



WPF

Introduzione a WPF e XAML

WPF

WPF, abbreviazione di Windows Presentation Foundation, è un UI framework usato per la costruzione di applicazioni desktop per Windows.

Fa parte del framework .NET e fornisce un modo per creare interfacce utente interattive utilizzando XAML.

Consente agli sviluppatori di creare interfacce utente reattive e adattive che si regolano automaticamente a varie dimensioni e risoluzioni dello schermo (responsive).





Supporta grafica e animazioni 2D e 3D, nonché grafica vettoriale e funzionalità multimediali.

Facilita la gestione e i flussi di dati tra l'interfaccia utente e la logica dell'applicazione, grazie al supporto integrato per l'associazione dei dati.

Primo progetto

Crea un nuovo progetto


Modelli di progetto recenti

 App Windows Forms (.NET Framework)	C#
 App Windows Forms	C#
 App console	C#
 Applicazione Python	Python


Cerca modelli (ALT+S) 

[Deseleziona tutto](#)


C# Windows Desktop

 App Windows Forms
Modello di progetto per la creazione di un'app Windows Forms (WinForms) .NET.

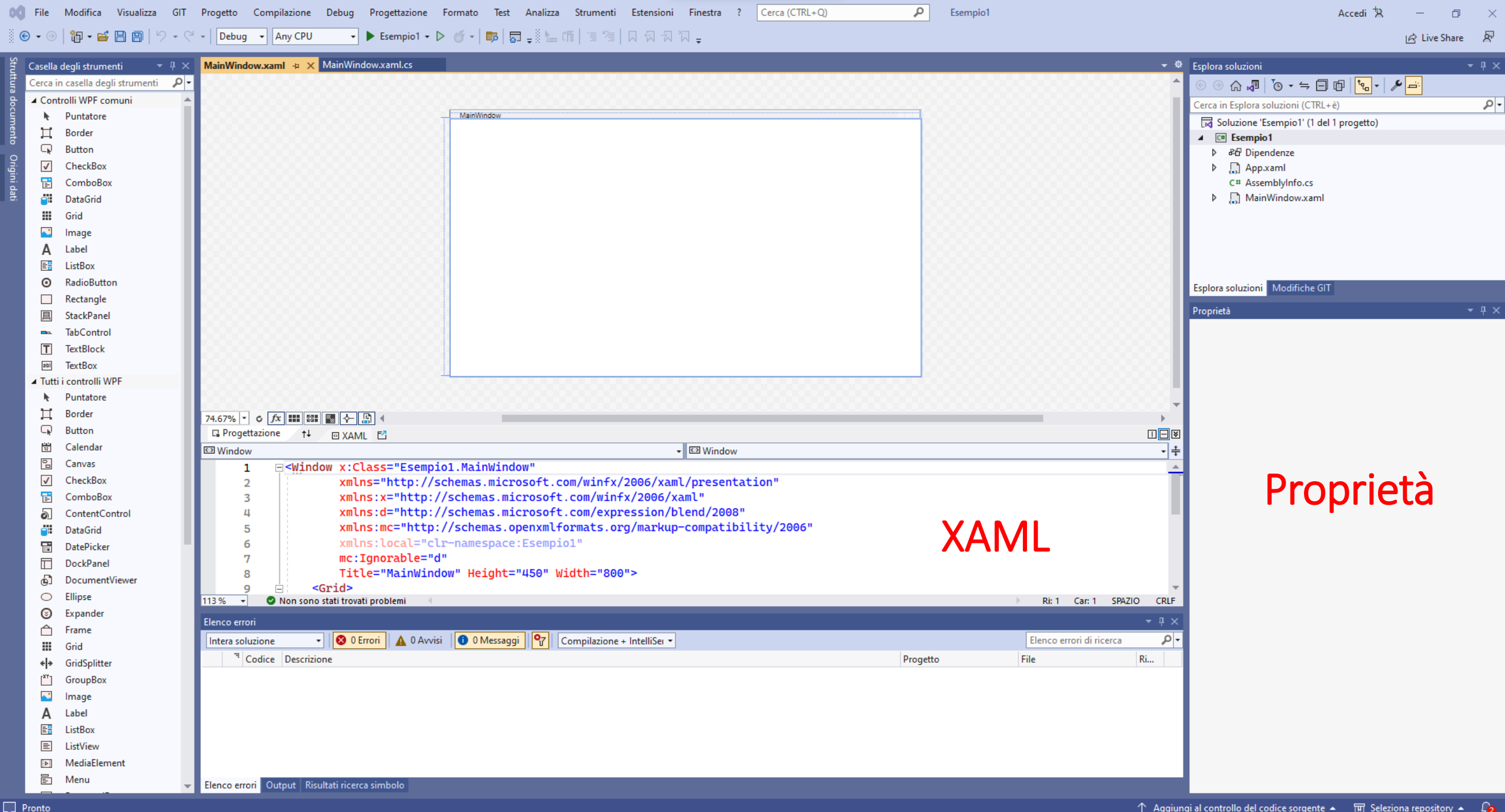
C# Windows Desktop

 App Windows Forms (.NET Framework)
Progetto per la creazione di un'applicazione con interfaccia utente Windows Forms (WinForms)

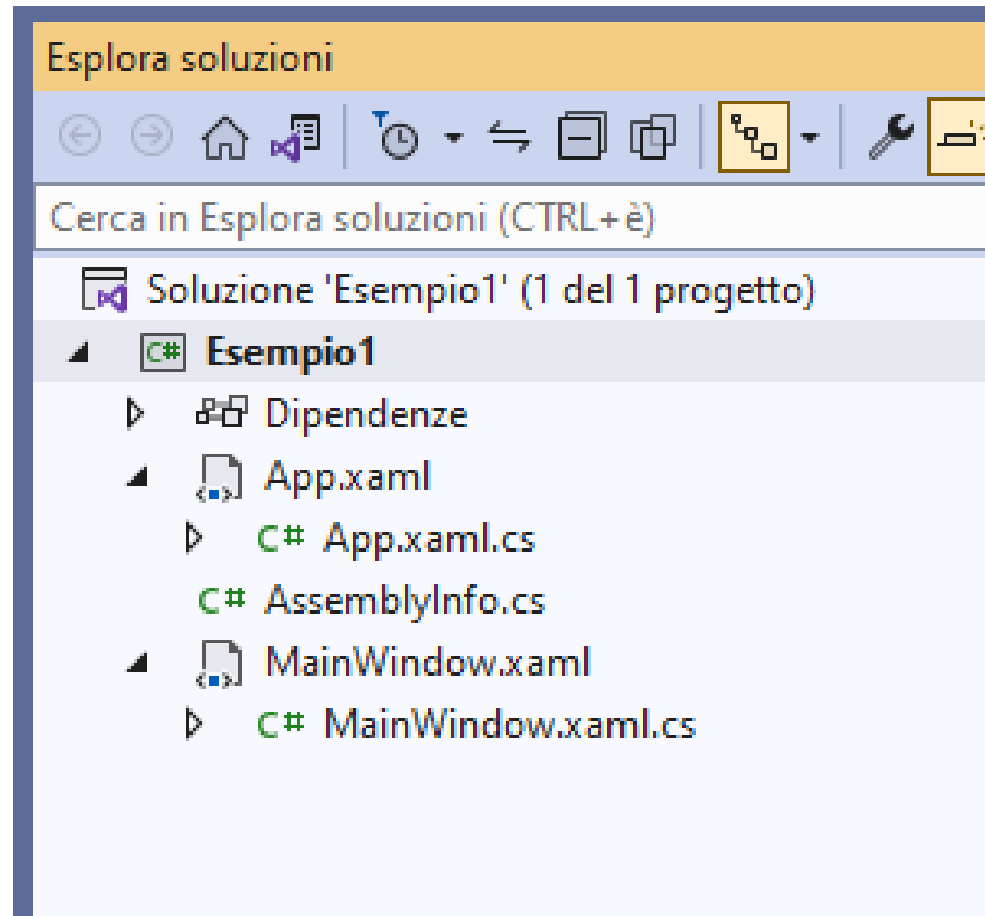
C# Windows Desktop

 Applicazione WPF
Progetto per la creazione di un'applicazione WPF .NET

C# Windows Desktop



Soluzione progetto



XAML

XAML è un linguaggio di markup basato su XML il quale permette di implementare l'aspetto (UI) di un'applicazione in modo dichiarativo.

Viene in genere usato per creare finestre, finestre di dialogo, pagine e controlli utente e per inserire in questi elementi controlli, forme e grafica.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Title="Window with Button"
  Width="250" Height="100">
  <Button Name="button">Click Me!</Button>
</Window>
```

Dichiarativo

La differenza fondamentale tra la programmazione dichiarativa e quella imperativa è che la programmazione dichiarativa si concentra su ciò che il programma dovrebbe realizzare mentre la programmazione imperativa si concentra su come il programma dovrebbe raggiungere il risultato.

App.xaml

Application x:Class="_01_BaseWPF.app"

↑ ↑
Namespace Classe

App.xaml

```
<Application x:Class="_01_BaseWPF.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Esempio1"
    StartupUri="MainWindow.xaml">
  <Application.Resources>

  </Application.Resources>
</Application>
```

App.xaml.cs

```
namespace Esempio1
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        ....
    }
}
```


App.xaml

StartupUri="MainWindow.xaml"



Prima view mostrata al momento di esecuzione del progetto

App.xaml

```
<Application x:Class="_01_BaseWPF.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Esempio1"
    StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```



MainWindows.xaml

<Window x:Class="_01_BaseWPF.MainWindow"

↑ ↑
Namespace Classe

MainWindows.xaml

```
<Window x:Class="_01_BaseWPF.MainWindow"
    ....
</Window>
```

















MainWindows.xaml.cs

```
namespace _01BaseWPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow :
    Window
    {
        public MainWindow()
        {
            ....
        }
    }
}
```

Controlli

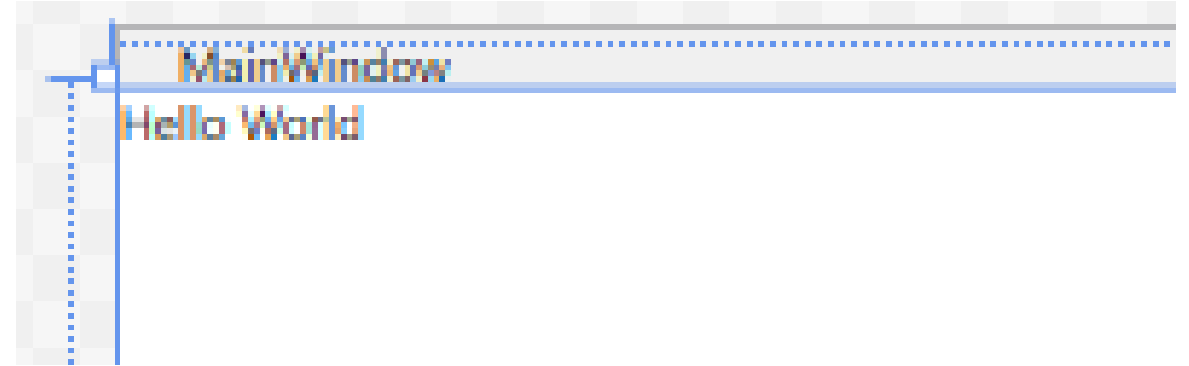

I controlli Windows Presentation Foundation (WPF) possono essere raggruppati logicamente in diverse categorie.

<https://learn.microsoft.com/it-it/dotnet/desktop/wpf/controls/controls-by-category?view=netframeworkdesktop-4.8>

	Puntatore
	Border
	Button
	CheckBox
	ComboBox
	DataGrid
	Grid
	Image
	Label
	ListBox
	RadioButton
	Rectangle
	StackPanel
	TabControl
	TextBlock
	TextBox

Hello World

Progettazione UI



```
<Window x:Class="_01_BaseWPF.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:Esempio1"
  mc:Ignorable="d"
  Title="MainWindow" Height="450" Width="800">
  <Grid>
    <Label x:Name="lHello" Content="Hello World"/>
  </Grid>
</Window>
```

Proprietà controlli

Come in Windows Form, anche WPF mette a disposizione diverse proprietà legate ai vari controlli.


Di seguito ne vediamo alcune inerenti il controllo Label.


```
...  
<Grid>  
    <Label x:Name="IHello" Content="Hello World" Foreground="Red" HorizontalAlignment="Center"/>  
</Grid>  
...
```

X:Name

Identifica in modo univoco gli elementi definiti in XAML.

```
...  
<Grid>  
    <Label x:Name="lHello" Content="..." Foreground="Blue" HorizontalAlignment="Center"/>  
    <Button x:Name="bSaluta" Content="Saluta" Foreground="Red" VerticalAlignment="Center"/>  
</Grid>  
...
```

A	Nome	<input type="text" value="lHello"/>	 
	Tipo	<input type="text" value="Label"/>	

	Nome	<input type="text" value="bSaluta"/>	 
	Tipo	<input type="text" value="Button"/>	

Evento

XAML

```
...
<Grid>
    <Button x:Name="bSaluta" Content="Saluta" Foreground="Red" VerticalAlignment="Center" Click="Button_Click"/>
    <Button x:Name="bSaluta" Content="Saluta" Foreground="Red" VerticalAlignment="Center"/>
</Grid>
...
```

MainWindow.xaml.cs

```
...
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        IHello.Content = "Hello World";
    }
}
...
```