

Desenvolupament d'interfícies

Unitat 3. Esdeveniments. Components visuals.



Índex

1. Introducció	2
2. Finestres i diàlegs	2
2.1. Diàlegs predefinits.....	3
2.2. Diàlegs basats en els predefinits.....	8
3. Controls per a mostrar informació	16
3.1. Control Label.	16
3.2. Control TextBlock.	16
4. Camps de text	17
4.1. Màscara.	17
4.2. Contrasenya	19
5. Botons	19
6. Organitzadors.....	20
6.1. Separator	20
6.2. Groupbox	21
7. Consells per a formularis	22
8. Esdeveniments	22
8.1. Controlador d'esdeveniments	24
8.2. Subscripció a esdeveniments desde C# (Code-behind).....	26
8.3. Exemples d'Esdeveniments.....	27
9. Bindings	34
9.1. Elements.....	34
9.2. Direcció.....	36
9.3. Sincronització.....	36
9.4. Crear un Binding.	36
9.5. Binding amb un model de dades.	37
9.6. Relative Source.....	38

1. Introducció

En aquest punt desenvoluparem alguns dels principals components utilitzats per a introduir informació en la nostra aplicació, és a dir, per a realitzar formularis.

Primer veurem la diferència entre diàlegs i finestres per a poder entendre millor l'entrada de dades.

Depenent del tipus d'informació a introduir, haurem d'utilitzar un diàleg, o bé un formulari en una finestra.

Farem un recorregut pels controls més comuns i les seues propietats i característiques més habituals.

2. Finestres i diàlegs

Els diferents tipus de finestres que ens podem trobar en la nostra aplicació són:

- **Finestra no modal (Finestra Principal):** permet alternar el focus a qualsevol altra finestra present dins de l'entorn gràfic. És el tipus més comú de modalitat. Si disposem de diverses finestres obertes ens podem moure per elles amb total llibertat sense esperar a introduir cap dada.
- **Finestra modal respecte a una aplicació (Diàlegs):** permet alternar el focus a altres finestres del sistema, però no a la finestra que li dona origen fins que es pren una acció sobre ella. Normalment s'utilitzen per a confirmar una acció de l'usuari, un exemple típic seria un administrador d'arxius que deté una acció de l'usuari demanant confirmació com Desitja eliminar aquest arxiu? i les opcions d'Acceptar i Cancel·lar (O Esborrar / No Esborrar).
- **Finestra modal respecte al sistema:** similar a l'anterior, però no cedeix el focus a cap altra aplicació fins que es pren determinada acció sobre ella. Per regla general només han de sorgir quan existeix un esdeveniment a nivell del sistema complet que exigisca atenció immediata de l'usuari, per exemple Desitja apagar l'equip?

La següent classificació la farem en funció de l'aplicació i la manera que te aquesta d'organitzar les diferents finestres obertes:

- **MDI (Multiple Document Interface):** són aquelles aplicacions en les que les finestres dels quals es troben dins d'una finestra pare (normalment amb l'excepció de les finestres modals), és a dir, a excepció dels diàlegs.
- **SDI (Single Document Interface):** és una manera d'organitzar les aplicacions gràfiques en finestres individuals que són manejades per separat pel gestor de finestres del sistema operatiu. Això significa normalment que cada finestra és mostrada com una entrada individual en la barra de tasques del sistema operatiu o en el gestor de tasques.
- **TDI (Tabbed Document Interface):** la navegació per pestanyes es refereix a la possibilitat que diversos panells amb informació estiguen continguts dins d'una sola

finestra principal, usant pestanyes per a alternar entre ells. Molt utilitzat en l'actualitat en els navegadors.

Per exemple, la mateixa Microsoft ha variat la seua política amb l'aplicació Office, la qual ha passat de SDI a MDI i en l'actualitat una altra vegada a SDI. Mentre que Eclipse utilitza una combinació de TDI + MDI.

2.1. Diàlegs predefinits.

En nombroses ocasions necessitem presentar informació en forma de diàleg per a desenvolupar correctament la nostra aplicació.

Principalment es divideixen en dues categories, la primera d'elles simplement busca mostrar informació curta i ràpida sobre diferents coses. Existeixen missatges d'informació, advertiment o error.

Mentre que en la segona categoria s'engloben els missatges que pregunten alguna cosa, especialment confirmacions sobre alguna acció, per exemple si es desitja confirmar l'esborrat d'un fitxer.

Important: Aquests diàlegs predefinits únicament podem utilitzar-los mitjançant codi C#, per tant sempre aniran en els fitxers amb extensió .cs.

Diàleg MessageBox:

En WPF existeix la classe `MessageDialog` que ens permet configurar diverses opcions depenent si volem un diàleg d'informació o bé de confirmació.

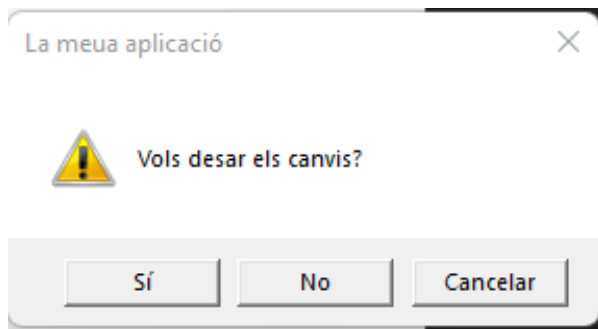
Ho invoquem mitjançant un mètode estàtic amb diverses possibilitats segons del nombre de paràmetres del mètode `Show`.

Per exemple:

```
// Configurem las opciones del diàleg
string messageBoxText = "Vols desar els canvis?";
// Títol de la finestra
string caption = "La meua aplicació";
// Combinació de botons a mostrar
MessageBoxButton button = MessageBoxButton.YesNoCancel;
MessageBoxImage icon = MessageBoxImage.Warning;
// Mostrem el MessageBox

MessageBox.Show(messageBoxText, caption, button, icon);
```

El resultat és el següent:



Si necessitem (òbviament que si) saber quin botó s'ha `polsat per a aplicar qualservol acció, codi, ..., haurem de replegar el resultat de la pulsació en una variable:

```
// Mostrem el MessageBox
MessageBoxResult result = MessageBox.Show(messageBoxText,
caption, button, icon);

// Processem el resultat de la pulsació
switch (result)
{
    case MessageBoxResult.Yes:
        // Ha polsat Si
        // ...
        Button1.Content = "Ha polsat Si";
        break;
    case MessageBoxResult.No:
        // Ha polsat No
        // ...
        Button1.Content = "Ha polsat No";
        break;
    case MessageBoxResult.Cancel:
        // Ha polsat Cancelar
        // ...
        Button1.Content = "Ha polsat Cancelar";
        break;
}
```

Observa que, mitjançant un switch de variable que replega el resultat de la pulsació, el que fem en cada cas és canviar el text d'un botó indicant la pulsació ocorreguda.

Si voleu aprofundir més podeu consultar Per a més informació consultar: [MessageBox](#), [MessageBox Sample](#), i [Dialog Box Sample](#).

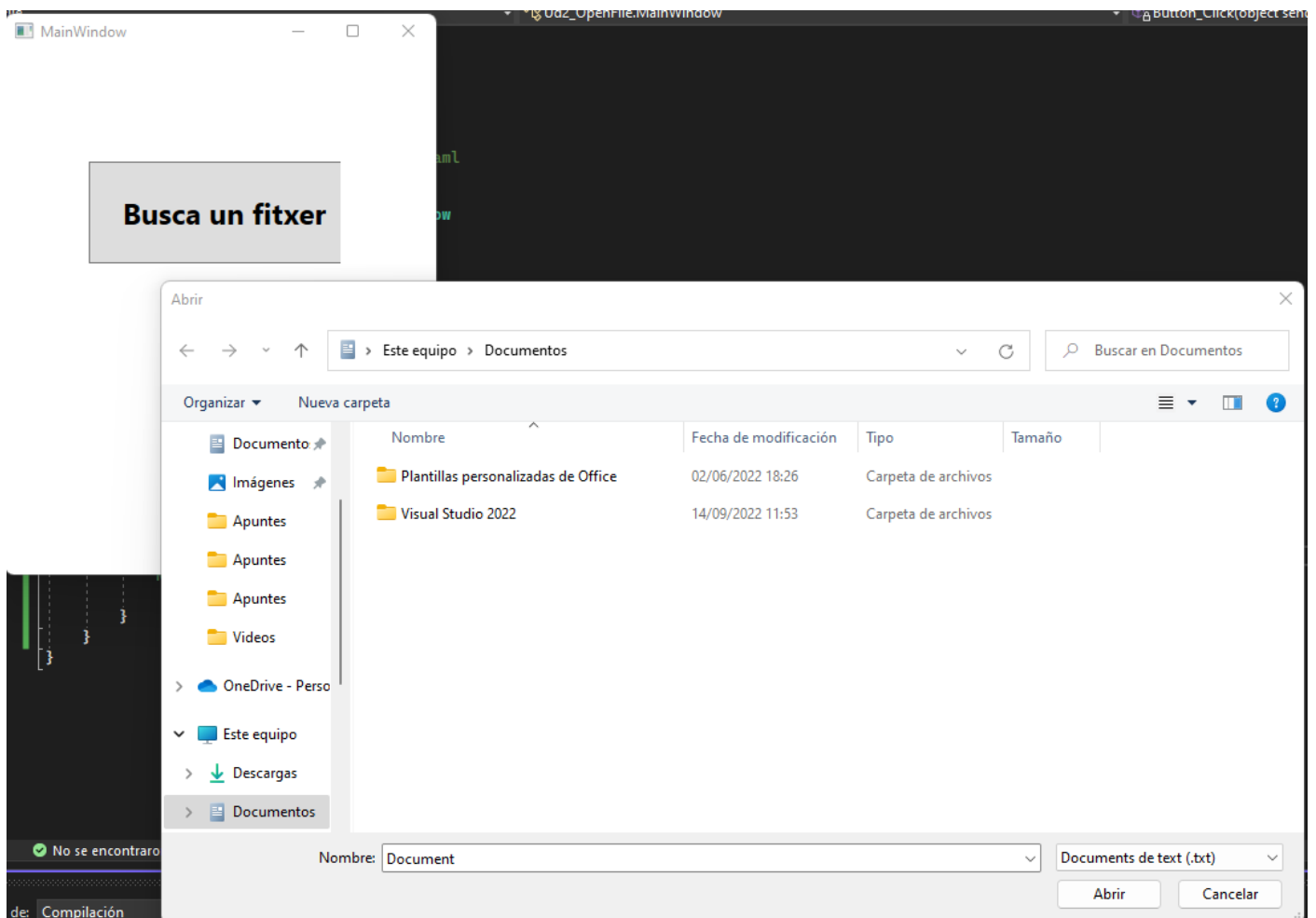
Diàleg OpenFileDialog:

Aquest diàleg s'utilitza per a obrir fitxers (per a després adjuntar-los, obrir-los, ...) . Tenim la possibilitat de configurar coses com el tipus de fitxer, ruta on buscar primer, ...

```
// Configurem el open file dialog box
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
dlg.FileName = "Document"; // Nom del fitxer per defecte
dlg.DefaultExt = ".txt"; // Extensió per defecte
dlg.Filter = "Documents de text (*.txt)|*.txt"; // Filtre per als fitxers d'extensió txt

// Mostrem el open file dialog box i repleguem en una variable el resultat

Nullable<bool> result = dlg.ShowDialog();
```

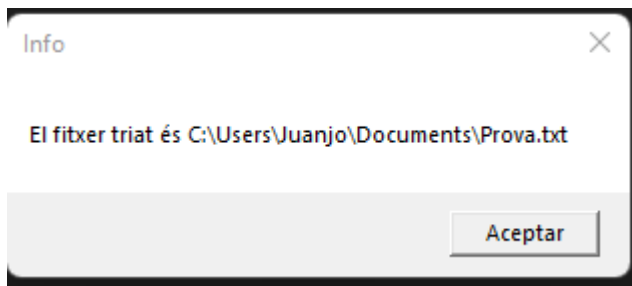


I seguidament hem de comprovar si s'ha triat un fitxer (el mètode showDialog tornarà true):

```
// Processem el fitxer triat si el resultat és true (és a dir s'ha triat un fitxer)
if (result == true)
{
    // Repleguem el nom del document triat
    string filename = dlg.FileName;

    MessageBox.Show("El fitxer triat és " + filename, "Info");
}
```

I l'exemple anterior mostraria un MessageBox amb la ruta i el nom del fitxer:

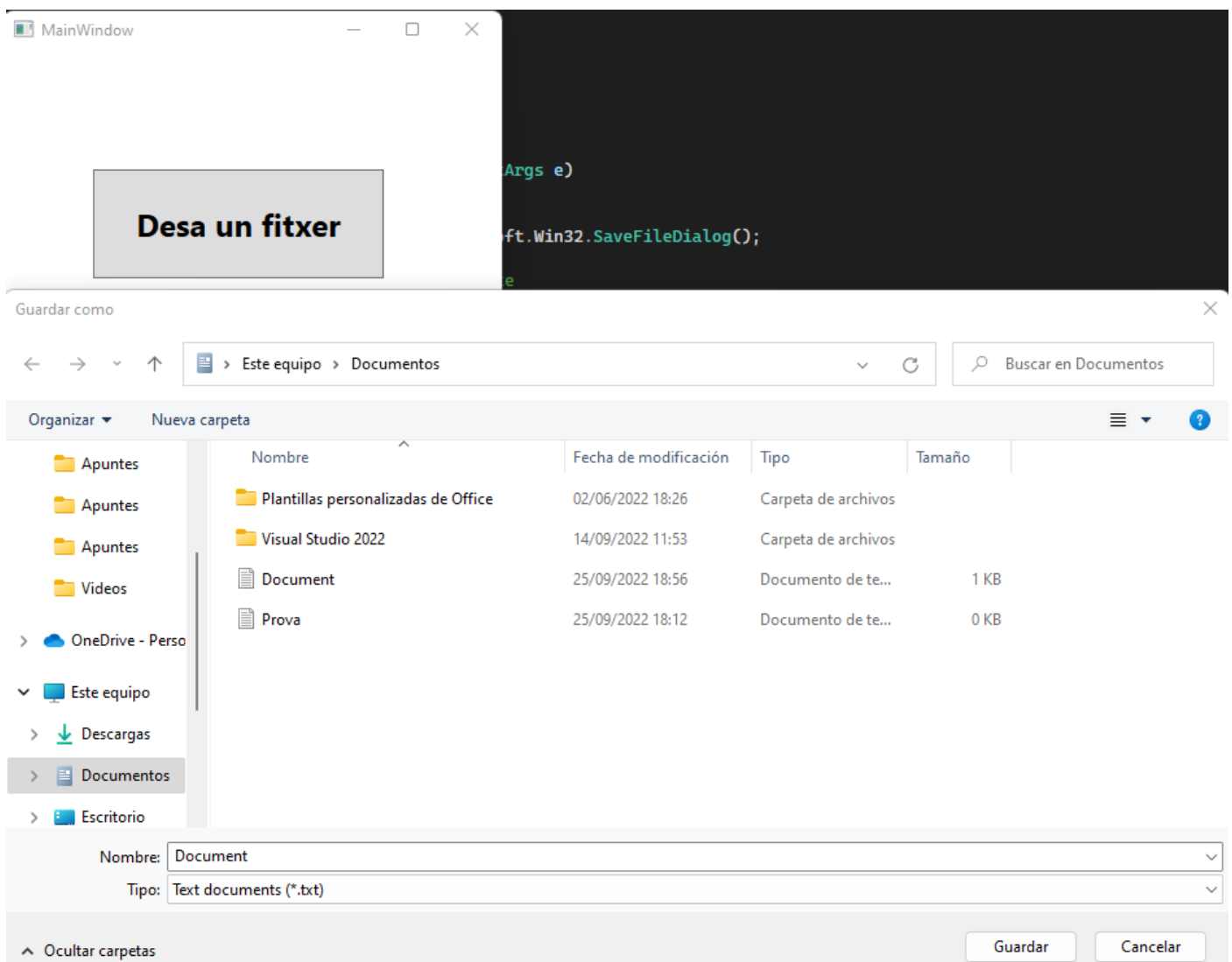


Diàleg SaveFile:

Aquest diàleg s'utilitza per a desar fitxers amb qualsevol contingut:

```
// Configurem el save file dialog box
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = "Document"; // Nom per defecte
dlg.DefaultExt = ".text"; // Extensió per defecte
dlg.Filter = "Text documents (*.txt)|*.txt"; // Filtrem per l'extensió txt

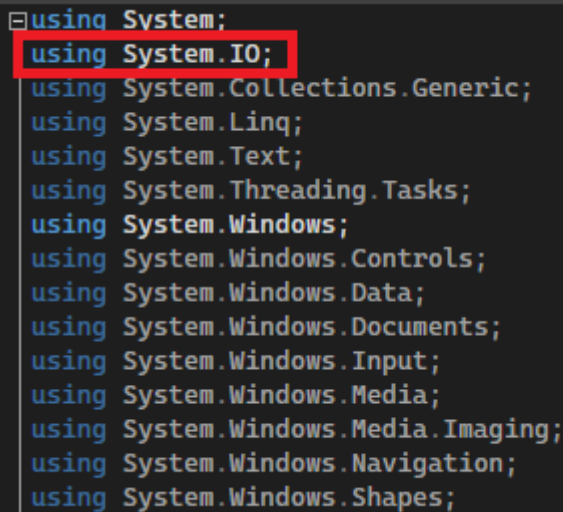
// Mostrem el save file dialog box
Nullable<bool> result = dlg.ShowDialog();
```



La següent part del codi hauria de ser per a comprovar si han pulsat el botó "Guardar" i això desar el fitxer:

```
// Procesem el resultat del save file dialog box
if (result == true)
{
    // Desem el document amb un text d'exemple
    string filename = dlg.FileName;
    File.WriteAllText(filename, "Text d'exemple per al contingut del fitxer");
}
```

Observa també (revisa els exemples que acompanyen a la unitat) que per a poder desar el fitxer cal afegir una referència a la classe System.IO al fitxer C# (normalment al fitxer MainWindow.xaml.cs):



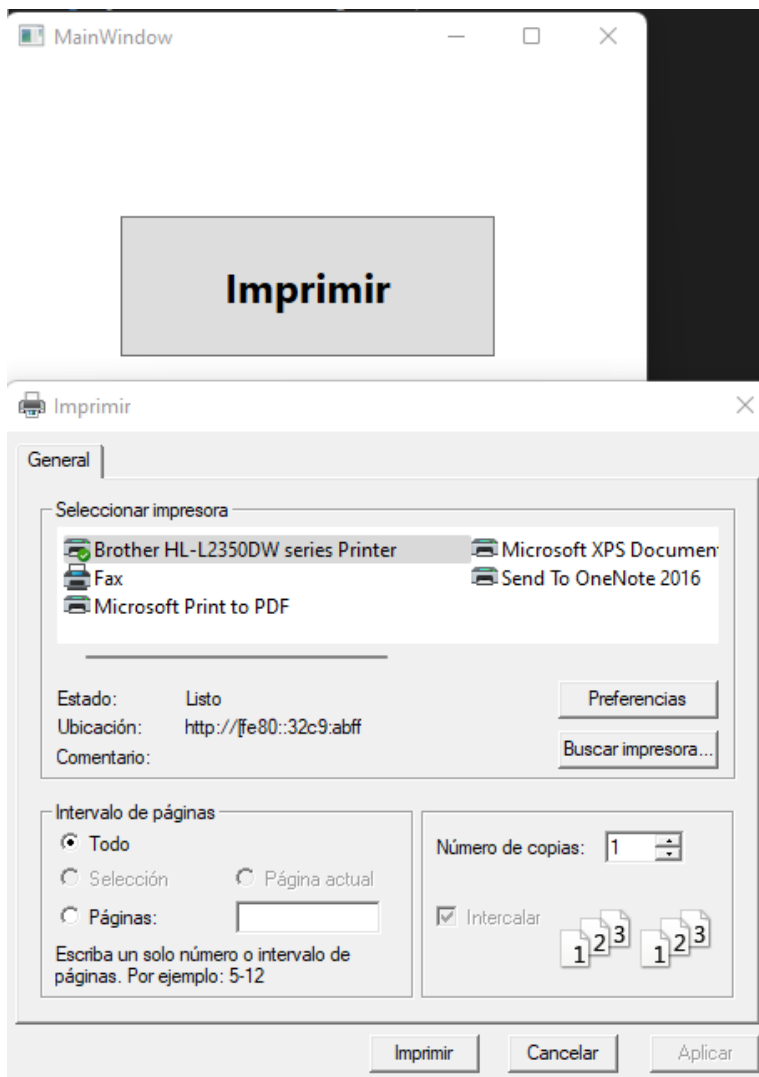
```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

Diàleg Print DialogBox:

Aquest diàleg s'utilitza per a triar una impressora al moment d'imprimir o per a configurar les seues propietats:

```
// Configurem el printer dialog box
System.Windows.Controls.PrintDialog dlg = new
System.Windows.Controls.PrintDialog();
dlg.PageRangeSelection = PageRangeSelection.AllPages;
dlg.UserPageRangeEnabled = true;

// Mostrem el save file dialog box
Nullable<bool> result = dlg.ShowDialog();
```

I finalment imprimim si s'ha polsat el botó imprimir:

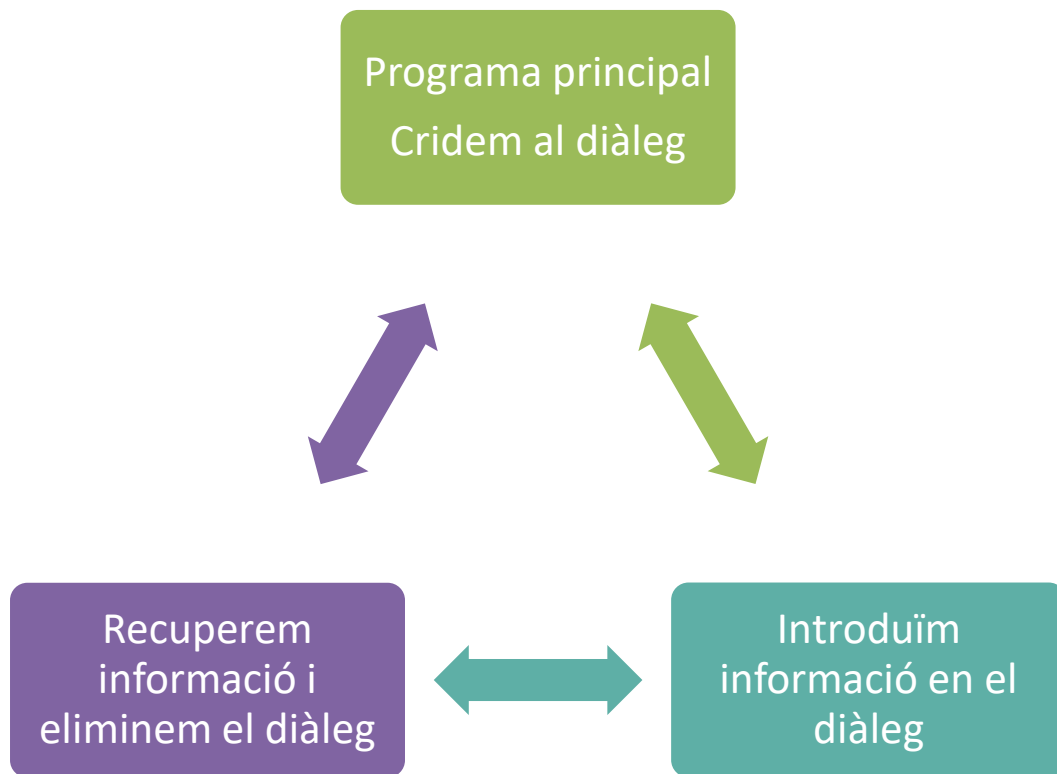
```
// Enviem la impressió si han polsat el botó Imprimir
if (result == true)
{
    // Imprimim un document
}
```

2.2. Diàlegs basats en els predefinitos.

Podem crear els nostres propis diàlegs per a una determinada aplicació, i fins i tot guardar-los en una llibreria per a posteriorment reutilitzar-los en diferents projectes.

Crearem classes que construïsquen la interfície dels nostres diàlegs i després els cridarem des del programa principal. Els diàlegs quasi sempre seran de tipus **modal**, és a dir, l'execució del programa no seguirà fins guardem els valors o cancel·lem el diàleg.

En WPF cal destacar que segons cridem al nostre diàleg, aquest serà **modal** o **no modal**. Si volem que el nostre **diàleg siga modal**, la majoria der casos, llavors hem de cridar-lo amb el mètode **ShowDialog()**. Per contra, si volem que el nostre diàleg siga **no modal** hem de cridar-lo amb el mètode **Show()**. És a dir, quan hàgem de cridar a una finestra normal utilitzarem el mètode **Show()**, mentre que per als diàlegs utilitzarem **ShowDialog()**.



Dissenyem la interfície del diàleg (la nostra **vista del diàleg**) i l'invoquem des del programa principal passant-li els paràmetres que siguin necessaris. Després de tancar el diàleg, bé mitjançant el botó de **Desar** o **Cancel·lar** recuperarem la informació o la rebutjarem, tot això ho farem des de la classe de maneig d'esdeveniments del diàleg (els esdeveniments els vorem més avant).

Els diàlegs s'utilitzen per a obtenir informació introduïda per l'usuari. Normalment tenim dues opcions, una per a cancel·lar, és a dir, l'usuari ha decidit no fer res, o l'opció d'acceptar, en aquest cas, l'usuari ha introduït informació.

Normalment els diàlegs tindran dos botons un per a cancel·lar i un altre per a desar o acceptar. En el primer cas hem de definir aquest botó de cancel·lar de la nostra aplicació mitjançant la propietat **IsCancel a true** en el la etiqueta XAML on es defineix.

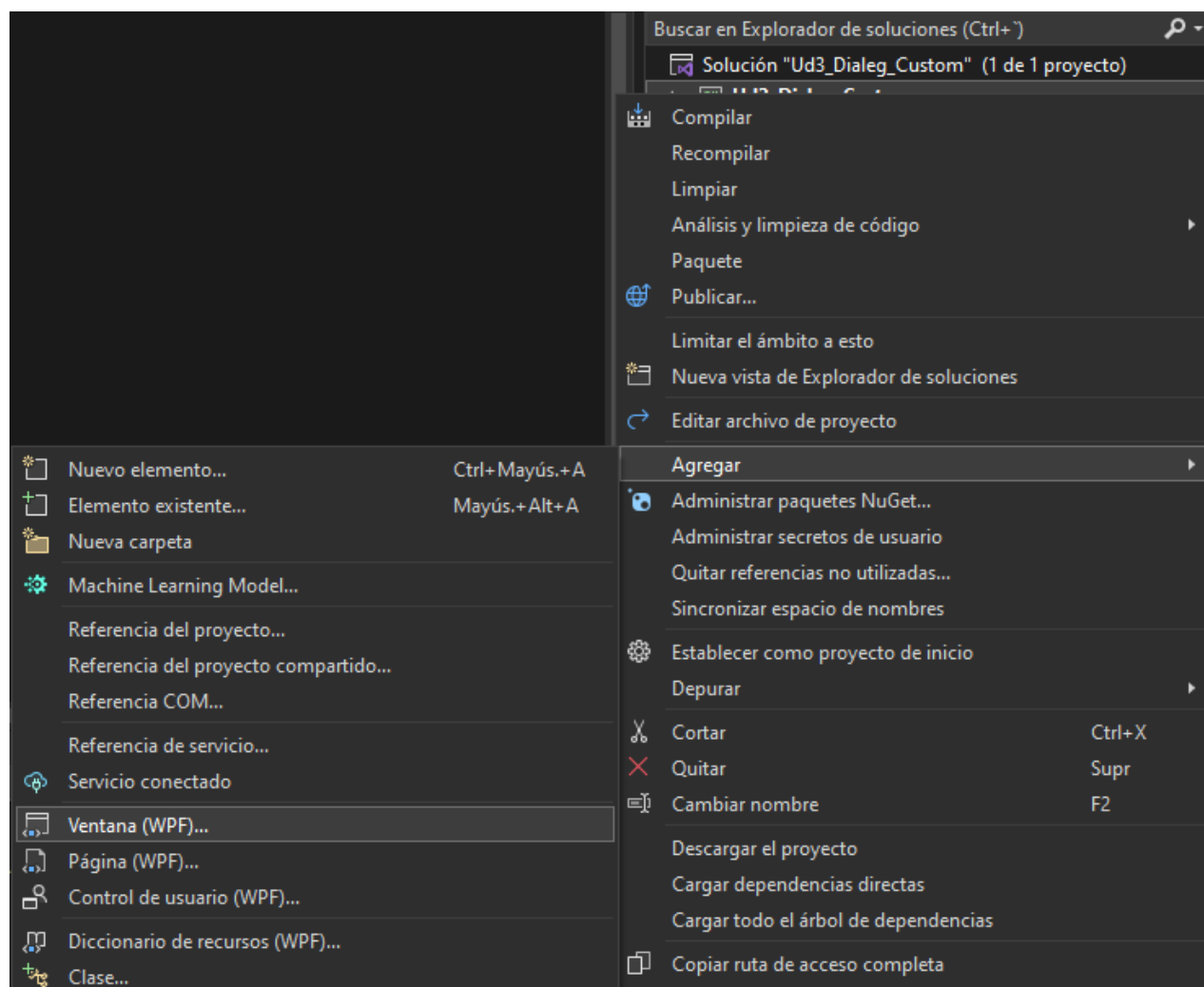
Mitjançant l'etiqueta **IsDefault a true** definirem el botó que s'utilitza per a la funció d'acceptar.

Perquè el nostre diàleg pose el focus en un camp determinat utilitzarem la propietat `FocusManager.FocusedElement = "{Binding ElementName = Nom_del_TextBox}"` en la definició de l'etiqueta `Window` o bé `UserControl`

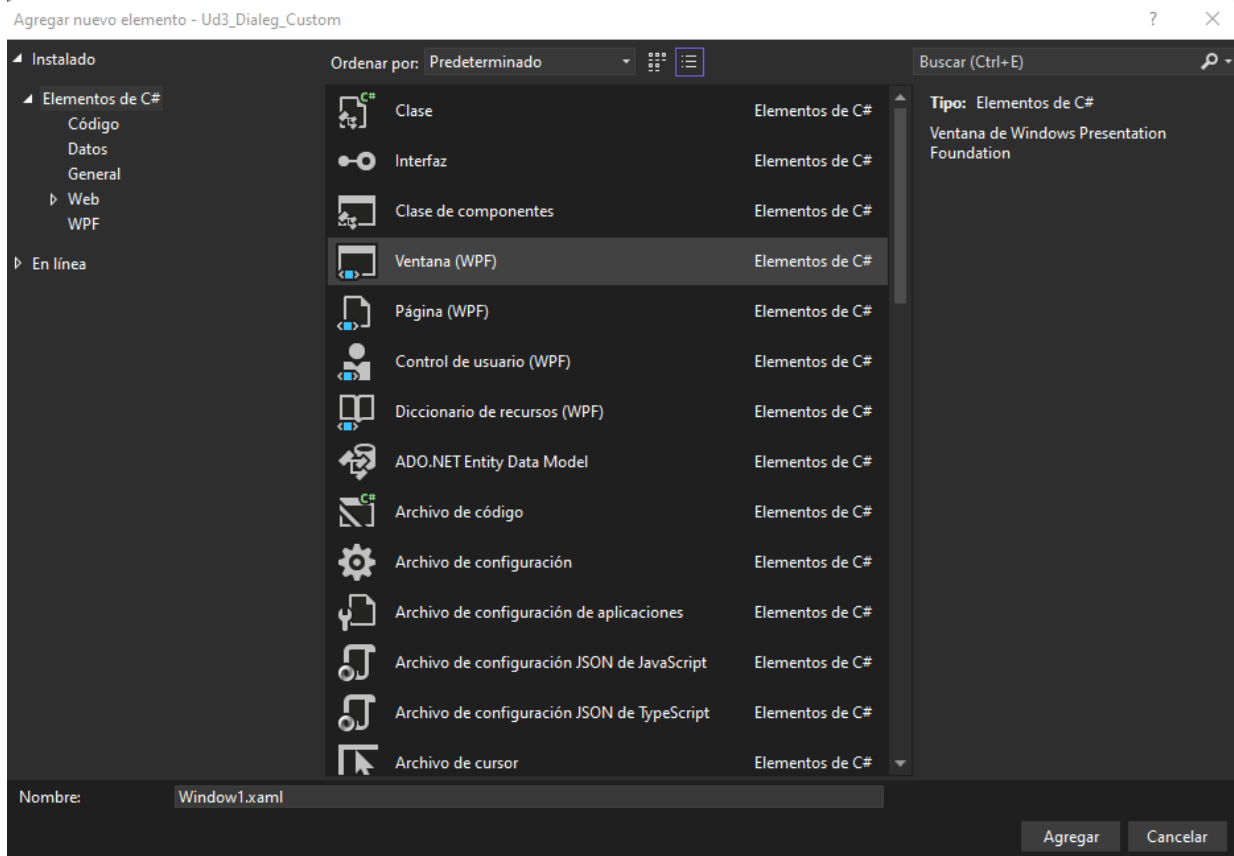
En tractar-se d'una interfície podem associar-li una classe Model-Vista igual que a la nostra pantalla principal.

Vegem un exemple pas a pas de com construir un quadre de diàleg personalitzat:

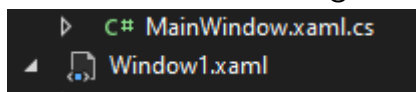
1. Una vegada creat el projecte amb la finestra principal, afegim una segona finestra (serà el nostre quadre de diàleg) polsant amb el botó dret del ratolí sobre el projecte (no sobre la sol·lució) i triant l'opció **Agregar** → **Finestra (WPF)**:



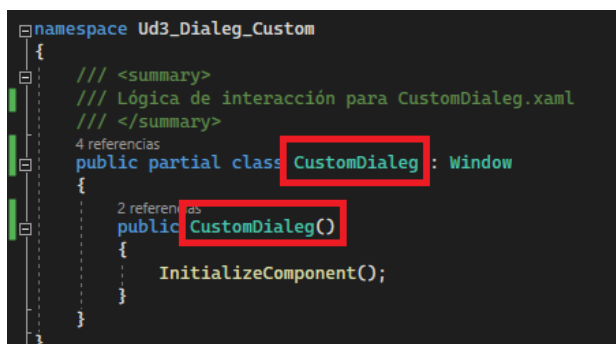
2. A continuació triem la finestra i polsem "Agregar":



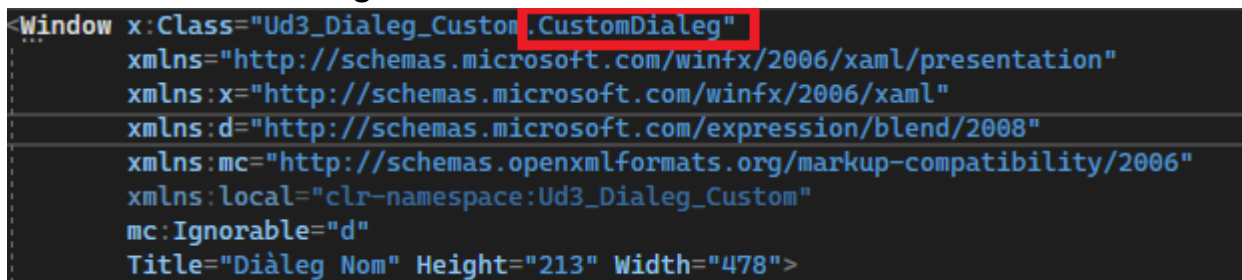
I voreu com s'ha afegit la nova finestra al explorador de sol·lucions:



Podeu canviar el nom a la finestra (l'exemple jo l'he canviat al nom CustomDialog). Si el feu recordeu canviar el codi dins del fitxer **CustomDialog.xaml.cs**:

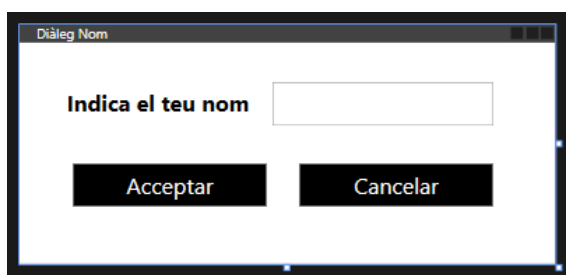


Y al fitxer **CustomDialog.xaml**:



3. A continuació podeu dissenyar la finestra al vostre gust, el meu disseny consta d'una etiqueta, un quadre de text i 2 botons:

```
<Grid>
    <Label Content="Indica el teu nom" HorizontalAlignment="Left" Margin="37,35,0,0"
    VerticalAlignment="Top" Width="184" FontSize="20" FontWeight="Bold" Height="39"/>
    <TextBox HorizontalAlignment="Left" Margin="226,35,0,0" TextWrapping="Wrap"
    VerticalAlignment="Top" Width="197" Height="39" FontSize="20"/>
    <Button x:Name="btnAcceptar" Content="Acceptar" HorizontalAlignment="Left"
    Margin="47,107,0,0" VerticalAlignment="Top" Height="39" Width="174" Foreground="White"
    Background="Black" FontSize="20" IsDefault="True"/>
    <Button x:Name="btnCancelar" Content="Cancelar" HorizontalAlignment="Left"
    Margin="249,107,0,0" VerticalAlignment="Top" Height="39" Width="174" Foreground="White"
    Background="Black" FontSize="20" IsCancel="True"/>
</Grid>
```



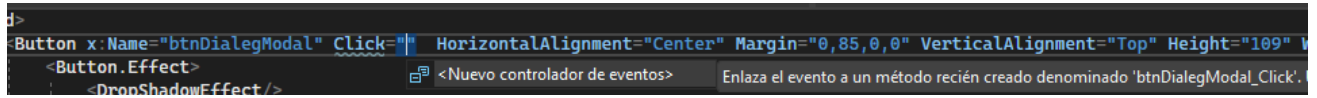
4. Ara crearem un botó a la finestra principal (**MainWindow.xaml**) per tal de poder obrir la finestra de diàleg de forma modal i una etiqueta (**lblNom**) per a rebre el text replegat pel diàleg:

```
<Grid>
    <Button x:Name="btnDialeModal" HorizontalAlignment="Center" Margin="0,85,0,0"
    VerticalAlignment="Top" Height="109" Width="300" Click="Button_Click" FontSize="24"
    FontWeight="Bold" BorderThickness="4,4,4,4" Content="Diàleg Modal">
        <Button.Effect>
            <DropShadowEffect/>
        </Button.Effect>
    </Button>
    <Label Content="Resposta " HorizontalAlignment="Left" Margin="86,240,0,0"
    VerticalAlignment="Top" FontSize="20" FontWeight="Bold"/>
    <Label x:Name="lblNom" Content="" HorizontalAlignment="Left" Margin="203,240,0,0"
    VerticalAlignment="Top" FontSize="20" FontWeight="Bold"/>
</Grid>
```



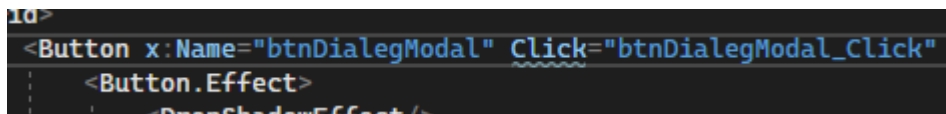
5. A continuació hem de assignar una funció o mètode per a que quan fem click amb el ratolí en el botó s'obri la finestra del diàleg personalitzat:

Escrivim l'esdeveniment Click dins de la definició del botó i apareixerà un requadre indicant si volem crear un nou Controlador d'Esdeveniments per al botó:

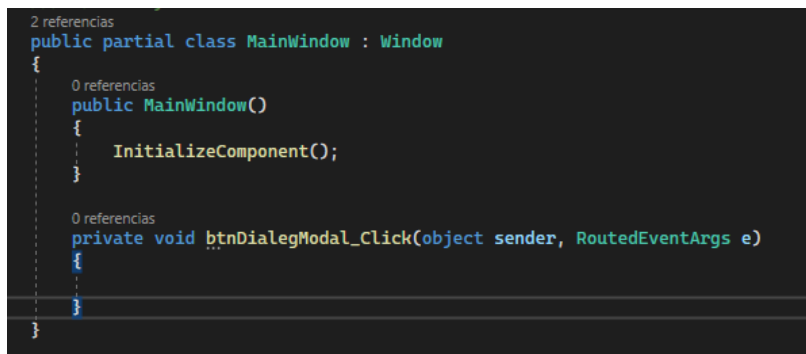


Polsem sobre l'etiqueta “*Nou controlador d'esdeveniments*” i ens crearà el mètode Click associat al botó al fitxer **MainWindows.xaml.cs** i la cridada al mateix dins de la definició del botó al fitxer **MainWindow.xaml**:

MainWindow.xaml

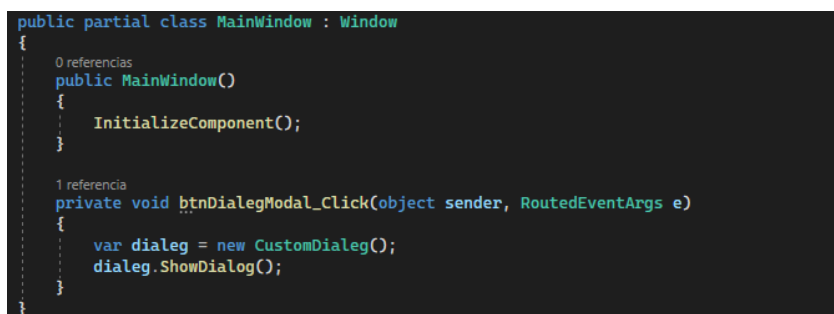


MainWindow.xaml.cs



Si no pots crear el mètode d'aquesta forma (l'entorn crea el mètode automàticament) sempre pots escriure directament el mètode de forma manual.

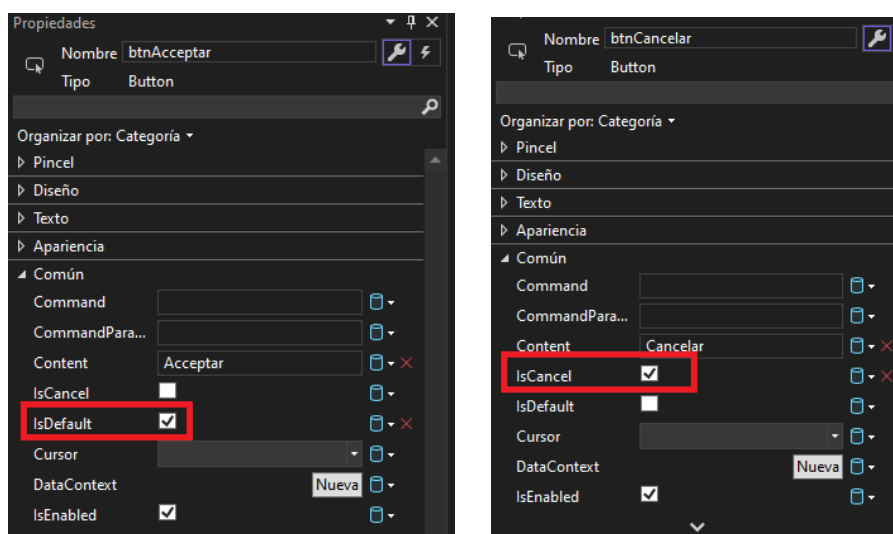
6. A continuació omplim el mètode creat en el punt anterior dins del fitxer **MainWindow.xaml.cs**:



Observa que el codi introduït es:

- Generem una instància de la finestra CustomDialog (el nostre diàleg personalitzat)
- Mostrem la finestra en forma modal (mètode ShowDialog())

7. En aquest moment ja és possible fer el clic al botó de la finestra principal i el diàleg se mostrarà. Ara modificarem les propietats dels botons del diàleg personalitzat per a que aquest retorne un valor en funció del botó pulsat. El botó **Acceptar** li canviarem la propietat **isDefault a true** i al botó **Cancel·lar** li canviarem la propietat **isCancel a true**:



8. Per acabar amb la finestra customDialog, realitzarem 3 canvis:

- Situarem el focus al quadre text per poder escriure en quant s'obriga la finestra.
- A l'esdeveniment **Click** del botó **Acceptar** establim a **true** el resultat del diàleg (i així la finestra es tancarà automàticament).
- Crearem una **propietat (Nom)** on es **retornarà** el text escrit al **TextBox** del diàleg.

```
public partial class CustomDialog : Window
{
    1 referencia
    public CustomDialog()
    {
        InitializeComponent();
        txtNom.Focus();
    }

    1 referencia
    private void btnAceptar_Click(object sender, RoutedEventArgs e)
    {
        this.DialogResult = true;
    }

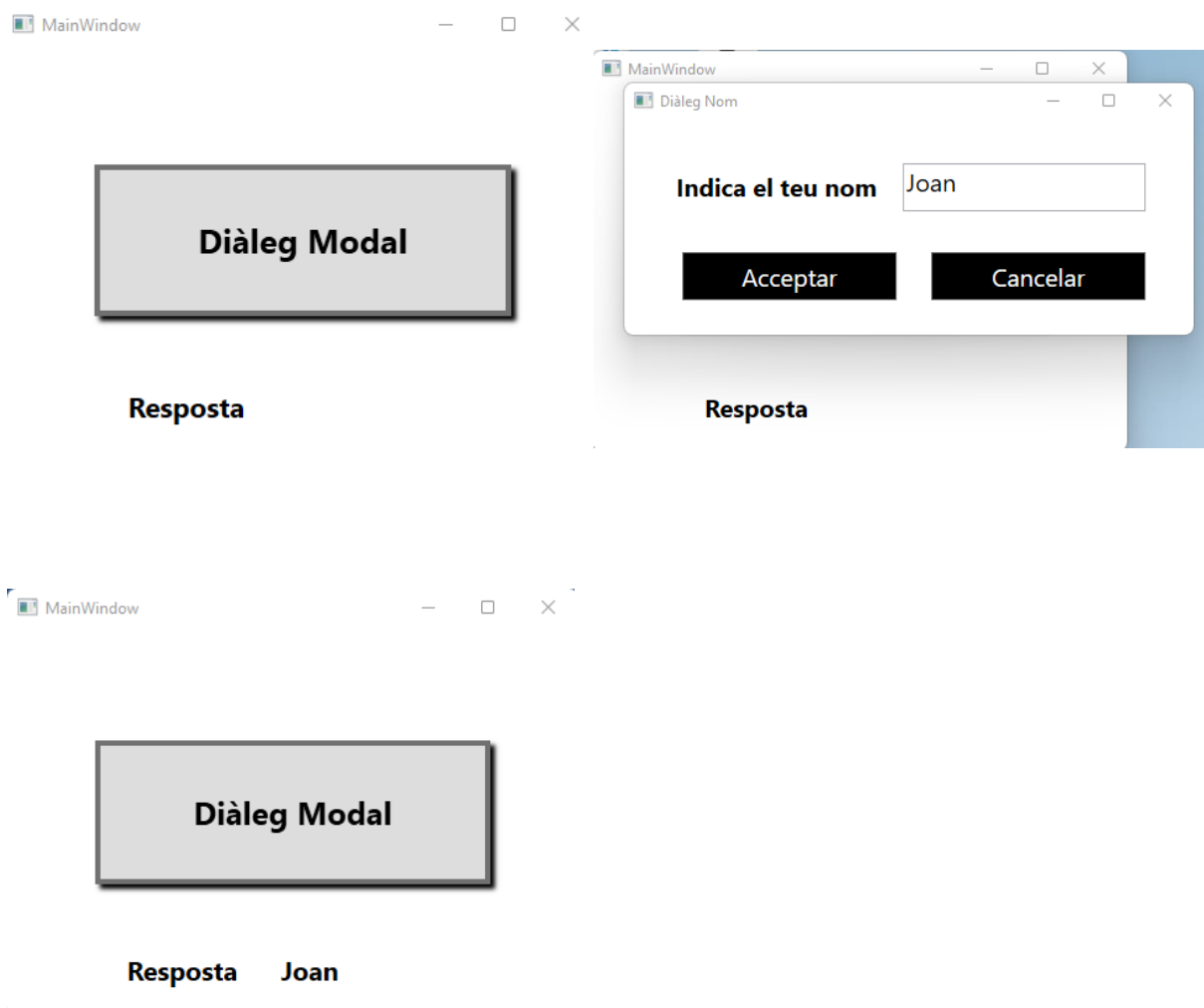
    1 referencia
    public string Nom
    {
        get { return txtNom.Text; }
    }
}
```

9. Finalment, en la finestra MainWindow (al codi C#) recuperarem el valor retornat pel diàleg (en cas d'haver polsat el botó Acceptar, es a dir el diàleg retorna un valor true):

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

    1 referencia
    private void btnDiallegModal_Click(object sender, RoutedEventArgs e)
    {
        var dialleg = new CustomDialleg();
        if (dialleg.ShowDialog() == true) lblNom.Content = dialleg.Nom;
    }
}
```

10. Finalment si executem aquest xicotet projecte, la seqüència de execució seria la següent:



3. Controls per a mostrar informació

Aquest tipus de components s'utilitzen simplement per a mostrar informació. Normalment s'utilitzen per a títols, capçaleres o descripcions dels camps de text que apareixen en un formulari.

A pesar que tenen esdeveniments associats no solen utilitzar-se, excepte en algun cas per a canviar alguna cosa en passar el ratolí per damunt.

3.1. Control Label.

Aquest component, com el seu nom indica, s'utilitza per a etiquetar o mostrar informació sobre un altre component al qual està associat. Les principals propietats a les quals està associat són les següents:

- **FontSize:** determina la grandària de la font de lletra utilitzada per l'etiqueta
- **FontWeight:** si es tracta d'una lletra negreta, semi negreta, negreta doble, ...
- **Foreground:** color de la font
- **FontFamily:** tipus de lletra
- **FontStyle:** si és cursiva, normal o obliqua
- **Content:** s'especifica la cadena de text a visualitzar.

Aquest component permet la utilització d'imatges en el seu interior.

3.2. Control TextBlock.

Aquest component també s'utilitza per a introduir text en les interfícies, no obstant això, mentre que les etiquetes s'utilitzen per a mostrar informació breu, aquest component ens permet manejar de forma més senzilla grans porcions de text. Fins i tot ens permet realitzar salts de línia.

De fet treballa millor amb text multi-línia que les etiquetes, si bé **no permeten incloure imatges** en aquesta mena de components, únicament text.

Les propietats utilitzades són les mateixes que per a una etiqueta excepte per les següents dues:

- **Text:** s'especifica el text a mostrar.
- **TextWrapping:** s'indica si es produeix un salt de línia en cas necessari per falta d'espai.

Podem introduir salts de línia en les nostres interfícies amb l'etiqueta **<LineBreak/>**

4. Camps de text

Són camps de text en els quals introduïm informació per a processar-la. Aquest tipus de controls s'utilitzen per a les dades que no poden associar-se a determinats components.

Solen estar íntimament relacionats amb els esdeveniments del teclat com: *KeyPressed*, *KeyRelease* o *KeyTyped*.

També tenen la possibilitat de desactivar-los o deixar-los únicament en mode lectura per a mostrar informació i que no es puga modificar.

Una altra propietat interessant és la de **Prompt** que ens mostra informació sobre la forma o la informació que s'ha d'introduir en el camp de text.

El component que s'utilitza en WPF per a introduir text és **TextBox**. Compte amb el component de text multi-línia que denomina de manera semblant.

WPF disposa d'aquest element per a l'entrada de text i té les següents característiques interessants:

- Té la possibilitat d'introduir text multilínia com si fora un editor. Propietat **AcceptsReturn**
- També podem activar la correcció ortogràfica del text introduït. Propietat **SpellCheck.IsEnabled** combinada amb **Language** per a indicar el llenguatge a utilitzar per a la correcció.
- Permet la selecció de text

A través de la propietat **Text** podem obtenir el text introduït.

4.1. Màscare.

Sovint hem d'afegir components d'introducció de text per a escriure DNI, codis postals, etc. No obstant això si deixem el camp de text tal qual l'usuari sol cometre errors d'introducció als quals hem d'estar atents quan recollim la dada per a processar-lo. Si no ho fem i l'usuari s'equivoca podem fer que el nostre programa falle per una cosa tan ximple.

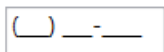
Per això existeixen components de text que ens permeten formatar l'entrada i fer que solament es puguin introduir determinades combinacions de caràcters per a evitar problemes. Per exemple, si hem de teclejar un codi postal, resulta interessant obligar al fet que solament s'introduïsquen 5 números. És cert que no garanteixes que es produïska algun error (pot introduir un codi que comence per 7), però ja els límits molt.

En WPF no tenim cap mena de control d'aquest estil encara que si en el conjunt estés que proporciona *WPFToolkit*.

El control és **MaskedTextBox** i té de diferent amb **TextBox** que la propietat **Mask** que ens permet especificar una expressió regular o màscara.

```
<xctk:MaskedTextBox x:Name="maskedTextBox" Margin="20"
Mask="(000) 000-0000" Width="80" Height="25"/>
```

Fixeu-vos en la propietat **Mask** que mostra zeros. El resultat és el que es mostra en la figura següent:



En la màscara s'especifiquen zeros, tanmateix en visualitzar-ho solament es veuen guions baixos. Això és pel fet que els zeros no són literals o caràcters fixos sinó que són expressions que tenen un significat, en aquest cas que solament es poden introduir números, més concretament només díigits del 0 al 9.

Aquest exemple s'especifica una màscara per a teclejar un número de telèfon, el qual solament pot contindre díigits. En aquest cas els parèntesis i el guió alt són **literals**, és a dir no tenen cap significat especial.

Les expressions regulars o màscares poden trobar-se en diversos llenguatges de programació o sistemes operatius per a representar en una sola expressió múltiples combinacions de caràcters.

En la taula següent podem veure els diferents caràcters que s'utilitzen per a representar diferents expressions regulars.

Mask Element	Description
0	Qualsevol dígit entre 0 i 9
9	Qualsevol dígit entre 0 i 9 o espai en blanc
#	Dígit, espai en blanc, + o -
L	Lletres de la a – z o de la A – Z . Força a posar una lletra, és requerida
?	Lletra a – z o A – Z , però és opcional posar-la
&	Qualsevol caràcter. El caràcter és requerit
C	Qualsevol caràcter, encara que és opcional
A	Qualsevol caràcter alfanumèric
A	Qualsevol caràcter alfanumèric
.	Separador de nombres decimals
,	Separador de milers
:	Separador d'hores
/	Separador de dates
\$	Símbol actual, dependrà de la propietat FormatProvider
<	Converteix tots els caràcters en minúscules
>	Converteix tots els caràcters en majúscules

	Deshabilita les expressions anteriors
\	Fuita. Si posem aquest caràcter davant d'algun dels anteriors perd el seu significat i es comporta com un literal. Per exemple \\ d'aquesta manera, la segona contrabarra es comporta com un literal

4.2. Contrasenya

Aquest control bàsicament s'utilitza per a introduir contrasenyes perquè els caràcters que s'introdueixen s'oculten amb un caràcter espacial que pot canviar-se.

O també podem limitar el nombre de caràcters que es poden a introduir.

```
<StackPanel>
    <Label Content="Password:" />
    <PasswordBox x:Name="passwordBox" Width="130" />
</StackPanel>
```

5. Botons

El control botó és un dels components més utilitzats en interfícies per a indicar alguna operació a realitzar. Sempre sol estar associat a l'esdeveniment clic.

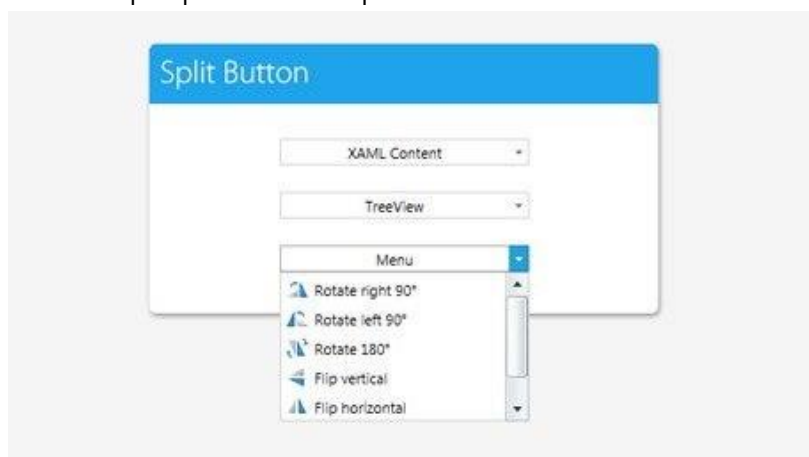
Els botons poden contindre text i imatges, per a això hem de modificar la seua propietat **Content**.

L'esdeveniment típic al qual està associat és el de Click que s'activa en punxar sobre la superfície del ratolí.

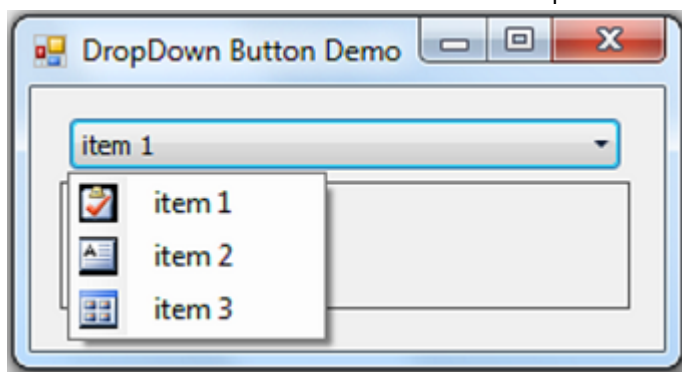
Quant a la resta d'atributs, el botó disposa dels més utilitzats per la resta de controls.

Existeixen altres tipus de botons a part de l'habitual control **button**, com per exemple **SplitButton**, **DropDownButton** o **ToggleButton**.

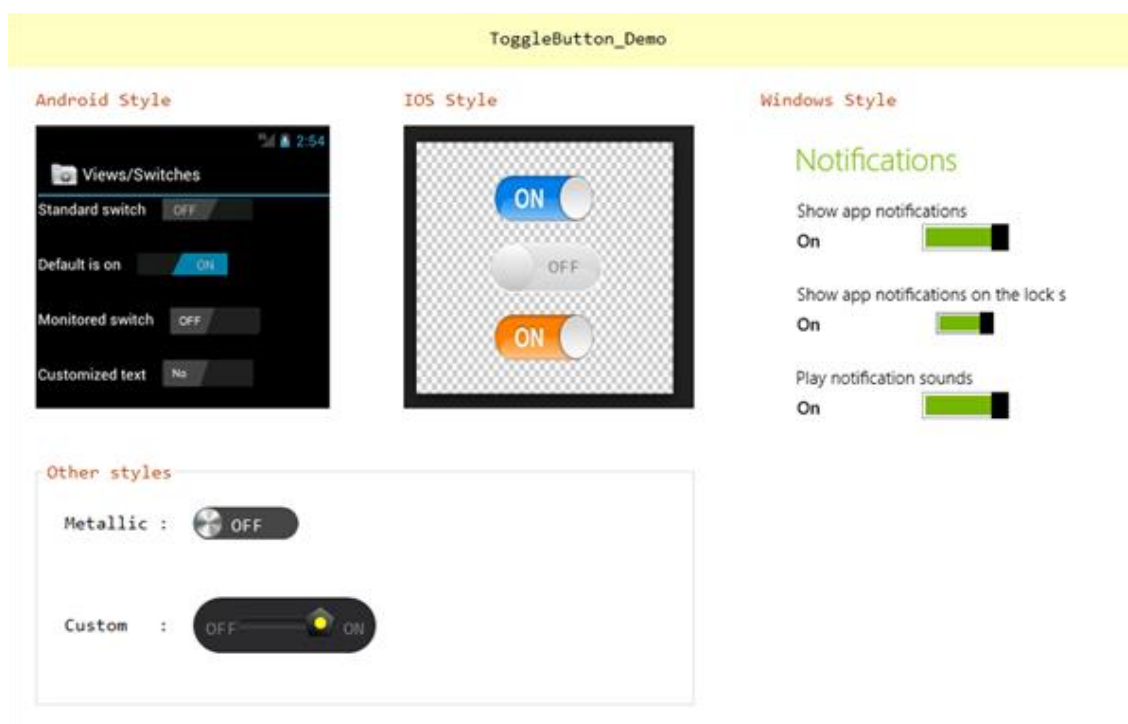
- **SplitButton**: aquest component és un botó que conté una llista d'ítems que poden ser seleccionats. El contingut del botó canvia en fer-lo l'ítem seleccionat en la llista. Disposa de la propietat **Icon** per a visualitzar una icona.



- **DropDownButton**: és similar a l'anterior però **no disposa d'una llista d'ítems sinó d'opcions de menú que poden ser seleccionades**. El contingut del botó no canvia en cas de seleccionar una de les opcions.



- **ToggleButton**: es tracta d'un botó que té dos estats: actiu o inactiu. Es comporta com un interruptor, mentre que un botó normal es comporta com un polsador.



6. Organitzadors

Aquests controls ens ajuden a organitzar i separar els components que componen un formulari. D'aquesta manera podem diferenciar els camps que tenim segons la informació que manegen. Principalment disposem de dos.

6.1. Separator

Es tracta d'un component que simplement dibuixa una línia que s'utilitza per a separar components els uns dels altres. No té cap altra funcionalitat.

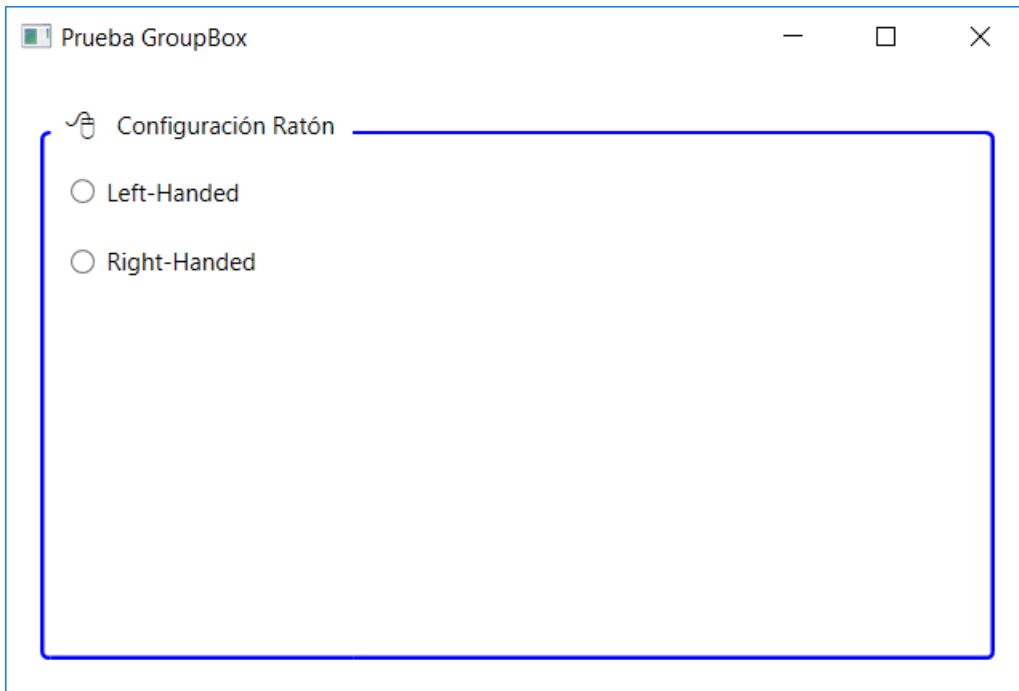
Les principals propietats utilitzades són:

- **Background:** per a canviar el color de la línia
- **BorderThickness:** per a establir el gruix de la línia

6.2. Groupbox

Aquest component és un panell amb títol i vora que s'utilitza per a col·locar controls en un formulari amb informació semblant.

Aquest component necessita col·locar una panell dins d'ell per a afegir els diferents components.



```
<GroupBox Margin="15" BorderBrush="Blue" BorderThickness="2">
  <GroupBox.Header>
    <StackPanel Orientation="Horizontal">
      <Label FontFamily="Wingdings" FontSize="17">8</Label>
      <Label Content="Configuración Ratón" />
    </StackPanel>
  </GroupBox.Header>
</GroupBox>
```

Les principals propietats a tindre en compte en aquest component són les següents:

- **Header:** es tracta del títol del panell
- **BorderBrush:** color de la vora
- **BorderThickness:** gruix de la vora

7. Consells per a formularis

- S'han d'utilitzar, sempre que es pugui, controls en els quals l'usuari no hagi de teclejar res, sinó simplement hagi de seleccionar una opció, com per exemple utilitzar llistes desplegable o controls tipus calendari. D'aquesta manera minimitzem els errors que es produeixen en teclejar, o bé els problemes per escriure les coses de diferents formes.
- Deixar clar en la interfície quins camps són obligatoris i quins no (per exemple amb un asterisc al costat).
- Introduir ajuda per a emplenar els diferents camps de manera correcta (per exemple fent ús de la propietat ToolTip dels controls).
- Introduir un assistent si s'han de completar diversos passos.
- Que el desplaçament del cursor entre els diferents camps siga ordenat i no hi haja salts estranys en passar d'un camp a un altre amb el teclat (tabulació).
- Organitzar les dades a introduir per categories i indicar les mateixes mitjançant un títol. Per exemple, si hem d'introduir dades d'un usuari, distingir entre les dades personals i els professionals.
- No col·locar en els formularis els camps o claus primàries de les taules d'una base de dades. Són d'ús intern i l'usuari no ha d'interactuar amb ells.

8. Esdeveniments

Quan es construeix una aplicació amb interfície gràfica d'usuari la manera de programar varia respecte a una aplicació de consola. En una aplicació amb interfície gràfica s'associa codi a diferents esdeveniments que pot produir l'usuari quan interactua amb l'aplicació.

De manera que, quan es genera un esdeveniment (per exemple, prémer un botó, prémer una tecla del teclat, passar el ratolí sobre una imatge, etcètera), s'executa el codi associat i, habitualment, es modifica la interfície gràfica d'usuari per a mostrar el resultat d'aqueixa execució. Quan finalitza l'execució d'aqueix codi associat, l'aplicació espera nous esdeveniments per part de l'usuari.

Depenent de la plataforma utilitzada (.NET, Java, ...) podem tindre diferents esdeveniments, no obstant això, els principals solen ser els mateixos. Els principals esdeveniments s'agrupen en diferents categories, les quals es descriuen a continuació (no confondre a com s'anomenen dins del codi, això varia en funció de la plataforma utilitzada):

Teclat

- **Key Pressed:** s'activa en pressionar una tecla (KeyDown)
- **Key Release:** s'activa en soltar una tecla (KeyUp)
- **Text Changed:** s'activa en canviar el text (TextChanged)

Ratolí:

- **Mouse Clicked:** s'activa al fer clic amb el botó del ratolí (Click)
- **Mouse Dragged:** s'activa en arrossegar el ratolí (múltiples esdeveniments Drag)
- **Mouse Entered:** s'activa en entrar el cursor del ratolí dins de l'àrea que forma el control (MouseEnter)
- **Mouse Exited:** s'activa en eixir el cursor del ratolí dins de l'àrea que forma el control (MouseLeave)
- **Mouse Move:** s'activa en moure el ratolí (MouseMove)
- **Mouse Pressed:** s'activa en prémer el botó del ratolí (Click o MouseDown)
- **Mouse Released:** s'activa en soltar el botó del ratolí (MouseUp)
- **Scroll Started:** s'activa en iniciar el scroll, bé siga en desplaçar la roda del ratolí o mitjançant la detecció del moviment del dit (MouseWheel)
- **Scroll Finished:** és l'esdeveniment complementari a l'anterior i indica la finalització del moviment de scroll. (MouseWheel)

Window

- **Window Shown:** en fer-se visible el component es produeix l'esdeveniment (Loaded)
- **Window Hidden:** s'activa en desaparèixer el component (IsVisibleChanged)
- **Window Moved:** s'activa en canviar de posició el component (LocationChanged)
- **WindowResized:** invocat quan el component canvia de grandària (SizeChanged)

Focus

- **Focus:** el component obté el focus (GotFocus)
- **Blur:** el component perd el focus (LostFocus)

Item

- **ItemChanged:** s'activa en canviar l'element. Se sol utilitzar en els components List i ComboBox per a esbrinar si se selecciona un ítem o element de la llista. (SelectionChanged)

Drag and Drop

- **On Drag Detected:** s'activa en detectar el moviment de drag and drop (múltiples esdeveniments Drag)
- **On Drag Done:** s'activa en finalitzar l'acció d'arrossegar i soltar
- **On Drag Entered:** s'activa en iniciar el moviment
- **On Drag Exited:** s'activa en finalitzar el moviment

Existeixen més esdeveniments que anirem analitzant posteriorment perquè ja no són comuns si no específics d'alguns components.

Una llista general d'esdeveniments podeu trobar-la ací <https://learn.microsoft.com/es-es/windows/uwp/xaml-platform/events-and-routed-events-overview> (quasi al final de la pàgina està la llista de **esdeveniments enrutats**)

Com ja hem vist abans, una forma de treballar és declarar l'esdeveniment i el mètode que s'ha d'executar quan és produïda al codi **xaml** i el contingut d'aquest mètode (el codi que es va a executar) en **C#**.

8.1. Controlador d'esdeveniments

Després de produir-se algun esdeveniment dels anteriors, el nostre programa ha de realitzar alguna acció. Els esdeveniments es notifiquen i se'ns indica el tipus d'esdeveniment i el component sobre el qual es produeix.

Per a associar **l'esdeveniment que es produeix amb el nostre codi es creen uns mètodes especials anomenats controladors d'esdeveniments (event handlers)**. Aquests mètodes són cridats en produir-se un esdeveniment i s'executa el codi que hi ha en ell. Si volem gestionar algun esdeveniment l'hem d'especificar en el nostre codi. Veurem un xicotet exemple i l'analizem pas a pas.

En primer lloc definim una interfície senzilla en la qual afegim una etiqueta i un botó:

```
<Window x:Class="WpfApplication1.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:WpfApplication1"
    mc:Ignorable="d"
    Title="MainWindow" Height="525" Width="707.836">
    <Grid Background="LightBlue">
        <Label Content="Label" HorizontalAlignment="Left" Margin="75,46,0,0"
            VerticalAlignment="Top" MouseUp="Label_MouseUp"/>
        <Button x:Name="button" Content="Button" HorizontalAlignment="Left" Margin="247,227,0,0"
            VerticalAlignment="Top" Width="109" Height="47"/>
    </Grid>
</Window>
```

De moment únicament hem creat un esdeveniment la etiqueta, concretament **MouseUp**. Li hem assignat el valor **Label_MouseUp** que no és més que el nom del mètode que definim en la classe associada a la pàgina principal perquè controle l'esdeveniment quan es produïda. El codi associat a la finestra principal és:

```

namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            private void Label_MouseUp(object sender, MouseButtonEventArgs e) {
                MessageBox.Show("You clicked me at " + e.GetPosition(this).ToString());
            }
        }
    }
}

```

S'associa el component i l'esdeveniment que es produeix sobre el component amb el mètode que manejarà l'esdeveniment. Quan l'usuari solte el botó del ratolí sobre l'etiqueta, llavors s'executarà el mètode associat, que en aquest cas traurà un missatge que indicarà la posició sobre la qual s'ha soltat el botó del ratolí.

Aquest esdeveniment l'hem definit manualment en el codi XAML i el codi de C#, no obstant això podem fer que l'entorn el faça de manera automàtica en prémer dues vegades sobre el component en la pantalla de disseny.

Aquesta manera d'actuar ens condiciona al fet que l'esdeveniment que s'utilitzarà serà el que el component té assignat per defecte. Per exemple, en el nostre cas si estrenyem dues vegades el botó en la pantalla de disseny s'associarà a l'esdeveniment **Click** que s'activarà en punxar amb el ratolí el botó.

Ara l'entorn ha afegit `Click="button_Click"` en la declaració de l'etiqueta associada al botó i de pas ha generat un nou mètode en la classe controladora de la pantalla principal.

```

private void button_Click(object sender, RoutedEventArgs e)
{
    ((Control)sender).Background = Brushes.BlueViolet;
}

```

8.2. Subscripció a esdeveniments desde C# (Code-behind).

Altra forma de treballar amb els events es associar-los o capturar-los directament d'un control concret, un botó, una caixa de text, ... dins del codi C# associat al fitxer xaml.

Això es realitza assignant la funció o mètode que gestionarà l'esdeveniment directament a l'esdeveniment de l'objecte, utilitzant la **sintaxi +=** per a afegir l'esdeveniment a l'element. Ací hi ha un exemple:

Tenint aquest codi XAML:

```
<Window x:Class="WpfApplication1.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:WpfApplication1"
        mc:Ignorable="d"
        Title="MainWindow" Height="525" Width="707.836">
    <Grid Background="LightBlue">
        <Label x:Name="label" Content="Label" HorizontalAlignment="Left" Margin="75,46,0,0"
        VerticalAlignment="Top" />
        <Button x:Name="button" Content="Button" HorizontalAlignment="Left" Margin="247,227,0,0"
        VerticalAlignment="Top" Width="109" Height="47"/>
    </Grid>
</Window>
```

El codi C# associat on assignaríem un controlador d'esdeveniments a l'esdeveniment MouseUp de l'etiqueta seria el següent:

```
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            label.MouseUp += new MouseButtonEventHandler(Label_MouseUp);
        }

        private void Label_MouseUp(object sender, MouseButtonEventArgs e) {
            MessageBox.Show("You clicked me at " + e.GetPosition(this).ToString());
        }
    }
}
```

8.3. Exemples d'Esdeveniments.

A continuació vorem alguns exemples de codi d'esdeveniments típics de cada element:

Esdeveniments de Ratolí:

Els esdeveniments típics del ratolí sobre un botó poden ser: Click, MouseRightButtonDown, MouseRightButtonUp, MouseMove, MouseEnter i MouseLeave.

Un exemple de codi XAML que capture tots aquests esdeveniments seria aquest:

```
Window x:Class="Ud3_Esdeveniments_Ratolí.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:Ud3_Esdeveniments_Ratolí"
mc:Ignorable="d"
Title="MainWindow" Height="561" Width="590">
<Grid>
<Button x:Name="btnEvent" Content="Esdeveniments" HorizontalAlignment="Left" Height="99" Margin="72,46,0,0" VerticalAlignment="Top"
Width="441" FontSize="24" Background="Black" Foreground="White" FontWeight="Bold"
Click="Button_Click"
MouseMove="Button_MouseMove"
MouseEnter="Button_MouseEnter"
MouseLeave="Button_MouseLeave"
MouseRightButtonDown="btnEvent_MouseRightButtonDown"
MouseRightButtonUp="btnEvent_MouseRightButtonUp"
/>
<TextBlock x:Name="LogEsdeveniments" HorizontalAlignment="Left" Margin="72,166,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="310" Width="415"/>
</Grid>
</Window>
```

I el codi C# associat podria ser aquest:

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

1 referencia
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Has fet Clic al botó " + System.Environment.NewLine;
}

1 referencia
private void Button_MouseEnter(object sender, MouseEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseEnter a la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}

1 referencia
private void Button_MouseLeave(object sender, MouseEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseLeave a la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}

1 referencia
private void Button_MouseMove(object sender, MouseEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseMove a la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}

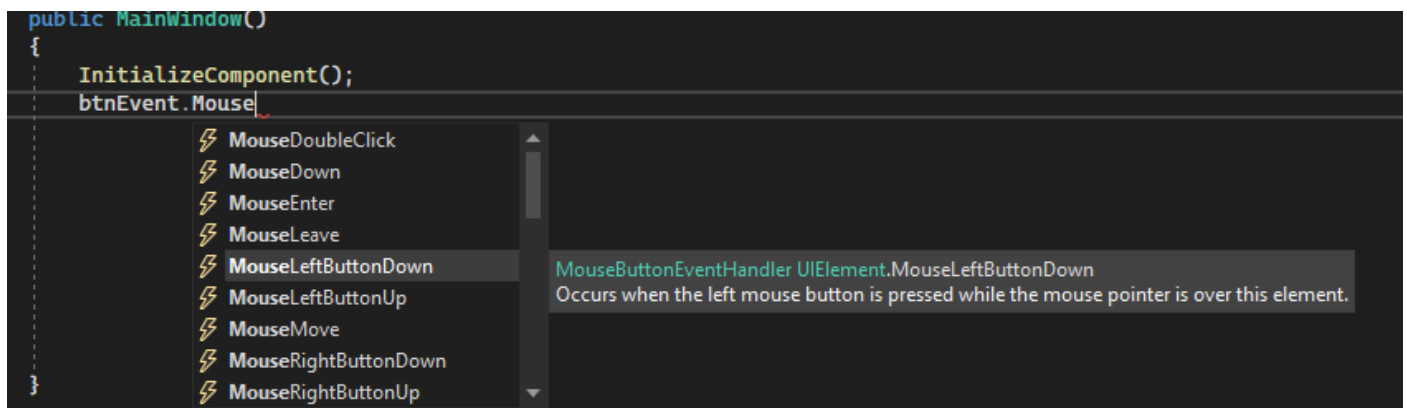
1 referencia
private void btnEvent_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseRightButtonDown a la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}

1 referencia
private void btnEvent_MouseRightButtonUp(object sender, MouseButtonEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseRightButtonUp a la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}
```

Observa que el codi va escrivint sobre un TextBlock els esdeveniments que van ocorrint i quan arriba a una grandària de 1000 caràcters s'esborra completament.

Nota important: Compte !! No tots els esdeveniments funcionen ambs tots els elements. Per exemple els esdeveniments MouseUp i MouseDown no estan presents al ratolí en canvi si ho està a l'element etiqueta.

Si vols conèixer quins son els esdeveniments presents en un element, una forma fàcil és escriure el nom de l'element y un "." i recórrer la llista que apareix buscant l'esdeveniment:



Esdeveniments de Teclat:

Els esdeveniments típics del teclat sobre un textbox poden ser: KeyDown, KeyUp, GotFocus, LostFocus.

Un exemple de codi XAML que capture tots aquests esdeveniments seria aquest:

```
<Window x:Class="Ud3_Esdeveniments_Teclat.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:Ud3_Esdeveniments_Teclat"
        mc:Ignorable="d"
        Title="MainWindow" Height="561" Width="590">
    <Grid>
        <TextBox x:Name="txtEvent" Text="Esdeveniments" HorizontalAlignment="Left" Height="59" Margin="72,40,0,0" VerticalAlignment="Top"
                Width="441" FontSize="24" Background="Black" Foreground="White" FontWeight="Bold"
                KeyDown="txtEvent_KeyDown"
                KeyUp="txtEvent_KeyUp"
                GotFocus="txtEvent_GotFocus"
                LostFocus="txtEvent_LostFocus"
                />
        <TextBlock x:Name="LogEsdeveniments" HorizontalAlignment="Left" Margin="77,128,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="100" />
    </Grid>
</Window>
```

I el codi C# associat podria ser aquest:

```

1 referencia
private void txtEvent_KeyDown(object sender, KeyEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment KeyDown. Tecla polsada: " + e.Key.ToString() + System.Environment.NewLine;
}

1 referencia
private void txtEvent_KeyUp(object sender, KeyEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment KeyUp. Tecla polsada: " + e.Key.ToString() + System.Environment.NewLine;
}

1 referencia
private void txtEvent_GotFocus(object sender, RoutedEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment GotFocus detectat al control " + e.Source.GetType().Name.ToString() + System.Environment.NewLine;
}

1 referencia
private void txtEvent_LostFocus(object sender, RoutedEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment LostFocus detectat al control " + e.Source.GetType().Name.ToString() + System.Environment.NewLine;
}

```

Esdeveniments de Finestra:

Els esdeveniments típics que afecten a una finestra poden ser: `SizeChanged`, `MouseMove`, `LocationChanged`, `StateChanged`.

Un exemple de codi XAML que capture tots aquests esdeveniments seria aquest:

```

<Window x:Class="Ud3_Esdeveniments_Finestra.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:Ud3_Esdeveniments_Finestra"
        mc:Ignorable="d"
        Title="MainWindow" Height="561" Width="590"

        SizeChanged="Window_SizeChanged"
        MouseMove="Window_MouseMove"
        LocationChanged="Window_LocationChanged"
        StateChanged="Window_StateChanged" >

    <Grid>

```

I el codi C# associat podria ser aquest:

```

1 referencia
private void Window_SizeChanged(object sender, SizeChangedEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment SizeChanged amb un nou tamany de " + e.NewSize.ToString() + System.Environment.NewLine;
}

1 referencia
private void Window_MouseMove(object sender, MouseEventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment MouseMove sobre la posició " + e.GetPosition(this).ToString() + System.Environment.NewLine;
}

1 referencia
private void Window_StateChanged(object sender, EventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment StateChanged " + this.WindowState.ToString() + System.Environment.NewLine;
}

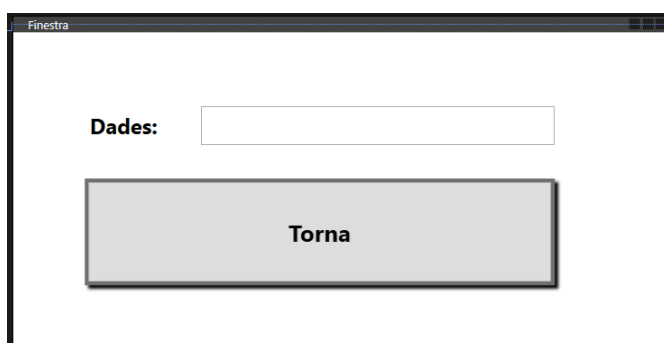
1 referencia
private void Window_LocationChanged(object sender, EventArgs e)
{
    if (LogEsdeveniments.Text.Length > 1000) LogEsdeveniments.Text = "";
    LogEsdeveniments.Text = LogEsdeveniments.Text + "Esdeveniment LocationChanged amb nova posició " + this.Left.ToString() + "," + this.Top.ToString() + System.Environment.NewLine;
}

```

Navegació entre finestres (forma 1):

Altra característica important de les finestres és la navegació entre elles, és a dir canviar el control de l'aplicació entre finestres. Per veure aquest exemple partirem d'un exemple similar al del diàleg personalitzat. Tindrem aquestes dues finestres:

- Una finestra principal (**MainWindow**) amb un botó (que obrirà l'altra finestra i és tancarà a si mateixa) i una etiqueta per a rebre dades de l'altra finestra.
- Altra finestra amb un TextBox i un botó per a tornar a la finestra anterior.



El codi XAML de la finestra principal seria el següent:

```
<Window x:Class="Ud3_Finestres.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:Ud3_Finestres"
        mc:Ignorable="d"
        Title="MainWindow" Height="341" Width="669">
    <Grid>
        <Button x:Name="btnFinestra" Click="btnFinestra_Click" HorizontalAlignment="Center" Margin="192,75,0,0">
            <Button.Effect>
                <DropShadowEffect/>
            </Button.Effect>
        </Button>
        <Label Content="Resposta Altra Finestra" HorizontalAlignment="Left" Margin="86,240,0,0">
        <Label x:Name="lblResposta" Content="" HorizontalAlignment="Left" Margin="203,240,0,0">
    </Grid>
</Window>
```

Observa el mètode associat a l'esdeveniment Click

El codi C# d'aquesta finestra seria aquest:

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

    1 referencia
    private void btnFinestra_Click(object sender, RoutedEventArgs e)
    {
        var finestra = new Finestra();
        finestra.Show();
        this.Close();
    }
}
```

Observa com s'obri l'altra finestra i es tanca a si mateixa.

Si pasem a la segona finestra, el seu codi XAML seria aquest:

```
<Grid>
    <Label Content="Dades: " HorizontalAlignment="Left" Margin="73,75,0,0" VerticalAlignment="Top" FontSize="22">
    <TextBox HorizontalAlignment="Left" Margin="192,75,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="300">
    <Button x:Name="btnTornar" Click="btnTornar_Click" HorizontalAlignment="Left" Margin="73,149,0,0" VerticalAlignment="Top">
        <Button.Effect>
            <DropShadowEffect/>
        </Button.Effect>
    </Button>
</Grid>
```

Observa el mètode associat a l'esdeveniment Click del botó.

I el codi C# seria aquest:

```
public partial class Finestra : Window
{
    1 referencia
    public Finestra()
    {
        InitializeComponent();
    }

    1 referencia
    private void btnTornar_Click(object sender, RoutedEventArgs e)
    {
        var principal= new MainWindow();
        principal.Show();
        this.Close();
    }
}
```

És molt evident com utilitzar els mètodes **show** y **close** per a obrir i tancar les finestres.

Però i ¿com ens passem les dades entre elles? Una forma molt ràpida es accedir al control de la finestra destí on volem situar les dades replegades. En aquest exemple podríem deixar les dades replegades a l'etiqueta de la finestra principal al moment de passar el control a eixa finestra:

```
1 referencia
private void btnTornar_Click(object sender, RoutedEventArgs e)
{
    var principal= new MainWindow();
    principal.lblResposta.Content = txtDades.Text;
    principal.Show();
    this.Close();
}
```

Navegació entre finestres (forma 2):

Altra forma de passar informació entre finestres seria sobrecarregar el constructor de la finestra que va a rebre la informació redefinint-lo d'aquesta forma (Pensem que la finestra principal **MainWindow** va a rebre la informació):

```
0 referencias
public MainWindow()
{
    InitializeComponent();
}

1 referencia
public MainWindow(string Dades)
{
    InitializeComponent();
    lblResposta.Content = Dades;
}
```

Es redefineix el constructor afegint un paràmetre (En aquesta ocasió tipus string) i se passa al control que ha de mostrar la informació (lblResposta).

Per tant la cridada a aquesta finestra des de la finestra secundaria variarà d'aquesta forma:

```
5 referencias
public partial class Finestra : Window
{
    1 referencia
    public Finestra()
    {
        InitializeComponent();
    }

    1 referencia
    private void btnTornar_Click(object sender, RoutedEventArgs e)
    {
        var principal = new MainWindow(txtDades.Text);
        principal.Show();
        this.Close();
    }
}
```

Al instanciar la finestra destí (MainWindow) se li passa el paràmetre a passar tal i com es va definir al constructor de la finestra destí.

Aquestes 2 formes poden utilitzar-se tant amb variables simples (string, integer, ...) com amb objectes, col·leccions o variables més complexes.

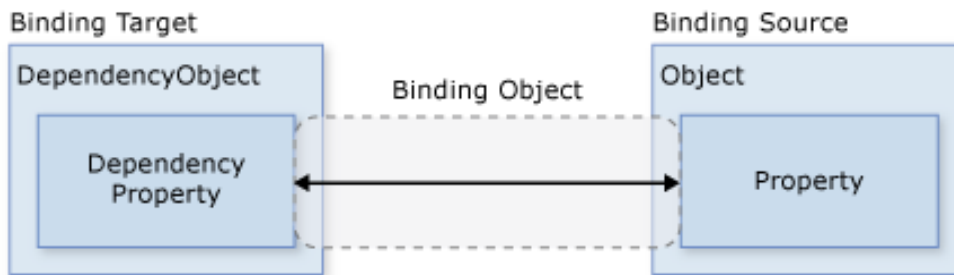
**Però, ¿aquestes serien les formes més correctes de passar les dades d'una finestra a altra?
¿Quina seria la forma més adequada?**

9. Bindings

Data Binding és el procés que estableix una connexió entre la interfície d'usuari de l'aplicació i la lògica de negoci o controlador. Quan les dades canvien el seu valor, els elements que estan vinculats a les dades reflecteixen els canvis de manera automàtica. És a dir, si escrivim en una caixa de text informació, aquesta s'actualitzarà en les variables de la nostra capa de lògica de negoci.

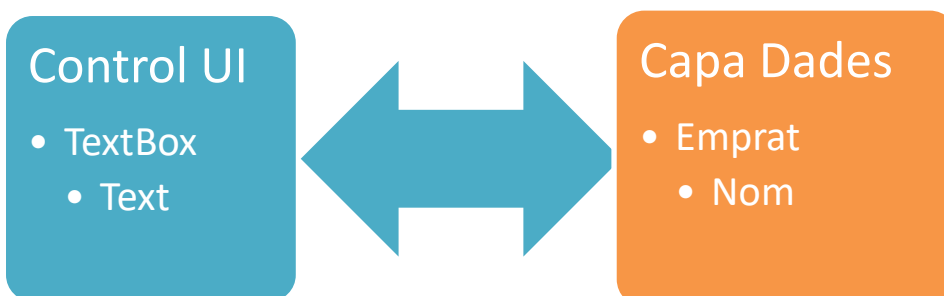
9.1. Elements.

Anem a veure els diferents elements involucrats en el procés de binding . Normalment hi ha quatre components involucrats en el procés de Data Binding:



- **Objecte destí (Binding Target):** normalment es tracta d'un component UI de WPF
- **Propietat destí (Target Property):** es tracta de la propietat del control que estarà sincronitzada amb la corresponent propietat en el model de dades. Aquestes propietats **obligatòriament han de ser Dependency Property**. La majoria de les propietats dels components són *Dependency Property* i per tant poden ser susceptibles d'utilitzar *Data Binding* excepte aquelles que siguin de només lectura.
- **Objecte origen (Binding Source):** generalment és l'objecte del nostre model de dades, encara que també pot ser un altre control UI. L'objecte del model de dades pot tindre diverses fonts: un objecte, o una llista.
- **Propietat origen (Source Property):** la propietat del model de dades que se sincronitza amb la propietat del control. També pot ser la propietat d'un altre control.
- **Objecte d'enllaç (Binding Object):** es tracta de l'objecte que s'encarrega de la **sincronització** entre les propietats dels objectes font i destí.

Posem un exemple:



Segons la imatge anterior volem que existisca un procés de sincronització (data binding) entre la propietat **Text** (propietat destí) d'un control **TextBox** (objecte destí) i el **Nom** (propietat origen) del nostre objecte **Emprat** (objecte origen) que es troba en la nostra capa d'accés a dades.

Aquesta unió fa que tot allò que escriguem en el nostre camp de text se sincronitzarà amb la propietat Nom del nostre objecte Emprat.

Posem un altre exemple que ens mostre l'enllaç o data binding entre components de la nostra interfície.

```
<Window x:Class="WpfDataBindingInControls.TextBsAndSlider"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="TextBsAndSlider" Height="300" Width="300">
    <Grid>
        <Slider HorizontalAlignment="Left" Margin="10,71,0,0"
            Name="slider" VerticalAlignment="Top" Width="272"
            Value="{Binding ElementName=textBox, Path=Text}"
            Minimum="10" Maximum="20"/>
        <TextBlock HorizontalAlignment="Left" Margin="10,107,0,0"
            TextWrapping="Wrap" Text="Eminem" VerticalAlignment="Top" Width="272" Height="152"
            Name="textBlock"
            FontSize="{Binding ElementName=slider, Path=Value}"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="10,26,0,0" Name="textBox"
            TextWrapping="Wrap" VerticalAlignment="Top" Width="272"
            Text="{Binding ElementName=slider, Path=Value}"/>
    </Grid>
</Window>
```

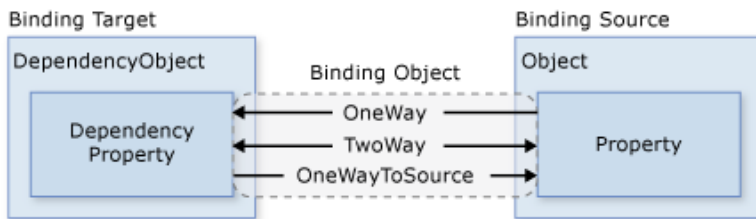
En aquest exemple veiem com hem enllaçat el valor d'un **slider** amb la **grandària de font d'un TextBlock**. La qual cosa fa que en canviar el valor del slider també canviarà la grandària de la font del Textblock.

De la mateixa forma hem enllaçat el **valor del slider** amb el **text que inserim en un Textbox**, amb la qual cosa veurem:



9.2. Direcció.

També tenim la possibilitat de seleccionar la direcció de la sincronització, és a dir, podem fer que en un sentit es facen els canvis però no en l'altre. Existeixen tres formes d'enllaçar la informació:



- **One-way:** en aquest cas solament s'actualitza si els canvis es realitzen en la propietat origen, no obstant això, si els canvis es realitzen en la propietat destí, aquests no s'actualitzaran. Aquest tipus d'enllaç té sentit en cas de disposar de controls de només lectura. Si és possible, resulta millor utilitzar aquest tipus de Data Binding que el següent per reduir la sobrecàrrega que té la comprovació del canvi de valor d'una propietat. **És la forma per defecte de Data Binding**, excepte en les propietats típicament editables per l'usuari com el text d'un TextBox o el check d'una Casella de selecció.
- **Two-way:** en aquest cas, si la font o origen canvia, llavors la destinació canvia i viceversa.
- **OneWayToSource:** és la contrària a la primera forma, és a dir, els canvis en la destinació són actualitzats en l'origen, però no al contrari.

9.3. Sincronització.

En els punts anteriors hem vist els elements involucrats, la direcció de l'actualització, i ara veurem el moment en el qual es realitza la sincronització.

Quan estem teclejant en un camp de text, **en quin moment es realitza l'actualització de la nostra propietat?**

Podem actualitzar-la mentre teclegem o quan el control perd el focus. Aquest comportament es pot modificar amb la propietat **UpdateSourceTrigger** que té els següents valors:

- **PropertyChanged:** s'actualitza cada vegada que la propietat canvia.
- **LostFocus:** s'actualitza quan el component perd el focus.

El valor per defecte de la majoria de les propietats és **PropertyChanged** excepte per a la propietat **Text** que té com a valor per defecte a **LostFocus**.

9.4. Crear un Binding.

Començarem per un enllaç (binding) senzill com és per exemple fer que el que escriguem en un camp de text es reproduïska en una etiqueta.

En aquest cas l'objecte destí serà el component **Label**, mentre que la propietat destí serà **Content** d'aquest Label.

L'objecte origen serà un **TextBox** i la seua propietat **Text** serà la propietat font o origen. Vegem el codi com quedaria.

```
<TextBox Width="150" x:Name="txtNombre"/>
<Label Margin="10,0,0,0" Content="{Binding Path=Text, ElementName=txtNombre}" />
```

Podem veure que en l'objecte i propietat destí especifiquem que realitzarem una sincronització amb la propietat **Text** (ho especifiquem amb la **propietat Path**) de l'objecte **TextBox** (L'especifiquem amb la **propietat ElementName**).

9.5. Binding amb un model de dades.

Aquest tipus de Binding s'utilitza per a sincronitzar propietats de diferents components. També podem fer que existisca una sincronització entre una propietat d'un control i un objecte dels nostres models de dades. Per exemple, podem tindre una llista en les nostres classes de C# que siga utilitzada per un component **ListBox** per a mostrar les seues dades.

A continuació descriurem l'exemple que utilitzarem per a explicar el concepte. Tenim dues classes per al nostre model de dades; **la primera es diu Client** que té les propietats d'un client; **la segona es una col·lecció de Clients** que genera una llista de clients (**ListaClients**).

El contingut del fitxer **MainWindow.xaml.cs** seria el següent:

```
namespace Ud3_Binding_ListBox
{
    public partial class MainWindow : Window
    {
        public ObservableCollection<Client> ListaClients { get; set; }
        public MainWindow()
        {
            ListaClients = new ObservableCollection<Client>() {
                new Client("Carles", "Aldaia", "23432543"),
                new Client("Joan", "València", "234325325"),
                new Client("Rosa", "Torrent", "545675676"),
                new Client("Pere", "València", "567658768")
            };
            InitializeComponent();
            DataContext = this;
        }
    }

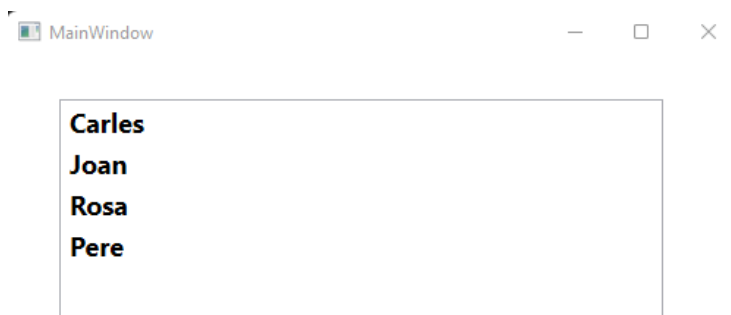
    public class Client
    {
        public string Nom { get; set; }
        public string Ciutat { get; set; }
        public string Telefon { get; set; }
        public Client(string nom, string ciutat, string telefon)
        {
            this.Nom = nom;
            this.Ciutat = ciutat;
            this.Telefon = telefon;
        }
    }
}
```

Després li indiquem al component la llista on ha d'agafar les dades per a mostrar-los mitjançant la propietat **ItemsSource** i quins camps mostrar amb la propietat **DisplayMemberPath**:

Contingut del fitxer **MainWindow.xaml**:

```
<ListBox Grid.Column="0" Grid.Row="2" Grid.ColumnSpan="2" Grid.RowSpan="5"
ItemsSource="{Binding ListaClients}" DisplayMemberPath="Nom"
Margin="36,30,54,119" FontSize="18" FontWeight="Bold"/>
```

Resultat:



9.6. Relative Source.

Hem vist com podem realitzar un data binding entre propietats de diferents objectes. No obstant això, imaginem que hem d'enllaçar dues propietats del mateix objecte, o bé propietats de l'actual objecte i una altra d'un antecessor o pare d'ell. Per a poder fer-ho utilitzarem l'etiqueta **RelativeSource**.

Enllaçar les propietats del mateix objecte

En el següent exemple enllacem la propietat amplària d'un rectangle amb la propietat altura del mateix per a aconseguir un quadrat.

```
<Rectangle Fill="Red" Height="100"
Stroke="Black"
Width="{Binding RelativeSource={RelativeSource Self},
Path=Height}"/>
```

També podem enllaçar una propietat del nostre objecte amb una propietat del seu pare mitjançant l'etiqueta **self**.

```
<TextBlock Width="{Binding RelativeSource={RelativeSource Self},
Path=Parent.ActualWidth}"/>
```

En aquest cas l'amplària del component **TextBlock** serà la mateixa que la del component pare.

Enllaçar amb un antecessor

En aquest cas s'enllaçarà la propietat del component amb el primer antecessor que siga d'un determinat tipus. Pot ser el seu pare, avi, besavi, ...

```
<TextBlock FontSize="16" Margin="50"  
  Text="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Border}}"  
  Width="200"/>
```

En aquest exemple mostrarem com a text del control **TextBlock** el nom del primer antecessor que siga de tipus **Border**.

També podem incloure el nombre de nivells en els quals buscar.

```
<TextBlock FontSize="16" Margin="50"  
  Text="{Binding RelativeSource={RelativeSource FindAncestor,  
    AncestorType={x:Type Border},  
    AncestorLevel=2},Path=Name}"  
  Width="200"/>
```