

Desenvolupament d'interfícies

Unitat 2. Plataforma .NET i Llenguatge de programació al disseny d'interfícies



Índex

1.	Recordem la Programació Orientada a Objectes (POO)	2
1.1.	Classes.....	2
1.2.	Propietats o atributs.....	2
1.3.	Mètodes.	2
2.	.NET Framework.	3
2.1.	Característiques .NET.....	3
2.2.	Components.....	4
3.	L'entorn de desenvolupament VSC (Visual Studio Community)	6
4.	WPF.....	11
4.1.	XAML.	13
4.2.	Codificant amb C#.....	14
5.	Layout.....	14
5.1.	GridPanel	16
5.2.	StackPanel	18
5.3.	DockPanel.....	19
5.4.	WrapPanel	20
5.5.	CanvasPanel.....	20
6.	Estils.....	21
6.1.	Nivells.....	22
6.2.	Destinació.	23
6.3.	Triggers (<i>disparadors</i>).	25
7.	Temes	27
7.1.	Temes nadius VSC.	28
7.2.	Temes de tercers (Paquet MahApps).	29
7.3.	Temes de tercers (Paquet Material Design).	34

1. Recordem la Programació Orientada a Objectes (POO)

El desenvolupament d'interfícies gràfiques permet la creació del canal de comunicació entre l'usuari i l'aplicació, per aquesta raó requereix d'especial atenció en el seu disseny. En l'actualitat, les eines de desenvolupament permeten la implementació del codi relatiu a una interfície a través de vistes disseny que faciliten i fan més intuïtiu el procés de creació. La programació orientada a objectes permet utilitzar entitats o components que tenen la seua pròpia identitat i comportament.

Començarem veient els principals tipus de components així com les seues característiques més importants. La distribució d'aquesta mena d'elements depèn dels anomenats **contenidors o layout**, els quals permeten col·locar els elements en un lloc o en un altre.

1.1. Classes.

Recordem que una classe representa un conjunt d'objectes que comparteixen una mateixa estructura (**PROPIETATS**) i comportament (**MÈTODES**). A partir d'una classe es podran *instanciar* tants objectes corresponents a una mateixa classe com es necessite. Per a això s'utilitzen els constructors.

Per a dur a terme la **instanciació** d'una classe i així crear un nou objecte, s'utilitza el nom de la classe seguit de parèntesi. Un constructor és sintàcticament molt semblant a un mètode.

RECORDA TAMBÉ QUE: El constructor pot rebre arguments, d'aquesta manera podrà crear-se més d'un constructor, en funció del nombre d'arguments que s'indiquen en seua definició. Encara que el constructor **no haja sigut definit** explícitament, a Java sempre existeix un **constructor per defecte** que posseeix el nom de la classe i no rep cap argument.

1.2. Propietats o atributs.

Un objecte és una càpsula que conté totes les dades i mètodes lligats a ell. La informació continguda en l'objecte serà accessible solo a través de l'execució dels mètodes adequats, creant-se una interfície per a la comunicació amb el món exterior.

Els atributs defineixen les característiques de l'objecte. Per exemple, si es té una classe cercle, els seus atributs podrien ser el radi i el color, aquests constitueixen l'estructura de l'objecte, que posteriorment podrà ser modelada a través dels mètodes oportuns.

L'estructura d'una classe a Java quedaria formada pels següents blocs, de manera general: **atributs, constructor i mètodes**.

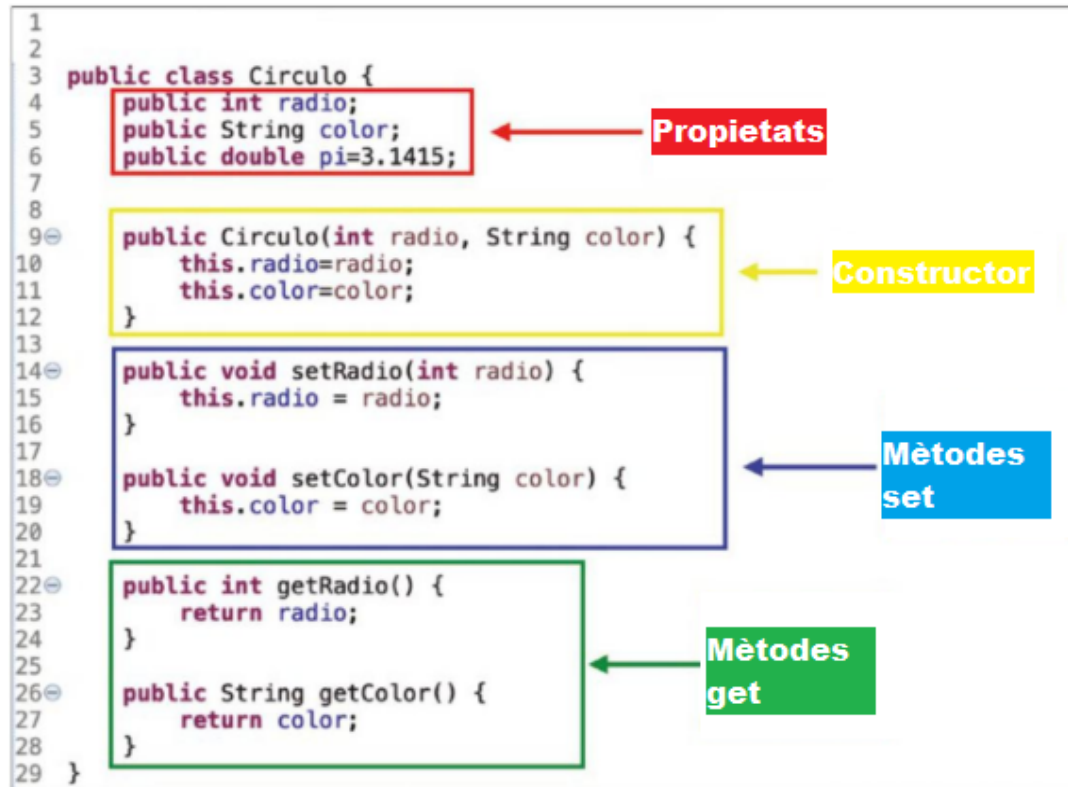
1.3. Mètodes.

Per últim, els mètodes defineixen el comportament d'un objecte, això vol dir que tota aquella acció que es desitja realitzar sobre la classe ha d'estar prèviament definida en un mètode.

Els mètodes (al igual que els constructors) poden rebre o no arguments, a més, en funció de la seua definició, retornaran un valor o realitzaran alguna modificació sobre els atributs de la classe.

Tradicionalment els mètodes anomenats get tenim la funció de tornar/obtindre informació i els mètodes anomenats set s'utilitzen per modificar informació/propietats de l'objecte.

Vegem-lo tot junt en una imatge a mode de resum/recordatori:



2. .NET Framework.

Com a resposta a la plataforma de desenvolupament de Java, Microsoft construeix la seua pròpia que anomena **.NET**.

.NET Framework és un entorn d'execució administrat que proporciona diversos serveis a les aplicacions en execució. Es tracta d'un entorn de desenvolupament d'aplicacions que ens aporta eines, biblioteques i entorns d'execució per a les nostres aplicacions desenvolupades en diferents llenguatges de programació.

Les aplicacions desenvolupades amb aquesta plataforma són interpretades per un entorn comú d'execució, similar a la màquina virtual de Java..

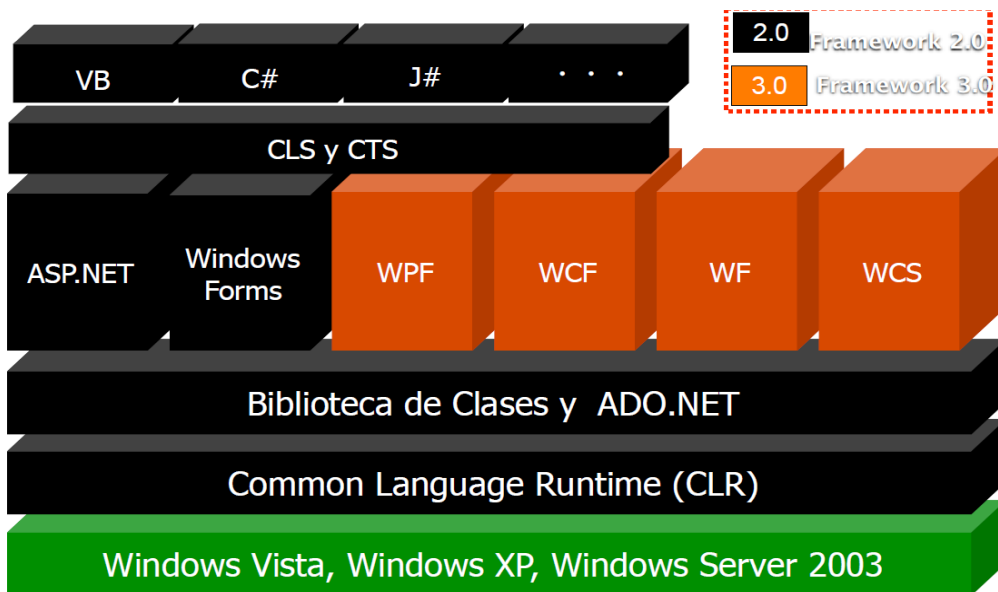
2.1. Característiques .NET.

- Les aplicacions .NET es poden desenvolupar en qualsevol llenguatge de programació que s'ajuste a les especificacions.
- Suporta una extensa col·lecció de llibreries de classes independents del llenguatge de programació per a la realització de tasques comunes d'accés a dades, sobre seguretat, ...

- Suporta la creació de components.
- Ofereix integració multi-llenguatge, reutilització de components, i herència entre components desenvolupats en diferents llenguatges.
- Ofereix una nova manera de desenvolupar aplicacions gràfiques usant WPF (Windows Presentation Foundation)
- Ofereix una nova manera de desenvolupar aplicacions basades en navegador Web a través d'ASP.NET
- Els tipus d'objectes són especificats pel framework i no pel llenguatge.

2.2. Components.

En la següent figura es mostra l'arquitectura i principals parts de l'entorn .NET



En la part més alta de la figura tenim els diferents llenguatges que podem utilitzar: Visual Basic, C#, J# (Java dissenyat per Microsoft), NETCOBOL ...

Després tenim el llenguatge intermedi que té la plataforma, igual que succeeix amb Java, en concret amb els bytecodes.

A continuació tenim les biblioteques amb funcionalitats comunes per a la major part d'aplicacions.

Finalment, tenim la capa de CLR que s'encarrega d'executar les aplicacions, de la mateixa forma que la màquina virtual de Java.

Common Language Specification (CLS)

Es tracta d'un llenguatge intermedi que genera el compilador del llenguatge utilitzat, és a dir, nosaltres programem en el llenguatge que vulguem, i després el compilador genera aquest llenguatge intermedi semblant als bytecodes de Java.

En anteriors versions aquest llenguatge intermedi rebia el nom de MSIL.

Actualment hi ha uns 60 llenguatges de programació que tenen compiladors que generen el codi intermedi.

Biblioteca de Clases

Es tracta d'una col·lecció de codi OO (orientat a objectes) que pot ser emprat des de qualsevol llenguatge .NET.

Defineix els tipus bàsics, classes per a l'entrada/eixida, seguretat, etc.

En tindre definits els tipus de dades per a tots els llenguatges, facilita l'intercanvi d'informació entre aplicacions desenvolupades en diferents llenguatges.

També disposa de la possibilitat de realitzar un accés a dades mitjançant **ADO.NET**; igual que el tractament de fitxers XML

També podem desenvolupar interfícies mitjançant **Windows Presentation Forms (WPF)**, generar pàgines web amb **ASP.NET**.

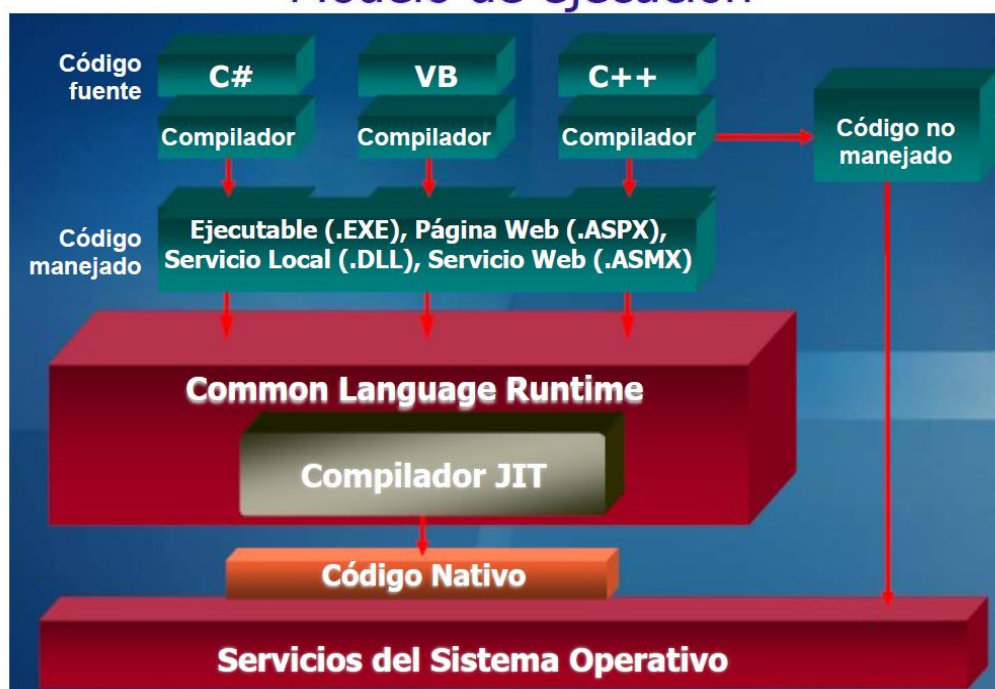
Common Language Runtime (CLR)

Un Runtime és un entorn en el qual s'executen els programes, en concret est s'encarrega d'executar les aplicacions de .NET

El CLR realitza una compilació Just in time (JIT) que tradueix el codi gestionat en codi natiu sobre l'arquitectura de maquinari sobre la qual s'executa. És a dir que interpreta el codi intermedi d'igual manera que fa la màquina virtual de Java.

El model d'execució que realitza és el següent:

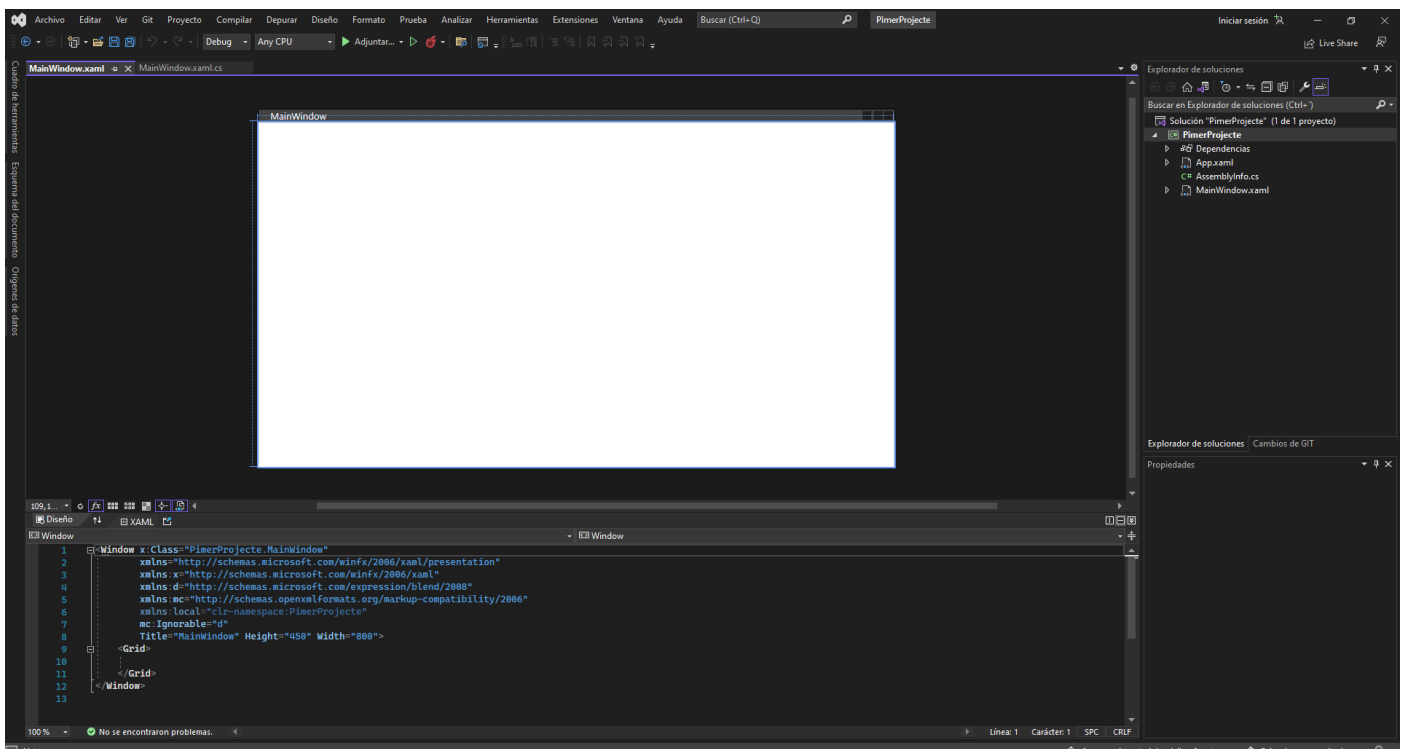
Modelo de ejecución



S'encarrega de tasques com carregar les caleres, recol·lector de fem, control de la pila, interactuar amb el sistema operatiu, control d'excepcions, control de seguretat o compilador de codi natiu.

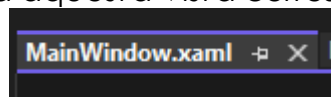
3. L'entorn de desenvolupament VSC (Visual Studio Community)

Visual Studio Community és una eina molt completa. Amb aquesta eina podrem crear la interfície de forma gràfica, i a la mateixa vegada treballar sobre amb codi. Primer identificarem les diverses àrees de l'entorn VSC.



A l'entorn VSC podem observar diferents àrees o zones de treball quan tenim un projecte obert. El primer que bot a la vista és el disseny visual, en blanc podem observa el disseny gràfic de la finestra de la aplicació, sobre ella podem arrossegar controls, canviar el seu aspecte, etc.

Si observeu dalt a la dreta aquesta vista correspon a tindre seleccionat o obert el fitxer XAML del projecte

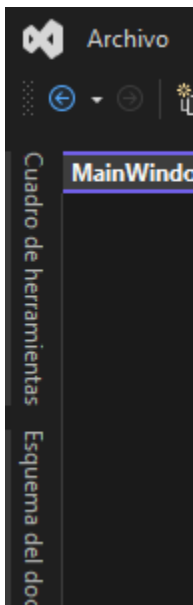


I baix tenim una zona on es veu en llenguatge XAML el codi font resultant del disseny visual de la part de dalt. Per a una finestra buida el codi és el que es veu en la primera imatge, ampliat seria aquest:

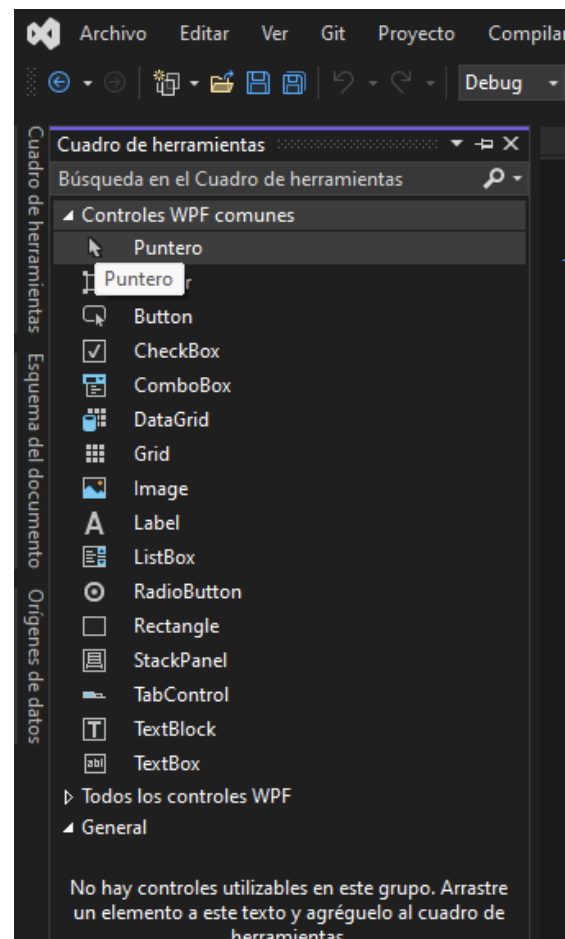
```
109, 1... fx
Diseño XAML
Window
1 <Window x:Class="PimerProjecte.MainWindow"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6     xmlns:local="clr-namespace:PimerProjecte"
7     mc:Ignorable="d"
8     Title="MainWindow" Height="450" Width="800">
9     <Grid>
10
11    </Grid>
12 </Window>
13
```

Aquest codi XAML canvia dinàmicament si anem afegint controls o canvis sobre el disseny visual de dalt.

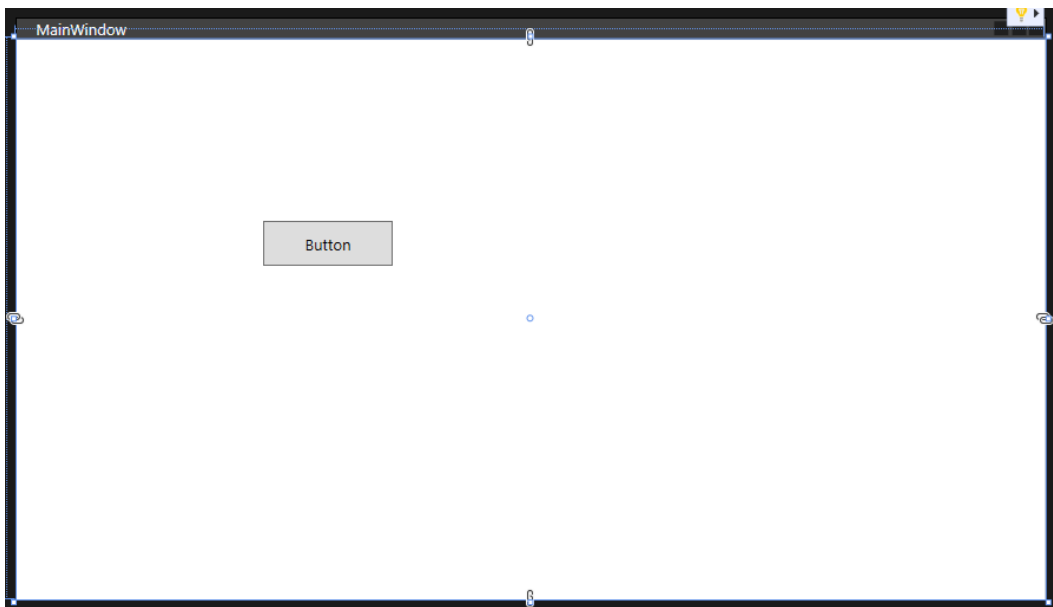
Per afegir un control de forma visual, primer hem de fer visible el **Quadre d'eines**. El Quadre d'eines està disponible dalt a l'esquerra:



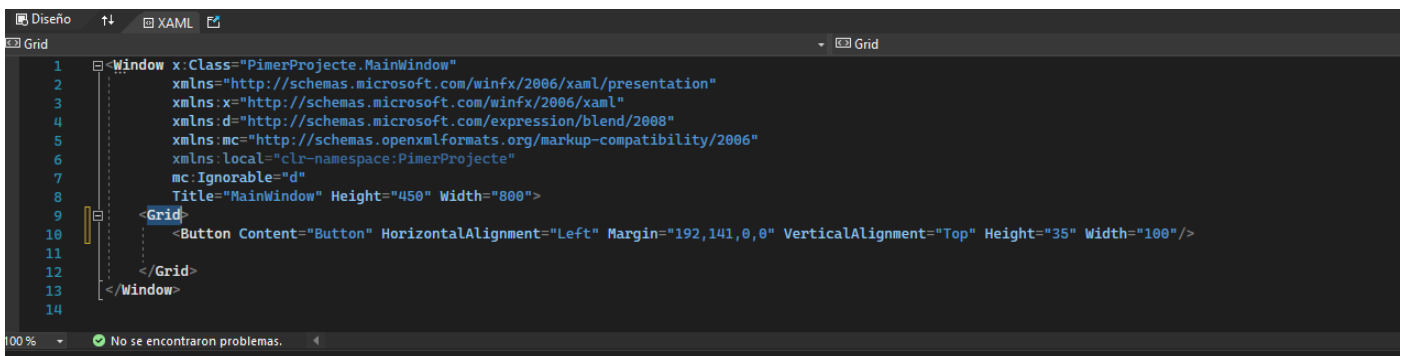
Polsant sobre la pestanya es desplega el **Quadre d'eines**



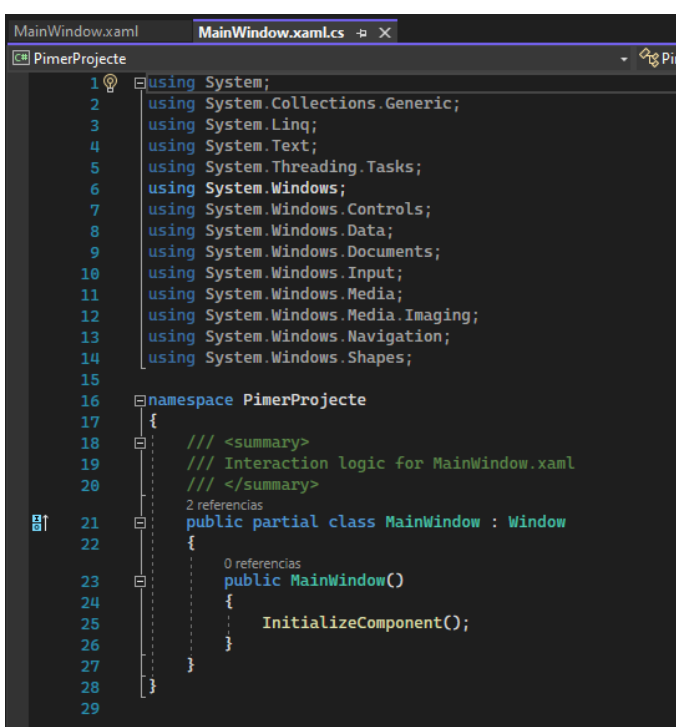
Per a afegir un control simplement hem d'arrossegar un d'ells, per exemple un botó, sobre la finestra inicial del projecte:



I immediatament s'actualitza el codi XAML en la finestra de baix (observeu l'etiqueta *Button* que s'ha generat):

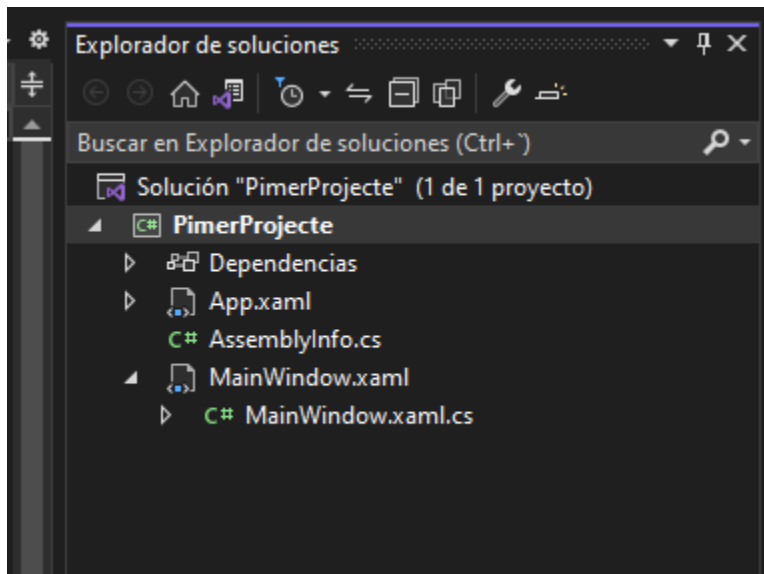


D'altra banda, si canviem de pestanya (polsem en la pestanya de la dreta del fitxer `MainWindow.xaml`, on diu **MainWindows.xaml.cs**) podem observar el fitxer C# associat al projecte:

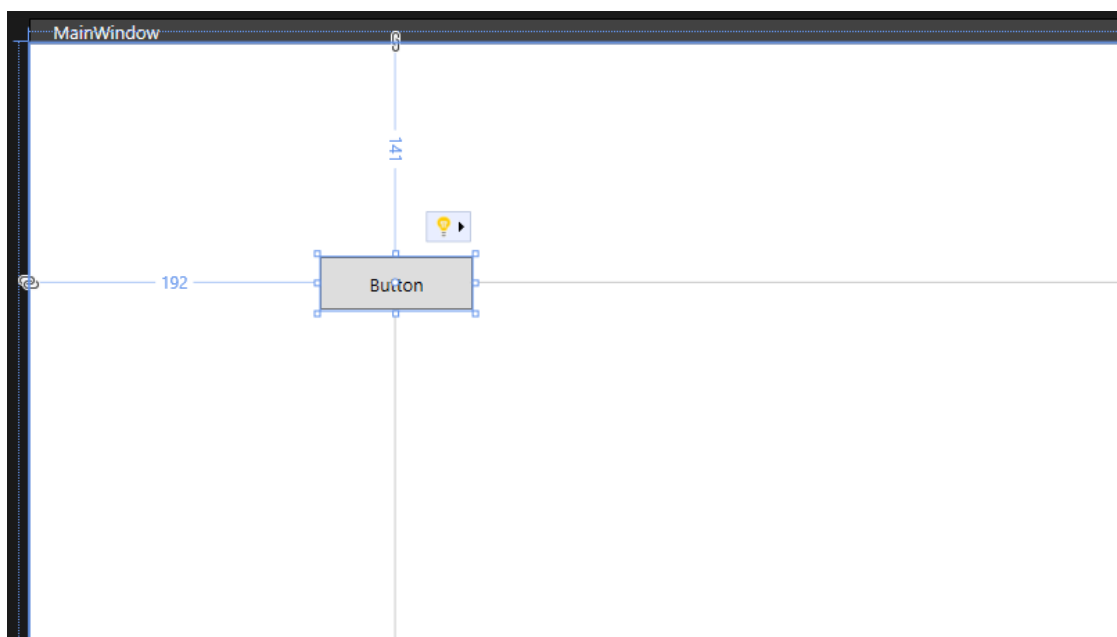


En aquest fitxer podem afegir codi que faci el mateix que el codi XAML, es a dir representar l'interfície, tant codi associat a altres aspectes de l'aplicació: control d'esdeveniments, accés a dades, ...

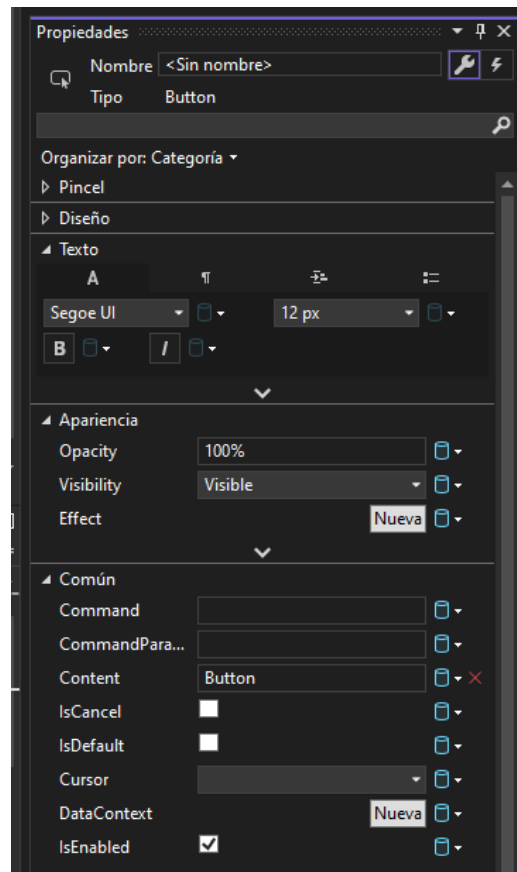
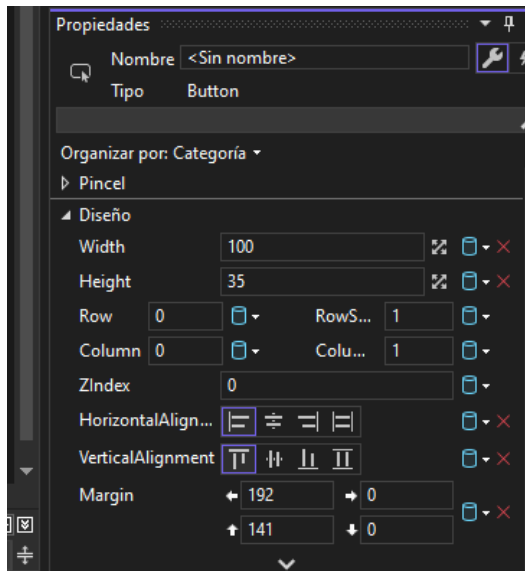
I per últim a la dreta tenim l'explorador de solucions, on podem veure l'arbre complet dels fitxers que componen el projecte:



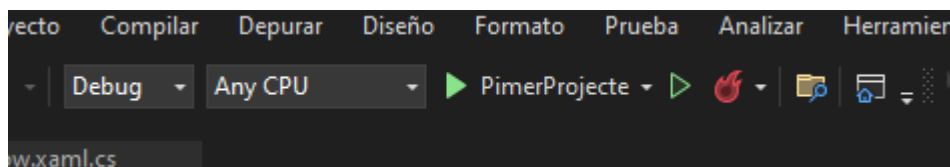
També dir que des de el disseny gràfic del projecte podem triar qualsevol control, seleccionar-lo i modificar les seues propietats (color, alineació, tipus de lletra, ..., inclús el comportament del mateix):



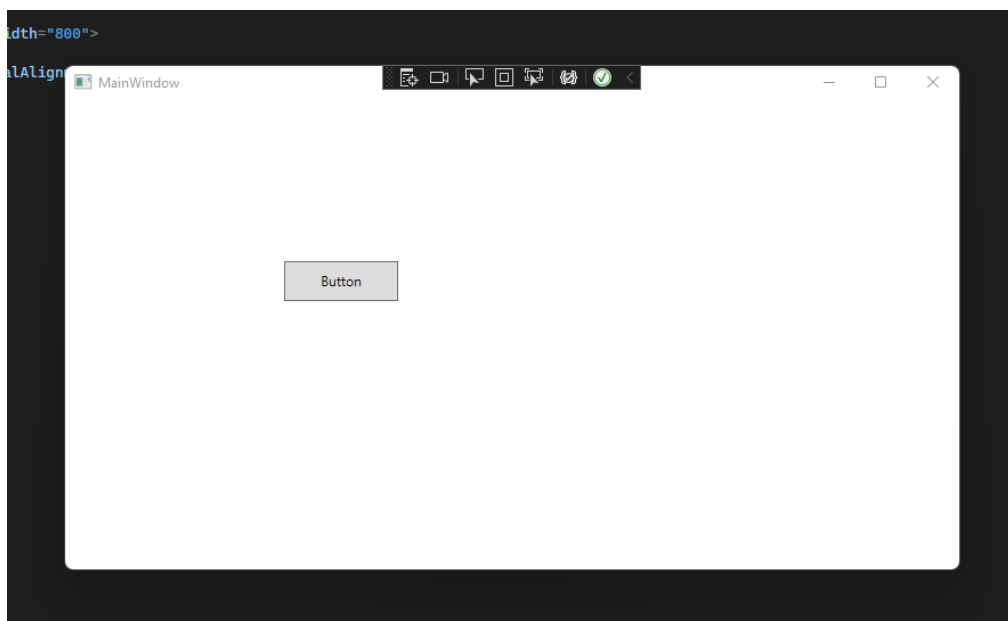
Aquestes propietats apareixen baix a la dreta (baix del explorador de solucions). Podem alterar propietats de disseny, de aparença, de text, ...):



Per acabar dalt al centre tenim els botons per a executar l'aplicació, parar-la i executar-la pas a pas:



Si polsem el triangle verd executarem l'aplicació:



En definitiva és un entorn molt complet i al principi costa trobar les coses (com Eclipse, Netbeans, ...), però amb una corba d'aprenentatge ràpida. No s'ha de tindre por a experimentar amb ell.

Podeu trobar una referència més completa de l'entorn VSC en aquest enllaç: <https://docs.microsoft.com/es-es/dotnet/desktop/wpf/get-started/create-app-visual-studio?view=netdesktop-6.0>

4. WPF

Windows Presentation Foundation (WPF) és una tecnologia de Microsoft. Permet el desenvolupament d'interfícies d'interacció en Windows.

Aquesta tecnologia ve a substituir a Windows Forms que s'utilitzava en la versió 2 de .NET Framework.

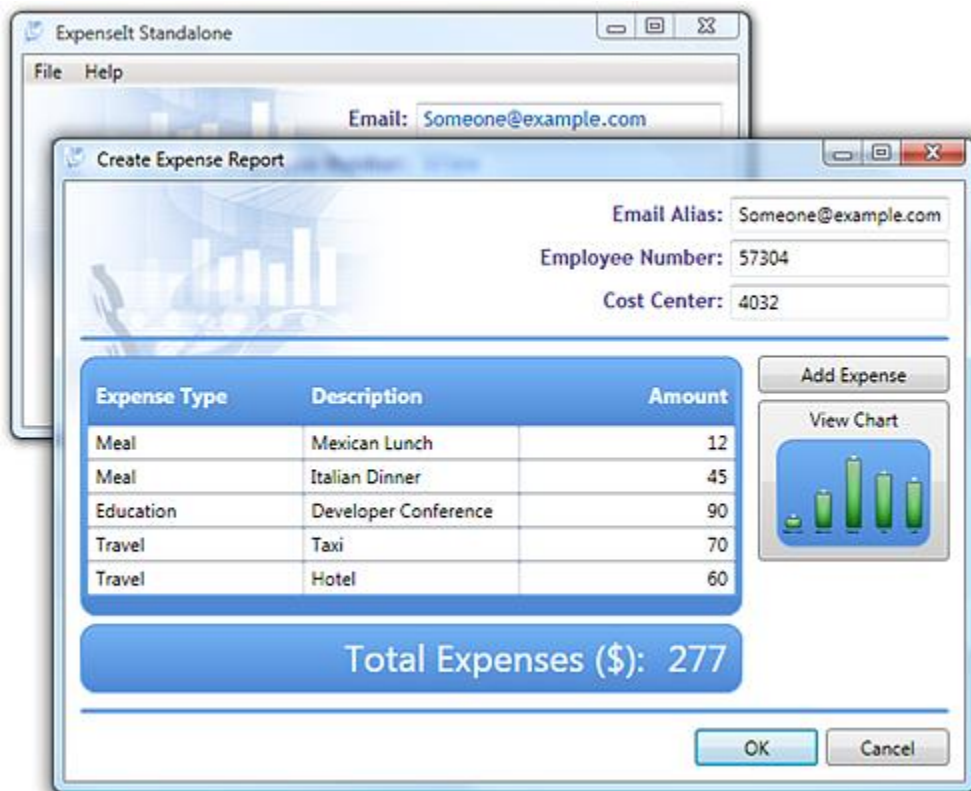
WPF ofereix una àmplia varietat de controls i components gràfics amb la qual és possible desenvolupar aplicacions visualment atractives, amb facilitats d'interacció que inclouen animació, vídeo, àudio, documents, navegació o gràfics 3D.

Separa clarament la definició de la pantalla o interfície de la programació de les seues funcionalitats, seguint de manera clara el patró de disseny Model Vista Controlador.

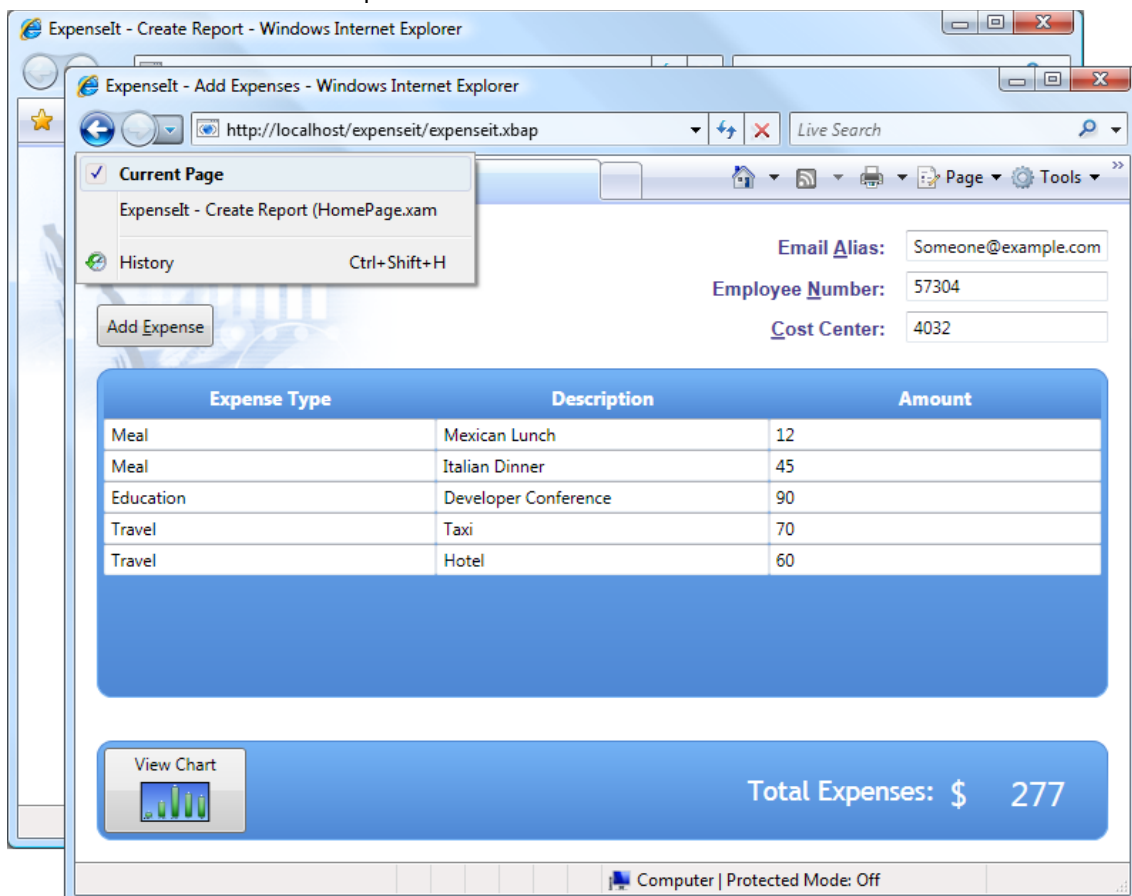
Per a especificar la interfície utilitza un llenguatge basat en XML anomenat XAML, mentre que les funcionalitats dels diferents components les realitza en el llenguatge triat d'entre l'àmplia gamma que es pot utilitzar.

També ens permet desenvolupar dues modalitats d'aplicacions:

- Aplicacions independents, a l'estil tradicional de Windows Forms que s'instal·len en l'equip client i s'executen des d'ell. Modalitat que he utilitzat per a aquest projecte.



- Aplicacions denominades XAML Browser Applications (XBAPs). Són aplicacions compostes de pàgines de navegació que es compilen i s'allotgen en exploradors web com a Internet Explorer o Mozilla.



L'ús de WPF està enfocat a aplicacions que fan ús de contingut multimèdia (vídeos, àudio, text enriquit) com per a facilitar la creació d'aplicacions empresarials de "tipus escriptori" o aplicacions web enriquides.

4.1. XAML.

Extensible Application Markup Language és un llenguatge de marques basat en XML que permet el disseny d'una interfície d'usuari d'una aplicació. XAML simplifica la creació d'interfícies d'usuari mitjançant la separació dels elements de desenvolupament i disseny de l'aplicació. XAML està en arxius de text que estan separats del codi de l'aplicació.

XAML és fàcil de llegir i entendre, i pot ser editat amb qualsevol editor de text. Es guarda en arxius de text amb una extensió d'arxiu .xaml.

Els objectes declarats mitjançant el llenguatge XAML tenen la seua corresponent instanciació en les classes del llenguatge utilitzat, de manera que podem utilitzar-los en el nostre programa de la mateixa manera que faríem si el definíem per codi.

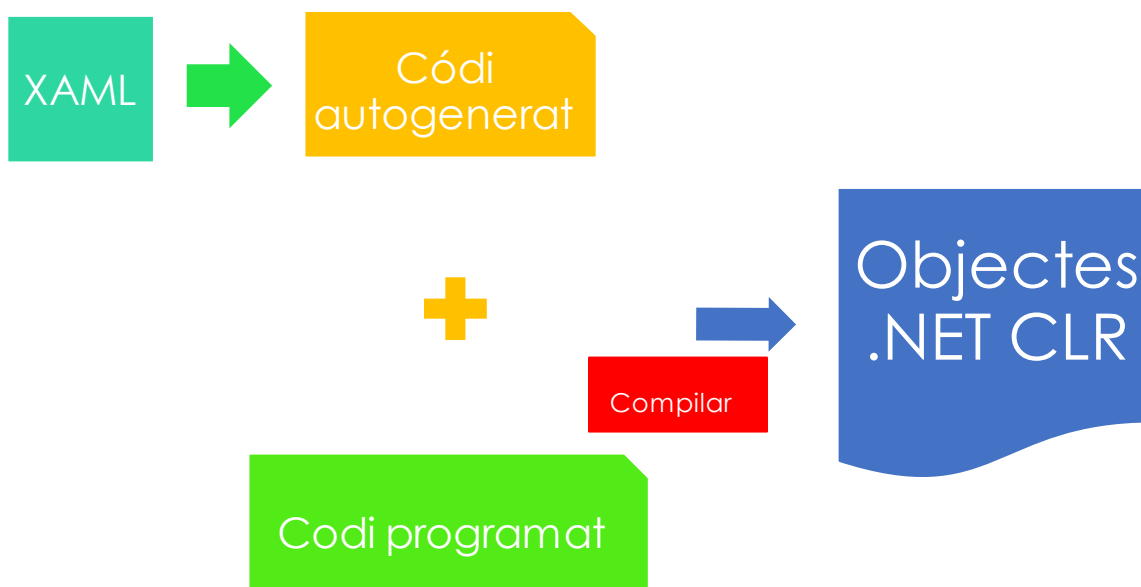
Existeix una classe que s'encarrega de llegir els objectes declarats en XAML i els crea a l'entorn d'execució del llenguatge utilitzat, en el nostre cas C#. Després des de C# podem utilitzar aqueixos objectes per a modificar el seu comportament, realitzar operacions sobre la base de dades o mostrar informació.

El mètode que s'utilitza per a carregar un fitxer XAML des d'un programa escrit en C#, o un altre llenguatge qualsevol, és **InitializeComponent()**. Aquest mètode llig el fitxer xaml, realitza un parse de xml perquè la sintaxi siga correcta i càrrega a l'entorn d'execució de l'aplicació els objectes que representen als diferents controls que es troben especificats en la nostra interfície.

Quan generem una interfície amb XAML, l'entorn crea automàticament unes classes amb el codi que representa els nostres components o controls.

Després unim aquest codi autogenerat al codi que nosaltres hem realitzat per a formar els objectes que els identifiquen en el nostre CLR (màquina virtual) i poder executar el programa.

En el següent esquema podem observar el procés:



A continuació tenim un exemple de declaració d'un parell de controls en XAML.

```
<StackPanel>
    <TextBlock Margin="20">Welcome to the World of XAML</TextBlock>
    <Button Margin="10" HorizontalAlignment="Right">OK</Button>
</StackPanel>
```

Aquest codi xml utilitza un layout o manera de col·locar elements mitjançant les etiquetes StackPanel. Després afegim un títol mitjançant les etiquetes TextBlock amb un marge de 20 px. Posteriorment afegim un botó que alineem a la dreta amb una distància de 10 px al marge, el text que mostra el botó OK.

Podeu consultar la guia completa de XAML ací: <https://docs.microsoft.com/es-es/windows/uwp/xaml-platform/xaml-syntax-guide>

4.2. Codificant amb C#.

També podem realitzar el mateix mitjançant C#, en el següent quadre es mostra l'exemple:

```
// Create the StackPanel
StackPanel stackPanel = new StackPanel();
this.Content = stackPanel;

// Create the TextBlock
TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to the World of XAML";
stackPanel.Children.Add(textBlock);

// Create the Button
Button button = new Button();
button.Margin = new Thickness(20);
button.Content = "OK";
stackPanel.Children.Add(button);
```

5. Layout

Les aplicacions tenen una premissa clara, es poden executar en qualsevol mena de plataforma, això que suposa un gran avantatge, ens obliga a anar amb compte amb el disseny de les interfícies gràfiques perquè els components es desquadraran en canviar de plataforma.

Per a solucionar-ho es va introduir el terme de layout, **amb això es pretén posicionar els elements o components en el nostre formulari independentment de la font, de la resolució de la pantalla o les diferències entre plataformes.**

També ens permetrà facilitar la traducció, ja que si augmenta la grandària de la cadena també el farà el component proporcionalment.

Un layout manager no és més que un delegat que s'encarrega d'organitzar els components que formen part d'un contenidor com per exemple puga ser una finestra. El layout manager és l'encarregat de decidir en quines posicions es renderitzaran els components, que grandària tindran, que porció del contenidor abastaran, etc. Tot això es realitza d'una manera transparent al programador que per tant s'estalvia l'haver d'escriure una gran quantitat de línies de control.

Els diferents layouts poden mesclar-se dins d'una mateixa aplicació per a aconseguir l'efecte desitjat, no són excloents.

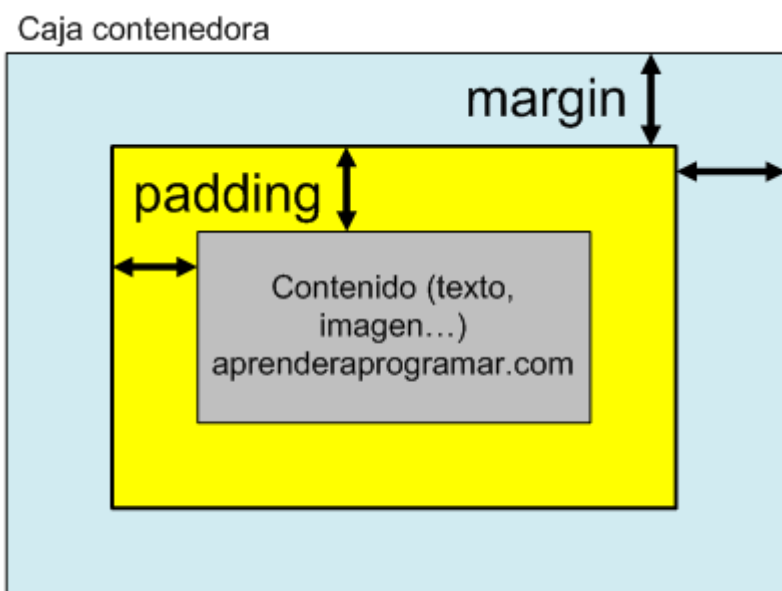
Perquè la nostra aplicació resulte versàtil i es puga adaptar a diferents resolucions, hem de seguir una sèrie de recomanacions:

- **Evita les posicions fixes i utilitza les propietats d'alineació (Alignment) i de marge (Margin) per a posicionar els elements en el panell.**
- **Evita les grandàries fixes per als controls. Utilitza el valor Acte sempre que siga possible.**
- **No abusar de Canvas Panell per a situar controls**
- **Utilitza un GridPanel per a col·locar els components dels formularis**

A continuació es mostra una imatge que ens especifica els valors de l'alineament vertical i horitzontal:

		HorizontalAlignment			
		Left	Center	Right	Stretch
VerticalAlignment	Top	Button	Button	Button	Button
	Center	Button	Button	Button	Button
	Bottom	Button	Button	Button	Button
	Stretch	Button	Button	Button	Button

En la següent figura s'explica el sentit de les propietats Margin i Padding:



5.1. GridPanel

Aquest layout estructura els components com una taula, és a dir per files i columnes. Té la capacitat de poder incloure diversos components en una cel·la, es poden expandir entre diverses cel·les, tant vertical com horitzontalment.

Per defecte es crea amb una sola fila i **columna**, la resta han de definir-se mitjançant **RowDefinition** o amb **ColumnDefinition**.

Per a especificar la **grandària de la fila o columna** podem especificar un valor **absolut**, un **percentatge** o deixar-lo amb un valor **automàtic**.

Si posem l'etiqueta **Auto** la columna o fila ocuparà l'espai exacte que necessita el seu contingut. Mentre que si posem ***** llavors la fila o columna ocuparà el màxim espai que puga.

Si per contra volem que la nostra columna o fila tinguin un valor exacte, llavors posarem el valor numèric, expressat en píxels, que volem que tinga.

L'última possibilitat ens permet definir percentatges, és a dir, que independentment de la grandària de la nostra finestra, l'aplicació guardarà la mateixa proporció per als seus components. Per exemple, si volem que una part de la nostra pantalla ocupe un 30% i l'altra un 70%, llavors haurem d'utilitzar les següents expressions: **3*** i **7***.

Vegem un exemple:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" /> // Automático
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
    <RowDefinition Height="28" /> // Fijo
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="200" />
  </Grid.ColumnDefinitions>
</Grid>
```

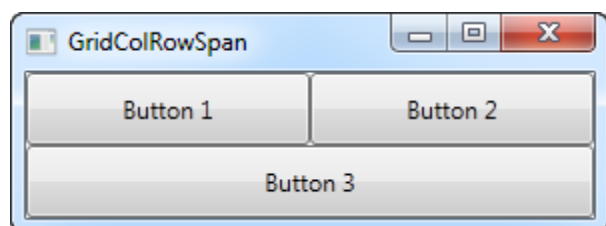
Per a col·locar elements dins del Grid ho farem especificant la fila i la columna on situarem el control:

```
<Label Grid.Row="0" Grid.Column="0" Content="Name:" />
<Label Grid.Row="1" Grid.Column="0" Content="E-Mail:" />
<Label Grid.Row="2" Grid.Column="0" Content="Comment:" />
<TextBox Grid.Column="1" Grid.Row="0" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="1" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="2" Margin="3" />
<Button Grid.Column="1" Grid.Row="3" HorizontalAlignment="Right"
  MinWidth="80" Margin="3" Content="Send" />
```

També disposa de la possibilitat d'expandir un control entre diverses files i/o columnes. Vegem un exemple:

```
<Window x:Class="WpfTutorialSamples.Panels.GridColRowSpan"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="GridColRowSpan" Height="110" Width="300">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="1*" />
      <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button>Button 1</Button>
    <Button Grid.Column="1">Button 2</Button>
    <Button Grid.Row="1" Grid.ColumnSpan="2">Button 3</Button>
  </Grid>
</Window>
```

El resultat és el següent:



Com podem observar en el requadre roig, col·loquem una propietat que indica que el component ocuparà 2 columnes, podríem fer que ocupara 3, 4, ...

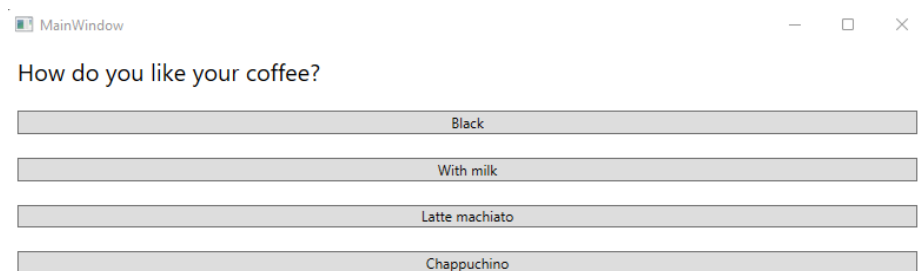
5.2. StackPanel

Es tracta d'un layout senzill que s'encarrega de realitzar una llista de components. La forma de col·locació dels controls pot ser en vertical, per defecte, o en horitzontal.

Sol ser útil per a col·locar un conjunt de controls de forma ordenada i senzilla com per exemple una sèrie de botons.

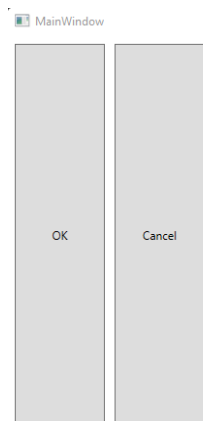
Vegem dos exemples:

```
<StackPanel>
  <TextBlock Margin="10" FontSize="20">How do you like your
coffee?</TextBlock>
  <Button Margin="10">Black</Button>
  <Button Margin="10">With milk</Button>
  <Button Margin="10">Latte machiato</Button>
  <Button Margin="10">Chappuchino</Button>
</StackPanel>
```



Ara veurem la forma horitzontal:

```
<StackPanel Margin="8" Orientation="Horizontal">
  <Button MinWidth="93">OK</Button>
  <Button MinWidth="93" Margin="10,0,0,0">Cancel</Button>
</StackPanel>
```

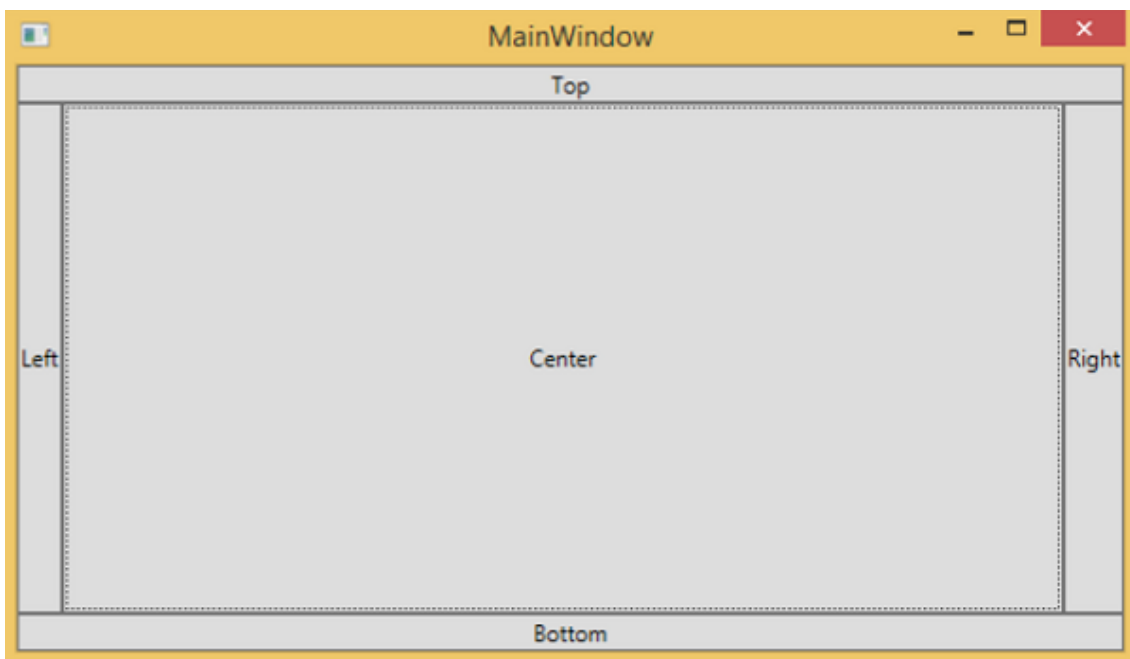


Com s'observa, es tracta d'un layout que apenas té configuració i els components es col·loquen de manera fàcil. No obstant això, no s'ha d'abusar d'aquesta mena de layout perquè té algunes mancances, com per exemple que no es pot col·locar una barra de Scroll dins d'ell.

5.3. DockPanel

Es tracta d'un layout que ens permet dividir la finestra en regions o zones on col·locar els elements.

Divideix la pantalla en cinc regions: *top*, *bottom*, *left*, *right* i *center*. En la següent figura s'observa les diferents regions en la pantalla:



El codi:

```
<DockPanel LastChildFill="True">
  <Button Content="Dock=Top" DockPanel.Dock="Top"/>
  <Button Content="Dock=Bottom" DockPanel.Dock="Bottom"/>
  <Button Content="Dock=Left"/>
  <Button Content="Dock=Right" DockPanel.Dock="Right"/>
  <Button Content="LastChildFill=True"/>
</DockPanel>
```

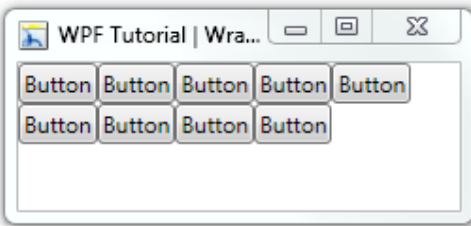
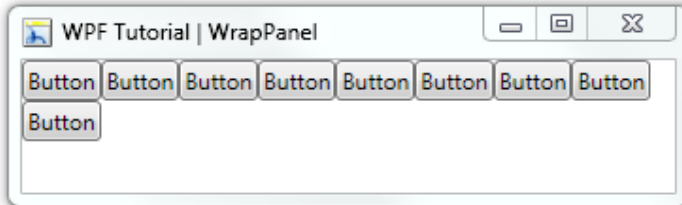
S'utilitza normalment per a organitzar **la finestra principal** d'una aplicació. D'aquesta manera podem distribuir els components adequats per cadascuna de les regions. **Dins d'aquest layout podem utilitzar altres layouts.**

5.4. WrapPanel

És molt similar a un Stack Panell, és a dir, amuntega els components segons es van afegint al panell.

La diferència radica en el fet que aquest layout afeg els elements en la mateixa línia fins que no hi ha lloc i salta a la següent.

També podem triar l'orientació entre horitzontal i vertical:



Sol utilitzar-se per a configurar pestanyes en un **TabControl**, afegir elements a una barra d'eines, ...

5.5. CanvasPanel

És el més bàsic dels layouts ja que els controls es posicionen en funció de les coordenades que establisca l'usuari. Les coordenades s'especifiquen relatives a qualsevol dels costats del panell utilitzant *Canvas.Left*, *Canvas.Top*, *Canvas.Bottom*, *Canvas.Right*.

Sol **utilitzar-se per a representar els components 2D relacionats amb el dibuix de formes**, ja que no és recomanable utilitzar-lo per a col·locar controls en la interfície perquè en utilitzar posicions absolutes pot haver-hi problemes quan es redimensione la finestra.

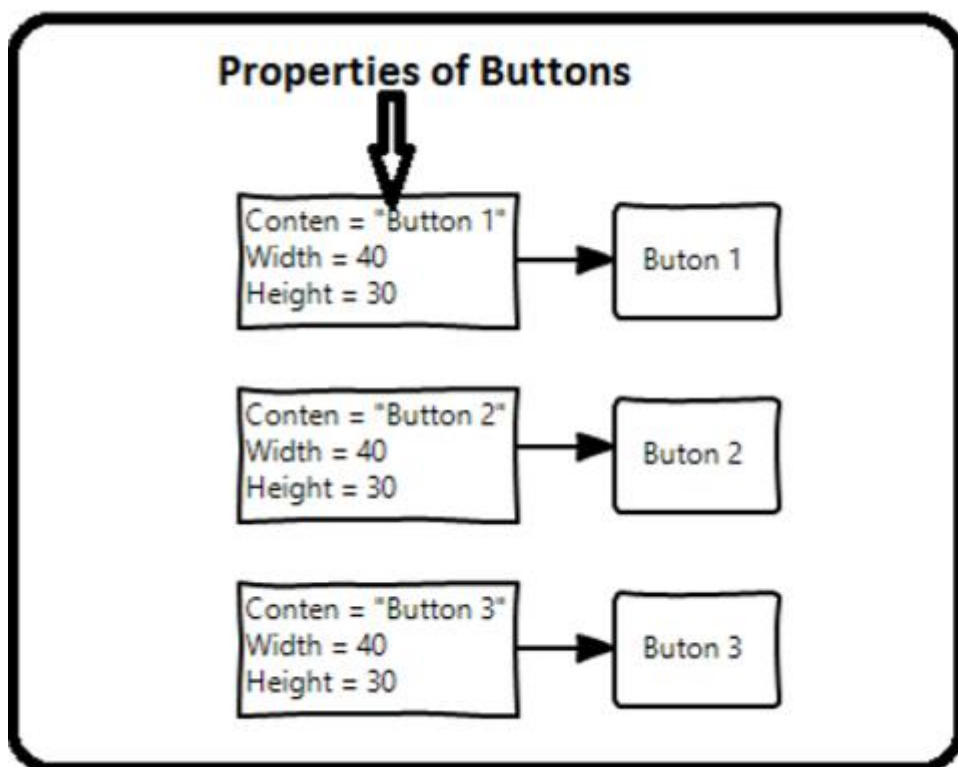
```
<Canvas>
  <Rectangle Canvas.Left="40" Canvas.Top="31" Width="63" Height="41" Fill="Blue" />
  <Ellipse Canvas.Left="130" Canvas.Top="79" Width="58" Height="58" Fill="Blue" />
  <Path Canvas.Left="61" Canvas.Top="28" Width="133" Height="98" Fill="Blue"
        Stretch="Fill" Data="M61,125 L193,28"/>
</Canvas>
```

MainWindow

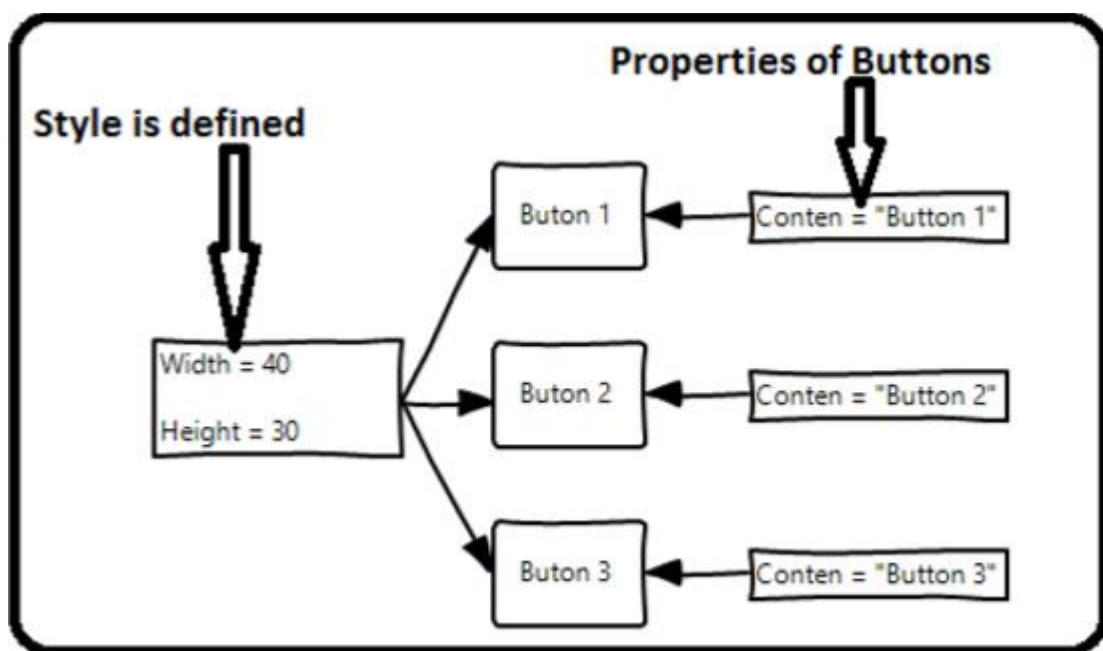


6. Estils

Com hem vist en els exemples anteriors per a poder personalitzar els diferents components d'una aplicació hem d'afegir les propietats en cadascun dels controls, encara que siguin del mateix tipus.



Per a evitar que hagem de replicar moltes propietats podem utilitzar els estils en WPF. Definim un estil i el podem aplicar als controls que vulguem en tota l'aplicació, en una pantalla o en un panell.



En l'estil podem definir propietats com el color de fons, l'amplària, l'altura, l'alineació, etc.

6.1. Nivells

Els estils es defineixen, com els controls en la finestra, mitjançant el llenguatge XAML. A més té diversos àmbits d'aplicació:

- **Control:** l'estil definit en aquest nivell solament s'aplicarà al control al qual afecta.

```
<TextBlock Text="Style test">
  <TextBlock.Style>
    <Style>
      <Setter Property="TextBlock.FontSize" Value="36" />
    </Style>
  </TextBlock.Style>
</TextBlock>
```

S'utilitza la propietat **Style** del control. A diferència dels nivells següents que s'utilitza la propietat **Resources**.

- **Layout:** en aquest cas, s'aplicarà a tot el panell en el qual es defineix l'estil.

```
<StackPanel Margin="10">
  <StackPanel.Resources>
    <Style TargetType="TextBlock">
      <Setter Property="Foreground" Value="Gray" />
      <Setter Property="FontSize" Value="24" />
    </Style>
  </StackPanel.Resources>
  <TextBlock>Header 1</TextBlock>
  <TextBlock>Header 2</TextBlock>
  <TextBlock Foreground="Blue">Header 3</TextBlock>
</StackPanel>
```

A tots els TextBlock del Layout se li aplicarà l'estil

- **Window:** de la mateixa forma s'aplicarà a tots els controls afectats dins de la finestra.

```
<Window.Resources>
  <Style TargetType="TextBlock">
    <Setter Property="Foreground" Value="Gray" />
    <Setter Property="FontSize" Value="24" />
  </Style>
</Window.Resources>
```

- **Application:** en aquest nivell afectarà a tota l'aplicació.

```
<Application.Resources>
  <Style TargetType="TextBlock">
    <Setter Property="Foreground" Value="Gray" />
    <Setter Property="FontSize" Value="24" />
  </Style>
</Application.Resources>
```

Perquè l'estil s'aplique a tota l'aplicació hem de modificar el fitxer **App.xaml**, en aquest fitxer hauríem de escriure el codi anterior.

També hi ha la possibilitat d'establir els estils en un fitxer a part per a no haver de repetir-los i situar-los en un punt centralitzat de l'aplicació.

Per a això hem de crear un fitxer **Resource Dictionary** que creguem i dins d'ell afegim els estils que vulguem.

Després ho cridem des de l'aplicació o des de la finestra amb la següent sintaxi:

```
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="Recursos/Estils/Estil.xaml"/>
    </ResourceDictionary.MergedDictionaries>
  </ResourceDictionary>
</Application.Resources>
```

El fitxer **Estil.xaml** hauria de tindre un contingut semblant a aquest:

```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Style TargetType="TextBlock">
    <Setter Property="Foreground" Value="Fuchsia" />
    <Setter Property="FontSize" Value="24" />
  </Style>
</ResourceDictionary>
```

6.2. Destinació.

Una vegada establert el nivell d'aplicació de l'estil hem de determinar sobre què controls s'aplica.

Per a especificar el tipus de control sobre el qual es modificarà l'aspecte hem d'utilitzar l'etiqueta **TargetType**. Per defecte s'aplicaran a tots els controls del nivell especificat.

Què succeeix si volem aplicar l'estil solament a alguns dels controls del tipus especificat per la propietat **TargetType** que hi ha en l'aplicació? Per defecte s'aplicaran a tots els controls.

Per a poder flexibilitzar l'aplicació de l'estil dins d'un mateix nivell utilitzarem la propietat **x:Key**. A continuació veiem un exemple:

```
<Window x:Class="WpfTutorialSamples.Styles.ExplicitStyleSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ExplicitStyleSample" Height="150" Width="300">
    <Window.Resources>
        <Style x:Key="HeaderStyle" TargetType="TextBlock">
            <Setter Property="Foreground" Value="Gray" />
            <Setter Property="FontSize" Value="24" />
        </Style>
    </Window.Resources>
    <StackPanel Margin="10">
        <TextBlock>Header 1</TextBlock>
        <TextBlock Style="{StaticResource HeaderStyle}">Header
2</TextBlock>
        <TextBlock>Header 3</TextBlock>
    </StackPanel>
</Window>
```

En aquest exemple veiem que l'estil s'aplicarà sobre tots els controls del tipus **TextBlock** que tinguen definit en l'etiqueta **Style** l'identificador **HeaderStyle**, solament a aqueixos se'ls aplicarà l'estil.

S'utilitza l'etiqueta **Setter** per a assignar les propietats.

Accés als estils des de C#

Podem definir estils en el nostre fitxer xaml i després necessitar aplicar-lo des del nostre manejador d'esdeveniments (vorem els esdeveniment a la propera unitat). Mitjançant el mètode **FindResource**, per exemple:

```
Style style = Application.Current.FindResource("LabelTemplate") as Style;
label1.Style = style;
```

És important comprovar si l'estil és nul abans d'aplicar-lo.

6.3. Triggers (*disparadors*).

Els estils poden activar-se o desactivar-se en funció del valor d'una propietat de l'objecte. Per exemple, en passar el punter del ratolí per un control podem canviar l'estil, i després que el punter haja deixat el component tornar a l'estil original. Per a realitzar aquestes accions amb els estils utilitzem aquesta funcionalitat anomenada **Triggers**. Existeixen tres tipus.

Property Trigger

És el disparador més comú i sol utilitzar-se per a determinar un estil en funció del valor d'una propietat. En aquest cas quan el ratolí passa per damunt del control (*IsMouseOver*)

```
<TextBlock.Style>
  <Style TargetType="TextBlock">
    <Setter Property="Foreground" Value="Blue"></Setter>
    <Style.Triggers>
      <Trigger Property="IsMouseOver" Value="True">
        <Setter Property="Foreground" Value="Red" />
        <Setter Property="TextDecorations" Value="Underline" />
      </Trigger>
    </Style.Triggers>
  </Style>
</TextBlock.Style>
```

Data Trigger

Aquest *disparador* sol utilitzar-se per a associar l'aplicació de l'estil amb propietats d'altres components. En aquest cas quan el control de tipus *CheckBox* està marcat (*isChecked*).

```
<TextBlock.Style>
  <Style TargetType="TextBlock">
    <Setter Property="Text" Value="No" />
    <Setter Property="Foreground" Value="Red" />
    <Style.Triggers>
      <DataTrigger Binding="{Binding ElementName=cbSample, Path=IsChecked}"
Value="True">
        <Setter Property="Text" Value="Yes!" />
        <Setter Property="Foreground" Value="Green" />
      </DataTrigger>
    </Style.Triggers>
  </Style>
</TextBlock.Style>
```

En aquest cas s'enllaça el Trigger amb la propietat **checked** del control denominat **cbSample**, que en aquest cas és una **Casella de selecció**.

Event Trigger

En aquest cas s'activa l'estil en detectar-se un esdeveniment i no en aconseguir una propietat un valor. Solen utilitzar-se per a crear animacions o efectes especials.

```
<Style TargetType="TextBox">
  <Style.Triggers>
    <EventTrigger RoutedEvent="GotFocus">
      <BeginStoryboard>
        <Storyboard>
          <ColorAnimation Storyboard.TargetProperty="Background.Color"
            To="Gold" Duration="0:0:0.25"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>

    <EventTrigger RoutedEvent="LostFocus">
      <BeginStoryboard>
        <Storyboard>
          <ColorAnimation Storyboard.TargetProperty="Background.Color"
            To="AntiqueWhite" Duration="0:0:0.25"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Style.Triggers>
</Style>
```

En aquest cas els esdeveniments capturats son quan te el foc (*GotFocus*) i quan el perd (*LostFocus*).

MultiTrigger

Si tenim la necessitat d'especificar més propietats per les quals s'activarà l'estil utilitzarem la possibilitat d'establir condicions sobre diferents propietats.

```
<Style TargetType="TextBox">
  <Style.Triggers>
    <MultiTrigger>
      <MultiTrigger.Conditions>
        <Condition Property="IsMouseOver" Value="True"/>
        <Condition Property="Text" Value=""/>
      </MultiTrigger.Conditions>
      <MultiTrigger.Setters>
        <Setter Property="Background" Value="Orange"/>
        <Setter Property="FontSize" Value="15"/>
      </MultiTrigger.Setters>
    </MultiTrigger>
  </Style.Triggers>
</Style>
```

Primer definim les condicions que han de complir-se i després establim les propietats que han de canviar-se. En aquest únicament funcionarà el Trigger (canviar el color de fons a Orange i tamany de lletra a 15) quan el control estiga buit i ratolí passe per damunt.

No es poden tindre diverses condicions amb **EventTriggers**, solament amb **Property Triggers** i amb **Data Triggers**.

Podeu consultar la guia completa de estils ací: <https://docs.microsoft.com/es-es/windows/apps/design/style/xaml-styles>









7. Temes

Com ha hem vist abans, un projecte WPF pot tindre diversos recursos d'interfície d'usuari que s'apliquen en tota l'aplicació. De manera col·lectiva, aquest conjunt de recursos es poden considerar el **tema** per a l'aplicació. WPF proporciona compatibilitat per a empaquetar recursos d'interfície d'usuari com un tema mitjançant un diccionari de recursos que s'encapsula com la classe **ResourceDictionary**.

Els temes de WPF es defineixen mitjançant el mecanisme d'aplicació d'estils i plantilles que WPF exposa per a personalitzar els objectes visuals de qualsevol element.

7.1. Temes nadius VSC.

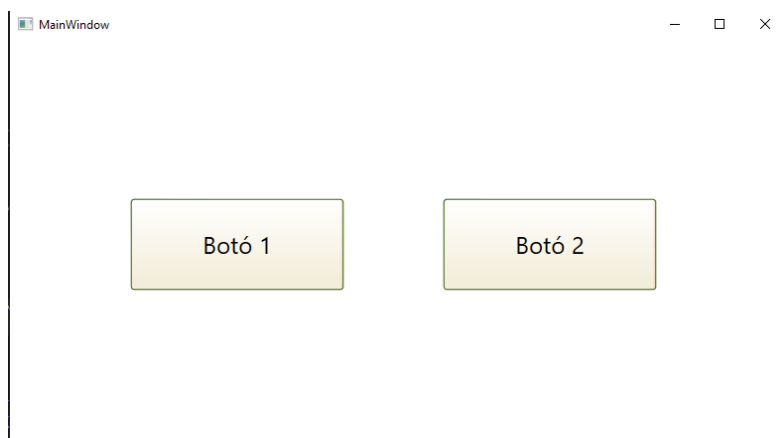
Dins de la instal·lació de VSC tenim alguns temes per defecte que podem utilitzar. Observem el contingut de la carpeta **C:\Program Files\Microsoft Visual Studio\2022\Community\DesignTools\SystemThemes\Wpf**

osoft Visual Studio > 2022 > Community > DesignTools > SystemThemes > Wpf				
Nombre	Fecha de modificación	Tipo	Tamaño	
 aero.normalcolor	17/08/2022 11:27	Windows Markup ...	567 KB	
 aero2.normalcolor	17/08/2022 11:27	Windows Markup ...	542 KB	
 aerolite.normalcolor	17/08/2022 11:27	Windows Markup ...	473 KB	
 classic	17/08/2022 11:27	Windows Markup ...	491 KB	
 luna.homestead	17/08/2022 11:27	Windows Markup ...	544 KB	
 luna.metallic	17/08/2022 11:27	Windows Markup ...	545 KB	
 luna.normalcolor	17/08/2022 11:27	Windows Markup ...	544 KB	
 royale.normalcolor	17/08/2022 11:27	Windows Markup ...	540 KB	

Si no canviem el tema per defecte d'un projecte WPF l'aspecte d'una finestra amb dues botons és aquesta:



Si per exemple canviem el tema per defecte al tema **luna.homestead** l'aspecte de la finestra i el botons serà la següent:



A la imatge podem veure com ha canviat de forma automàtica l'aspecte i color dels botons.

Per a carregar qualsevol dels temes inclosos podem fer us d'aquest codi dins del fitxer principal del projecte **App.xaml**

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary
Source="/PresentationFramework.Luna;V3.0.0.0;31bf3856ad364e35;component\themes\Luna.homestead.
xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

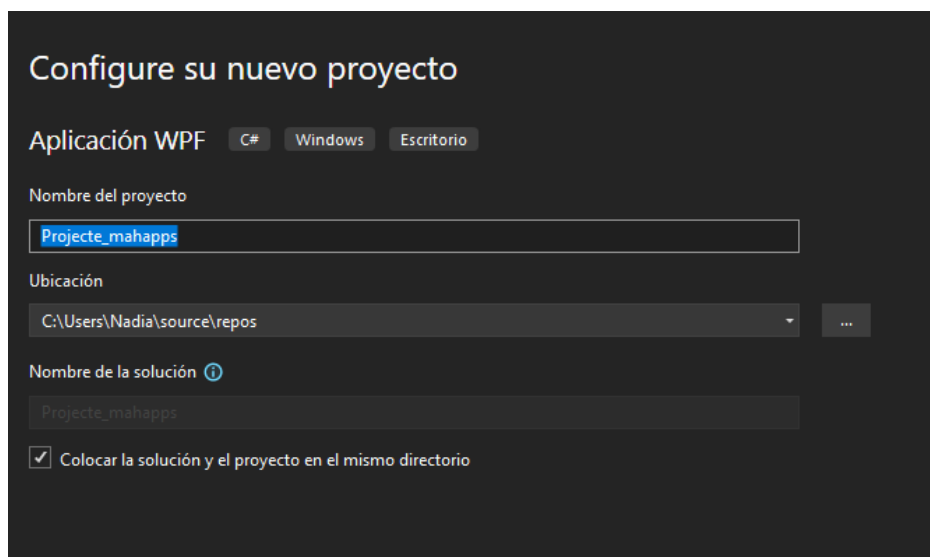
També podríem indicar la ruta de forma absoluta, així:

```
<ResourceDictionary
Source="pack://application:,,,/PresentationFramework.Luna;V3.0.0.0;31bf3856ad364e35;component\
themes\Luna.homestead.xaml" />
```

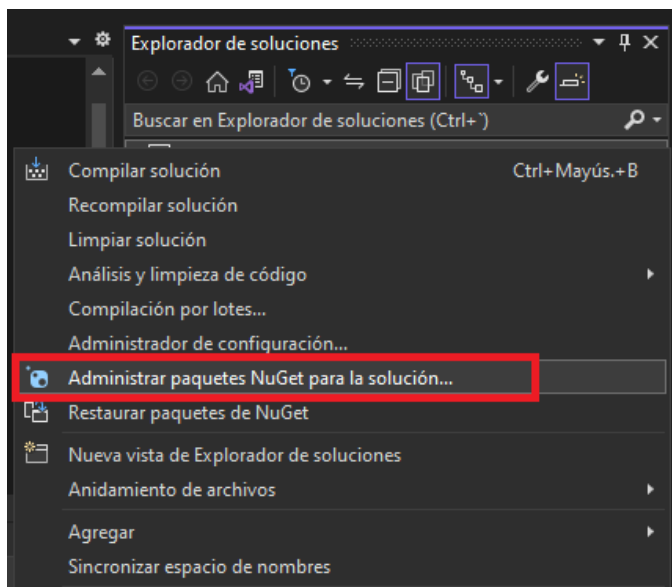
7.2. Temes de tercers (Paquet MahApps).

També és possible descarregar i utilitzar altres temes creats per altres dissenyadors o altres empreses. Aquests temes podem descarregar-los de la web o repositori del creador o des de el propi entorn VSC. Vegem com fer-ho des de VSC descarregant el paquet **Mahapps Metro**:

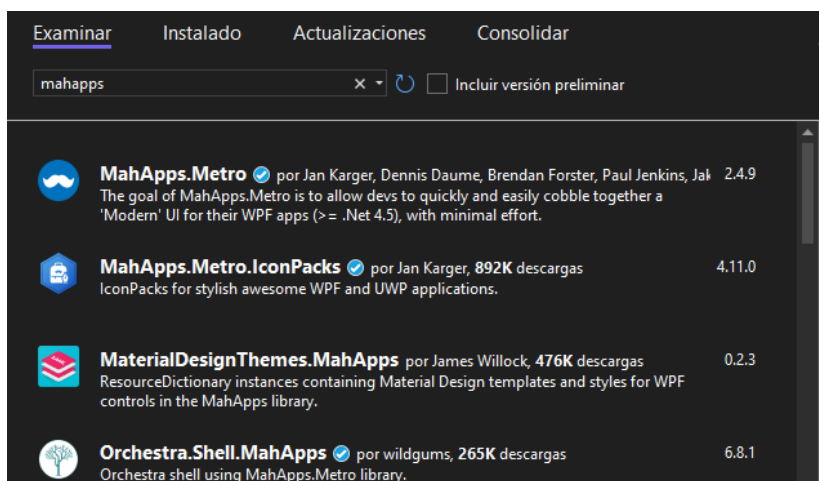
Creem un projecte nou (l'anomenem Projecte_Mahapps, per exemple):



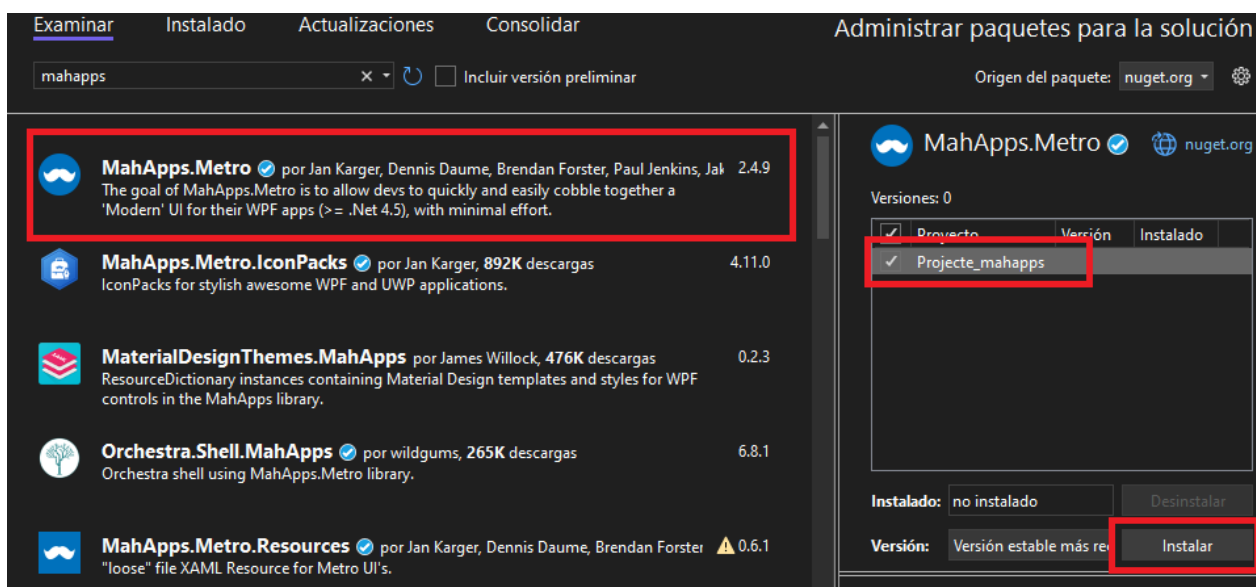
Una vegada creat el primer que hem de fer és descarregar l'última versió del framework, per a això ens anem al projecte i premem botó dret perquè ens mostre les opcions i seleccionem **Manage Nugets Packages...**



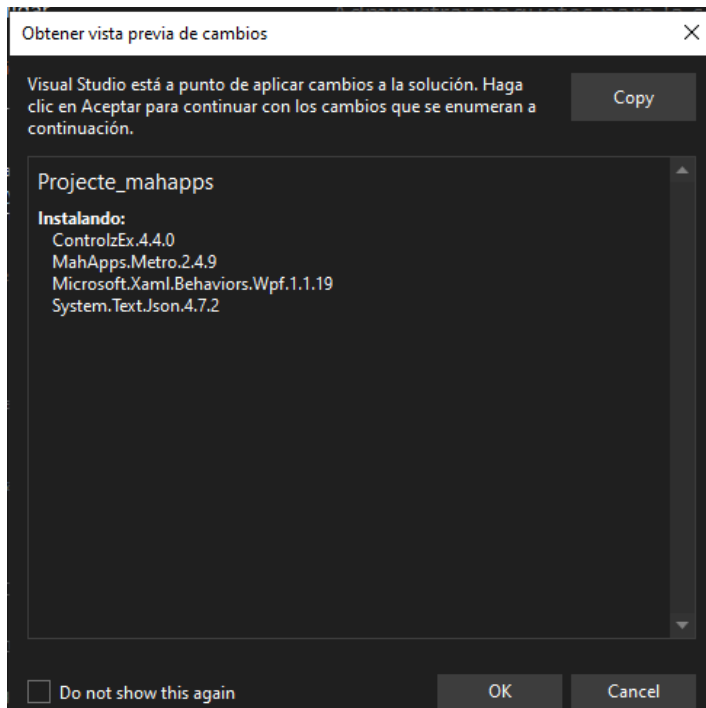
S'obri la finestra de **Nuget**, el gest or de paquets on hem de buscar en l'apartat en **Examinar** i instal·lar dos paquets de **Mahapps Metro**.



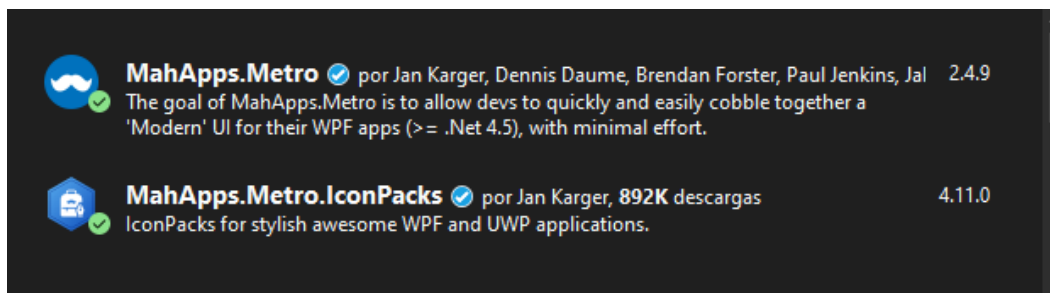
Marquen cada paquet a instal·lar, indiquen per a quin projecte el volem afegir i polsem instal·lar (aquest procés hem de fer-ho amb el paquet **MahApps.Metro** i **MahApps.Metro.IconPacks**):



VSC ens avisa de que és van a fer canvis en el projecte:



Després d'instal·lar els dos paquets, aquests quedaran marcats amb un tic verd:



Només ens queda configurar el nostre projecte per a tindre accés a totes les millors de la interfície gràfica que ens aporta **Mahapps**. Per a això hem d'editar l'arxiu **App.xaml** i afegir el següent codi:

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Fonts.xaml" />
            <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Controls.xaml" />
            <ResourceDictionary
Source="pack://application:,,,/MahApps.Metro;component/Styles/Themes/Light.Blue.xaml" /> />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

Observa també que dins del paquet existeixen diferents temes (en aquest cas seleccionem el tema **LightBlue**). Pots consultar les possibilitats en

<https://mahapps.com/docs/themes/usage>

A continuació, per poder explotar tota la potència d'aquest paquet canviarem la finestra que es crea per defecte quan creguem una aplicació WPF (fitxer **MainWindow.xaml**). El codi XAML que genera per defecte és el següent:

```
<Window x:Class="Projecte_mahapps.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:Projecte_mahapps"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

I hem de canviar-lo per aquest:

```
<Controls:MetroWindow x:Class="Projecte_mahapps.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:Projecte_mahapps"
        xmlns:Controls="http://metro.mahapps.com/winfx/xaml/controls"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
    <Grid>

    </Grid>
</Controls:MetroWindow>
```

Observa que ara es crea una finestra de tipus **MetroWindow** i que es fa referència als controls mahapps.

Per últim hem de canviar la classe de la qual deriva la nostra finestra per la classe **MetroWindow** en el C# (**MainWindow.xaml.cs**). El codi final que hauria de tindre aquest arxiu seria el següent:

```
using MahApps.Metro.Controls;
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 Projecte_mahapps
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml

```

```

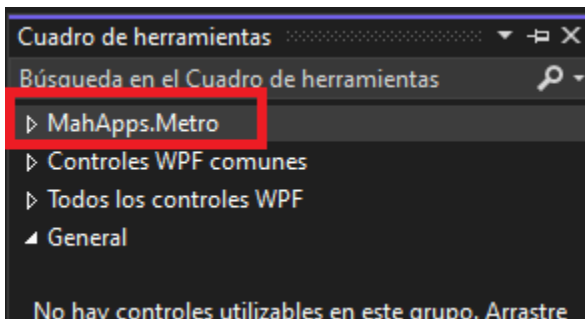
/// </summary>
public partial class MainWindow : MetroWindow
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
}

```

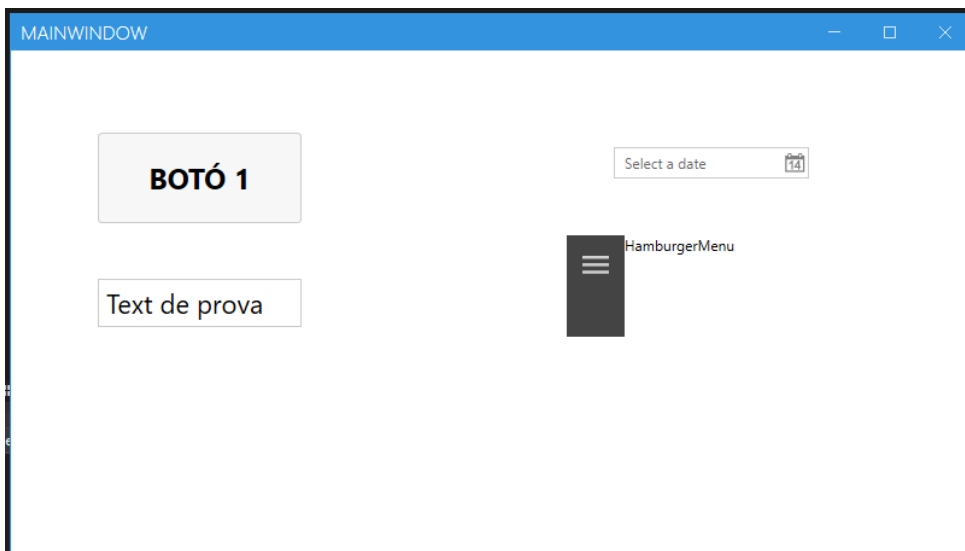
Observa que el canvis fets son:

- Importar la llibreria controls de MahApps `using MahApps.Metro.Controls;`
- Canviar la classe de la finestra principal `public partial class MainWindow : MetroWindow`

Si observem el Quadre d'eines observarem que aquest paquet ha aportat nous controls:



Afegim un parell de controls per observar l'impacte d'aquest paquet al projecte (A l'esquerra controls WPF estàndard i a la dreta controls MahApps):

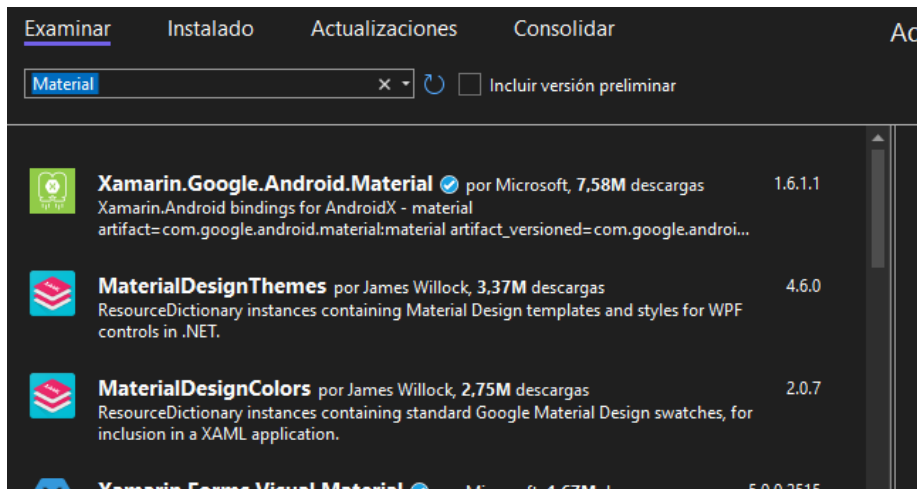


L'aspecte de l'aplicació es molt diferent a més que el paquet afegeix altres controls extra (En la imatge un DatePicker i un menú tipus Hamburger).

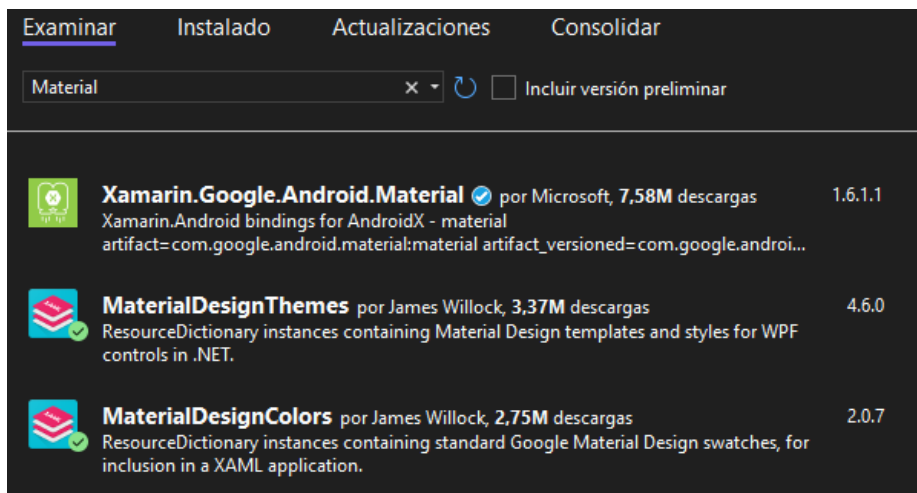
A la web del creador d'aquest paquet [MahApps.Metro - Home](http://MahApps.Metro-Home) podeu trobar molta més informació i com traure més profit d'ell.

7.3. Temes de tercers (Paquet Material Design).

Vegem ara com importar altre paquet molt popular al mon WPF. El paquet a buscar és **Material Design**. Escrivim "material" al buscador de paquets:



Instal·lem els paquets **MaterialDesignThemes** i **MaterialDesignColors**:



A la web del creador [Material Design In XAML](#) s'explica com editar el projecte per a utilitzar el paquet. Bàsicament cal fer el següent:

Afegir els Recursos de Diccionaris al fitxer **App.xaml**:

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary
Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Light.xaml" />
            <ResourceDictionary
Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/MaterialDesignTheme.Defaults.xaml" />
            <ResourceDictionary
Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Primary/MaterialDesignColor.DeepPurple.xaml" />
            <ResourceDictionary
Source="pack://application:,,,/MaterialDesignColors;component/Themes/Recommended/Accent/MaterialDesignColor.Lime.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

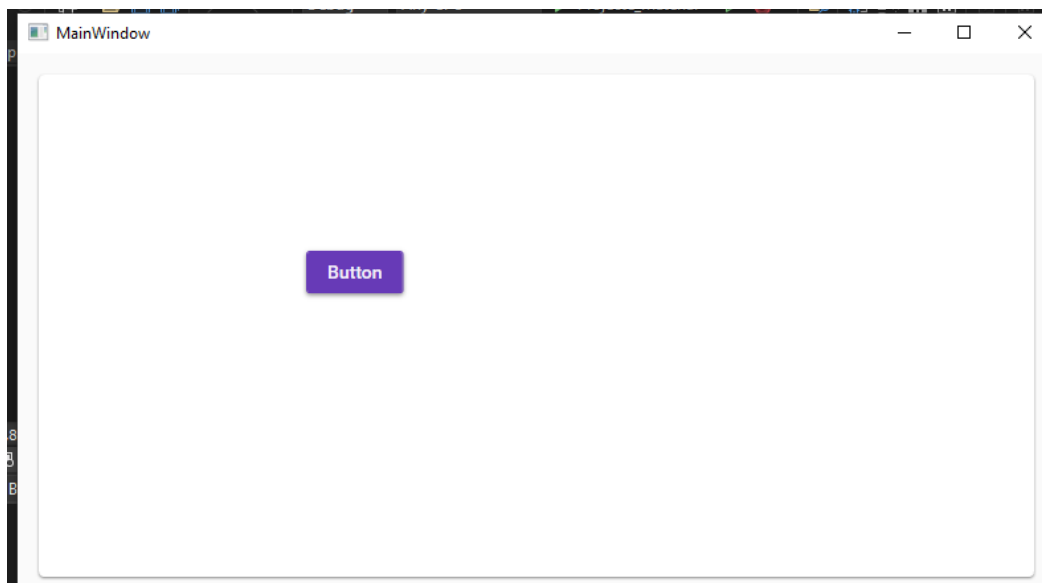
I modificar el fitxer **MainWindow.xaml** així (afegint un botó):

```
Window x:Class="Projecte_material.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:Projecte_material"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="800"
    xmlns:materialDesign="http://materialdesigninxaml.net/winfx/xaml/themes"
    TextElement.Foreground="{DynamicResource MaterialDesignBody}"
    TextElement.FontWeight="Regular"
    TextElement.FontSize="13"
    TextOptions.TextFormattingMode="Ideal"
    TextOptions.TextRenderingMode="Auto"
    Background="{DynamicResource MaterialDesignPaper}"
    FontFamily="{DynamicResource MaterialDesignFont}">

    <Grid>
        <materialDesign:Card Padding="32" Margin="16">
            <Button Content="Button" HorizontalAlignment="Left" Margin="170,101,0,0"
VerticalAlignment="Top"/>
        </materialDesign:Card>
    </Grid>

</Window>
```

Si executem el projecte:



Aquest paquet, al igual que MahApps, també incorpora controls específics.