TabControl, Binding et Mappage

Contenu :

1. TabControl

Un contrôle d'onglets (TabControl) et une façon de diviser l'interface utilisateur en différentes zones appelées « onglets ». Chaque onglet est accessible en cliquant sur l'entête de l'onglet. Il est souvent utilisé dans les applications Windows et on le trouve fréquemment dans l'interface du système d'exploitation Windows lui-même.

Par exemples :

- Dans la fenêtre des propriétés des fichiers ou des dossiers.
- Dans les navigateurs Web, etc.

Dans un « TabControl », chaque onglet est représenté avec un élément « TabItem » dont la propriété « Header » permet la définition du texte à afficher sur l'onglet. Un seul élément peut être définit à l'intérieur d'un « TabItem », mais si on veut y placer plusieurs contrôles, il suffit d'utiliser un panel avec des contrôles à l'intérieur.

Le contenu d'un « TabItem » ne sera visible que lorsque l'onglet est sélectionné.

Voici un exemple avec trois onglets :

```
<Grid Name="MonTabControl">
  <TabControl>
      <TabItem Header="onglet 1">
         <Label Content="Texte de l'onglet 1"/>
      </TabItem>

      <TabItem Header="onglet 2">
      </TabItem>

      <TabItem Header="onglet 3">
      </TabItem>
  </TabControl>
</Grid>
```

- Il est possible de créer un quatrième onglet à partir du « **code behind** ». Ajoutez le code métier nécessaire dans la méthode principale après l'initialisation des contrôles de l'interface graphique.

- Par défaut, les onglets du TabControl sont placés en haut du contrôle. Cependant, la propriété « TabStripPlacement » peut être utilisée pour changer l'emplacement des onglets à gauche ("Left"), en bas ("Bottom") ou à droite ("Right").

```
<TabControl TabStripPlacement="Bottom">
```

2. Insertion d'une image

Le premier onglet de notre application sera dédié aux prévisions météorologiques. Remplacez le texte de son en-tête par « Météo ».

- Utilisez une grille de 2 lignes et de 2 colonnes.

- Définissez un style local à la fenêtre « MainWindow » ciblant les contrôles « Border » : couleur d'arrière-plan « DarkBlue », couleur des bordures « SkyBlue » et épaisseur des bordures « 2 »
- Appliquez ce style aux deux dernières bordures (Border).
- Ajoutez un style local au Panel « Grid » ayant pour cible la mise en forme de tous ses « TextBlock » pour que le texte soit de couleur « blanche » et de taille « 25 ».
- Faites une recherche sur Internet d'une image de nuage avec flocons en utilisant les mots clés « cloud weather snowflake » et sauvegarder une des images sous le nom « flocons.jpg ». À partir de « l'explorateur de solutions », ajoutez cette image au projet dans un dossier « images ».
- À la suite du code XAML écrit dans le « TabItem » de l'onglet « Météo », ajoutez l'élément Image.

2.1. Ajout d'une icône à une fenêtre

Il sera possible d'ajouter une icône à une interface utilisateur en utilisant la propriété « Icon » dans l'élément « Window ».

- Ajoutez l'image de flocon comme icône de l'interface graphique.

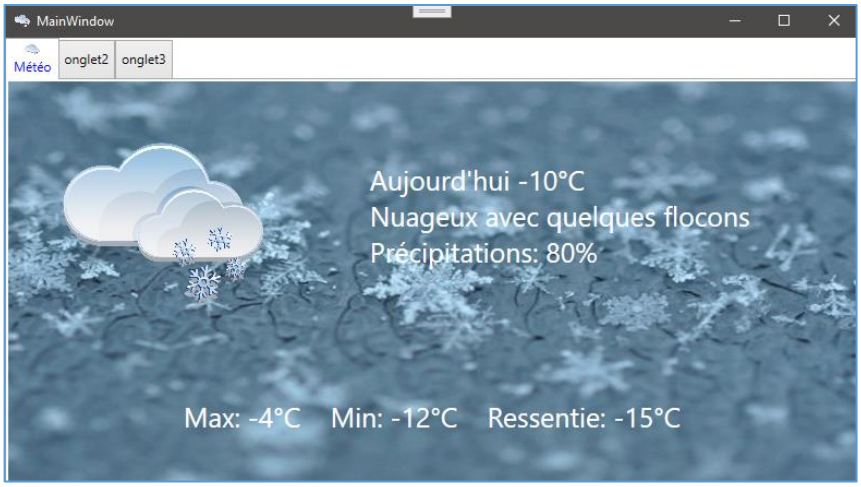
2.2. Personnalisation de l'en-tête d'un onglet avec une image

Il est également possible de personnaliser l'en-tête d'un onglet en insérant une image. Dans ce cas, il faut remplacer la propriété « Header » dans l'élément « TabItem » par un sous élément « TabItem.Header », d'y ajouter un « StackPanel » et d'y mettre l'image et un « TextBlock » pour l'en-tête de l'onglet « Météo ».

2.3. Image d'arrière-plan

« ImageBrush » peint un élément avec une « ImageSource ». Nous voulons ajouter une image d'arrière-plan au premier onglet, par exemple, l'image de chutes de neige. Faites la recherche d'une image en utilisant le mot « snowing ». Sauvegardez l'image puis ajoutez-la au projet dans le dossier « images ».

- Mettez en commentaire les trois contrôles « Border » pour enlever les couleurs de fond déjà définies.
- Dans l'élément « Grid » du premier onglet, ajoutez les balises nécessaires pour définir l'image « snowing.jpg » comme image d'arrière-plan du « Grid » principal comme dans la figure ci-dessous.

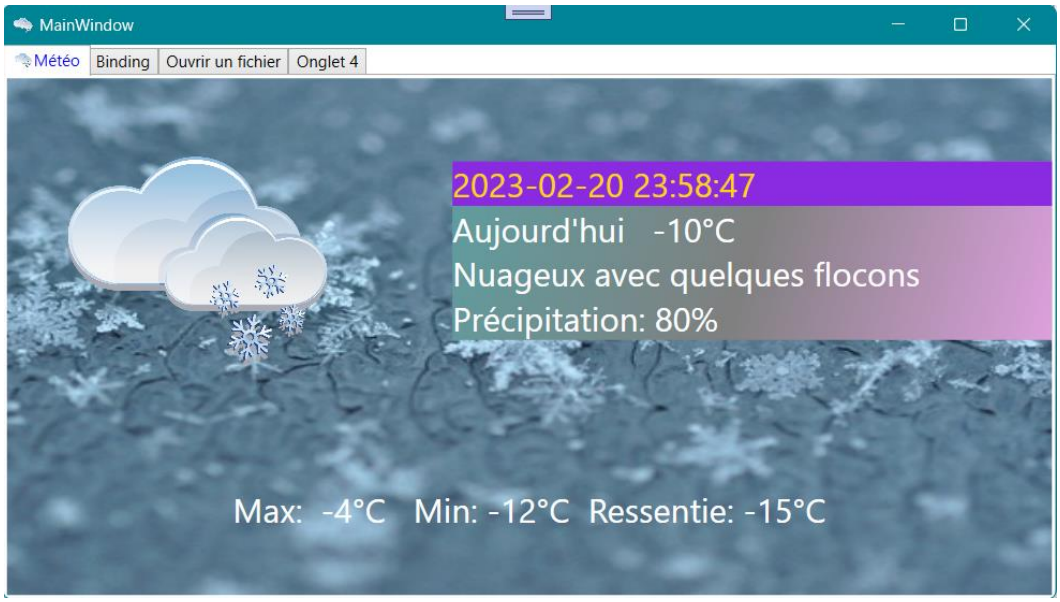


3. Peinture d'un contrôle à partir du code-behind

Tous les contrôles qui sont à l'écran sont visibles car ils ont été peints au pinceau. Par exemple, un pinceau est utilisé pour colorier l'arrière-plan d'un bouton, l'arrière-plan d'une zone de texte, d'une grille, etc.

3.1. Peinture avec une couleur unie

- Ajoutez un TextBlock en haut du stackPanel pour y afficher la date et l'heure et nommez-le « horloge ».



Pour le TextBlock « horloge », il est possible de définir la couleur d'arrière-plan ou la couleur du texte en définissant des pinceaux avec la classe « Brushes » (System.Windows.Media.Brushes).

- Ajoutez les instructions nécessaires dans le constructeur de « MainWindow » pour que le « TextBlock » ait un arrière-plan de couleur « BlueViolet » et une couleur de texte « Gold ».

NB : Vous trouvez dans l'adresse ci-dessous la palette des couleurs disponibles en WPF

https://docs.microsoft.com/en-us/dotnet/api/system.windows.media.colors?view=net-5.0

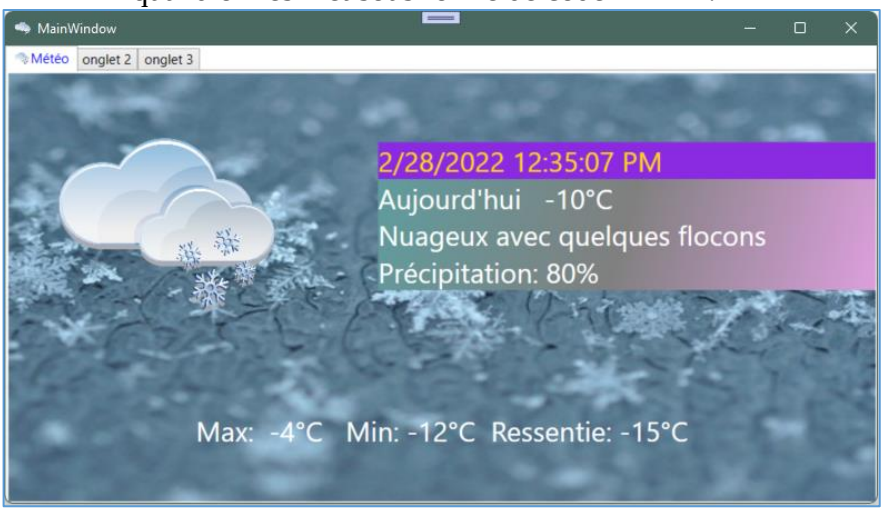
3.2. Peinture avec un dégradé linéaire

La peinture avec un dégradé linéaire est possible avec « LinearGradientBrush ». Un dégradé linéaire mélange deux couleurs ou plus sur une ligne, qui est l'axe du dégradé.

« GradientStop » permet de spécifier les couleurs du dégradé et leurs positions « offset » (une valeur entre 0 et 1).

- Nous voulons peinturer l'arrière-plan du « StackPanel ». Nommez-le « spPrevisions »
- Dans le constructeur, ajoutez le code métier permettant d'ajouter un dégradé linéaire des couleurs CadetBlue, Gray et Plum à 0, 0.5 et 1 respectivement.
- Quel est le code XAML correspondant ?

NB : Remarquez que contrairement au code-behind, les dégradés apparaissent directement sur l'interface utilisateur quand on les met sous forme de code XAML.



4. Liaison de deux contrôles de l'interface utilisateur

- Le Data Binding (liaison de données) fournit un moyen simple pour relier deux informations ou sources de données dans une application et assurer la synchronisation des valeurs entre elles, de telle manière que si l'une change, l'autre la suivra.
- Une fois les champs liés, le changement se fait automatiquement.

- Changez le titre de l'en-tête du deuxième onglet à « Binding ».

- En utilisant le « Binding », liez la valeur « d'un label » pour correspondre à la valeur saisie dans un TextBox.

  NB : On remarque que le contenu du label est mis à jour automatiquement en saisissant ou en enlevant du texte du TextBox nommé « txtValue ». Le binding nous a donc permis d'éviter d'écouter les événements du TextBox pour mettre à jour le label. La connexion entre les deux contrôles est donc établie via {Binding}.

- La source de données est définie explicitement avec la propriété « `ElementName` » qu'on lui affecte le nom du contrôle lié.
- « `Path` » indique vers quelle propriété du contrôle on veut effectuer le binding.
- On aurait pu avoir le même résultat en utilisant un « Data binding » via le code-behind. Mettez en commentaire le code permettant le Binding et ajoutez le code correspondant dans le constructeur.
- Ajoutez le bouton « btn_Accept » et la zone de texte « txt_TextBtn » qui aura le contenu du bouton.
- Question : Quel est le « code behind » équivalent qu'on pourrait utiliser pour relier la zone de texte au bouton ?

- On peut définir un « DataContext » pour le contrôle « Window » et l'utiliser pour tous les contrôles enfants de cette fenêtre. Ajoutez le code XAML suivant :

```
<WrapPanel Margin="10,10,0,0">
    <Label Content="Titre de la fenêtre: "/>
    <TextBox Text="{Binding ElementName=Wnd, Path=Title}" Width="150"/>
</WrapPanel>

<WrapPanel Margin="0,10,0,0">
  <Label Content="Dimensions de la fenêtre:"/>
  <Label Content="{Binding ElementName=Wnd, Path=Width}" Width="30"/>
  <Label Content=" x "/>
  <Label Content="{Binding ElementName=Wnd, Path=Height}" Width="30"/>
</WrapPanel>
```

→ On remarque que les labels qui affichent les dimensions de la fenêtre changent en redimensionnant l'interface utilisateur. Toutefois, le changement fait dans le TextBox contenant « Titre de la fenêtre » n'est pas immédiatement renvoyé à la source et le titre de la fenêtre ne change que lorsque le TextBox perd le focus.

- Ce comportement est contrôlé par une propriété du binding appelé « UpdateSourceTrigger ». Sa valeur par défaut est « default ». Pour que le binding se fait au changement du texte dans le TextBox, nous pouvons ajouter la propriété « UpdateSourceTrigger » tout en lui affectant la valeur « PropertyChanged ».
- Remplacez « `PropertyChange` » par « `LostFocus` ». Quand est ce que les modifications dans la zone de texte prennent effet dans le titre de la fenêtre ?

4.1. Débogage des erreurs de binding :

- Supposons que le développeur a commis une erreur en écrivant à la place de ligne du code suivante :

```
lblValue.SetBinding(Label.ContentProperty, liaison);
```

en écrivant à la place le code suivant :

```
lblValue.SetBinding(Label.ContentProperty, liaison.Source.ToString());
```

Est-ce que le compilateur détecte une erreur dans le code ? Non, l'exécution de l'application est plutôt lancée. Mais, il n'y a pas de valeur affichée dans le label « lblValue » et donc l'application ne fonctionne pas comme prévu.

Est-ce qu'il y'a un message d'erreur dans la fenêtre « Liste d'erreurs » ? Pas de message d'erreur car le compilateur n'a pas détecté d'erreurs.

Il s'agit ici d'une erreur de logique. Dans le cas d'une erreur similaire, il sera possible de trouver l'erreur dans la fenêtre « Sortie ». Si elle n'est pas visible, vous pouvez l'afficher avec le Menu Affichage | Sortie.

Cependant, si vous rencontrez une erreur de binding mais que vous ne voyez aucune référence à celle-ci dans la fenêtre « Sortie », cela peut être dû au fait que votre Visual Studio n'est pas actuellement configuré pour y envoyer des informations de débogage.

Vous pouvez activer cette fonctionnalité dans la fenêtre de dialogue Options de Visual Studio. Accédez à Outils | Options | Débogage | Fenêtre de sortie

- Dans la rubrique « Paramètres généraux de sortie », mettez toutes ses options à l'état « Actif » (excepté « Messages de diagnostic Natvis »).
- Dans la rubrique « Paramètres de trace WPF » (en haut), mettez l'option « Liaison de données » à « Erreur » ou « Tous ».

Lancez l'exécution de l'application. L'erreur de binding sera affichée dans la fenêtre de sortie, parmi plusieurs autres lignes, dans le format suivant :

```
System.Windows.Data Information: 10 : Cannot retrieve value using the binding
and no valid fallback value exists; using default instead.
BindingExpression:Path=System.Windows.Controls.TextBox; DataItem=null; target
element is 'Label' (Name='lblValue'); target property is 'Content' (type
'Object')
```

On peut comprendre que l'erreur provient d'une expression de binding en relation avec le contenu de l'élément « lblValue » d'un côté et le « TextBox » de l'autre côté.

5. Mappage vers des classes et des assemblys personnalisés

- Dans le premier onglet « Météo », on veut afficher la date et l'heure au moment de démarrage de l'application en utilisant la propriété « Now » de la struct « DateTime ».
- Pour que le binding fonctionne avec la méthode « DateTime » dans le fichier « .xaml », ajoutez la ligne suivante dans l'élément « Window » :

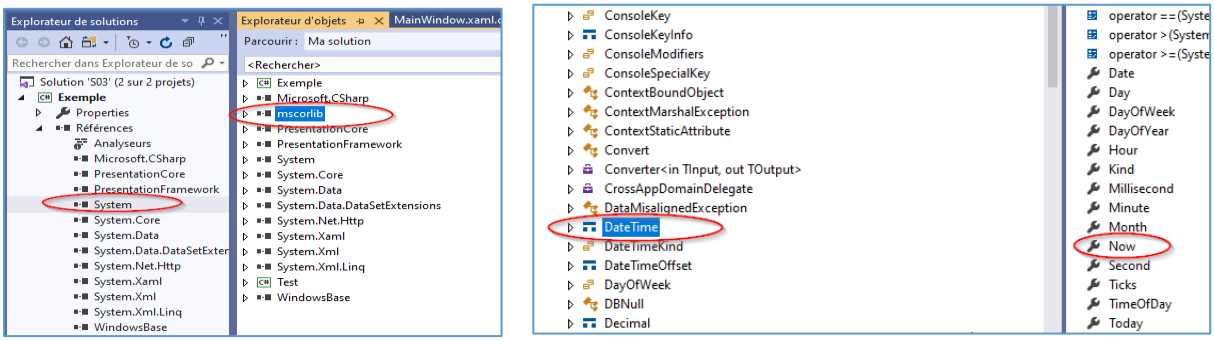
```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

- **xmlns:** Définit l'espace de noms par défaut par lequel le nom d'un élément XML doit être résolu.
- Le nom « sys » est choisi par le programmeur.
- **clr-namespace:** dans notre exemple, la valeur associée est « `System` ». C'est l'espace de noms CLR (Common Language Runtime) déclaré dans l'assembly qui contient les types publics à exposer en tant qu'éléments. CLR est le composant de la machine virtuelle du framework .NET
- **assembly=** dans notre exemple, la valeur associée est « `mscorlib` ». C'est l'assembly qui contient le tout ou une partie de l'espace de noms CLR référencé. On met juste le nom de l'assembly, pas de chemin ni d'extension (.dll ou .exe)

- Faites le binding au niveau de la propriété « Text » du TextBlock « horloge » :

```
<TextBlock Name="horloge" Text="{Binding Source={x:Static sys:DateTime.Now}}" />
```

- La struct « DateTime » se trouve dans l'Assembly « mscorlib.dll » qui lui-même se trouve dans l'espace de noms « System » comme c'est illustré ci-dessous :



NB : On aurait pu avoir le même résultat en utilisant le code-behind. Par défaut, nous avons l'instruction « `using System;` » permettant l'utilisation de « DateTime.Now ». Mettez donc en commentaire l'utilisation du binding pour la date dans le fichier « .xaml », puis ajoutez l'instruction suivante dans le code-behind du constructeur de la classe partielle « MainWindow.xaml.cs ».

```
horloge.Text = DateTime.Now.ToString();
```

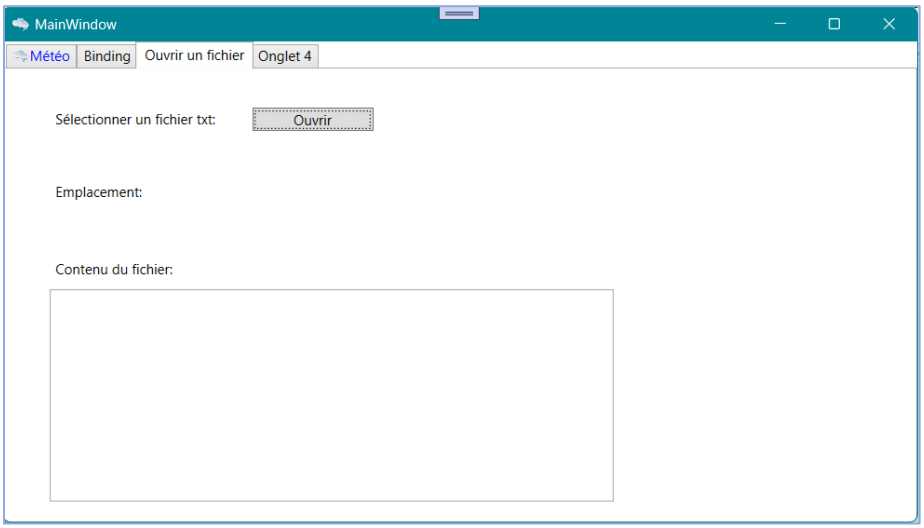
6. Exécution répétée d'une action à chaque intervalle de temps

- Pour avoir une horloge digitale qui affiche la date et l'heure actuelles en temps réel et qui se met à jour à chaque intervalle de temps, on peut utiliser un « `DispatcherTimer` » qui permet d'exécuter une action de manière répétée selon un intervalle de temps choisi.

La classe « DisptacherTimer » fonctionne en spécifiant un intervalle et en s'abonnant à l'événement « Tick » qui se déclenche chaque fois que l'intervalle est rencontré. L'objet de type « DispatcherTimer » est démarré lors de l'appel de la méthode « Start ».

7. Fenêtre « Ouvrir » pour sélectionner un fichier

- Pour afficher la fenêtre "Ouvrir" et permettre à l'utilisateur de sélectionner un fichier à partir de l'interface utilisateur, vous pouvez utiliser la classe « OpenFileDialog » de la bibliothèque System.Windows.Forms.



- Dans « l'onglet 3 » de « MainWindow.xaml », ajoutez les contrôles suivants : un Label et un Bouton nommé « btnOuvrir » pour afficher la fenêtre de sélection d'un fichier .txt.
- Ajoutez aussi un Label et un TextBlock « tbChemin » pour contenir l'emplacement du fichier choisi.
- Ajoutez un Label et un TextBox « txtContenu » pour contenir le contenu du fichier sélectionné avec le bouton.
- Sur clique sur le bouton « btnOuvrir », on fait appel à la méthode « btnOuvrir_Click » permettant d'ouvrir une fenêtre de sélection du fichier.
- Dans le code behind, définissez la méthode « btnOuvrir_Click » permettant de sélectionner un fichier en utilisant la classe « OpenFileDialog » de la bibliothèque System.Windows.Forms.