Utilisation des contrôles

La section précédente vous a rapidement montré la structure d'une application WPF. À cette occasion, vous avez pu remarquer que **l'interface graphique est composée de contrôles** définis par du **code XAML** ou, plus rarement, C#.

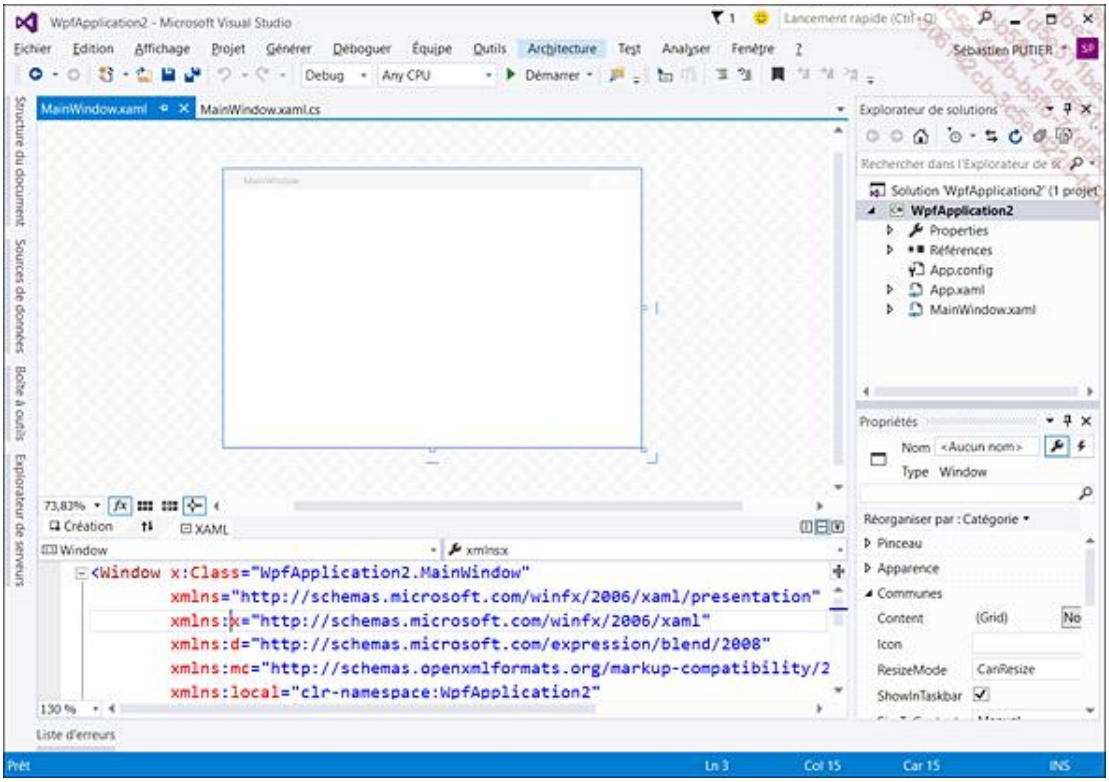
Ces contrôles font tous partie d'une hiérarchie leur permettant d'avoir des propriétés communes, la plupart du temps liées au positionnement. Chaque contrôle hérite ainsi d'un contrôle parent auquel il ajoute des fonctionnalités de manière à fournir un nouvel ensemble constitué d'un visuel et d'un comportement qui lui est associé.

Visual Studio intègre nativement un **concepteur visuel** qui permet la manipulation des contrôles d'une manière plus graphique, donc plus simple. Il génère ainsi lui-même le code XAML qui définit l'interface. Contrairement au concepteur visuel Windows Forms, le concepteur de WPF crée un code XAML qu'il est possible de manipuler directement, toute modification du code étant répercutée sur la partie de conception graphique et inversement.

1. Ajout de contrôles

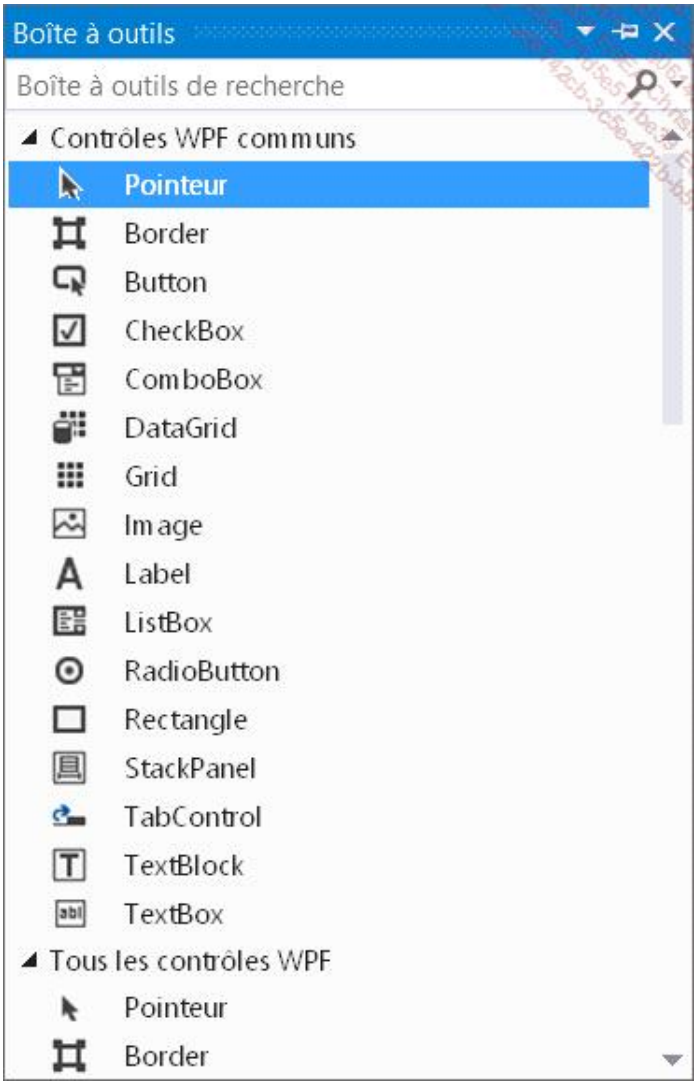
À la création d'un nouveau projet d'application WPF, Visual Studio affiche par défaut plusieurs éléments :

- l'**Explorateur de solutions**,
- la fenêtre **Propriétés**,
- une **fenêtre d'édition** scindée en deux parties : le concepteur visuel dans sa partie haute, l'éditeur de code XAML étant quant à lui situé dans la partie basse.



La fenêtre d'édition est destinée à la modification de la fenêtre principale de l'application, nommée `MainWindow`. Le concepteur visuel présente une vue de cette fenêtre, qui n'a pour l'heure aucun contenu.

Plusieurs fenêtres masquées automatiquement sont positionnées sur la gauche de l'environnement. Dans le contexte d'une création ou modification d'interface graphique, la plus importante est la fenêtre **Boîte à outils**.

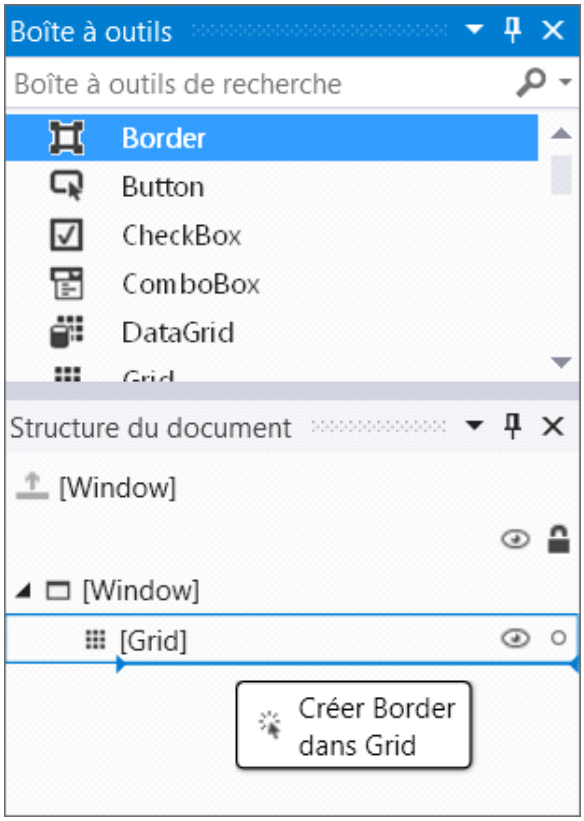


Cette fenêtre liste les différents contrôles utilisables dans une interface WPF. Elle permet également leur sélection et leur placement à l'aide de la souris. Cette approche s'avère beaucoup plus rapide que l'édition du code source XAML.

Trois solutions peuvent être utilisées pour le placement de contrôles dans une fenêtre WPF :

- Un **double clic** sur un contrôle dans la boîte à outils place un exemplaire du contrôle dans la fenêtre. Le contrôle est positionné par défaut dans le coin supérieur gauche de la fenêtre. Ses propriétés de dimensionnement sont spécifiées par défaut.
- L'utilisation du **glisser-déposer** sur un élément de la boîte à outils vers le concepteur visuel permet le positionnement du contrôle à l'endroit souhaité. Ses dimensions sont toutefois définies par défaut par le concepteur visuel.
- La sélection d'un élément dans la boîte à outils donne la capacité de **dessiner une zone rectangulaire** dont la position et la taille sont définies par l'utilisateur. Le contrôle sera positionné dans cette zone.

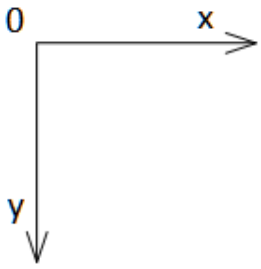
La fenêtre **Structure du document** offre une visualisation rapide de l'arborescence de la fenêtre en cours d'édition. Elle permet également d'ajouter un contrôle à un emplacement précis de cette arborescence. Vous pouvez utiliser le glisser-déposer d'un élément de la boîte à outils vers cette fenêtre. Certaines indications apparaissent lors du survol de cette fenêtre pour vous aider à insérer le contrôle à l'emplacement logique souhaité.



Le contrôle créé dans l'arborescence est positionné par défaut dans son contrôle conteneur, et ses propriétés de dimensionnement sont elles aussi valorisées par défaut par le concepteur visuel. Ce mode de création de contrôle est particulièrement pratique lorsque l'interface est complexe.

2. Positionnement et dimensionnement des contrôles

Les contrôles sont positionnés par défaut aux coordonnées 0;0 de leur conteneur (lorsque celui-ci le permet). Celles-ci correspondent au coin supérieur gauche du contrôle parent. WPF utilise en effet un système de coordonnées cartésiennes dans lequel l'axe des ordonnées est orienté vers le bas. La figure suivante fournit une représentation visuelle de ce repère.



Le positionnement par défaut est évidemment rarement celui qui est souhaité. Les contrôles WPF possèdent un grand nombre de propriétés permettant de modifier ce comportement ainsi que leur dimensionnement.

Height et Width

Les propriétés `Height` et `Width` permettent de définir de manière absolue ou relative la hauteur et la largeur d'un contrôle. Elles peuvent être valorisées avec trois types de valeur afin de remplir cet objectif :

- Une **valeur numérique absolue** en pixels.
- Une **valeur numérique** absolue suivie d'une **unité de mesure**. Cette unité peut être `px` (pixels), `in` (pouces), `cm` (centimètres), `pt` (points - unité utilisée en typographie).
- La valeur `Auto` indique que le contrôle doit être **dimensionné selon ses besoins et dans les limites de son contrôle conteneur**. Certains contrôles prendront par défaut tout l'espace disponible, tandis que d'autres adapteront leur taille à leur contenu.

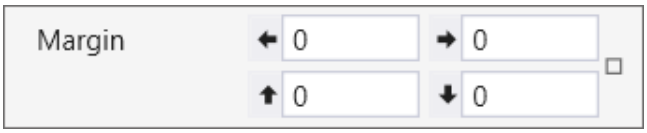
HorizontalAlignment et VerticalAlignment

Le positionnement d'un contrôle dans son parent peut être géré de manière automatique en fonction d'un alignement. Les propriétés `HorizontalAlignment` et `VerticalAlignment` permettent de mettre en œuvre ce type de positionnement en leur fournissant une valeur d'énumération indiquant l'alignement à utiliser : `Left`, `Center`, `Right` ou `Stretch` horizontalement et `Bottom`, `Center`, `Top` ou `Stretch` verticalement.

Les trois premières valeurs sont relativement explicites. Elles permettent d'aligner le contrôle sur le bord gauche, le centre ou le bord droit de son parent. La valeur `Stretch` permet quant à elle d'imposer au contrôle d'utiliser tout l'espace disponible horizontalement ou verticalement.

Margin

La propriété `Margin` d'un contrôle définit une zone vide autour du contrôle permettant de le **décaler par rapport à sa position normale**. Quatre valeurs sont associées à cette propriété, elles sont associées aux quatre bords du contrôle. La figure ci-dessous montre la définition de ces valeurs dans la fenêtre **Propriétés** de Visual Studio.



Lorsque ces valeurs sont éditées directement dans le code XAML, elles doivent être fournies dans l'ordre suivant : **gauche**, **haut**, **droite**, **bas**.

```
<Button Margin="0 10 0 0" />
```

Ici, le contrôle `Button` est décalé de 10 pixels vers le bas par rapport à son positionnement normal dans le conteneur.

Les valeurs données à cette propriété peuvent, comme les propriétés `Width` et `Height`, spécifier une unité de mesure particulière. Visual Studio ne supporte pas ces unités de mesure dans la fenêtre de propriétés, il est donc impératif, si vous souhaitez les utiliser, de les fournir dans l'éditeur de code source XAML.

```
<Button Margin="1cm 2px 3in 4pt"/>
```

C'est la propriété `Margin` qui est utilisée par le concepteur visuel de Visual Studio pour le positionnement d'un contrôle dans un conteneur de type `Grid`, par exemple. Il est donc intéressant d'étudier ce positionnement à travers le placement d'un contrôle et de voir comment les valeurs peuvent varier en fonction de la structure de l'objet `Grid` (lignes, colonnes et positionnement du contrôle dans celles-ci).

Padding

La propriété `Padding` permet de définir une zone, située à l'intérieur du contrôle, dans laquelle le contenu ne doit pas être affiché. Pour le contrôle `Button`, par exemple, cette propriété peut être utilisée afin que le contenu n'atteigne jamais les bords du contrôle. Sa valeur est définie de la même manière que la valeur de la propriété `Margin`.

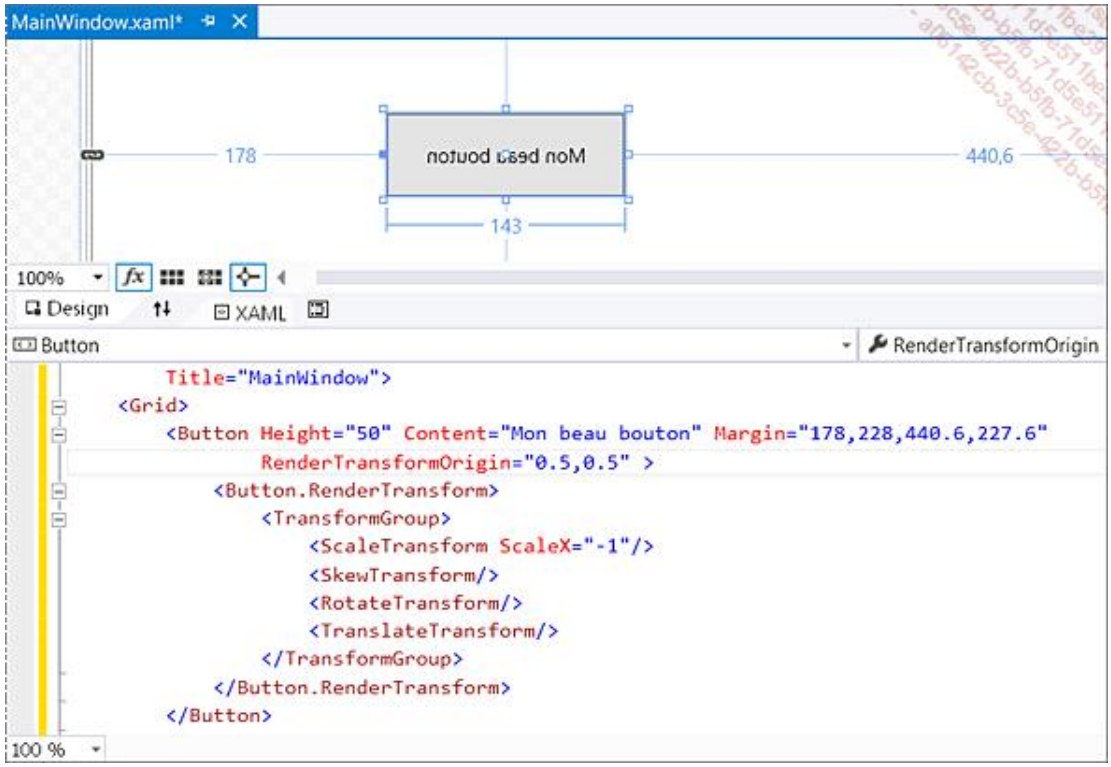
Le positionnement dans un Canvas

Lorsqu'un conteneur est de type `Canvas`, le positionnement peut être effectué en valorisant les propriétés `Canvas.Top`, `Canvas.Left`, `Canvas.Right` et `Canvas.Bottom` sur chacun des contrôles qu'il contient. Celles-ci permettent de spécifier la distance entre un bord du `Canvas` et le bord associé du contrôle. Elles acceptent des valeurs numériques, avec ou sans unité de mesure associée.

Ces propriétés de positionnement sont un peu particulières puisqu'elles appartiennent au contrôle `Canvas` mais opèrent sur les contrôles qu'il contient. On les appelle **propriétés attachées**. Vous pourrez être amenés à en créer, notamment pour étendre les fonctionnalités d'un contrôle sans en hériter.

Le concepteur visuel permet d'agir sur le positionnement des contrôles très simplement. En effet, un glisser-déposer sur un contrôle déjà placé permet de le déplacer et ainsi de modifier la valeur de sa propriété `Margin` ou des propriétés attachées issues du contrôle `Canvas`, en fonction du conteneur utilisé.

Il est également possible de redimensionner un contrôle en mode de conception. Pour cela, il suffit de sélectionner un contrôle, puis de positionner le curseur de la souris sur une des ancres situées sur les bordures et enfin de l'étirer en maintenant le bouton gauche de la souris appuyé tout en déplaçant le curseur. Le concepteur ne vous empêche pas de déplacer le bord droit au-delà du bord gauche du contrôle. Il considère en effet que ce comportement est assimilable à un retournement du contrôle et génère une transformation permettant de visualiser un contrôle "à l'envers" !



3. Ajout d'un gestionnaire d'événements à un contrôle

Les contrôles visuels peuvent tous déclencher de nombreux événements pour lesquels il est possible de s'abonner dans le code source C# associé à la fenêtre. Il est également possible de s'abonner à ces événements à partir du code source XAML.

Pour cela, il suffit d'ajouter à la déclaration XAML d'un contrôle un attribut ayant la forme suivante :

```
<nom d'événement>="<nom du gestionnaire>"
```

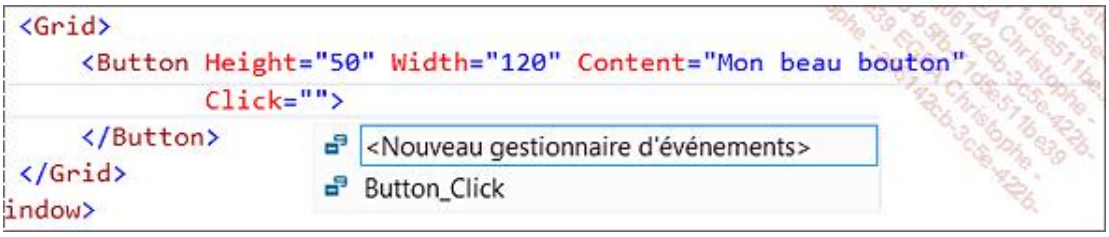
```
<Grid>
    <Button Height="50" Width="120" Content="Mon beau bouton"
            Click="Button_Click">
    </Button>
</Grid>
```

Le gestionnaire d'événements correspondant doit se trouver dans le fichier de code-behind de la fenêtre en cours d'édition. Ici, la gestion du clic sur le bouton est effectuée par la méthode `Button_Click` dont la définition est la suivante.

```
private void Button_Click(object sender, RoutedEventArgs e)
{

}
```

IntelliSense fournit une aide précieuse pour effectuer cette opération. Il permet en effet de générer automatiquement un gestionnaire correspondant au type de délégué associé à l'événement. Il propose également, quand cela est possible, d'associer l'événement à un gestionnaire d'événements existant.



Un double clic sur un contrôle génère également un gestionnaire pour l'événement par défaut du contrôle. Le gestionnaire et l'événement sont associés automatiquement dans le code XAML.

Lorsqu'un gestionnaire n'existant pas dans le code C# est lié à un événement dans le code XAML, une erreur de compilation est levée.