore this category for now.

The References category shows a list of references to other code. We are going to use a lot of already existing functionality when writing our own code. Not all of the references are essential to what we will do, but you can leave the list as is for now.

The important files for this workshop are the two .xaml files and their corresponding .cs file. In WPF there is an explicit approach to separate the design from the code behind. The XAML files contain a special form of XML code to describe the layout and design of your application, while the CS files contain the C# code that will do all the background work.

In this workshop the default App.xaml provides all the functionality we need. The MainWindow.xaml file is what we will be working with and as the name states; it’s the main window of our application and entry point for the user.

# Let me see something!

## Goal

Compile the code and run the application.

## Instructions

Press F6 to compile (build) your code. Press F5 to compile (build) and run your code in debug mode.

Run and close your first (empty) application.

## Explanation

When compiling (building) your code, a tool will take all the code files in your project and put them together into one executable package which you may know as .exe files. You can either run the project directly from within Visual Studio or you can go to your projects directory and look for the executable file under the bin directory.

# Now I want to code something myself!

## Goal

Add a Button to your main window.

## Instructions

Open your MainWindow.xaml and locate the grid section and add a button element in between.

<Grid>

**<Button Content="This is my first button!"/>**

</Grid>

## Explanation

XAML is based on XML and works very similar in many aspects. The basic concept is as follows:

Every XAML document consists of a number of elements (surrounded by angle brackets). Every element has an opening and closing tag - like the Grid - unless you use the short form - like the Button. Depending on the type of element it can have none, one or many children. Any element can have an unlimited number of attributes - like Content. Every attribute can either be written as shown in the example or as nested element.

XAML elements always represent some kind of object that exists in the WPF framework. You can create a WPF user interface without any XAML, but it would be very inconvenient.

# Great, but it doesn’t do anything

## Goal

Add an event handler to the click event of the button and show a message box.

## Instructions

Within this button element write Click="" and you will now be prompted whether you want to add a new event handler. Use this to quickly create a method template you can use to execute your own code.

Open the MainWindow.xaml.cs file and locate the Button\_Click method. Within the curly braces of the method write the following code:

MessageBox.Show("Thanks for pressing me!");

Run the application and see what happens when you click the button.

## Explanation

Click is an event which exists for different controls, like a button. By adding an event handler to this event you can specify any number of actions that should happen whenever this event occurs. In this example we use an existing functionality that will open a message box and display any string you want.

# Now, let's do something useful

## Goal

Present the user with a dialog which lets him or her choose a folder to load images from and confirm the selection.

## Instructions

Right click the projects references and add a new reference to the .NET component named System.Windows.Forms.

Modify the existing button click event handler (Button\_Click) to present a FolderBrowserDialog to let the user choose any folder he or she likes.

**var browser = new System.Windows.Forms.FolderBrowserDialog();**

**browser.ShowDialog();**

MessageBox.Show("You chose: " + browser.SelectedPath);

## Explanation

The FolderBrowserDialog is a functionality that is not part of the default WPF components. We add the reference to use a piece of code that is located in another file of the .NET framework.

First we create a new instance (object) of the dialog and store it in the variable browser. Then we call the method ShowDialog which will present the dialog to the user and waits for a confirmation. The result of the selection can then be accessed via the SelectedPath property of the dialog.

# Let's read some data

## Goal

Read the content of the selected folder.

## Instructions

Use the following code to analyze the content of the selected folder (directory).

**var selectedDirectory = new System.IO.DirectoryInfo(browser.SelectedPath);**

**var files = selectedDirectory.GetFiles();**

MessageBox.Show("You chose: " + selectedDirectory + " containing " + files.Length + " files.");

Set a breakpoint at the first line by selecting the line and pressing F9. Run the application, choose a folder and step through the application line by line by pressing F10. Hover over any variable to view information about the object stored in that variable. Get an overview of all the properties that are available for use. Note that files variable contains an array of objects with information about any file in the chosen directory.

## Explanation

The .NET framework already provides functionality to read information about files and directories. Since the selected path is just a simple string to the system which does not know what it's content means, we first create a DirectoryInfo object. The DirectoryInfo object provides much information about the directory and has a method GetFiles to get the files inside. Every file is then itself described by a FileInfo object, which also contains lots of different properties about the file.

# How to display the images you promised?

## Goal

Add a control to your application which dynamically converts an unknown number data items into visual controls that can be displayed.

## Instructions

Add a DataGrid to the Grid in XAML and give it a name.

<Button Content="This is my first button!" Click="Button\_Click"/>

**<DataGrid x:Name="dataGridPictures">**

**</DataGrid>**

## Explanation

The DataGrid is an existing control which uses templates to convert a data item into any number of controls to represent that data. By default the DataGrid generates a text column for any property of a data item. This is why we don't need to define any templates for now. Note that we give the DataGrid a name dataGridPictures which allows us to address the control in the code behind.

# Cool, but where is my button?

## Goal

Set the layout of your application by defining a number of rows to place any controls at specific positions.

## Instructions

Add row definitions into you Grid.

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

Set the row number for each control.

<Button **Grid.Row="0"** Content="This is my first button!" Click="Button\_Click"/>

<DataGrid x:Name="dataGridPictures" **Grid.Row="1"**>

</DataGrid>

## Explanation

RowDefinitions is a property of Grid and contains a list of RowDefinitions which you can use to lay out your controls. The first row will automatically set its height based on the height of the biggest control in that row. This is why the button is getting smaller and will only use the space required to be correctly displayed. A row without a height defined will use all the space that is left.

Note that indices in C#/XAML are zero based. So row number zero is the first row.

# Connect data and user interface

## Goal

Present the file information in the DataGrid.

## Instructions

Uncomment the message box and use the array of file information as source for the DataGrid.

**//**MessageBox.Show("You chose: " + selectedDirectory + " containing " + files.Length + " files.");

**dataGridPictures.ItemsSource = files;**

## Explanation

A double forward slash is used in C# and many other languages as indicator, that the rest of the line is a comment and should be ignored by the compiler.

With a name set on the DataGrid, we can now access the object in our code and set the ItemsSource property to the array of FileInfo objects. Changing the ItemsSource will instruct the DataGrid to analyze the data and generate controls to represent that data.

# Customizing the data displayed

## Goal

Present only selected information about the files and show the actual image.

## Instructions

Modify the DataGrid so that columns are no longer auto-generated and specify your own columns.

<DataGrid x:Name="dataGridPictures" Grid.Row="1" **AutoGenerateColumns="False"**>

**<DataGrid.Columns>**

**<DataGridTemplateColumn Header="Picture" Width="100">**

**<DataGridTemplateColumn.CellTemplate>**

**<DataTemplate>**

**<Image Source="{Binding Path=FullName}"/>**

**</DataTemplate>**

**</DataGridTemplateColumn.CellTemplate>**

**</DataGridTemplateColumn>**

**<DataGridTextColumn Header="Name" Binding="{Binding Path=Name}"/>**

**</DataGrid.Columns>**

</DataGrid>

## Explanation

Similar to the Grid, we specify a list of column definitions. There are various types of columns, but all have one thing in common. A DataTemplate is used to describe the visual representation of data. Since displaying an image not a common task we have to create our own DataTemplate, while there is a predefined column for text.

Please take note of the Binding element in curly braces. This element is a crucial part of WPF, because it allows you to quickly define a connection between different properties. In this example we are binding the content of the FullName property of the FileInfo object to the Source property of the Image object. Same goes for the text column, where we bind the Name property of the FileInfo object to the Text property of the text column.

# Congratulations

You have done it! You have created your (first) WPF application which references other code, interacts with the user, reads files, contains various controls, templates, bindings and a bit of magic.

# More customization

## Goal

Make your application look cool.

## Instructions

Here are a few examples of what you can do:

* Adjust the grid lines of the DataGrid (GridLinesVisibility)
* Convert the DataGridTextColumn into a DataGridTemplateColumn with a TextBlock element inside and set various properties, like VerticalAlignment and FontSize
* Add a search pattern to GetFiles to filter for images.
* Add another row to the Grid and display the selected folder in a TextBlock.
* …