

Desenvolupament d'interfícies

Unitat 7. Documentació d'aplicacions. Validació de dades i Logging.



Índex

Índex.....	1
1. Per què és important documentar?	2
2. Tipus de documentació	3
2.1. Documentació de proves.....	3
2.2. Documentació tècnica.....	3
3. Manuals i guies	4
3.1. Manual i guia d'usuari	4
3.2. Manual i guia d'explotació	4
3.3. Guia ràpida i guia de referència.....	5
4. Fitxers d'ajuda. Formats. Eines per a generar-los	6
4.1. Formats.....	6
4.2. Eines per a generar fitxers d'ajuda	7
4.3. Ajuda en format HTML amb el control WebBrowser de WPF	7
4.4. Ajuda genèrica i sensible al context.....	11
4.5. Creació d'un tooltip en WPF	11
5. Taules de continguts.....	13
6. Validació.....	14
6.1. Validation Rules	14
6.2. Exception Validation Rules.....	17
6.3. IDataErrorInfo	19
6.4. DataAnnotations.....	21
7. Creació de Logs o Logging	23

1. Per què és important documentar?

Quan es parla de documentació d'aplicacions de programari ens referim a tots els elements que detallen i aclareixen les característiques d'una aplicació, i que poden ser necessaris per al seu ús o modificació. En concret, es tracta de fitxers d'ajuda, manuals i altres guies d'ús i/o manteniment, ja siguin dirigides per als usuaris de l'aplicació, per a equips de suport o fins i tot per a altres programadors.



Per a un sistema programari és especialment important comptar amb una documentació completa i que siga senzilla de consultar. Els motius per a això són diversos:

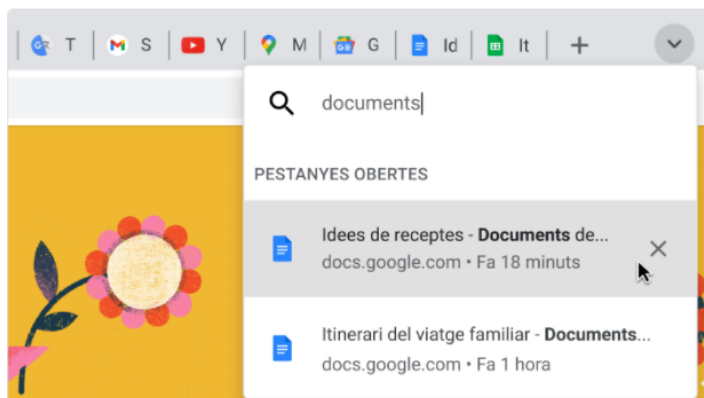
- Facilitar la seua comprensió i possibilitar el seu ús per part de l'usuari, independentment de quin siga el seu perfil.
- Facilitar el manteniment de l'aplicació.
- Facilitar el desenvolupament de millores per altres equips de treball.
- Facilitar que el sistema pugui ser comprés per altres programadors.

PRODUCTIVITAT

Utilitza la cerca de pestanyes per trobar les pestanyes més ràpidament

Tens moltes pestanyes obertes? Si et costa trobar les pestanyes ràpidament, prova la funció de cerca de pestanyes de Chrome.

A la part superior de la finestra de Chrome, fes clic a la icona de cerca de pestanyes  o  per veure una llista de totes les pestanyes de Chrome que tens obertes.



Exemple d'ajuda a Google Chrome

Malgrat tractar-se d'un treball que pot resultar tediós, se li ha d'atorgar tota la importància que té. No és prou amb desenvolupar aplicacions funcionals i atractives, sinó que, a més, aquestes han de comptar amb una documentació adequada. Es tracta d'una tasca tan important com la de realitzar el propi codi de l'aplicació.

2. Tipus de documentació

A grans trets, podem dividir la documentació de programari en dos grups: documentació de proves i documentació tècnica.

2.1. Documentació de proves

Documentar les proves realitzades sobre una aplicació determinada és fonamental per a detectar i corregir possibles errors. Podem distingir-ne dos tipus:

- a) **Documentació d'entrada:** en la qual s'especifiquen els escenaris de prova i es detallen els procediments d'aquests.
- b) **Documentació d'eixida:** es tracta dels informes resultants d'aplicar les proves sobre l'aplicació definides al punt anterior. Aquests informes recolliran l'històric de proves realitzades, documentant-se així cada problema que haja pogut sorgir i la seua posterior correcció.

2.2. Documentació tècnica

Pertany a aquest grup la resta de documentació: guies, fulles de especificacions, manuals. Igual que en el cas anterior, pot dividir-se en dos grups:

- a) **Documentació interna:** es tracta dels comentaris inclosos pel desenvolupador en el codi, que han de descriure diferents aspectes sobre aquest.
S'ha d'incidir en la importància de realitzar un programa ordenat i clar, en el qual els comentaris ajuden a entendre el codi, però aquest ja de per si mateix ha de ser clar. En aquest àmbit, s'ha de tindre en compte el següent:
 - Incloure comentaris al començament de mòduls.
 - Comentar variables, constants, procediments i funcions.
 - Incloure comentaris introductoris sobre blocs de codi, funcions o mòduls.
 - No comentar l'obvi, ja que l'excés de comentaris pot ser contraproduent.

```
1 referencia
public DataSet categories()
{
    DataSet ds = new DataSet();
    try
    {
        //Obrim la connexió utilitzant el mètode de la classe
        this.ObrirConnexio();
        //Llancem la consulta amb un DataAdapter
        SqlDataAdapter dataAdapter = new SqlDataAdapter("SELECT * FROM CATEGORIA", this.connexio);
        //Oplim el DataSet amb els registres rebuts al DataAdapter
        dataAdapter.Fill(ds, "categorias");
        //Tanquem la connexió
        this.TancarConnexio();
        //Tornem el DataSet
        return ds;
    }
    catch (Exception ex)
    {
        //Capturem l'excepció i la mostrem amb un MessageBox
        MessageBox.Show(ex.Message);
        return ds;
    }
}
```

- b) **Documentació externa:** en aquest cas, sol tractar-se d'un manual tècnic (orientat a programadors per a facilitar el manteniment i desenvolupament de l'aplicació a futur) i d'un manual i/o guia d'usuari (per a facilitar el seu ús).

3. Manuals i guies

Existeixen diferents tipus de manuals i guies, diferents entre si en funció del contingut de cadascun i de la seua manera d'utilització. Aquests documents poden crear-se en diferents formats i/o suports: paper, format electrònic, web d'ajuda, presentació, vídeo, combinació de alguns d'ells, segons quina siga la complexitat de l'aplicació resultant i del perfil d'usuari a qui vaja dirigit.

3.1. Manual i guia d'usuari

El manual d'usuari conté tota la informació necessària amb la finalitat de fer més fàcil l'ús i la comprensió del programa. Els àmbits d'aplicació són varis i diversos: formació de l'usuari, guia de consulta abans dubtes, ajuda per a detectar i corregir errors, etc.

És cert que no existeix cap norma sobre com s'ha d'elaborar, encara que hi ha uns certs patrons i usos que s'han de seguir. En primer lloc, el document en si mateix ha de ser clar i atractiu des del punt de vista de l'usuari, ja que al final és qui l'utilitzarà. Per a això, pot resultar aconsellable utilitzar imatges i gràfics per a ajudar a entendre el text, utilitzar un vocabulari clar i concís, exemples pràctics, etc.

Una possible estructura del manual d'usuari pot ser la següent:

Portada	
Títol i copyright	
Pròleg	
Índex i continguts	
Guia bàsica de l'aplicació	
Secció FAQ	
Glossari	

3.2. Manual i guia d'explotació

El manual i guia d'explotació conté la informació necessària per a utilitzar i explotar l'aplicació. És a dir, va molt orientat a la instal·lació, configuració i posada en marxa, per la qual cosa, evidentment, anirà molt lligat al context en el qual es vaja a realitzar la instal·lació (tipus d'organització, requeriments, nombre d'usuaris).

Encara que en aquest cas tampoc hi ha una norma per a la seua elaboració, ha de tindre almenys els següents punts:

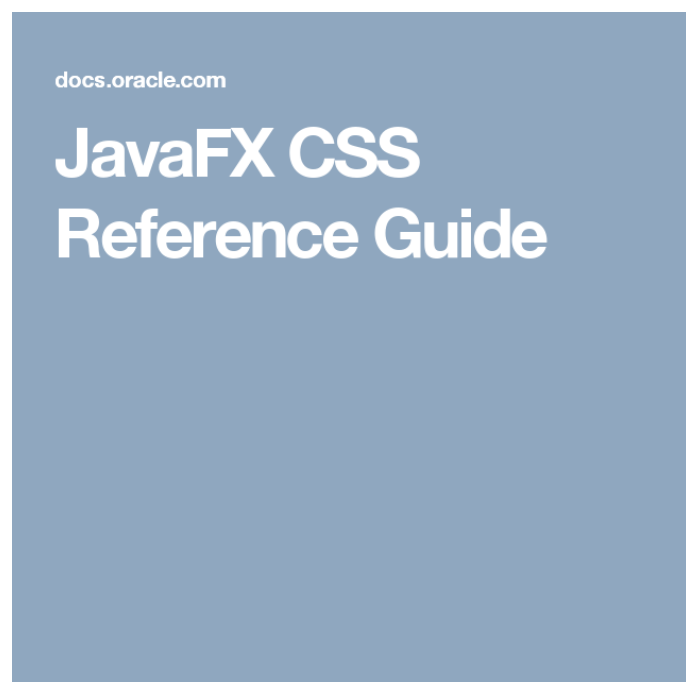
- Requeriments mínims del sistema quant a processador, memòria, espai lliure.
- Procés d'instal·lació:
 - Necessitat o no de preparació prèvia, si és necessària alguna tasca per a això (per exemple , habilitar la connexió a Internet, activar o desactivar permisos).
 - Opcions d'instal·lació: a través d'USB, xarxa, etc.
 - Procediment d'instal·lació pas a pas, detallant i explicant les seues diferents opcions.
- Actualitzacions del sistema i còpies de seguretat.
- Glossari.
- Índex.

En funció de la complexitat de l'aplicació, pot dividir-se en manual d'instal·lació i manual de configuració.

3.3. Guia ràpida i guia de referència

D'una forma addicional als manuals estudiats anteriorment, la documentació d'una aplicació pot completar-se amb aquesta mena de guies, en funció del cas en concret. Les seues característiques són les següents:

1. **Guia ràpida:** s'orienta a usuaris del sistema i/o encarregats de manteniment. En funció de la complexitat de l'aplicació pot ser necessari realitzar diverses guies, cadascuna amb diferent temàtica. Les guies ràpides aporten informació molt concreta i molt diferent, relacionada amb diversos procediments o camps d'una aplicació.
2. **Guia de referència:** al contrari que en el cas de les guies ràpides, aquestes solen estar pensades per a usuaris que tenen un major nivell de coneixement sobre l'ús de l'aplicació, així com una major experiència. Per això, és habitual que continguin informació relacionada amb aspectes més tècnics: tipus de missatges d'error i la seua causa, tipus de dades d'entrada que li són permesos a l'aplicació, tipus de comandaments o instruccions, ús de les funcions, ús de la API, etc.



4. Fitxers d'ajuda. Formats. Eines per a generar-los

Un fitxer és un element que pot contindre diversos tipus d'informació en diferents formats, ja siga en suport físic o en suport digital, que sol ser el procediment habitual en el camp de les aplicacions informàtiques.

D'aquesta manera, un fitxer d'ajuda és un document que conté informació d'ajuda sobre un determinat camp. Un exemple de fitxer d'ajuda podria ser un manual d'ús d'una eina concreta, dirigit als usuaris d'aquesta. Els fitxers d'ajuda estan formats per dues parts diferenciades:

- **Mapa del fitxer:** es tracta d'un apartat que facilita la navegació per el fitxer, que identifica clarament el contingut d'aquest a través d'identificadors.
- **Vista d'informació:** mostra a l'usuari continguts en forma de glossari o taula de continguts.

4.1. Formats

En el següent quadre es mostren alguns dels principals formats de fitxers d'ajuda, juntament amb les principals característiques d'aquests:

Format	Característiques	Plataforma
CHM	Generat a partir de HLP. Utilitza HTML per a generar l'ajuda. Enllaços mitjançant enllaços a la taula de contingut. Permet fusionar diversos fitxers d'ajuda. Pot ser creat a partir d'HTML Help Workshop.	Windows
HLP	Pot incloure taula de contingut en fitxers .cnt. Inclou informació extra en fitxer .gid. Utilitza fitxers RTF per a generar l'ajuda. Necessita compilació (per exemple mitjançant HTML Help Workshop)	Windows
HPJ	Conté taula de contingut (fitxer .cnt). Es crea mitjançant eina tipus Help Workshop.	Windows
IPF	Similar a HTML. S'utilitza per a ajuda en línia o web. És necessari compilar per a generar-ho.	Web (ajuda en línia)

JavaHelp	Implementat a Java, per la qual cosa és utilitzat habitualment per a implementar l'ajuda d'aplicacions desenvolupades en aquest llenguatge	Diverses
-----------------	--	----------

4.2. Eines per a generar fitxers d'ajuda

A continuació, es detallen les principals característiques d'algunes eines que es poden utilitzar per a generar fitxers d'ajuda:

- **Help Work Shop:** possibilita crear fitxers d'ajuda en entorns Windows. És una aplicació que consta d'un editor d'imatges, un administrador de projectes i un compilador.
- **HELPMAKER:** és una eina de caràcter gratuït que compta amb múltiples opcions per a personalitzar els fitxers. Utilitza un editor de text per a facilitar la generació dels fitxers d'ajuda.
- **JavaHelp:** JavaHelp per si mateix és un sistema de creació de fitxers d'ajuda i, a més, un format de fitxers d'ajuda pròpiament dit. Es caracteritza per:
 - Utilitzar Java per a la seua implementació.
 - Utilitzar-se en aplicacions Java.
 - Ser independent de la plataforma.
 - Facilitar la realització d'ajuda en línia.
 - Sistema antic i reemplaçat per altres tècniques en les versions modernes de Java i JavaFX.
- **SHALOM HELP MAKER:** es tracta d'una altra eina gratuïta, que utilitza un editor de text per a la creació de fitxers d'ajuda, de manera similar a HELPMAKER. Permet incloure enllaços externs, inserir o crear imatges, crear índexs, etc.

4.3. Ajuda en format HTML amb el control WebBrowser de WPF

Una forma actual d'afegir ajuda a una aplicació WPF és crear l'ajuda en format HTML i carregar-la en un component de WPF anomenat WebBrowser.

El component WebBrowser és capaç de mostrar pàgines web (HTML, CSS, SVG, JavaScript) dins d'una aplicació WPF. Com a tal, el WebBrowser és un mini navegador. El component WebBrowser és molt pràctic en aquest cas, quan és necessita mostrar textos d'ajuda o altres continguts que s'han de descarregar d'un servidor web en temps d'execució.

Donar forma a l'ajuda

El primer pas és construir l'ajuda de la nostra aplicació, creant tants fitxers HTML com necessitem. És important construir adequadament l'estructura de l'ajuda, ja que és fonamental que siga clara i concisa per a facilitar la localització de la informació per part de l'usuari.

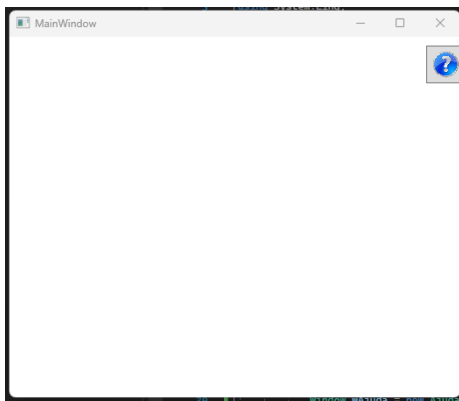
En el nostre exemple, en tractar-se d'un cas senzill, únicament hem creat dos temes d'ajuda, al costat d'una pàgina principal.

El codi HTML del fitxer principal és el següent:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8 />
  <title>Aquesta és la nostra ajuda de prova </title>
</head>
<body>
  <h1>Aquesta es una ajuda que carregarem en un WebBrowser</h1>
  <h2>Aquesta ajuda s'ha creat com a exemple</h2>
  <p>Per a crearla podem utilitzar qualsevol editor de codi HTML</p>
</h2>
  <p>Podem també afegir imatges o referenciar les diferents pàgines de la ajuda</p>
  
  <p>Aquest és l'enllaç a la <a href="unitat1.html">Unitat 1</a> de la ajuda </p>
  <p>Aquest és l'enllaç a la <a href="unitat2.html">Unitat 2</a> de la ajuda </p>
</body>
</html>
```

Exemple amb WebBrowser

A l'exemple plantejarem dues finestres, una finestra principal amb l'aplicació amb un botó d'ajuda i altra finestra en la que es carregarà el WebBrowser i l'ajuda corresponent. Observem primer el codi XAML de la finestra principal:



```
<Window x:Class="Ud7_AjudaWeb.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:Ud7_AjudaWeb"
  mc:Ignorable="d"
  Title="MainWindow" Height="450" Width="531"
  KeyDown="Window_KeyDown">
  <Grid>
    <Button x:Name="btnAjuda" Click="btnAjuda_Click" HorizontalAlignment="Left"
      Margin="476,10,0,0" VerticalAlignment="Top" Height="43" Width="45">
      <Image HorizontalAlignment="Left" Height="30" VerticalAlignment="Top" Width="30" Source="/75471.png"/>
    </Button>
  </Grid>
</Window>
```

Observa com llançarem la segona finestra. Ho farem de dues formes:

- Polsant al botó que porta dins una imatge.
- Polsat una tecla (és controlarà que siga la tecla F1).

El codi C# de la finestra principal seria aquest:

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

    1 referencia
    private void btnAjuda_Click(object sender, RoutedEventArgs e)
    {
        Window wAjuda = new Ajuda();
        wAjuda.Show();
    }

    1 referencia
    private void Window_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.Key == Key.F1)
        {
            Window wAjuda = new Ajuda();
            wAjuda.Show();
        }
    }
}
```

Carregarem la finestra d'ajuda tant si polsem el botó d'ajuda com si polsem la tecla F1.

Ara observem la aparença i el codi XAML de la finestra que carregarà el WebBrowser:



Conté dues botons amb imatges de fletxes per a navegar (darrere i avant), un TextBlock per informar la URL visitada (en aquest cas es local, però l'ajuda podria estar a internet) i el WebBrowser.

```

Window x:Class="Ud7_AjudaWeb.Ajuda"
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:Ud7_AjudaWeb"
mc:Ignorable="d"
Title="Ajuda" Height="450" Width="545">
<DockPanel>
<ToolBar DockPanel.Dock="Top">
<Button x:Name="btnBack" Click="btnBack_Click">
<Image Source="/left.png" Width="16" Height="16" />
</Button>
<Button x:Name="btnForward" Click="btnForward_Click">
<Image Source="/right.png" Width="16" Height="16" />
</Button>
<Separator />
<TextBlock Name="txtUrl" Width="300" />
</ToolBar>
<WebBrowser x:Name="wbAjuda" Navigating="wbAjuda_Navigating" />
</DockPanel>
</Window>

```

Com s'observa el codi XAML conté els elements descrits. També inclou la referència als mètodes que s'han d'executar quan es polsen els botons i quan canvia la ruta de navegació (esdeveniment Navigating).

El codi C# associat seria aquest:

```

public partial class Ajuda : Window
{
    2 referencias
    public Ajuda()
    {
        InitializeComponent();
        string curDir = Directory.GetCurrentDirectory();
        wbAjuda.Navigate(new Uri(Environment.CurrentDirectory + "/html/principal.html"));
    }

    1 referencia
    private void btnBack_Click(object sender, RoutedEventArgs e)
    {
        wbAjuda.GoBack();
    }

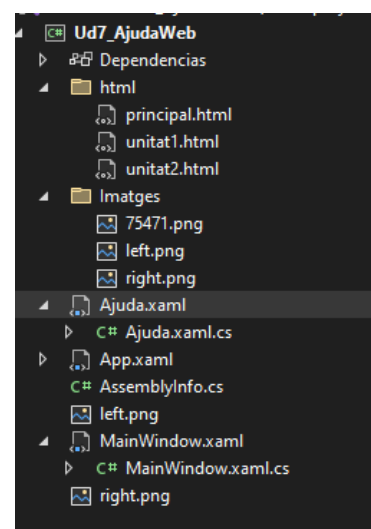
    1 referencia
    private void btnForward_Click(object sender, RoutedEventArgs e)
    {
        wbAjuda.GoForward();
    }

    1 referencia
    private void wbAjuda_Navigating(object sender, System.Windows.Navigation.NavigatingCancelEventArgs e)
    {
        txtUrl.Text = e.Uri.ToString();
    }
}

```

Observa com es carrega per primera vegada el WebBrowser amb la Url del fitxer html d'ajuda i com es navega avant i darrere amb es mètodes **GoBack** i **GoForward** del control WebBrowser.

Finalment, l'estructura del projecte complet seria aquesta:



4.4. Ajuda genèrica i sensible al context

Encara que a priori poguera semblar una distinció evident, hem de tindre en compte les diverses formes en les quals pot presentar-se i/o elaborar una documentació d'ajuda relacionada amb una aplicació:

Ajuda genèrica	Ajuda sensible al context
Conté tota la informació d'ajuda sobre una determinada aplicació, dividida per temes o camps. Generalment presenta un índex o menú. Permet navegar o buscar entre els elements que la formen	Es tracta d'ajuda particularitzada per a un tema o acció concreta. Pot presentar-se de diferents formes: ajuda en línia, breu detall de l'ús d'un menú en el programa.

4.5. Creació d'un tooltip en WPF

Tooltips, infotips o hints , diversos noms però el concepte és el mateix: la capacitat d'obtindre informació extra sobre un control específic, col·locant el ratolí sobre ell. WPF òbviament suporta aquest concepte molt bé, i poden fer-ho servir utilitzant la propietat `ToolTip` dins de quasi qualsevol control WPF.

Vegem un exemple complet de com crear un Tooltip senzill i altre més complex:

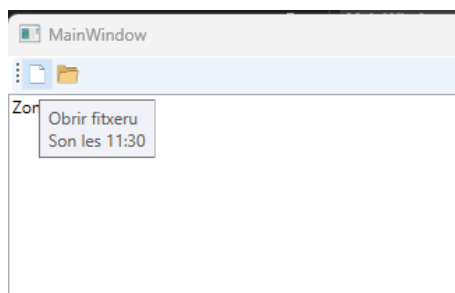
Creem un nou projecte i de forma similar al anterior afegirem una barra d'eines amb un parell de botons i un `TextBox`, simulant un editor de text:

```
Window x:Class="Ud7_ToolTip.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:Ud7_ToolTip"
    mc:Ignorable="d"
    Title="MainWindow" Height="450" Width="542">
    <DockPanel>
        <ToolBar DockPanel.Dock="Top">
            <Button ToolTip="Obrir fitxer" x:Name="toolTip1" ToolTipService.ShowDuration="5000" ToolTipOpening="toolTip1_ToolTipOpening">
                <Image Source="/nou.png" Width="16" Height="16" />
            </Button>
            <Button>
                <Image Source="/obrir.png" Width="16" Height="16" />
            </Button>
            <Button.ToolTip>
                <StackPanel>
                    <TextBlock FontWeight="Bold" FontSize="14" Margin="0,0,0,5">Obrir fitxer</TextBlock>
                    <TextBlock> Cerca al teu ordinador o a la xarxa</TextBlock>
                    <LineBreak />
                    <TextBlock> un fitxer per obrir i/o editar.</TextBlock>
                    <Border BorderBrush="Silver" BorderThickness="0,1,0,0" Margin="0,8" />
                    <WrapPanel>
                        <Image Source="/Ajuda.png" Width="32" Height="32" Margin="0,0,5,0" />
                        <TextBlock FontStyle="Italic">Polsa F1 per a més ajuda</TextBlock>
                    </WrapPanel>
                </StackPanel>
            </Button.ToolTip>
        </ToolBar>
        <TextBox>
            Zona d'edició
        </TextBox>
    </DockPanel>
</Window>
```

Observa com col·loquem el Tooltip al primer botó. A la mateixa línia també podem indicar el temps que estarà el Tooltip visible (**ToolTipService.ShowDuration**) i associar un mètode a l'esdeveniment **ToolTipOpening** en el qual modificarem de forma dinàmica el contingut del Tooltip.

Amb les propietats del element **ToolTipService** (present en quasi tots els controls WPF) podem configurar aspectes del Tooltip.

El funcionament d'aquest Tooltip és molt simple, al passar el ratolí sobre el botó **Nou Fitxer**:



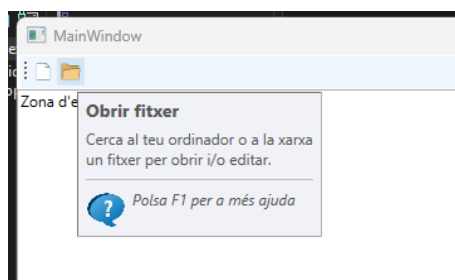
Si observeu al XAML únicament estava configurat que al Tooltip es mostrarà el text "Obrir fitxer", però com hem afegit la resta de text. Amb el mètode associat a l'esdeveniment **ToolTipOpening**. Així el codi C# de la finestra seria així:

```
1 referencia
private void tooltip1_ToolTipOpening(object sender, ToolTipEventArgs e)
{
    Button b = sender as Button;
    b.ToolTip = b.ToolTip + "\n" + "Son les " + DateTime.Now.ToShortTimeString();
}
```

Per acabar observem el Tooltip del segon botó, que és un poc més complex:

```
<Button>
  <Button.Content>
    <Image Source="/obrir.png" Width="16" Height="16" />
  </Button.Content>
  <Button.ToolTip>
    <StackPanel>
      <TextBlock FontWeight="Bold" FontSize="14" Margin="0,0,0,5">Obrir fitxer</TextBlock>
      <TextBlock> Cerca al teu ordinador o a la xarxa
      <LineBreak />
      un fitxer per obrir i/o editar.
    </TextBlock>
    <Border BorderBrush="Silver" BorderThickness="0,1,0,0" Margin="0,8" />
    <WrapPanel>
      <Image Source="/Ajuda.png" Width="32" Height="32" Margin="0,0,5,0" />
      <TextBlock FontStyle="Italic">Polsa F1 per a més ajuda</TextBlock>
    </WrapPanel>
  </StackPanel>
</Button.ToolTip>
</Button>
```

En aquest exemple col·loquem dins del Tooltip del botó un panell amb components dins d'ell. Des de uns TextBlock fins una imatge, per a aconseguir un Tooltip com aquest:


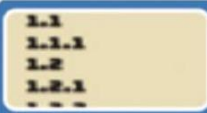




5. Taules de continguts

Les taules de continguts ens ajuden a modelar tant el contingut com l'estructura d'un document determinat. La informació estarà organitzada en forma d'esquema, amb diversos nivells entre els quals es cal destacar títols i subtítols. Lògicament, el disseny de les mateixes depèn de qui faça el desenvolupament, així com de molts altres factors (objectiu, usuari al qual es dirigeixen, etc.). No obstant això, existeix una sèrie de característiques comunes a totes elles:

- És possible o no que mostren el nombre de pàgines que contenen, depèn del disseny que es realitzi sobre les mateixes. Es tracta d'un element completament opcional.
- Les taules de continguts habitualment contenen enllaços directes a cada títol o subtítol, que apunta al contingut d'aquests. Així s'aconsegueix una navegació i utilització més fluida i senzilla per a l'usuari.
- Solen situar-se al començament de cada document, de manera similar a l'índex dels llibres.
- Encara que pot resultar senzill, el seu procés d'elaboració no és elemental, ja que és necessari realitzar una anàlisi prèvia i detallada de tota la documentació que contenen. A més, és necessari que els títols siguin el més clars i intuïtius possible. És també important que no hi haja informació duplicada en diverses parts del document.

Com a resum, en la següent figura es mostren alguns consells que s'han de tindre en compte per a elaborar una taula de continguts clara i de qualitat, que siga útil per a l'usuari:

	Selecció de tots els temes i subtemes que tindrà la taula
	Numerar temes i subtemes segons un patró de numeració
	Crear la taula de contingut, procurant afegir els enllaços necessaris en aquesta
	Actualitzar i modificar la taula cada vegada que es necessite

6. Validació

Una de les funcions més comunes que ha de realitzar una interfície gràfica per a una aplicació orientada a base de dades és la comprovació de les dades que l'usuari introdueix en els formularis.

Qualsevol dada errònia que s'intente guardar en la base de dades pot provocar excepcions o inconsistència en les dades emmagatzemades.

Resulta molt important que es vaig guiar a l'usuari perquè la informació introduïda siga correcta.

Tots els frameworks per al desenvolupament d'interfícies ofereixen funcionalitats per a la validació de les dades que introdueix l'usuari. WPF ens ofereix diverses opcions de com fer-ho.

A més en WPF el procés de validació està íntimament relacionat amb **Data Binding**.

A continuació veurem diferents mètodes de validació a WPF,

6.1. Validation Rules

En aquest mètode de validació generem una regla personalitzada per a un **binding** específic. Per exemple si tenim quatre camps de text, en els quals tenim dos camps que no poden ser buits, un altre ha de seguir la sintaxi d'una adreça de correu i un altre és un rang numèric.

Cada regla és una classe que hereta de la classe **ValidationRule** i en la qual s'ha d'implementar el mètode **Validate**.

Aquesta classe té també una propietat a destacar anomenada **ValidationStep** que determina quan s'executa el mètode de validació. Les opcions són les següents:

- *RawProposedValue*: s'executa abans de realitzar la conversió. **És el comportament per defecte.**
- *ConvertedProposedValue*: s'executa després de la conversió i abans que el *setter* de la propietat siga anomenat.
- *UpdatedValue*: s'executa després que la propietat origen siga actualitzada.
- *CommittedValue*: s'executa després que el valor de la propietat siga confirmada

Per exemple, si tenim dos `TextBox` diferents que hem de validar, però el tipus de validació és diferent (un dni i un correu) haurem d'utilitzar dues classes diferents, una per a cada `TextBox`.

Comencem amb un exemple. Imaginem que volem validar l'edat d'una persona. Hauriem de crear una classe que herete de **ValidationRule** similar a aquesta:


```

0 referencias
public class StringToIntValidationRule : ValidationRule
{
    0 referencias
    public override ValidationResult Validate(object value, System.Globalization.CultureInfo cultureInfo)
    {
        int i;
        if (int.TryParse(value.ToString(), out i))
            return new ValidationResult(true, null);

        return new ValidationResult(false, "La edat ha de ser un nombre enter");
    }
}

```

Aquest mètode comprova que l'objecte que es passa es tracta d'un nombre enter i a través de l'objecte **ValidationResult** informem del resultat. En cas que no siga un número l'indiquem passant-li false el primer paràmetre i en el segon indiquem el missatge que mostrarem més tard.

Observa que s'ha implementat el mètode **Validate** que torna un **ValidationResult**. Aquest **ValidationResult** pot tornar simplement un valor **boolean** o un valor boolean acompanyat del text amb la descripció de l'error.

Per a simplificar l'exemple creem una propietat dins de classe MainWindow anomenada edat:

```

2 referencias
public partial class MainWindow : Window
{
    0 referencias
    public MainWindow()
    {
        DataContext = this;
        InitializeComponent();
    }

    0 referencias
    public int edat { get; set; }
}

```

I a continuació l'associem a la finestra del projecte dins del codi XAML:

```

<Window x:Class="Ud7_ValidationRules.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:Ud7_ValidationRules"
        mc:Ignorable="d"
        Title="MainWindow" Height="234" Width="452">

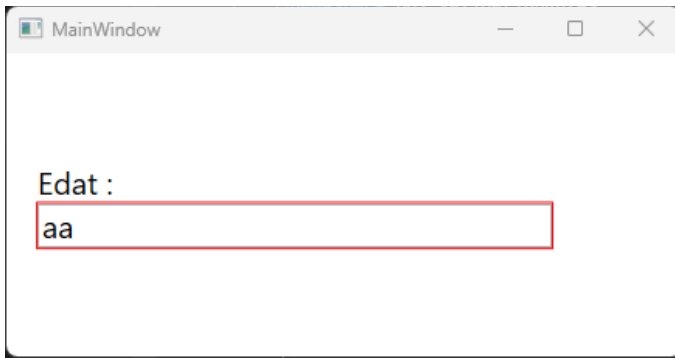
    <StackPanel VerticalAlignment="Center" Margin="20">
        <TextBlock Text="Edat :" FontSize="20"/>
        <TextBox x:Name="txtEdat" FontSize="20" Width="332" HorizontalAlignment="Left" >
            <TextBox.Text>
                <Binding Path="edat" UpdateSourceTrigger="PropertyChanged">
                    <Binding.ValidationRules>
                        <local:StringToIntValidationRule/>
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>
    </StackPanel>
</Window>

```

Observa com es fa el **binding** amb la propietat edat i la definició de la propietat **UpdateSourceTrigger** per a notificar dels canvis al TextBox.

A continuació s'especifica el mètode a executar dins de la propietat **ValidationRules** del **Binding**.

Si executem i escrivim lletres al Textbox observarem aquest canvi a la finestra, o més concretament al control:



El control colorea el seu bord de color roig, però no vegem per cap costat el missatge de error definit.

Per poder visualitzar el missatge de error hem de crear un control (label, textblock o similar) i associar-lo amb el missatges d'error de la validació i el control que pot generar el missatge de error. Podria ser així:

```
<TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtEdat}"
Foreground="Red"/>
```

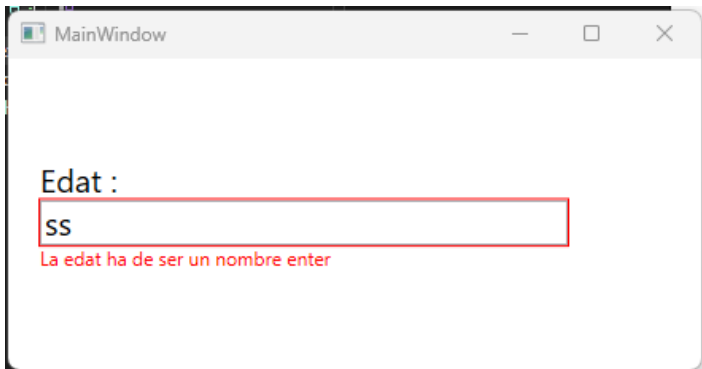
El que estem fent és vincular o enllaçar el missatge que tornen els mètodes de validació a la propietat visible del control (en aquest cas text), quin és el control a “observar” (amb la propietat **ElementName** i els canvis que ocorraran al control (posa el color del text en roig).

Per tant si afegim aquest control per a visualitzar els missatges de error, el codi XAML quedaria així:

```
<Window x:Class="Ud7_ValidationRules.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:Ud7_ValidationRules"
mc:Ignorable="d"
Title="MainWindow" Height="234" Width="452">

    <StackPanel VerticalAlignment="Center" Margin="20">
        <TextBlock Text="Edat :" FontSize="20"/>
        <TextBox x:Name="txtEdat" FontSize="20" Width="332" HorizontalAlignment="Left" >
            <TextBox.Text>
                <Binding Path="edat" UpdateSourceTrigger="PropertyChanged">
                    <Binding.ValidationRules>
                        <local:StringToIntValidationRule/>
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>
        <TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtEdat}" Foreground="Red"/>
    </StackPanel>
</Window>
```

I si executem ara l'exemple, el resultat al introduir una edat no vàlida seria aquest:



També es possible crear mètodes de validació amb expressions regulars. A internet pots trobar moltes, per exemple:

```
public class RegexValidation : ValidationRule
{
    public override ValidationResult Validate(object value, CultureInfo cultureInfo)
    {
        Regex regex = new Regex(@"^[^\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");
        Match match = regex.Match(value.ToString());
        if (match == null || match == Match.Empty)
        {
            return new ValidationResult(false, "El correu no és vàlid");
        }
        else
        {
            return ValidationResult.ValidResult;
        }
    }
}
```

6.2. Exception Validation Rules

Amb aquesta tècnica la validació es realitza en el mètode setter de la propietat de l'objecte que està enllaçada amb el component de la vista (finestra).

Si tornem a l'exemple anterior la validació s'hauria de fer al mètode setter de la propietat **edat**:

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

    private int _edat;
    0 referencias
    public int edat
    {
        get { return _edat; }
        set
        {
            if (value < 10 || value > 100)
                throw new ArgumentException("L'edat ha de estar entre 10 i 100");
            _edat = value;
        }
    }
}
```

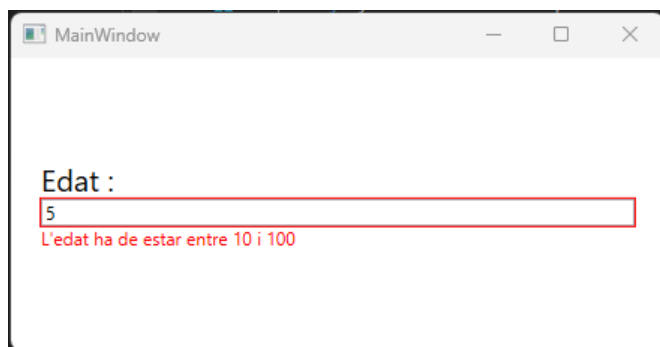
Amb aquesta tècnica el que fem es llançar una excepció amb d'instrucció **throw new ArgumentException**

El codi XAML associat variarà un poc. Ara el **binding** no comprova una validació, el que comprova es si ha ocorregut una **excepció**:

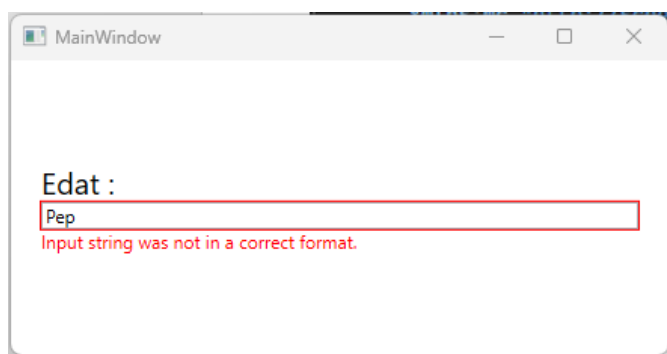
```
Window x:Class="Ud7_ExceptionValidationRule.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:Ud7_ExceptionValidationRule"
mc:Ignorable="d"
Title="MainWindow" Height="234" Width="452">

<StackPanel VerticalAlignment="Center" Margin="20">
    <TextBlock Text="Edat :" FontSize="20"/>
    <TextBox x:Name="txtEdat" Text="{Binding Path=edat, UpdateSourceTrigger=PropertyChanged, ValidatesOnExceptions=True}"/>
    <TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtEdat}" Foreground="Red"/>
</StackPanel>
</Window>
```

I com ja hem afegit el control per a visualitzar l'error el resultat seria aquest:



De fet si escrivim un nom en lloc d'un nombre també apareix un missatge:



Aquest comportament és degut a que el TextBox enllaçat amb la propietat edat observa les excepcions que provoca aquest control amb **ValidatesOnExceptions=True**.

I al estar enllaçat o vinculat amb una propietat de tipus int, escriure un text llança una excepció.

Aquesta manera de funcionar ens ve bé si la regla de validació la utilitzem **una sola vegada en el nostre codi**, no obstant això, si s'utilitza en diverses parts del nostre codi estarem escrivint més que si utilitzàrem **la forma anterior en la qual podem reutilitzar les regles**.

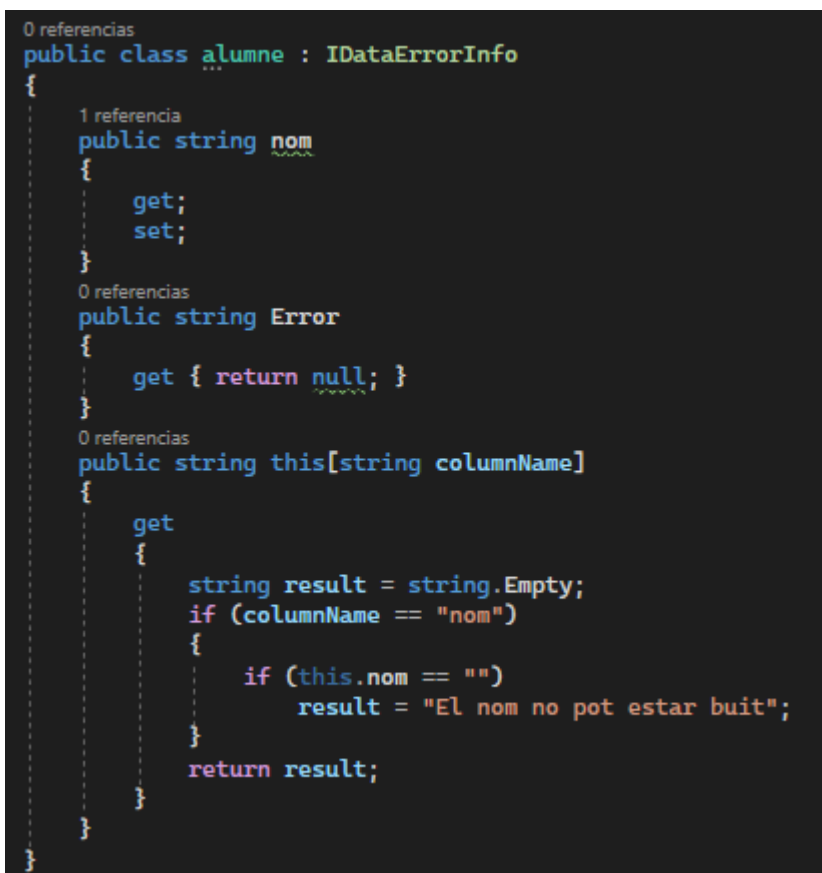
6.3. IDataErrorInfo

Igual que passava amb el mètode anterior, aquesta forma **desplaça la validació a la capa del model de l'aplicació**. És a dir, en les classes de la nostra aplicació que utilitzem per a representar les entitats.

La diferència amb el mètode anterior és que no utilitzem les excepcions per a informar de l'error, sinó que heretem d'una interfície que ha d'implementar dues propietats de tipus **string**:

- **Error**: s'utilitza per a comprovar si hi ha errors
- **Indexer**: accepta el nom d'una propietat i retorna **null** en cas que el valor que s'ha introduït siga correcte o un missatge amb l'error en cas de no passar la validació.

Per a fer un exemple hem de fer que la nostra classe herete de la classe `IDataErrorInfo` i implemente les dues propietats esmentades:



```
0 referencias
public class alumne : IDataErrorInfo
{
    1 referencia
    public string nom
    {
        get;
        set;
    }
    0 referencias
    public string Error
    {
        get { return null; }
    }
    0 referencias
    public string this[string columnName]
    {
        get
        {
            string result = string.Empty;
            if (columnName == "nom")
            {
                if (this.nom == "")
                    result = "El nom no pot estar buit";
            }
            return result;
        }
    }
}
```

En aquest cas la validació a realitzar és que la propietat `nom` no pot estar buida.

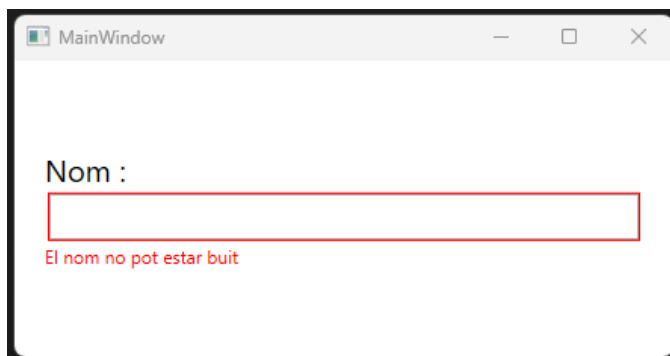
La propietat **Error** no s'utilitza en WPF, encara que s'ha de definir. L'altra propietat, **indexer** (es refereix a si mateixa), té com a paràmetre el nom de la propietat. Després en el *getter* comprovem si la propietat compleix amb la regla de validació. En cas de complir-la retornarem el valor de `null`, mentre que en cas de no complir-la retornarà un **string** informant de l'error.

En el fitxer XAML de la nostra aplicació activarem la propietat **ValidatesOnDataError="True"** en el control associat en la nostra interfície amb aquesta propietat.

Per tant el codi XAML de la finestra quedaria així:

```
Window x:Class="Ud7_IDataErrorInfo.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"
mc:Ignorable="d"
Title="MainWindow" Height="234" Width="452"
xmlns:local="clr-namespace:Ud7_IDataErrorInfo">
<Window.Resources>
<local:alumne x:Key="alumne1" nom="Pere"/>
</Window.Resources>
<StackPanel VerticalAlignment="Center" Margin="20">
<TextBlock Text="Nom :" FontSize="20"/>
<TextBox x:Name="txtNom" Height="30" Margin="3">
<TextBox.Text>
<Binding Source="{StaticResource alumne1}" Path="nom" ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged"/>
</TextBox.Text>
</TextBox>
<TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtNom}" Foreground="Red"/>
</StackPanel>
</Window>
```

I el resultat de l'execució seria aquest:



També cal destacar que si volem que es llancen **esdeveniments de validació** hem d'afegir la propietat **NotifyOnValidationError** i posar-la amb el valor **true**. Per exemple, si volem que el nostre botó d'Acceptar estiga desactivat mentre hi haja errors, llavors hem d'activar aquesta propietat perquè es llancen els esdeveniments corresponents de validació, i d'aquesta manera manejar-ho i poder desactivar el botó.

Si volem aplicar aquesta funcionalitat, el codi podria quedar així:

```
<StackPanel VerticalAlignment="Center" Margin="20">
<TextBlock Text="Nom :" FontSize="20"/>
<TextBox x:Name="txtNom" Height="30" Margin="3">
<TextBox.Text>
<Binding Source="{StaticResource alumne1}" NotifyOnValidationError="True" Path="nom"
ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged"/>
</TextBox.Text>
</TextBox>
<TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtNom}" Foreground="Red"/>
<Button Name="btnAcceptar" Content="Acceptar">
<Button.Style>
<Style TargetType="Button">
<Setter Property="IsEnabled" Value="True" />
<Style.Triggers>
<DataTrigger Binding="{Binding ElementName=txtNom, Path=(Validation.HasError)}" Value="True">
<Setter Property="IsEnabled" Value="False" />
</DataTrigger>
</Style.Triggers>
</Style>
</Button.Style>
</Button>
</StackPanel>
```

6.4. DataAnnotations

Aquest tipus de validacions trasllada la responsabilitat de la mateixa a la capa *Model de dades*, és a dir, a les classes.

Es tracta d'unes anotacions que es realitzen en cadascuna de les propietats d'una entitat. Vegem un exemple de com definiríem una classe amb aquestes anotacions de errades:

```
U referencias
public class usuari : IDataErrorInfo
{
    [Required(ErrorMessage = "El nickName és necessari.")]
    [StringLength(10, MinimumLength = 4, ErrorMessage = "El nickName ha de tindre entre 4 i 10 caracters")]
    [RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "El nickName únicament pot contindre aquestes lletres (a - z, A - Z).")]
    public string nickName { get; set; }

    [Required(ErrorMessage = "S'ha d'introduir un nom")]
    public string name { get; set; }
}
```

La primera propietat, **nickName**, és un camp requerit i que ha de tindre una longitud entre 4 i 10 caràcters. A més, mitjançant una expressió regular, comprova que el nom solament pugui tindre lletres. La segona propietat solament és obligatòria.

El missatge que es mostrarà per pantalla és l'especificat en la propietat **ErrorMessage**.

Les anotacions que podem utilitzar són les següents:

- **Required**: marca que la propietat és un camp obligatori. Aquesta anotació pot ser utilitzada al costat de **ErrorMessage** per a indicar un missatge personalitzat d'error en el cas que no es complisca aquesta validació.
- **Range**: marca un rang entre el qual ha d'estar comprés el valor passat.
- **StringLength**: indica una grandària del camp string. Aquesta anotació pot anar en conjunció amb **MinimumLength** per a indicar fins i tot una grandària mínima del camp string.
- **MaxLength**: estableix la longitud màxima d'una cadena de caràcters.
- **MinLength**: estableix la longitud mínima d'una cadena de caràcters.
- **RegularExpression**: indica una expressió regular que ha de ser utilitzada per a validar el membre.
- **CreditCard**: valida el número d'una targeta de crèdit.
- **EmailAddress**: valida una adreça de correu
- **Url**: valida una adreça web.
- **FileExtensions**: valida si l'extensió d'un fitxer és correcta.
- **Phone**: estableix que es tracta d'un número de telèfon vàlid.
- **DataType**: indica un nom d'una mena de dades.
- **CustomValidation**: ens permet realitzar validacions personalitzades.

Però únicament amb aquesta definició no podem validar el error. Hem de completar-la amb una tècnica que informe de l'errada i pugam visualitzar el problema.

La millor solució és combinar-la amb la tècnica el sistema *IDataErrorInfo* de forma que haurem de afegir les propietats necessàries a la classe:

```

public class usuari : IDataErrorInfo
{
    [Required(ErrorMessage = "El nickName és necessari.")]
    [StringLength(10, MinimumLength = 4, ErrorMessage = "El nickName ha de tindre entre 4 i 10 caracters")]
    [RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "El nickName únicament pot contindre aquestes lletres (a - z, A - Z).")]
    0 referencias
    public string nickName { get; set; }

    [Required(ErrorMessage = "S'ha d'introduir un nom")]
    0 referencias
    public string name { get; set; }

    0 referencias
    public string Error
    {
        get { return null; }
    }

    0 referencias
    public string this[string columnName]
    {
        get
        {
            var validationResults = new List<ValidationResult>();

            if (Validator.TryValidateProperty(
                GetType().GetProperty(columnName).GetValue(this),
                new ValidationContext(this)
                {
                    MemberName = columnName
                },
                validationResults))
            {
                return null;
            }

            return validationResults.First().ErrorMessage;
        }
    }
}

```

Observa que ara la propietat indexar (`public string this[string columnName]`) es genèrica per poder utilitzar qualsevol propietat.

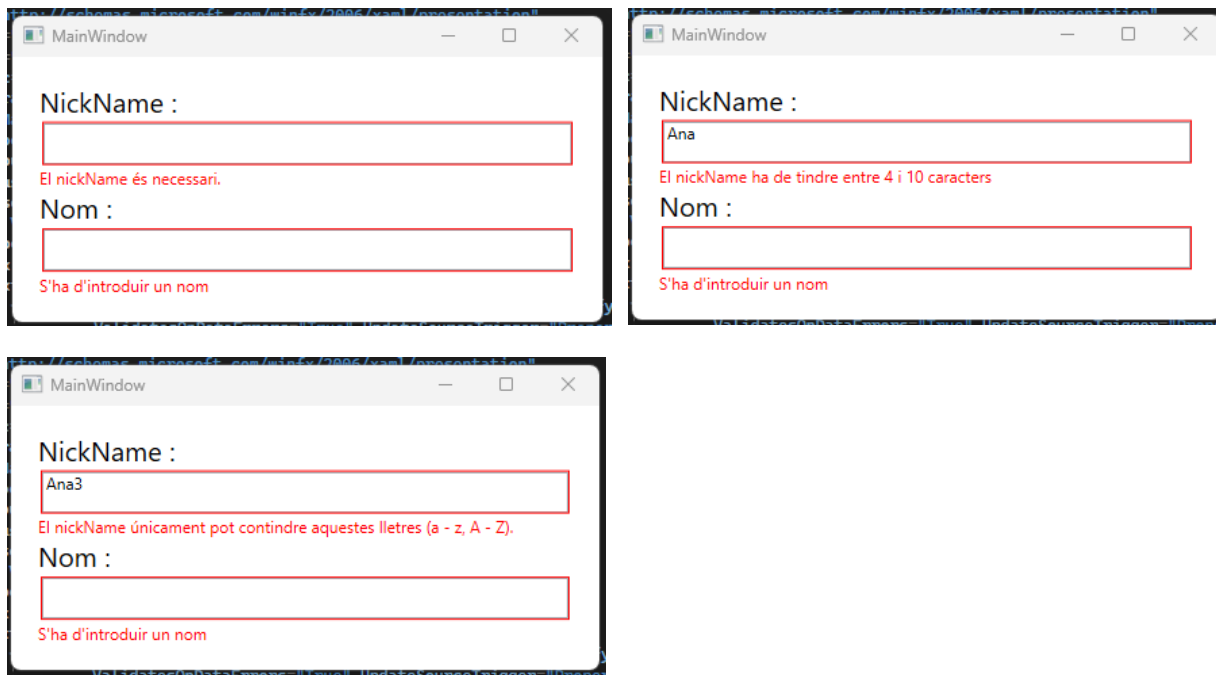
Per últim, el codi XAML de la finestra seria aquest:

```

Window x:Class="Ud7_DataAnnotations.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"
    mc:Ignorable="d"
    Title="MainWindow" Height="234" Width="452"
    xmlns:local="clr-namespace:Ud7_DataAnnotations">
<Window.Resources>
    <local:usuari x:Key="usuari1" nickName="hook" name="Pere"/>
</Window.Resources>
<StackPanel VerticalAlignment="Center" Margin="20">
    <TextBlock Text="NickName :" FontSize="20"/>
    <TextBox x:Name="txtNick" Height="30" Margin="3">
        <TextBox.Text>
            <Binding Source="{StaticResource usuari1}" Path="nickName" NotifyOnValidationError="True"
                ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged"/>
        </TextBox.Text>
    </TextBox>
    <TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtNick}" Foreground="Red"/>
    <TextBlock Text="Nom :" FontSize="20"/>
    <TextBox x:Name="txtNom" Height="30" Margin="3">
        <TextBox.Text>
            <Binding Source="{StaticResource usuari1}" Path="name" NotifyOnValidationError="True"
                ValidatesOnDataErrors="True" UpdateSourceTrigger="PropertyChanged"/>
        </TextBox.Text>
    </TextBox>
    <TextBlock Text="{Binding (Validation.Errors)[0].ErrorContent, ElementName=txtNom}" Foreground="Red"/>
</StackPanel>
</Window>

```

Algunes captures de mostra d'execució podrien ser aquestes:



Aquesta tècnica es una de les més utilitzades per l'estalvi de missatges alhora de validar diferents aspectes d'un mateix control.

7. Creació de Logs o Logging

En totes les aplicacions es produeixen errors o advertiments que hem d'atendre per a millorar la nostra aplicació. Quan l'aplicació està en producció i es produeix un incident hem de tindre eines que ens permeten saber què és el que ha succeït.

Tots els entorns de programació disposen de frameworks , amb llibreries pròpies o bé amb llibreries externes, que ens permeten registrar aquestes incidències. A aquest registre se'n diu en programació **Logging**.

Podriem crear una classe específica per aquesta tasca i cridar al seus mètodes en cada lloc de l'aplicació que ens interesse, però al mercat hi ha molts Frameworks que ens faciliten aquesta tasca.

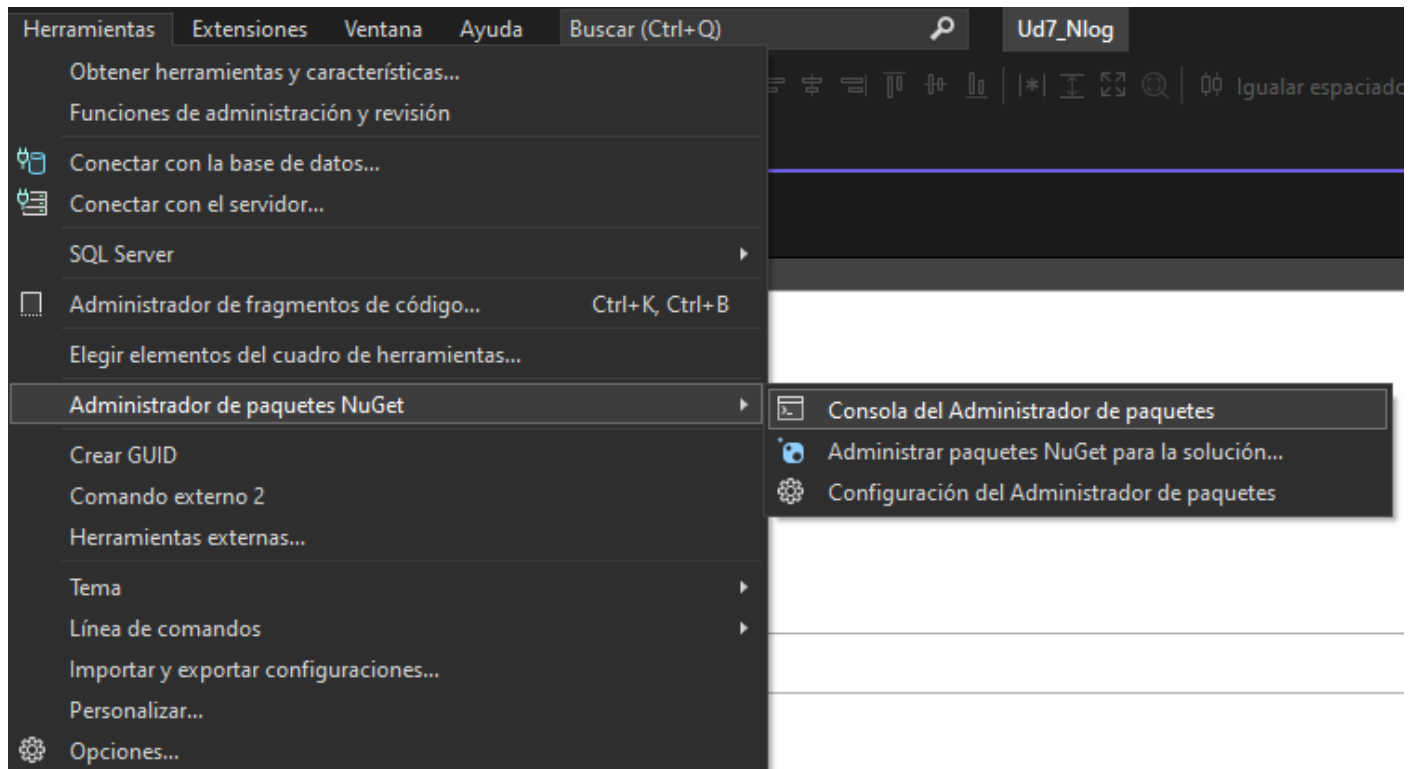
Frameworks per a logs hi ha molts, no obstant això cal decantar-se per un i en aquest curs utilitzarem **NLog**.

També és molt conegut **Log4net**, no obstant això, per rendiment, senzillesa en la configuració inicial i facilitat d'ús treballarem amb **NLog**.

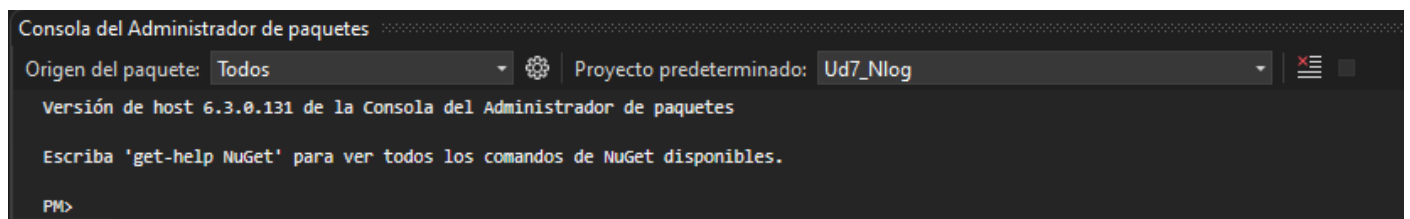
Instal·lació al projecte

Utilitzarem una còpia del projecte anterior per posar en pràctica el Nlog.

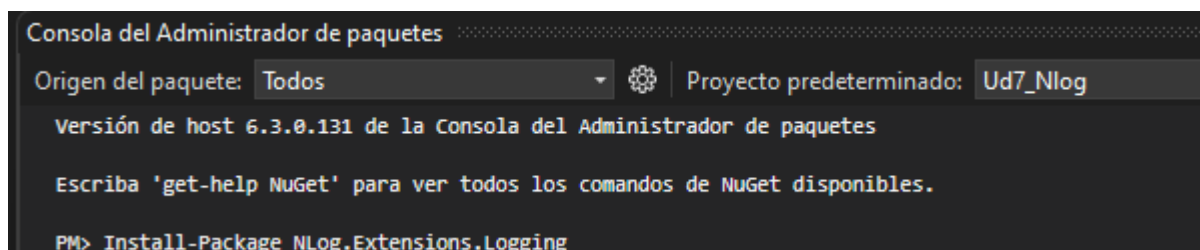
El primer pas per a utilitzar-lo és instal·lar-lo al projecte i per a això anirem al menú principal del VSC, a l'opció **Eines** → **Administrador de Paquets NuGet** → **Consola del Administrador de Paquets**.



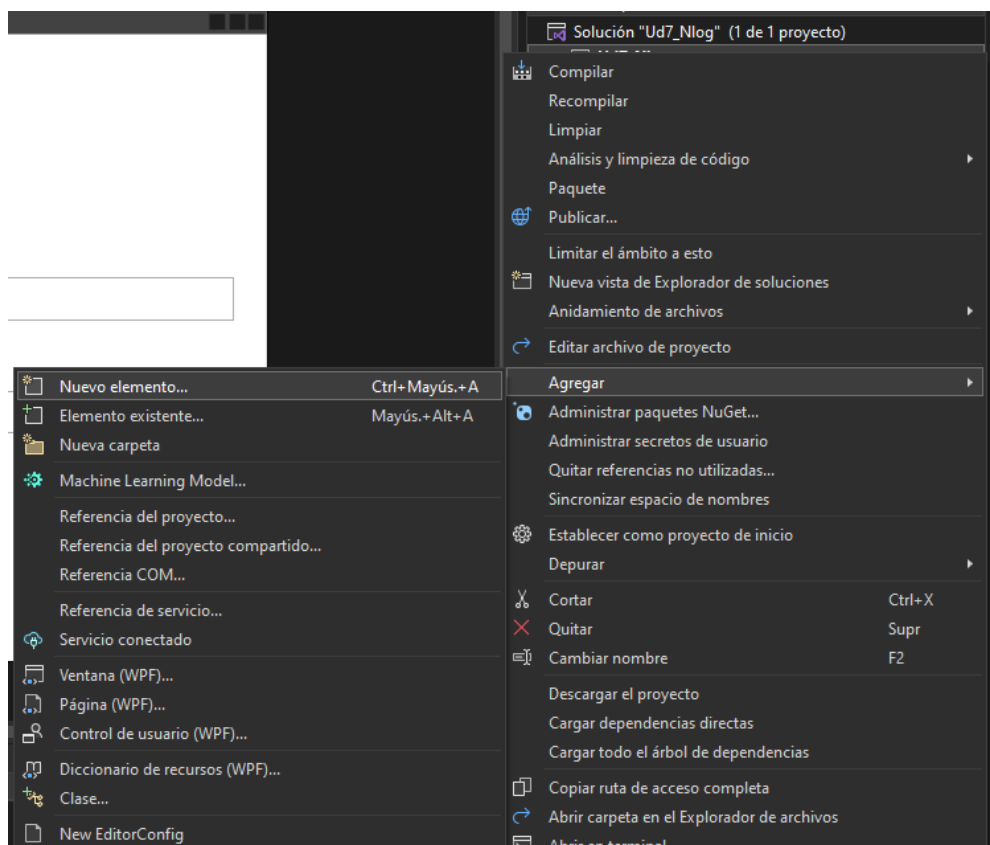
Això ens portarà a la següent finestra:



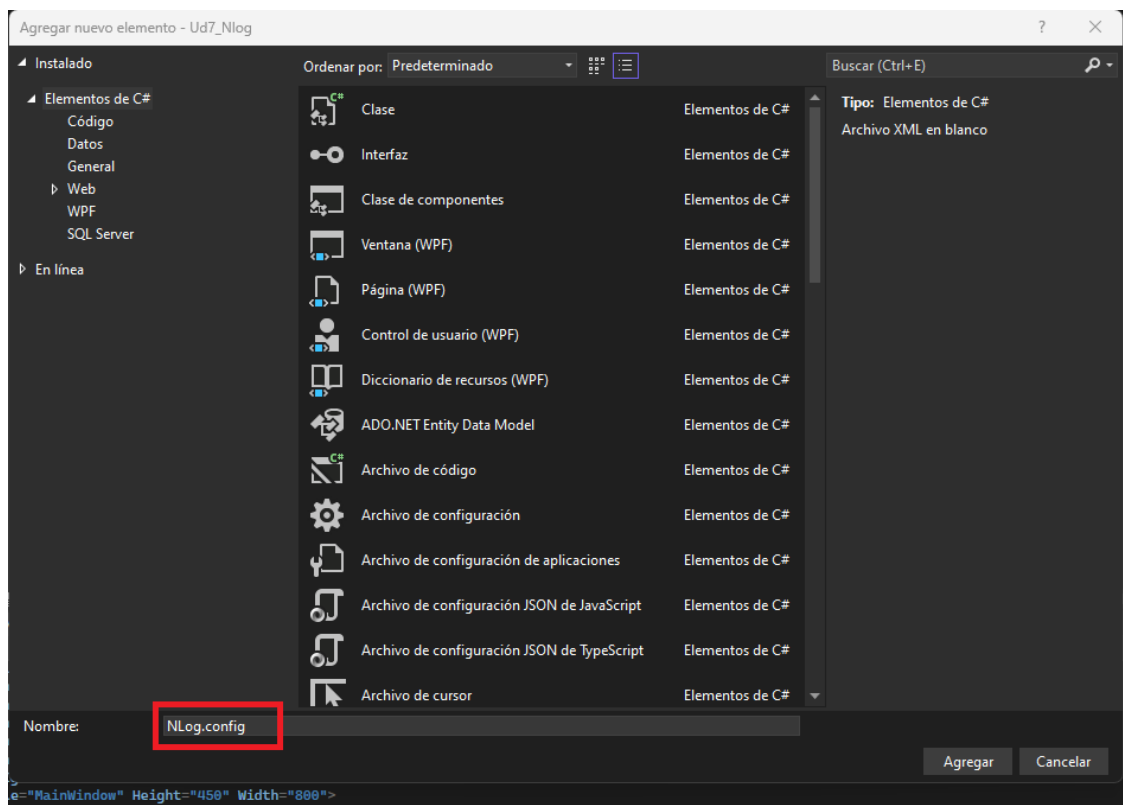
I en ella escrivim i executem la següent instrucció:



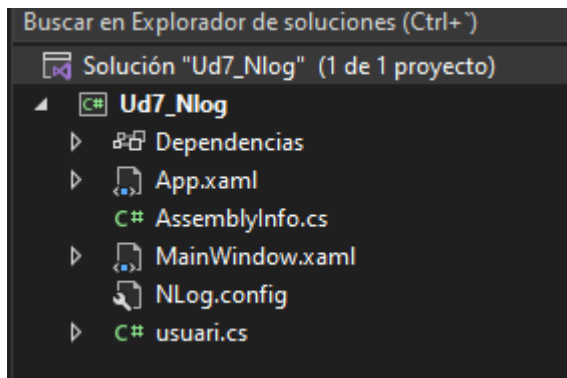
Els següent pas es afegir el fitxer de configuració de NLog al projecte. Per a fer això polsem amb el botó dret del ratolí sobre el projecte i triem l'opció **"Afegir Nou Element"**:



Hem d'anomenar-lo NLog.config



Finalment el projecte quedarà així:



A continuació hem d'editar i configurar el fitxer NLog.config. En ell decidirem quins logs crearem. Un exemple senzill de configuració és el següent:

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.nlog-project.org/schemas/NLog.xsd NLog.xsd"
      autoReload="true"
      throwExceptions="false"
      internalLogLevel="Off" internalLogFile="c:\temp\nlog-internal.log">

  <!-- optional, add some variables
  https://github.com/nlog/NLog/wiki/Configuration-file#variables
  -->

  <!--
  See https://github.com/nlog/nlog/wiki/Configuration-file
  for information on customizing logging rules and outputs.
  -->

  <targets>
    <!-- write logs to the files -->
    <target xsi:type="File" name="all_logs_file" fileName="${basedir}\Logs\all.log"/>
    <target xsi:type="File" name="important_logs_file" fileName="${basedir}\Logs\important.log"/>
    <!-- write logs to the console-->
    <target xsi:type="Console" name="logconsole" />
  </targets>

  <!-- rules to map from logger name to target -->
  <rules>
    <logger name="*" minlevel="Trace" writeTo="logconsole" />
    <logger name="*" minlevel="Debug" writeTo="all_logs_file" />
    <logger name="*" minlevel="Warn" writeTo="important_logs_file" />
  </rules>
</nlog>
```

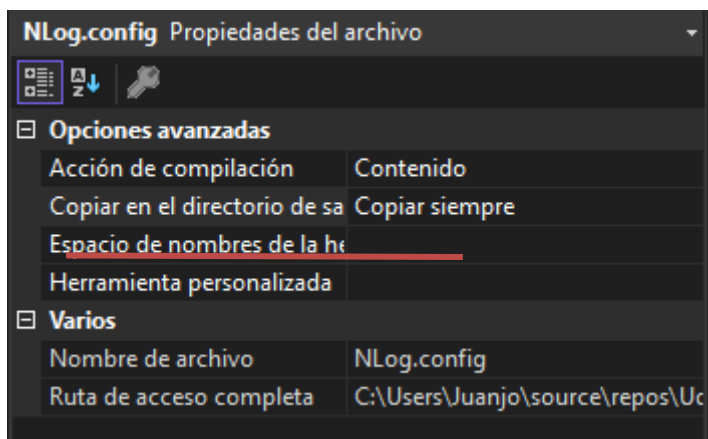
Bàsicament estem afegint tres *Target's* , és a dir, tres destins on s'emmagatzemarà la informació de Logging . Els dos primers emmagatzemaran la informació a un fitxer i el tercer a la consola.

El fitxers es guardarà en el directori **Logs** amb noms diferents.

El tercer destí serà el de la consola de l'entorn de desenvolupament.

En l'apartat **rules** indiquem quin nivells de log es deixaran en cada fitxer de log i també en la consola. El log de la consola es mantindrà en el període de desenvolupament, però no l'inclourem en producció.

Perquè funcione el log **hem de modificar una propietat** del fitxer de configuració (NLog.config). Concretament **Copiar al directori d'eixida** i canviar el seu valor a **Copiar Sempre**. També es recomana canviar la propietat **Acció de compilació** al valor **Contingut**.



Una vegada que ja hem configurat el framework ja podem utilitzar-ho. Les dues classes implicades són les següents:

- **Logger**: es tracta de la classe que s'utilitza per a emetre els missatges de log. Es recomana crear **una variable de log per classe** i declarar la **variable com a estàtica** perquè la creació de les variables de log és un procés costós.
- **LogManager**: crea les instàncies de la classe Logger.

```
namespace MyNamespace
{
    public class MyClass
    {
        private static Logger log = LogManager.GetCurrentClassLogger();
    }
}
```

Ara ja podem utilitzar la variable **log** per a incloure missatges en la consola i en el nostre fitxer de logging .

Per a poder incloure missatges és necessari comprendre els diferents nivells de logging que té el framework:

- **Trace**: és el nivell més detallat i el que inclou més informació. Sol ser utilitzat solament en l'etapa de desenvolupament.
- **Debug**:
- **Info**: s'utilitza per a missatges informatius
- **Warn**: són missatges no crítics que adverteixen sobre problemes que poden causar fallades.
- **Error**: en aquest nivell es mostren errors. Per exemple els missatges de les excepcions.
- **Fatal**: Un error molt que pot provocar la caiguda de l'aplicació.

Un exemple complet (per exemple a la classe de la finestra) podria ser aquest:

```
public partial class MainWindow : Window
{
    private static Logger logger = LogManager.GetCurrentClassLogger();

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

    public void provarLog()
    {
        logger.Trace("Sample trace message");
        logger.Debug("Sample debug message");
        logger.Info("Sample informational message");
        logger.Warn("Sample warning message");
        logger.Error("Sample error message");
        logger.Fatal("Sample fatal error message");

        // alternatively you can call the Log() method
        // and pass log level as the parameter.
        logger.Log(LogLevel.Info, "Sample informational message");
    }
}
```

Si executem observarem que al directori bin\Debug\net6.0-windows dins del projecte s'ha una carpeta amb dues fitxers (all.log i important.log):



```
all.log: Bloc de notas
Archivo  Editar  Ver

2022-12-04 19:20:12.9952|DEBUG|Ud7_Nlog.MainWindow|Sample debug message
2022-12-04 19:20:13.0104|INFO|Ud7_Nlog.MainWindow|Sample informational message
2022-12-04 19:20:13.0104|WARN|Ud7_Nlog.MainWindow|Sample warning message
2022-12-04 19:20:13.0104|ERROR|Ud7_Nlog.MainWindow|Sample error message
2022-12-04 19:20:13.0104|FATAL|Ud7_Nlog.MainWindow|Sample fatal error message
2022-12-04 19:20:13.0104|INFO|Ud7_Nlog.MainWindow|Sample informational message

Ln 1, Col 1

important.log: Bloc de notas
Archivo  Editar  Ver

2022-12-04 19:20:13.0104|WARN|Ud7_Nlog.MainWindow|Sample warning message
2022-12-04 19:20:13.0104|ERROR|Ud7_Nlog.MainWindow|Sample error message
2022-12-04 19:20:13.0104|FATAL|Ud7_Nlog.MainWindow|Sample fatal error message

Ln 1, Col 1
```

En cada fitxer s'ha emmagatzemat el nivell indicat al fitxer de configuració.

Per a aprendre més sobre aquest framework es poden consultar els següents llocs web:

<https://github.com/nlog/nlog/wiki/tutorial#log-levels>