Ressourcesinformatiques

C# sous Windows Phone

Développez et publiez vos applications sur le Store

Christopher MANEU

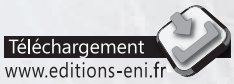


Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **RIWPCSHA** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 2
Présentation de Visual Studio

Chapitre 3
Les bases du langage C#

Chapitre 4
La plateforme Windows Phone

Chapitre 5
La programmation orientée objet

Chapitre 6
La création de l'interface utilisateur

Chapitre 7
La gestion des données

Chapitre 8
Gérer le cycle de vie d'une application

Chapitre 9
Aller plus loin avec l'interface utilisateur

Chapitre 10
Intégrer les applications à Windows Phone

Chapitre 11
Conception et ergonomie des applications

Chapitre 12
Diffuser et faire évoluer l'application

Chapitre 6
La création de l'interface utilisateur

1. Organiser et positionner les contrôles

1.1 Retour sur XAML

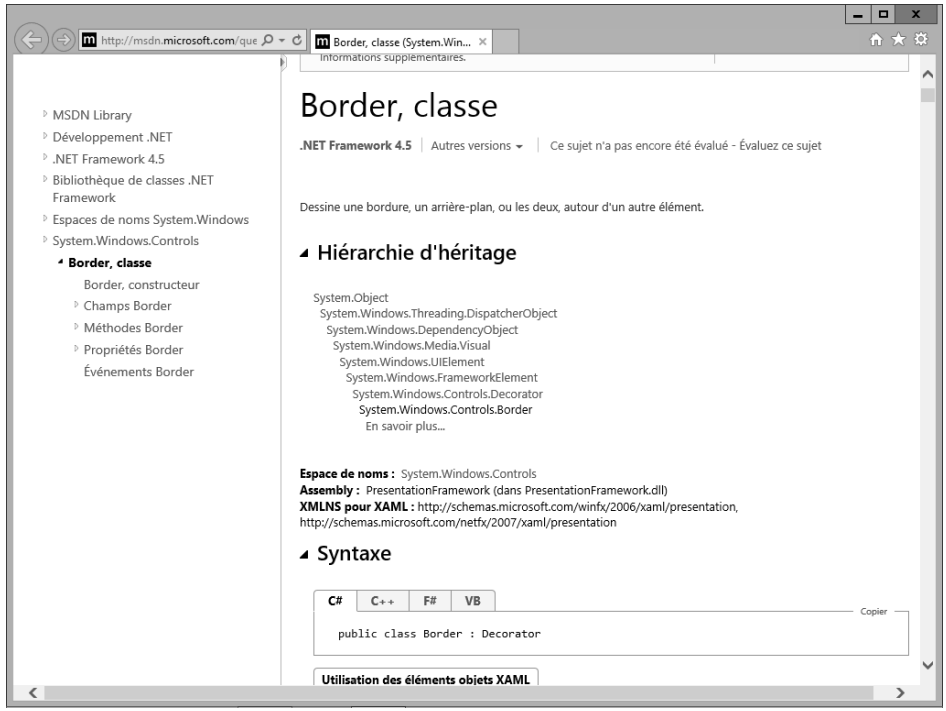
Nous avons déjà abordé l'écriture de code XAML dans le chapitre La plateforme Windows Phone. Ce chapitre va aborder les détails de la création de vos interfaces utilisateur avec ce langage.

La plateforme XAML disponible sous Windows Phone 8 (qui peut différer des autres plateformes disponibles, par exemple, pour les applications Windows Store ou WPF) offre un ensemble de contrôles afin de créer votre interface. Ces contrôles partagent un ensemble de propriétés ou de comportements. Nous allons tout d'abord nous pencher sur ces similitudes avant de voir ce que ces contrôles peuvent vous apporter.

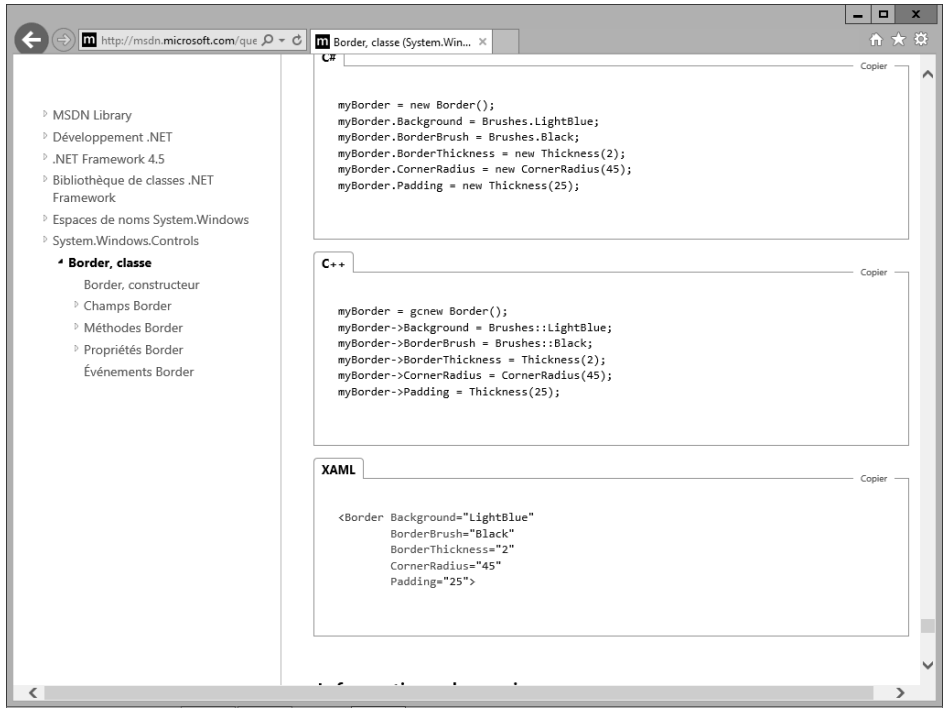
Remarque

Il existe des contrôles créés par des sociétés (telles que Telerik) ou disponibles sous licences libres. Ils peuvent vous faire gagner du temps ou enrichir rapidement vos applications. Nous verrons une partie de ces contrôles dans le chapitre Aller plus loin avec l'interface utilisateur.

Tous les contrôles disponibles sont accessibles directement via l'Intellisense dans un fichier XAML, ou depuis la barre d'outils. Si vous positionnez le curseur de l'éditeur au sein d'une balise XAML et que vous appuyez sur la touche [F1] de votre clavier, le navigateur s'ouvre directement sur la page d'aide du contrôle courant.



Cette page nous apporte plusieurs informations intéressantes. Tout d'abord, cela nous permet de nous remémorer que tous les objets graphiques ne sont que des classes. Ils sont donc accessibles également depuis du code C#, que cela soit pour créer des objets, ou pour modifier les propriétés d'objets définis dans du code XAML. La documentation en ligne apporte également des exemples de code pour utiliser ces contrôles.



Remarque

Il vous est possible de consulter la documentation sans connexion Internet. La procédure est expliquée sur http://blog.maneu.net/acceder-a-la-documentation-visual-studio-windows-phone-net-hors-connexion/.

Cette page de documentation nous apporte également une autre information intéressante : la hiérarchie d'héritage. Tous les contrôles graphiques héritent de la classe `DispatcherObject`. Ils sont donc intégrés au thread Dispatcher que nous avons abordé dans le chapitre La plateforme Windows Phone. D'autres classes parentes apportent des propriétés utiles pour un grand nombre de contrôles, telles que les propriétés `Width` et `Height` que nous aborderons plus loin dans ce chapitre.

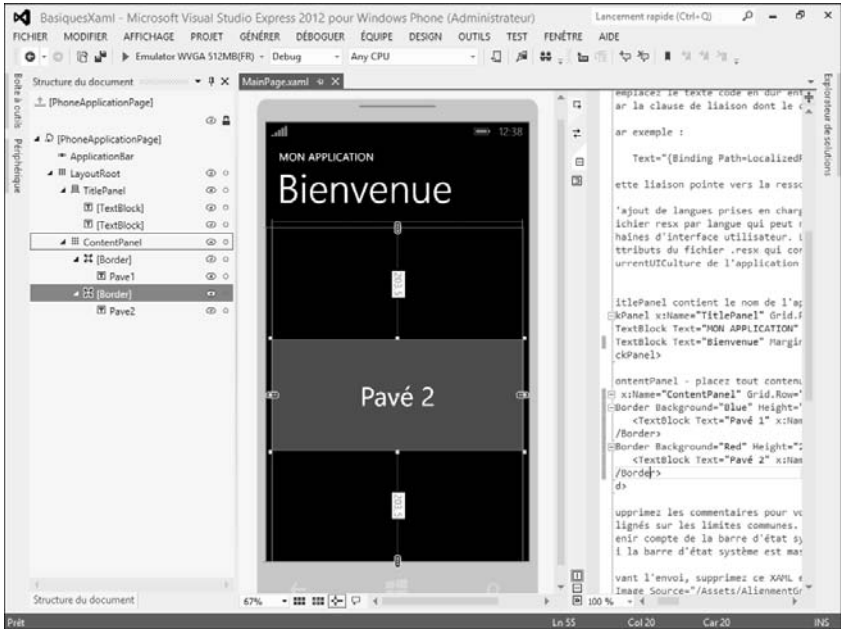
Avant de nous pencher sur les propriétés directement liées au positionnement, il faut comprendre comment fonctionne le moteur XAML pour calculer la mise en page des objets graphiques. Lorsqu'une propriété impactant le positionnement change de valeur, telle que `Height` ou `HorizontalAlignment`, le recalcul de l'écran n'est pas effectué directement. C'est le système qui décide quand est exécuté le prochain cycle de mise en page. Celui-ci est réalisé en deux phases: **Measure** et **Arrange**.

La phase **Measure** va déterminer la taille nécessaire à l'objet pour être dessiné. La méthode `Measure()` est ainsi appelée sur tous les contrôles qui vont en retour indiquer la taille d'affichage qu'ils demandent. À partir de ces résultats, la phase Arrange va finir le processus de mise en page en définissant ainsi la taille réelle qu'obtiendra chacun des contrôles. La manière dont la taille de chaque élément est déterminée dépend des objets conteneurs, que nous verrons en détail plus loin dans ce chapitre.

Remarque

Ce principe de fonctionnement a de nombreuses implications, notamment en termes de performances, ou dans les animations. Il est donc important de le conserver à l'esprit dès que l'on crée des animations ou que l'on modifie via le code des propriétés impactant le positionnement.

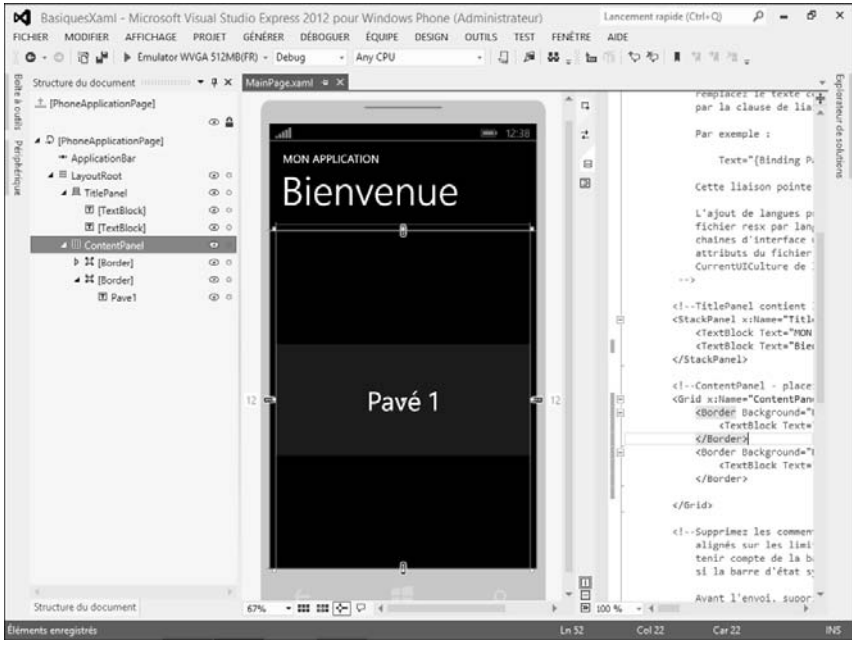
Pour calculer la mise en page d'une page entière, il suffit que le système appelle les méthodes `Measure()` et `Arrange()` sur le contrôle enfant de la page. Il va ainsi appeler ces deux mêmes méthodes sur l'ensemble de ses enfants, qui vont faire de même et ainsi de suite, afin de couvrir l'ensemble de la hiérarchie des contrôles. Cette **hiérarchie de contrôles XAML** n'a rien à voir avec la notion d'héritage telle qu'on l'a vu en C#, ou au début de ce chapitre. Il s'agit ici de l'ensemble de l'arborescence de contrôles qui a été définie en XAML (ou en C#). On peut visualiser cette hiérarchie dans l'éditeur XAML avec le panneau **Structure du document**.



Cette structure, et l'ordre des contrôles dans cette hiérarchie, a un impact sur la mise en page. Par exemple, prenons le code utilisé dans la capture d'écran précédente. Nous pouvons voir qu'au sein de l'élément `Grid` nommé `ContentPanel`, nous avons défini deux contrôles `Border`. Un seul de ces éléments est visible, celui qui a été défini en dernier dans le code XAML.

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Border Background="Blue" Height="200">
        <TextBlock Text="Pavé 1" x:Name="Pave1"
HorizontalAlignment="Center"  VerticalAlignment="Center"
FontSize="48"/>
    </Border>
    <Border Background="Red" Height="200">
        <TextBlock Text="Pavé 2" x:Name="Pave2"
HorizontalAlignment="Center"  VerticalAlignment="Center"
FontSize="48"/>
    </Border>
</Grid>
```

Si nous inversons les deux blocs afin que le pavé 1 soit déclaré en dernier, c'est bien ce second pavé qui est visible. Ce comportement est dû à deux facteurs : le fonctionnement du contrôle `Grid`, et l'ordre de déclaration. Les contrôles déclarés en fin de fichier ont la priorité sur ceux déclarés précédemment.



1.2 Les propriétés liées au positionnement

1.2.1 Width et Height

De nombreuses propriétés peuvent affecter directement le positionnement. Dans cette liste, `Width` et `Height` sont certainement les plus simples. Elles permettent, respectivement, d'indiquer la largeur et la hauteur souhaitées. Afin de mieux comprendre comment ces deux propriétés fonctionnent, nous allons prendre les quatre exemples suivants, en appliquant ces propriétés sur les objets `Border`.

```
<StackPanel x:Name="ContentPanel" Grid.Row="1"
Margin="12,0,12,0">
    <Border BorderBrush="Aqua" BorderThickness="3">
        <TextBlock Text="Bloc 1" />
    </Border>
    <Border Width="100" BorderBrush="Red" BorderThickness="3">
        <TextBlock Text="Bloc 2" />
    </Border>
    <Border Width="Auto" BorderBrush="MediumPurple"
BorderThickness="3">
        <TextBlock Text="Bloc 3" />
    </Border>
    <Border Height="180" BorderBrush="Orange"
BorderThickness="3">
        <TextBlock Text="Bloc 4" />
    </Border>
</StackPanel>
```

Le premier bloc n'a aucune de ces propriétés définie explicitement. C'est donc l'objet parent – ici le `StackPanel` – qui va déterminer la hauteur et la largeur de ce bloc.

Dans le second bloc, nous avons explicitement défini une largeur, exprimée en pixels. Cette largeur étant inférieure à la largeur du `StackPanel` parent, elle est directement appliquée. Il est possible de définir une largeur ou une hauteur plus grande que la taille dont nous disposons, mais le résultat dépend alors du conteneur utilisé.

Le troisième bloc a le même positionnement que le premier bloc, alors qu'il définit la valeur `Auto` comme largeur. `Auto` est en fait une valeur spécifique en XAML (traduite en `Double.NaN` en code) qui indique au système qu'il doit évaluer la valeur en fonction d'autres propriétés et du conteneur.