

Créez des applications en C# pour Windows Phone

8

Par nico.pyright



www.openclassrooms.com

Sommaire

| | |
|--|-----|
| Sommaire | 2 |
| Partager | 3 |
| Créez des applications en C# pour Windows Phone 8 | 5 |
| Partie 1 : La théorie et les bases | 6 |
| Windows Phone, un nouveau périphérique | 6 |
| Historique, la mobilité chez Microsoft | 6 |
| Le renouveau : Windows Phone 7 | 6 |
| Windows Phone 8 | 7 |
| L'esprit Modern UI | 8 |
| Le Windows Phone Store | 9 |
| Les outils de développements | 10 |
| Prérequis | 11 |
| Installer Visual Studio Express pour Windows Phone | 12 |
| L'émulateur | 20 |
| XAML et code behind | 24 |
| Blend pour le design | 25 |
| Notre première application | 26 |
| Hello World | 27 |
| L'interface en XAML | 31 |
| Le code-behind en C# | 34 |
| Le contrôle Grid | 35 |
| Le contrôle StackPanel | 36 |
| Le contrôle TextBox | 36 |
| Le contrôle TextBlock | 38 |
| Les événements | 39 |
| Le bouton | 40 |
| Et Silverlight dans tout ça ? | 42 |
| Les contrôles | 42 |
| Généralités sur les contrôles | 43 |
| Utiliser le designer pour ajouter un CheckBox | 44 |
| Utiliser Expression Blend pour ajouter un ToggleButton | 46 |
| Le clavier virtuel | 48 |
| Afficher le clavier virtuel | 49 |
| Intercepter les touches du clavier virtuel | 49 |
| Les différents types de clavier | 50 |
| Les conteneurs et le placement | 52 |
| StackPanel | 53 |
| ScrollViewer | 56 |
| Grid | 58 |
| Canvas | 63 |
| Alignement | 64 |
| Marges et espacement | 66 |
| Ajouter du style | 69 |
| Afficher des images | 69 |
| Les ressources | 71 |
| Les styles | 74 |
| Les thèmes | 80 |
| Changer l'apparence de son contrôle | 85 |
| TP1 : Création du jeu du plus ou du moins | 87 |
| Instructions pour réaliser le TP | 88 |
| Correction | 88 |
| Dessiner avec le XAML | 91 |
| Dessin 2D | 92 |
| Pinceaux | 95 |
| Les transformations | 99 |
| Créer des animations | 102 |
| Principe généraux des animations | 103 |
| Création d'une animation simple (XAML) | 103 |
| Création d'une animation complexe (Blend) | 107 |
| Projections 3D | 112 |
| Partie 2 : Un mobile orienté données | 116 |
| Une application à plusieurs pages, la navigation | 116 |
| Naviguer entre les pages | 116 |
| Gérer le bouton de retour arrière | 124 |
| Ajouter une image d'accueil (splash screen) | 126 |
| Le tombstonning | 128 |
| TP 2 : Créez une animation de transition entre les pages | 137 |
| Instructions pour réaliser le TP | 137 |
| Correction | 137 |
| Les propriétés de dépendances et propriétés attachées | 141 |
| Les propriétés de dépendances | 141 |
| Les propriétés attachées | 142 |

| | |
|--|-----|
| Où est mon application ? | 143 |
| Le .XAP | 143 |
| Affichage d'images en ressources | 144 |
| Accéder au flux des ressources | 144 |
| ListBox | 145 |
| Un contrôle majeur | 145 |
| Gérer les modèles | 148 |
| Sélection d'un élément | 151 |
| La manipulation des données (DataBinding & Converters) | 155 |
| Principe du Databinding | 156 |
| Le binding des données | 156 |
| Binding et mode design | 165 |
| Utiliser l'ObservableCollection | 172 |
| Les converters | 177 |
| MVVM | 181 |
| Principe du patron de conception | 182 |
| Première mise en place de MVVM | 183 |
| Les commandes | 190 |
| Les frameworks à la rescousse : MVVM-Light | 194 |
| D'autres frameworks MVVM | 210 |
| Faut-il utiliser systématiquement MVVM ? | 211 |
| Gestion des états visuels | 212 |
| Les états d'un contrôle | 212 |
| Modifier un état | 217 |
| Changer d'état | 222 |
| Créer un nouvel état | 222 |
| Le traitement des données | 224 |
| HttpRequest & WebClient | 225 |
| Linq-To-Json | 231 |
| La bibliothèque de Syndication | 237 |
| Asynchronisme avancé | 241 |
| Le répertoire local | 244 |
| Partie 3 : Une bibliothèque de contrôles | 247 |
| Panorama et Pivot | 248 |
| Panorama | 248 |
| Pivot | 255 |
| Navigateur web | 259 |
| Naviguer sur une page web | 260 |
| Événements de navigation | 261 |
| Navigation interne | 261 |
| Communiquer entre XAML et HTML | 263 |
| TP : Création d'un lecteur de flux RSS simple | 270 |
| Instructions pour réaliser le TP | 271 |
| Correction | 271 |
| Aller plus loin | 285 |
| Gérer l'orientation | 286 |
| Les différentes orientations | 287 |
| Déetecter les changements d'orientation | 290 |
| Stratégies de gestion d'orientation | 292 |
| Gérer les multiples résolutions | 304 |
| Les différentes résolutions | 305 |
| Gérer plusieurs résolutions | 306 |
| Les images | 307 |
| L'image de l'écran d'accueil | 309 |
| L'application Bar | 310 |
| Présentation et utilisation | 310 |
| Appliquer le Databinding | 318 |
| Mode plein écran | 321 |
| Le toolkit Windows Phone | 322 |
| Présentation et installation du toolkit Windows Phone | 322 |
| PerformanceProgressBar | 322 |
| ListPicker | 325 |
| WrapPanel | 332 |
| LongListSelector | 335 |
| Avantages & limites du toolkit | 342 |
| Les autres toolkits | 342 |
| Le contrôle de cartes (Map) | 343 |
| Présentation et utilisation | 344 |
| Interactions avec le contrôle | 347 |
| Epingler des points d'intérêt | 353 |
| Afficher un itinéraire | 363 |
| TP : Une application météo | 368 |
| Instructions pour réaliser le TP | 369 |
| Correction | 371 |
| Partie 4 : Un téléphone ouvert vers l'extérieur | 382 |
| La gestuelle | 383 |
| Le simple toucher | 383 |
| Les différents touches | 384 |
| Gestuelle avancée | 384 |

| | |
|--|------------|
| Le toolkit à la rescousse | 384 |
| L'accéléromètre | 386 |
| Utiliser l'accéléromètre | 387 |
| Utiliser l'accéléromètre avec l'émulateur | 388 |
| Exploiter l'accéléromètre | 390 |
| Les autres capteurs facultatifs | 392 |
| La motion API | 392 |
| TP : Jeux de hasard (Grattage et secouage) | 395 |
| Instructions pour réaliser le TP | 396 |
| Correction | 398 |
| Aller plus loin | 405 |
| La géolocalisation | 405 |
| Déterminer sa position | 406 |
| Utiliser la géolocalisation dans l'émulateur | 409 |
| Utiliser la géolocalisation avec le contrôle Map | 410 |
| Les Tasks du téléphone | 411 |
| Les choosers | 412 |
| Les launchers | 415 |
| Etat de l'application | 419 |
| Les tuiles | 421 |
| Que sont les tuiles ? | 421 |
| Des tuiles pour tous les goûts | 423 |
| Personnaliser les tuiles par défaut | 426 |
| Créer des tuiles secondaires | 433 |
| Modifier et supprimer une tuile | 437 |
| Les notifications | 438 |
| Le principe d'architecture des notifications | 439 |
| Le principe de création du serveur de notification | 440 |
| Les différents messages de notifications | 441 |
| Création du client Windows Phone recevant la notification | 442 |
| TP : Améliorer l'application météo avec géolocalisation et tuiles | 448 |
| Instructions pour réaliser le TP | 449 |
| Correction | 449 |
| Aller plus loin | 453 |
| Une application fluide = une application propre ! | 456 |
| Un thread, c'est quoi ? | 456 |
| Le thread d'interface | 456 |
| Utiliser un thread d'arrière-plan | 457 |
| Utiliser le Dispatcher | 459 |
| Utiliser un BackgroundWorker | 460 |
| Enchaîner les threads dans un pool | 463 |
| Le DispatcherTimer | 464 |
| Thread de composition | 466 |
| Les outils pour améliorer l'application | 467 |
| Utiliser des tâches Background Agent | 467 |
| Créer un Background Agent pour une tâche périodique | 468 |
| Créer une tâche aux ressources intensives | 474 |
| Remarques générales sur les tâches | 474 |
| Envoyer une notification avec un agent d'arrière-plan | 475 |
| Utiliser Facebook dans une application mobile | 477 |
| Créer une application Facebook | 478 |
| Simplifier les connexions à Facebook avec l'API | 479 |
| Sécuriser l'accès à Facebook avec OAuth | 480 |
| Se connecter à Facebook | 481 |
| Exploiter le graphe social avec le SDK | 487 |
| Récupérer des informations | 487 |
| Obtenir la liste de ses amis | 489 |
| Publier un post sur son mur Facebook | 491 |
| Utiliser les tasks | 495 |
| Publier son application | 498 |
| Créer un compte développeur | 498 |
| Inscrire un téléphone de développeur | 503 |
| Proposer une version d'essai de votre application | 509 |
| Certifier son application avec le Store Test Kit | 509 |
| Publier son application sur le Windows Phone Store | 515 |



Créez des applications en C# pour Windows Phone 8

Par [nico.pyright](#)

Mise à jour : [10/10/2013](#)

Difficulté : Facile



La révolution de la mobilité est en marche. Nous connaissons tous l'iPhone qui a su conquérir un grand nombre d'utilisateurs, ainsi que les téléphones Android dont le nombre ne cesse de croître... Ces téléphones intelligents (ou smartphones) deviennent omniprésents dans nos usages quotidiens. Microsoft se devait de monter dans le TGV de la mobilité ! Sont donc apparus, peu après ses deux grands concurrents, les téléphones Windows. Avec un peu plus de retard sur eux, Microsoft attaque ce marché avec plus de maturité qu'Apple qui a foncé en tant que pionnier et nous propose son système d'exploitation : **Windows Phone**.

C'est une bonne nouvelle pour nous ! C'est aujourd'hui un nouveau marché qui s'ouvre à nous avec plein d'applications potentielles à réaliser grâce à nos talents de développeur. Si c'est pour ça que vous vous trouvez sur cette page, alors restez-y ! Dans ce cours, nous allons apprendre à réaliser des applications pour Windows Phone grâce à notre langage préféré, le **C#**. Bonne nouvelle non ?

Il est possible de réaliser deux grands types d'application Windows Phone :

- des applications dites de gestion
- des jeux

Il est aussi possible de développer des applications pour Windows Phone en VB.NET et en F#, ainsi qu'en C++. Je ne traiterai que le C# dans ce cours.

Dans ce cours, nous allons apprendre à développer des applications de gestion avec **Silverlight** pour Windows Phone, qui est utilisé dans les versions 7 de Windows Phone, mais également des applications **XAML/C#**, utilisé pour développer des applications pour Windows Phone 8. Vous ne connaissez pas Silverlight, ni XAML/C# ? Ce n'est pas grave, nous allons les introduire rapidement (mais sûrement) dans les prochains chapitres.



Sachez que vous pouvez suivre beaucoup de chapitres de ce cours sans posséder forcément de Windows Phone, ce qui est idéal lorsque l'on souhaite simplement découvrir le sujet. Le seul pré-requis sera de maîtriser un tant soit peu le C#. Pour ceux qui auraient besoin d'une piqûre de rappel, vous pouvez consulter mon [cours C#](#) sur le Site OpenClassrooms.

Avant de commencer, je dois quand même vous signaler que le développement pour Windows Phone peut rendre accroc ! Vous allez avoir envie de créer des applications sans jamais vous arrêter ! Si vous êtes prêts à assumer cette probable dépendance, c'est que vous êtes bons pour continuer. Alors, c'est parti !



Partie 1 : La théorie et les bases

Windows Phone, un nouveau périphérique

Windows Phone est la nouvelle plateforme pour téléphones mobiles de Microsoft. Elle permet de réaliser des applications pour les smartphones équipés du système d'exploitation Windows Phone. Apparues dans un premier temps sous sa version 7, Windows Phone en est actuellement à sa version 8.

Microsoft arrive en rupture avec son nouveau système d'exploitation pour smartphone afin de concurrencer les deux géants que sont Apple et Google.

Historique, la mobilité chez Microsoft

Cela fait longtemps que Microsoft dispose de matériels mobiles. On a eu dans un premier temps les périphériques équipés de Windows CE ; ce système d'exploitation était une variation de Windows destinée aux systèmes embarqués. Il a notamment été beaucoup utilisé dans les PC de poche (Pocket PC). Cette version de Windows était optimisée pour les appareils possédant peu d'espace de stockage.

Est arrivée ensuite la gamme de Windows Mobile. Ce système d'exploitation était utilisé sur des smartphones, PDA ou PC de poche. Il est arrivé pour concurrencer les Blackberry et permettait de recevoir ses emails, d'utiliser la suite bureautique, etc. Ces mobiles ressemblaient beaucoup au Windows que l'on connaît, avec son fameux menu démarrer. On utilisait en général un stylet à la place de la souris.

Avec les nouvelles interfaces tactiles, on a vu apparaître Apple et son fameux iPhone venu pour révolutionner la façon dont on se servait jusqu'à présent des appareils mobiles. Ce qui a donné un sérieux coup de vieux à Windows Mobile...



Un Windows CE et un iPhone

Le renouveau : Windows Phone 7

Microsoft a, à son tour, changé radicalement son système d'exploitation afin de prendre ce virage de la mobilité en proposant Windows Phone, dont la première version est la version 7, sortie en octobre 2010. Il ne s'agit pas d'une évolution de Windows Mobile, à part au niveau de la numérotation des versions (dans la mesure où Windows Mobile s'est arrêté avec la version 6.5). Windows Phone 7 a été redéveloppé de zéro et arrive avec un nouveau look, nommé dans un premier temps « Métro », plus épuré, fortement réactif et intuitif, et valorisant l'information structurée.

Microsoft se positionne ainsi en rupture avec ses systèmes d'exploitation précédents et propose des concepts différents pour son nouvel OS, comme la navigation exclusive au doigt par exemple. Il se veut plutôt grand public qu'uniquement destiné aux entreprises.

Pour être autorisé à utiliser le système d'exploitation Windows Phone 7, un smartphone doit respecter un minimum de spécifications. Ces spécifications garantissent qu'une application aura un minimum de puissance, évitant d'avoir des applications trop lentes. L'écran doit être **multipoint d'au moins 3.5 pouces**, c'est-à-dire qu'il doit pouvoir réagir à plusieurs pressions simultanées et permettant une **résolution de 480x800**. Les téléphones doivent également être munis obligatoirement de quelques équipements, comme le fait d'avoir un GPS, d'avoir une caméra, un accéléromètre, etc ... Chaque téléphone possédera également trois boutons faisant partie intégrante de son châssis. Le premier bouton permettra de revenir en arrière, le second d'accéder au menu et le dernier de faire des recherches.



Windows Phone 7

Windows Phone 8

C'est tout dernièrement, avec la sortie de Windows 8, que le système d'exploitation Windows Phone a changé de version pour passer également à la version 8. L'objectif de Microsoft est d'unifier au maximum le cœur de Windows 8 et de Windows Phone 8, permettant de faire facilement des passerelles entre eux. Windows 8 étant un système d'exploitation créé avant tout pour des tablettes, il paraît logique que Windows 8 et Windows Phone 8 partagent beaucoup de fonctionnalités. Windows 8 s'est largement inspiré de Windows Phone pour créer son style, Modern UI, et c'est désormais au tour de Windows Phone de subir une évolution majeure – Windows Phone 8 – afin de se rapprocher de son grand frère, Windows 8.

Beaucoup de choses sont partagées entre les deux systèmes, c'est ce qu'on appelle le « Shared Windows Core ». Ainsi, il deviendra très facile de créer des applications pour Windows Phone 8 qui ne nécessiteront que très peu d'adaptation pour fonctionner sur Windows 8. C'est une des grandes forces de Windows Phone 8.



Notez que les applications Windows Phone 7 fonctionnent également sur Windows Phone 8, c'est une bonne chose si vous possédez déjà des applications Windows Phone 7 et que celles-ci n'ont pas été portées pour WP8.

Windows Phone 8 est également plus performant grâce au support du code natif. Il est ainsi possible de développer des jeux en C++, utilisant DirectX.

Windows Phone 8 apporte en plus des nouvelles résolutions d'écran : WVGA (800x480 pixels), WXVGA (1280x768), et "True 720p" (1280x720), avec une adaptation automatique de chacune.



Windows Phone 8

L'esprit Modern UI

Anciennement appelé « Métro », Modern UI est le nom donné par Microsoft à son langage de design. Plutôt que d'adapter l'interface des anciens Windows Mobile, pour Windows Phone 7 il a été décidé de repartir sur un tout nouveau design.

Le nom « Métro » a été inspiré par les affichages qui se trouvent effectivement dans les stations de métro et qui guident efficacement les voyageurs jusqu'à leurs destinations. Les affichages sont clairs, précis, souvent minimalistes et sans fioritures, par exemple une flèche et un gros 5 pour indiquer que c'est par là qu'on va trouver le métro numéro 5... Voilà à quoi doivent ressembler les interfaces pour Windows Phone. Elles doivent valoriser l'information et la fluidité plutôt que les interfaces lourdes et chargées. Le but est de faire en sorte que l'utilisateur trouve le plus rapidement possible l'information dont il a besoin et d'éviter les images ou animations superflues qui pourraient le ralentir.

Dans cette optique, les applications doivent être fluides et répondre rapidement aux actions de l'utilisateur, ou du moins lui indiquer que son action a été prise en compte. Ras le bol des applications Windows ou autre où on ne sait même plus si on a cliqué sur un bouton car rien ne se passe !



Courant 2012, Métro a changé de nom pour devenir Modern UI. Les applications Windows 8 et Windows Phone 8 doivent chacune suivre les normes édictées par les principes de « Modern UI ».

Modern UI se veut donc simple, épuré et moderne. Les fonctionnalités sont séparées en Hubs qui sont des espaces regroupant des informations de plusieurs sources de données. Ainsi, il existe plusieurs Hubs, comme le Hub « contacts » où l'on retrouvera les contacts du téléphone mais aussi les contacts Facebook, Twitter, ... Nous avons aussi le Hub « Photos », « Musique et vidéos », « Jeux », « Microsoft Office », « Windows Phone Store » ou encore le hub « Entreprise » qui permettra d'accéder aux applications métiers via un portail que les entreprises peuvent personnaliser.



Ecran d'accueil de l'émulateur où l'on voit l'accès aux Hubs et aux

applications

Une fois rentré dans un Hub, nous avons accès à plusieurs informations disposées sous la forme d'un panorama. Nous verrons un peu plus loin ce qu'est le panorama mais je peux déjà vous dire qu'il permet d'afficher des écrans de manière cyclique avec un défilement latéral. Ainsi, dans le Hub de contact, on arrive dans un premier temps sur la liste de tous les contacts. L'écran de panorama que l'on peut faire glisser avec le doigt nous permet d'obtenir sur l'écran suivant la liste des dernières activités de nos contacts, puis la liste des contacts récents, etc. Et c'est pareil pour les autres Hub, le but est d'avoir un point d'entrée qui centralise toutes les informations relatives à un point d'intérêt.

C'est avec tous ces principes en tête que vous devrez développer votre application. N'hésitez pas à observer comment sont faites les autres, on trouve souvent de bonnes sources d'inspirations permettant de voir ce qui fait la qualité du design d'une application.

Le Windows Phone Store

Les applications que nous créons sont ensuite téléchargeables sur la place de marché Windows Phone, appelée encore **Windows Phone Store**. Elles peuvent être gratuites ou payantes, permettant ainsi à son créateur de générer des revenus. Sur le store, on trouvera également des musiques et des vidéos.

Comme nous l'avons déjà dit en introduction, nous allons apprendre à développer pour Windows Phone sans forcément posséder un Windows Phone. C'est un point important, même s'il sera très utile d'en posséder un, vous pouvez tout à fait débuter sans.



Par contre, pour publier une application sur le Windows Phone Store, il faudra posséder un compte développeur. Celui-ci est facturé 19\$ par an, ce qui correspond à 14€.

Voilà, vous savez tout sur Windows Phone, il est temps d'apprendre à réaliser de superbes applications !

- Windows Phone est le nouveau système d'exploitation de Microsoft pour Smartphone.
- Il arrive avec une nouvelle philosophie de design des applications : Modern UI.
- Les applications réalisées sont téléchargeables via le magasin en ligne (store) de Microsoft.

Les outils de développements

Ça a l'air super ça, de pouvoir développer des applications pour les téléphones ! C'est très à la mode et ces mini-ordinateurs nous réservent plein de surprises. Voyons donc ce qu'il nous faut pour nous lancer dans le monde merveilleux du développement pour Windows Phone.

Nous allons apprendre à développer pour Windows Phone équipés de la **version 8**, qui est la dernière version à l'heure où j'écris ces lignes. Vous serez également capables de créer des applications pour Windows Phone 7.5 et 7.8. Même si la version 8 semble très alléchante, les versions 7 ne sont pas à oublier trop rapidement. En effet, tous les utilisateurs n'ont pas encore acheté de Windows Phone 8 et seraient sûrement déçus de manquer votre application révolutionnaire. De plus, Windows Phone 8 sera également capable de faire tourner des applications 7.X. Un marché à ne pas oublier...

Prérequis

Avant toute chose, je vais vous donner les éléments qui vont vous permettre de choisir entre l'environnement de développement dédié au développement d'applications pour Windows Phone 7.5 et celui dédié au développement d'applications pour Windows Phone 7.5 et 8.



Pourquoi choisir entre un environnement qui fait tout et un environnement qui ne fait que la moitié des choses ?

Bonne question... ! En effet, qui peut le plus peut bien sûr le moins. Mais bien que les environnements de développement soient gratuits, vous n'allez peut-être pas avoir envie de changer de machine de développement pour autant.

Voici le matériel requis pour développer pour Windows Phone 7.5 :

- La première chose est de posséder **Windows Vista SP2, ou bien Windows 7 ou encore Windows 8**, qui sont les seules configurations supportées permettant de développer pour Windows Phone 7.5.
- Il est également grandement conseillé de posséder une **carte graphique compatible avec DirectX 10** afin de pouvoir utiliser correctement l'émulateur. Je reviendrai plus tard sur ce qu'est l'émulateur.

La plupart des PC est aujourd'hui équipée de cette configuration. Ce qui est très pratique pour se lancer et découvrir le développement pour Windows Phone. Par contre, pour pouvoir développer pour Windows Phone 8, c'est un peu plus délicat :

- Il vous faut avant tout **Windows 8** en version **64 bits**, rien d'autre. La version conseillée est d'ailleurs la version PRO ou la version Entreprise qui vont permettre d'utiliser l'émulateur.
- Pour faire tourner l'émulateur, il faut que votre processeur supporte la technologie SLAT (qui permet de faire de la virtualisation) et qu'elle soit activée dans le bios ; ce qui est le cas généralement des PC récent (à partir de 2011). Il faut également installer le système de virtualisation de Microsoft, [Hyper-V](#), qui est disponible avec les versions PRO ou Entreprise de Windows 8. Si vous n'êtes pas certain que votre processeur supporte cette technologie, vous pouvez [le vérifier avec cette procédure](#) (en anglais) – ou alors, tentez l'installation et il vous dira une fois que tout est fini s'il y a un problème ou pas.
- Il vous faut aussi une bonne **une carte graphique compatible avec DirectX 10** ainsi que **4 Go de mémoire**.

Ces contraintes matérielles peuvent rendre difficile d'accès le développement pour Windows Phone 8 à quelqu'un qui souhaite simplement s'initier ou découvrir cette technologie. Si c'est votre cas et que vous ne possédez pas ce matériel, alors je vous conseille d'installer l'environnement de développement pour Windows Phone 7.5, qui vous permettra de suivre 95% du cours. J'indiquerai au cours de ce cours ce qui est réservé exclusivement à Windows Phone 8. Sinon, si votre matériel le permet, installez sans hésiter ce qu'il faut pour développer pour Windows Phone 8.



Si vous possédez un téléphone équipé de Windows Phone 8, que vous disposez d'un abonnement de développeur et que vous ne souhaitez pas vous servir de l'émulateur, il est possible de n'utiliser que la version normale de Windows 8.

Ces éléments expliqués, voici la suite des prérequis :

- Bien sûr, vous aurez intérêt à posséder **un smartphone équipé de Windows Phone** : il est primordial de tester son application sur un téléphone avant de songer à la rendre disponible sur le Windows Phone Store.
- Enfin le dernier prérequis est de **savoir parler le C#**.



Le C# est le langage de développement phare de Microsoft et permet la création d'applications informatiques de toutes sortes. Il est indispensable de connaître un peu le langage de programmation C# afin de pouvoir développer des applications pour smartphones équipés de Windows Phone. Son étude n'est pas traitée dans ce cours, mais vous pouvez retrouver un [cours C# complet sur le site OpenClassrooms](#).

Pour résumer ce qu'est le C#, il s'agit d'un langage orienté objet apparu en même temps que le framework .NET qui n'a cessé d'évoluer depuis 2001. Il permet d'utiliser les briques du framework .NET pour réaliser des applications de toutes sortes et notamment des applications pour Windows Phone. C'est le ciment et les outils qui permettent d'assembler les briques de nos applications.

Installer Visual Studio Express pour Windows Phone

Puisqu'on est partis dans le bâtiment, il nous faut un chef de chantier qui va nous permettre d'orchestrer nos développements. C'est ce qu'on appelle l'*IDE*, pour *Integrated Development Environment*, ce qui signifie « Environnement de développement intégré ». Cet outil de développement est un logiciel qui va nous permettre de créer des applications et qui va nous fournir les outils pour orchestrer nos développements. La gamme de Microsoft est riche en outils professionnels de qualité pour le développement, notamment grâce à Visual Studio.

Pour apprendre et commencer à découvrir l'environnement de développement, Microsoft propose gratuitement Visual Studio dans sa version express. C'est une version allégée de l'environnement de développement qui permet de faire plein de choses, mais avec moins d'outils que dans les versions payantes. Rassurez-vous, ces versions gratuites sont très fournies et permettent de faire tout ce dont on a besoin pour apprendre à développer sur Windows Phone et suivre ce cours !

Pour réaliser des applications d'envergure, il pourra cependant être judicieux d'investir dans l'outil complet et ainsi bénéficier de fonctionnalités complémentaires qui permettent d'améliorer, de faciliter et d'industrialiser les développements.

Pour développer pour Windows Phone gratuitement, nous allons avoir besoin de Microsoft Visual Studio Express pour Windows Phone. Pour le télécharger, rendez-vous sur <http://dev.windowsphone.com>. Il est important de noter que les images de ce site changent régulièrement, donc ne vous étonnez pas si celles que vous trouvez en ligne diffèrent légèrement de celles que je vais maintenant vous présenter !



Si vous possédez la version payante de Visual Studio, rendez-vous également sur <http://dev.windowsphone.com> pour télécharger les outils complémentaires et notamment le SDK permettant le développement sur Windows Phone.

Attention, le site est en anglais, mais ne vous inquiétez pas, je vais vous guider. Cliquez ensuite sur Get SDK qui va nous permettre de télécharger les outils gratuits (voir la figure suivante).

The screenshot shows the Windows Phone Dev Center homepage. At the top, there's a navigation bar with links for Design, Develop, Publish, Community, and Dashboard. A search bar is located on the right. Below the navigation, there's a grid of various app icons, including eBay, CNN, and SoundHound. To the right of the grid, a large purple banner with white text reads "The Windows Phone 8 developer platform is here!". At the bottom, there are three calls-to-action: "SUBMIT APP", "GET SDK" (which is highlighted with a red border), and "VIEW SAMPLES". Each action has a brief description below it: "Join Dev Center and publish your app in the Windows Phone Store.", "Download the tools to build great Windows Phone apps.", and "View code samples from Microsoft and the community to get started.".

Obtenir le SDK depuis la page d'accueil du dev center

On arrive sur une nouvelle page où il est indiqué que l'on doit télécharger le « Windows Phone SDK ». SDK signifie *Software Development Kit* que l'on peut traduire par : Kit de développement logiciel. Ce sont tous les outils dont on va avoir besoin pour développer dans une technologie particulière, ici en l'occurrence pour Windows Phone.

On nous propose de télécharger soit la version 8.0 du SDK qui va nous permettre de développer pour Windows Phone 7.5 et 8.0, soit la version 7.1 du SDK qui nous permettra de développer uniquement pour Windows Phone 7.5. La version 7.11 du SDK est une mise à jour de la version 7.1 permettant de développer sous Windows 8. Téléchargez la version qui vous convient en cliquant sur le bouton Download correspondant (voir la figure suivante).



Notez ici que je télécharge et installe la version 8.0 du SDK, les actions sont sensiblement les mêmes pour la version 7.1 du SDK.

The screenshot shows the Windows Phone Dev Center website at <https://dev.windowsphone.com/en-us/downloads>. The main heading is "Windows Phone SDK". On the left, there are three buttons: "SUBMIT APP", "GET SDK", and "VIEW SAMPLES". The "GET SDK" button is highlighted with a red box. Below it, text says "The Windows Phone Software Development Kit (SDK) includes all of the tools that you need to develop apps for Windows Phone." To the right, there are two sections: "SDK 8.0" and "SDK 7.1". The "SDK 8.0" section has a "Download" button with a red box around it, and text indicating it's up to 1.6 GB, English. The "SDK 7.1" section has text about developing apps for Windows Phone 7.0 devices. At the bottom, a link "Télécharger le SDK" is shown.

On arrive sur une nouvelle page où nous allons enfin pouvoir passer en français (voir la figure suivante).

The screenshot shows the Windows Phone SDK 8.0 download page. At the top, there's a red Windows logo followed by the text "Windows Phone SDK 8.0". Below this, a "Quick links" sidebar lists "Overview", "System requirements", and "Instructions". A "Looking for support?" section with a wrench icon and a link to the Microsoft Support site is also present. The main content area is titled "Quick details" and contains a "Version:" dropdown menu. The menu is open, showing options for Chinese (Simplified), Chinese (Traditional), English (which is selected and highlighted in blue), French, German, Italian, Japanese, Korean, Russian, and Spanish. The "File name" field below the dropdown is empty. A call-to-action button at the bottom says "Obtenir le SDK en français".

Quick links

- ↓ Overview
- ↓ System requirements
- ↓ Instructions

Looking for support?

Visit the Microsoft Support site now >

Windows Phone SDK 8.0

Quick details

Version: English

Change language:

Chinese (Simplified)
Chinese (Traditional)
English
French
German
Italian
Japanese
Korean
Russian
Spanish

File name

Obtenir le SDK en français

Une nouvelle page se charge et nous allons pouvoir télécharger l'installateur qui va nous installer tout ce qu'il nous faut. Comme sa taille le suggère, il ne s'agit que d'un petit exécutable qui aura besoin de se connecter à internet pour télécharger tout ce dont il a besoin (voir la figure suivante).

Développement logiciel Windows Phone 8.0



Le Kit de développement logiciel Windows 8.0 vous offre les outils nécessaires pour développer des applications et jeux pour Windows Phone 8 et Windows Phone 7.5.

Résumé

| | | | |
|----------------------|----------|-----------------------|------------|
| Version : | 8.0 | Date de publication : | 26/10/2012 |
| Modifier la langue : | Français | | |

Fichiers contenus dans ce téléchargement

Les liens figurant dans cette section correspondent aux fichiers disponibles pour ce téléchargement. Téléchargez les fichiers qui vous conviennent.

| Nom du fichier | Taille | |
|------------------------------------|--------|-----------------------------|
| fulltrial30\exe\WPexpress_full.exe | 1.0 MB | TÉLÉCHARGER |
| Windows Phone 8 Release Notes.htm | 38 KB | TÉLÉCHARGER |

[Télécharger l'installateur](#)

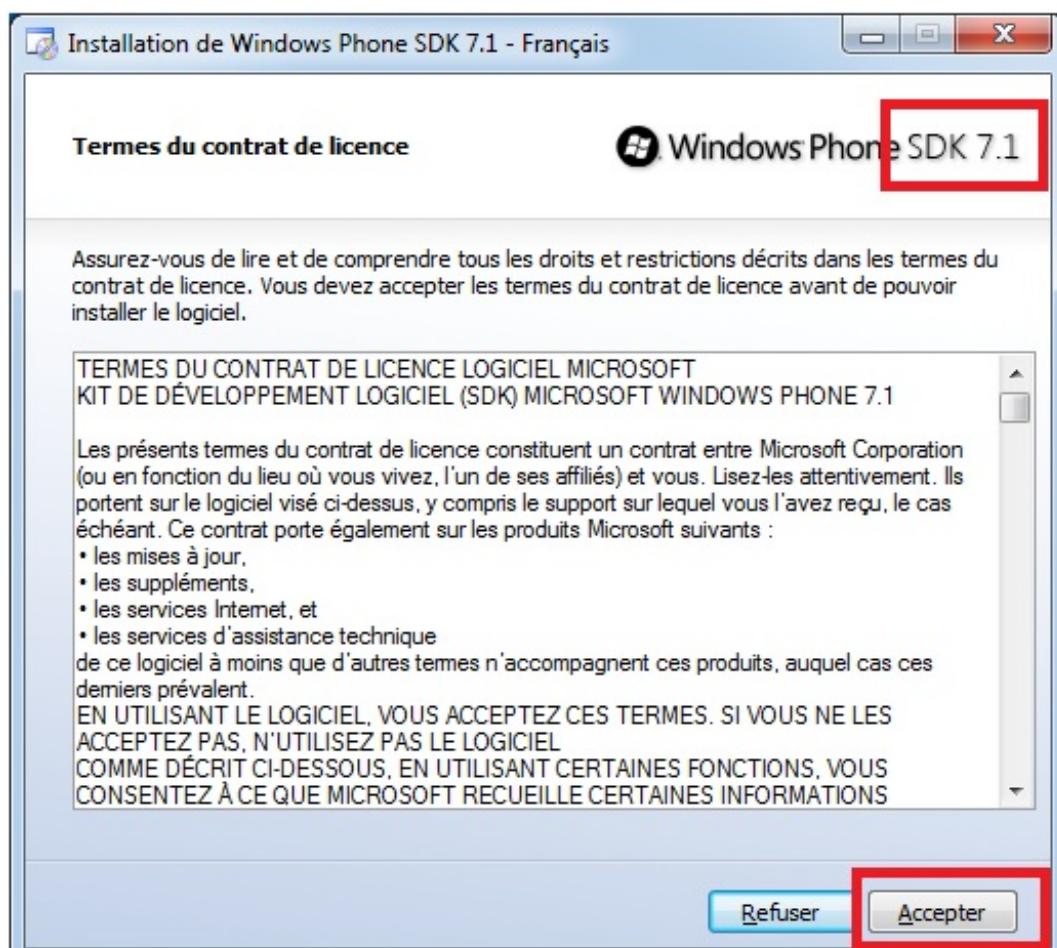
Donc, démarrez le téléchargement et enchaînez tout de suite sur l'installation, tant que vous êtes connectés à internet (voir la figure suivante).



Logo du SDK

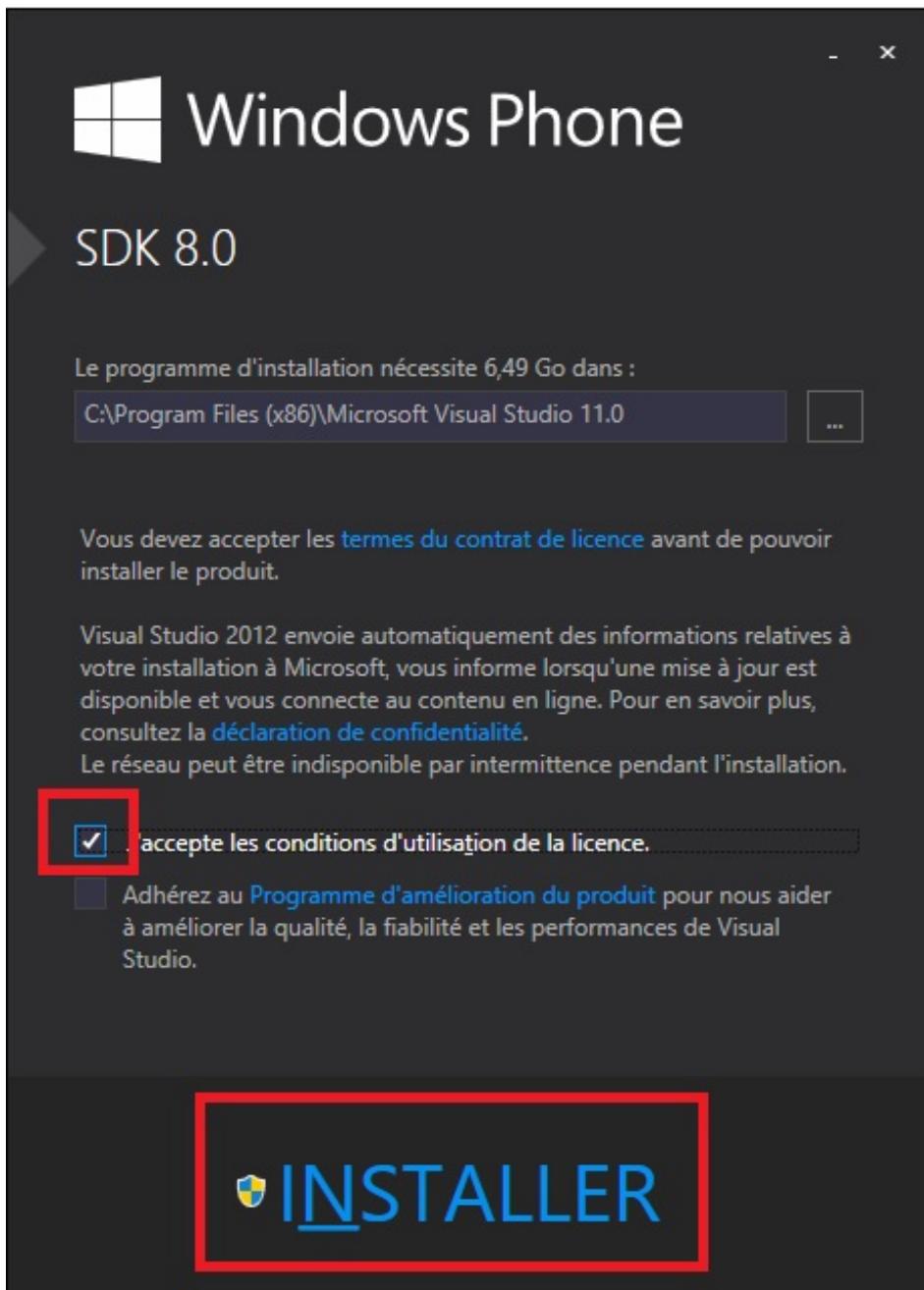
pour Windows Phone 8

L'installation est plutôt classique et commence par l'acceptation du contrat de licence (voir la figure suivante).



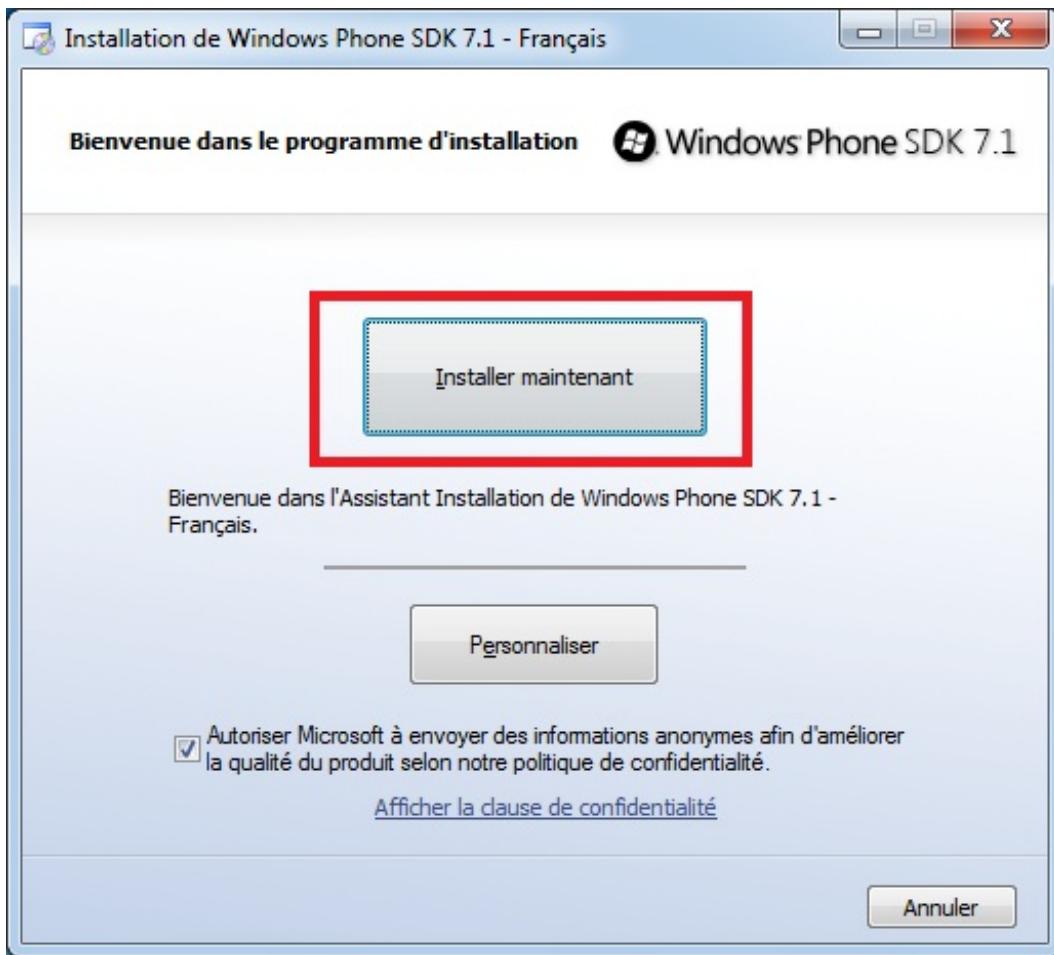
L'acceptation du contrat de

licence pour le SDK 7.1



le SDK 8.0

Pour le SDK 7.1, il y a un écran supplémentaire pour choisir d'installer les outils maintenant (voir la figure suivante).



7.1

L'installation est globalement plutôt longue, surtout sur un PC fraîchement installé. J'espère que vous réussirez à contenir votre impatience !

Enfin, nous arrivons à la fin de l'installation et vous pouvez démarrer Visual Studio.



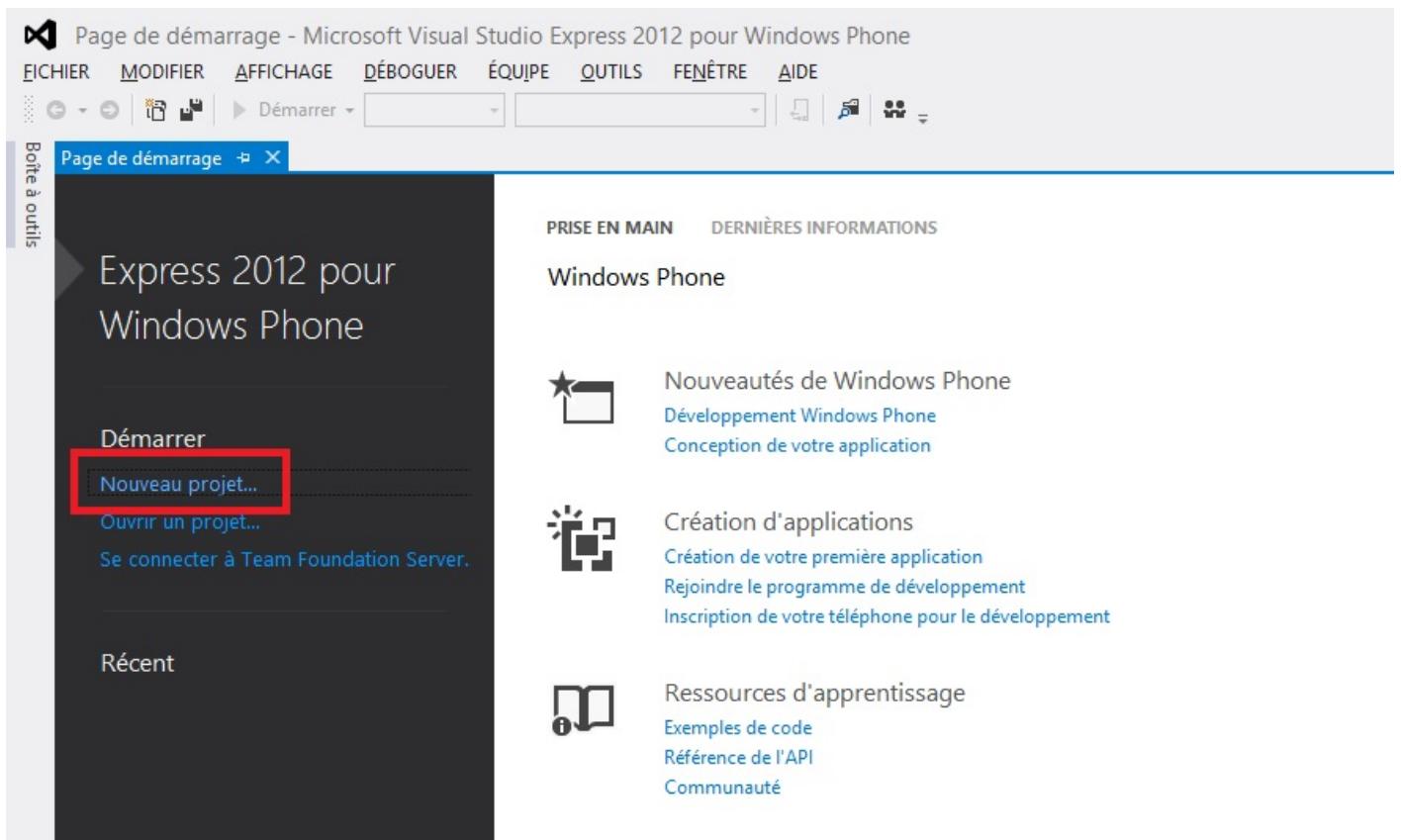
Remarquez que si vous avez installé le SDK 8.0, vous aurez la version 2012 de Visual Studio Express pour Windows Phone alors que si vous avez installé la version 7.1 du SDK, vous aurez la version 2010 de Visual Studio Express pour Windows Phone.

Vous pouvez également démarrer Visual Studio Express pour Windows Phone à partir du Menu Démarrer. Si vous possédez une version payante de Visual Studio, vous pouvez à présent le lancer.



Pour la suite du cours, je me baserai sur la version gratuite de Visual Studio 2012 Express que nous venons d'installer, mais tout ceci sera valable avec la version 2010 ou avec les versions payantes de Visual Studio. Les captures seront sans doute différentes, mais je suppose que vous vous y retrouverez !

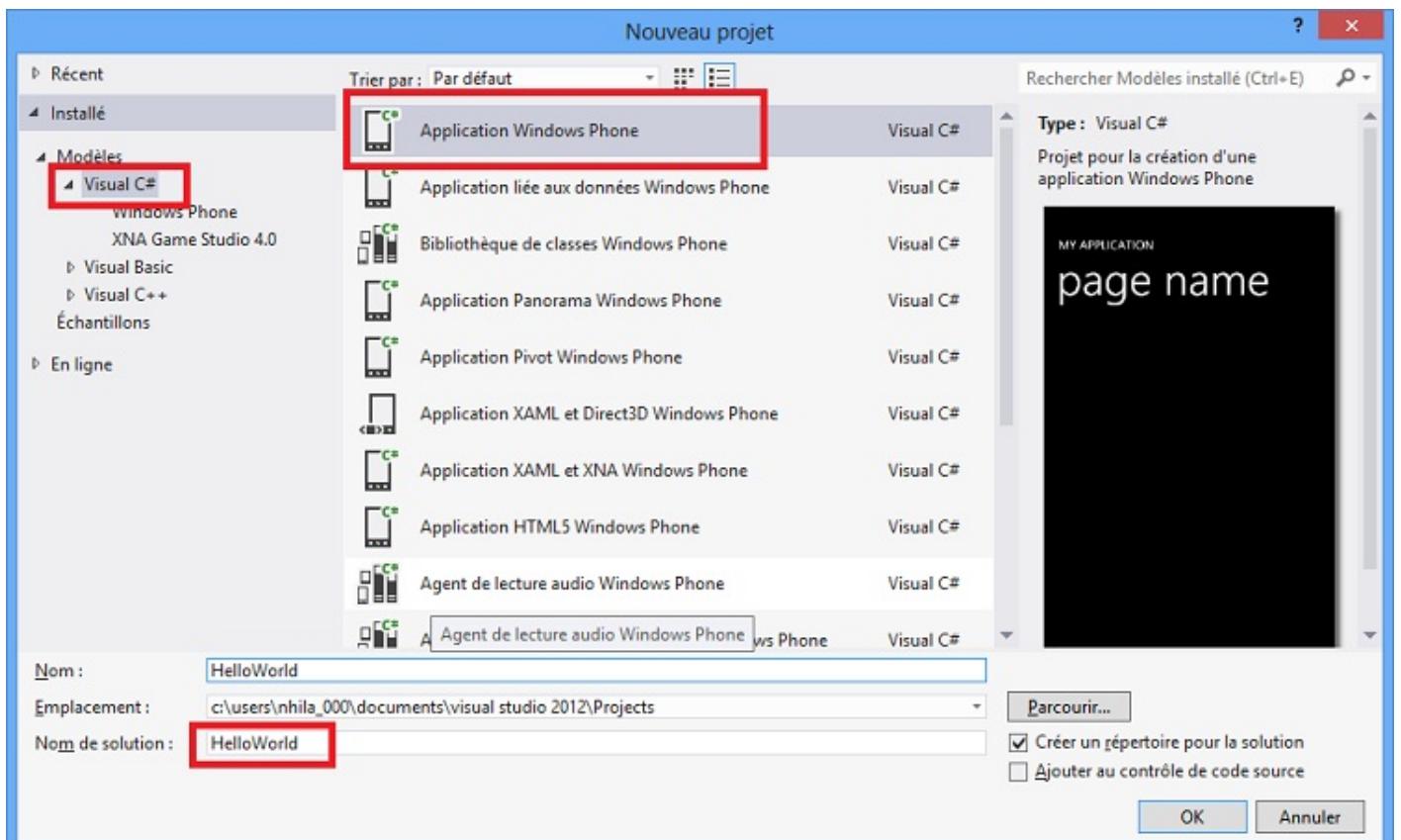
À son ouverture, vous pouvez constater que nous arrivons sur la page de démarrage (voir la figure suivante).



Page de démarrage de Visual Studio permettant de créer un nouveau projet

Nous allons donc créer un nouveau projet en cliquant sur le lien (comme indiqué sur la capture d'écran), ou plus classiquement par le menu Fichier > Nouveau projet.

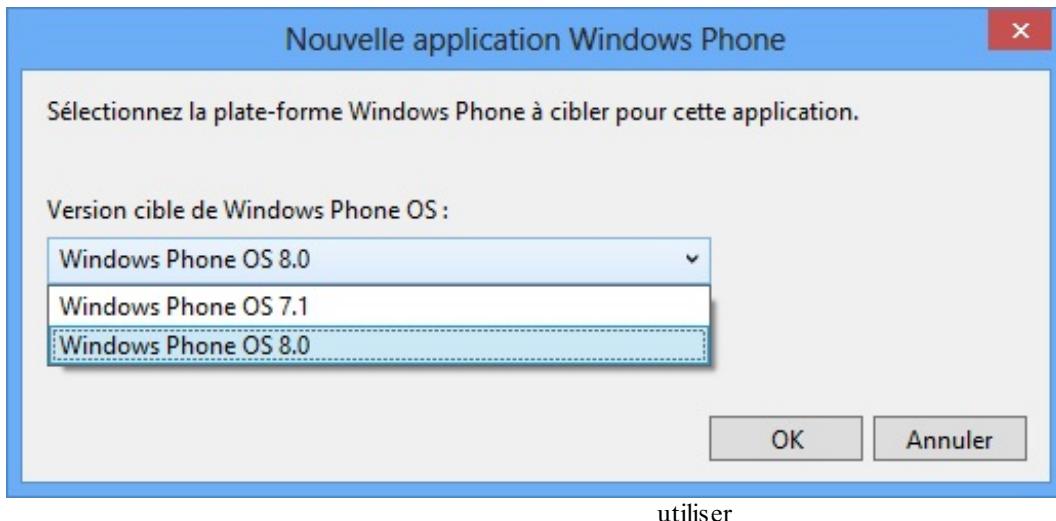
À ce moment-là, Visual Studio Express 2012 pour Windows Phone (que j'appellerai désormais Visual Studio pour économiser mes doigts et les touches de mon clavier) nous ouvre sa page de choix du modèle du projet (voir la figure suivante).



Modèle de projet pour créer une application Windows Phone

Nous allons choisir de créer une Application Windows Phone, qui est la version la plus basique du projet permettant de réaliser une application pour Windows Phone avec le XAML. Remarquons que le choix du langage est possible entre Visual Basic, Visual C++ et Visual C#. Nous choisissons évidemment le C# car c'est le langage que nous maîtrisons. J'en profite pour nommer mon projet « HelloWorld »... (ici, personne ne se doute quel type d'application nous allons faire très bientôt).

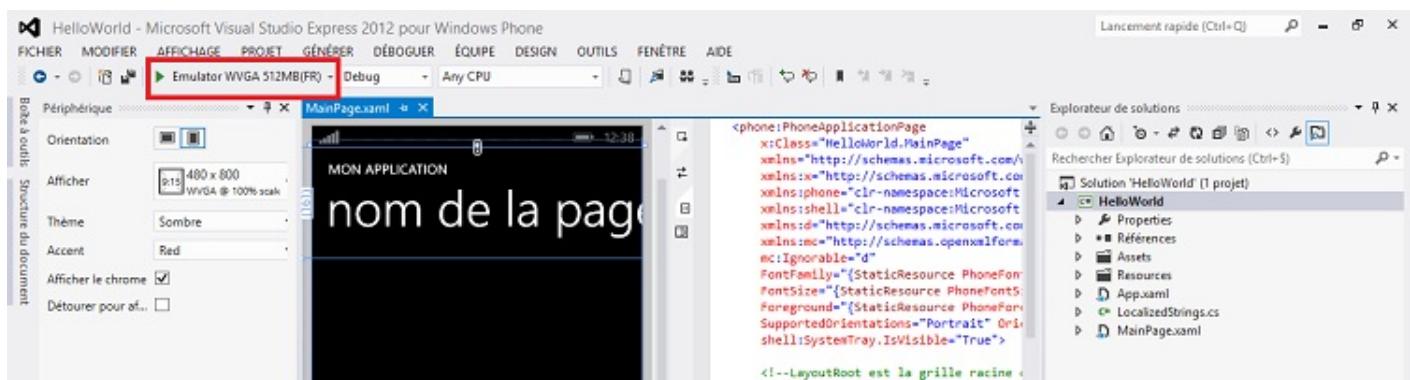
Enfin, après avoir validé la création du projet, il nous demande la version à cibler (voir la figure suivante).



Choix de la version du SDK à utiliser

Choisissez « 8.0 » pour développer pour Windows Phone 8 ou 7.1 pour développer pour Windows Phone 7.5. Rappelez-vous qu'une application 7.5 sera exécutable sur les téléphones équipés de Windows Phone 8 (c'est l'avantage) mais ne pourra pas utiliser les nouveautés de Windows Phone 8 (c'est l'inconvénient).

Visual Studio génère son projet, les fichiers qui le composent et s'ouvre sur la page suivante (voir la figure suivante).



L'interface de Visual Studio, ainsi que la liste déroulante permettant de choisir l'émulateur

Nous allons revenir sur cet écran très bientôt. Ce qu'il est important de remarquer c'est que si nous démarrons l'application telle quelle, elle va se compiler et s'exécuter dans l'émulateur Windows Phone. Vous le voyez dans le petit encadré en haut de Visual Studio, c'est la cible du déploiement. Il est possible de déployer soit sur l'émulateur, soit directement sur un téléphone relié au poste de travail. Il ne reste plus qu'à réellement exécuter notre application...

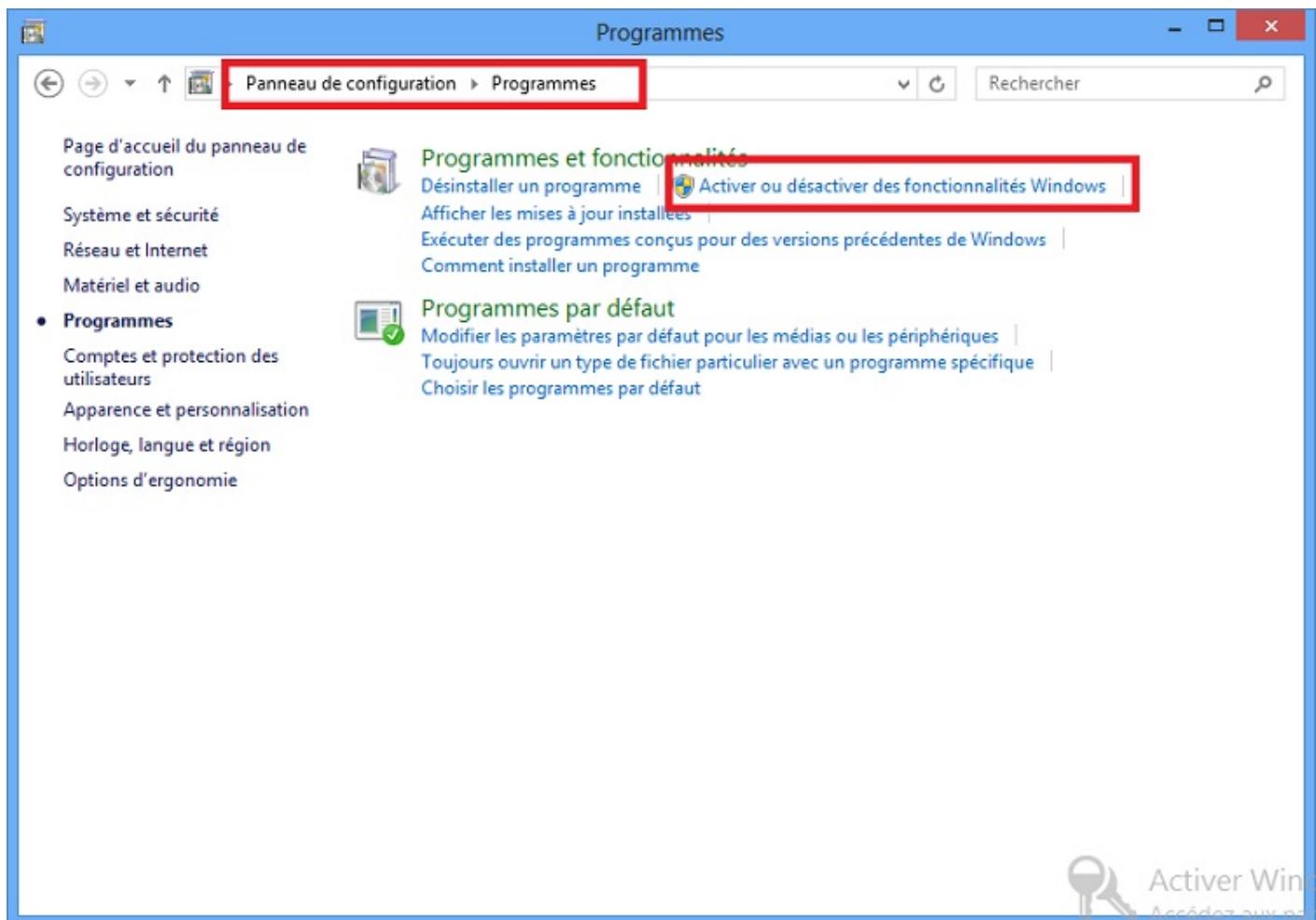


Il ne sera pas possible de déployer notre application sur le téléphone sans une manipulation préalable que nous découvrirons dans le dernier chapitre. Si vous n'avez pas les prérequis matériels pour installer l'émulateur et que vous possédez un Windows Phone, vous pouvez d'ores et déjà vous reporter à ce chapitre afin de pouvoir continuer à suivre le cours.

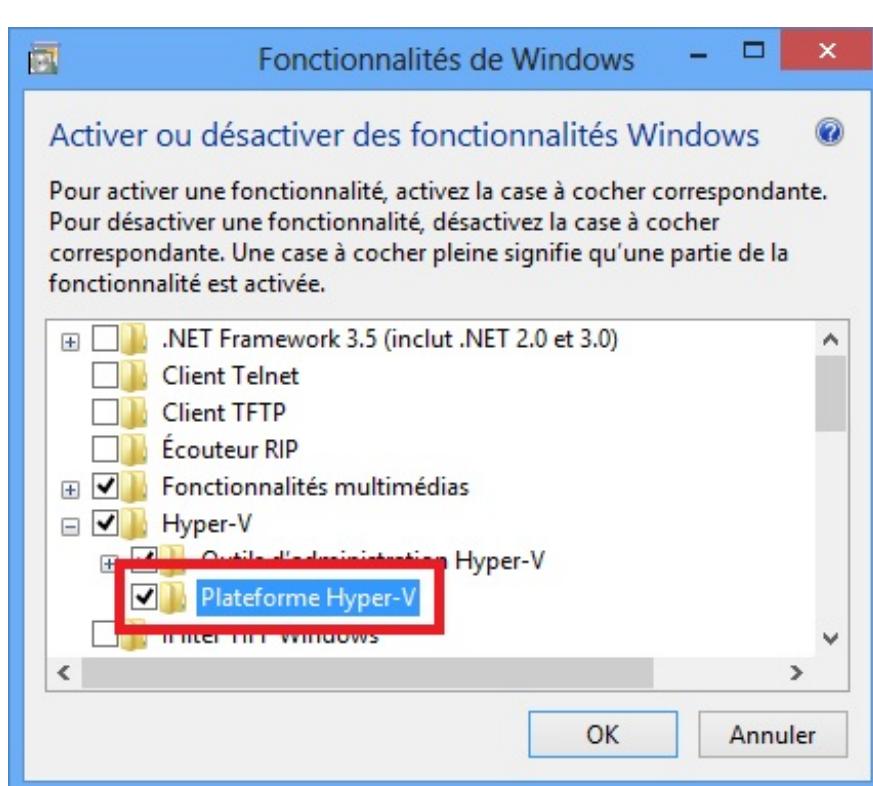
L'émulateur

Attention, si vous avez installé le SDK 8.0, vous allez avoir besoin également d'installer le logiciel gestion de la virtualisation : Hyper-v. Celui-ci n'est disponible qu'avec les versions PRO ou Entreprise de Windows 8 et uniquement si votre processeur supporte la technologie SLAT.

Allez dans le panneau de configuration, programmes, et choisissez d'activer des fonctionnalités Windows (voir la figure suivante).



Puis, installez hyper-V en cochant la case correspondante (voir la figure suivante).

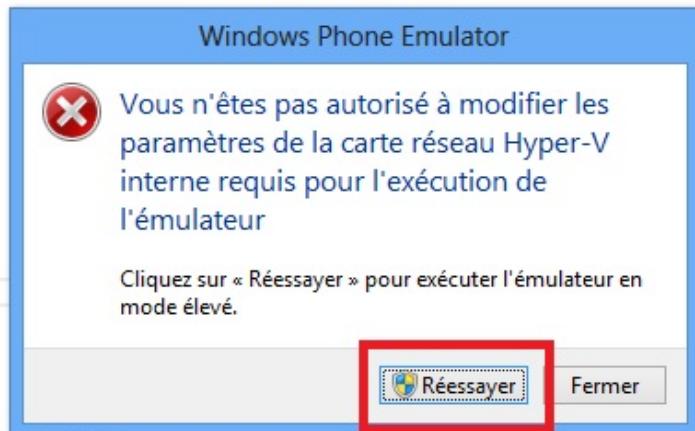


Exécutons donc notre application en appuyant sur F5 qui nous permet de démarrer l'application en utilisant le débogueur.



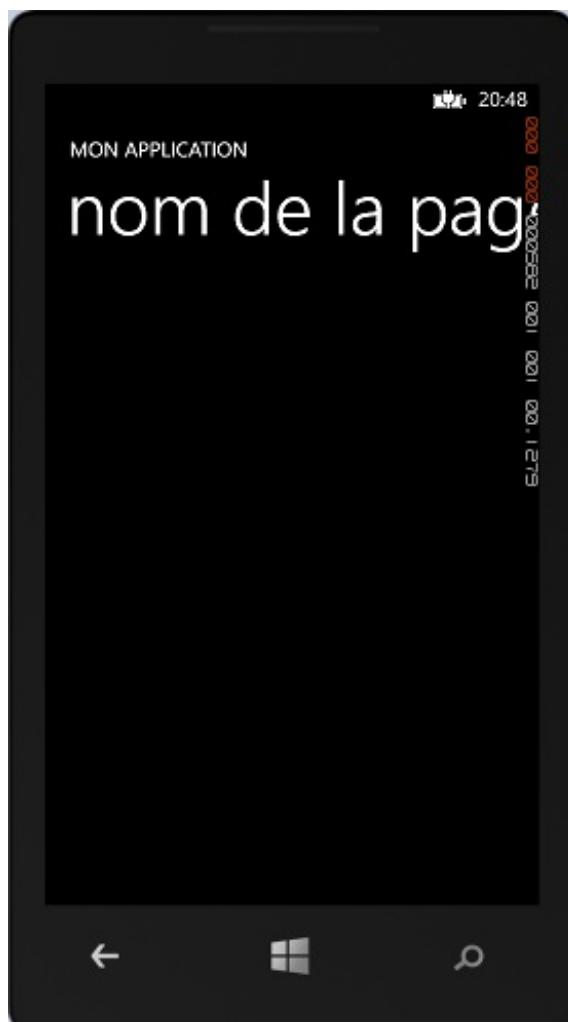
Vous aurez peut-être la fenêtre d'élevation des privilèges (voir figure suivante) qui permet de configurer l'émulateur. En ce cas, cliquez sur Réessayer.

```
s;  
s.Controls;  
s.Navigation;  
ne.Controls;  
ne.Shell;  
ources;  
  
class MainPage : PhoneApplicationPage  
  
teur  
Page()  
  
izeComponent();  
  
ple de code pour la localisation d'Applicationbar  
LocalizedAoalicationBar():
```



Démarrage de l'émulateur avec élévation de privilège

Nous constatons que l'émulateur se lance, il ressemble à un téléphone Windows Phone... On les reconnaît d'un coup d'œil car ils ont les trois boutons en bas du téléphone, la flèche (ou retour arrière), le bouton d'accès au menu et la loupe pour faire des recherches (voir la figure suivante).



Emulateur Windows Phone 8

L'émulateur possède également des boutons en haut à droite qui permettent (de haut en bas) de :

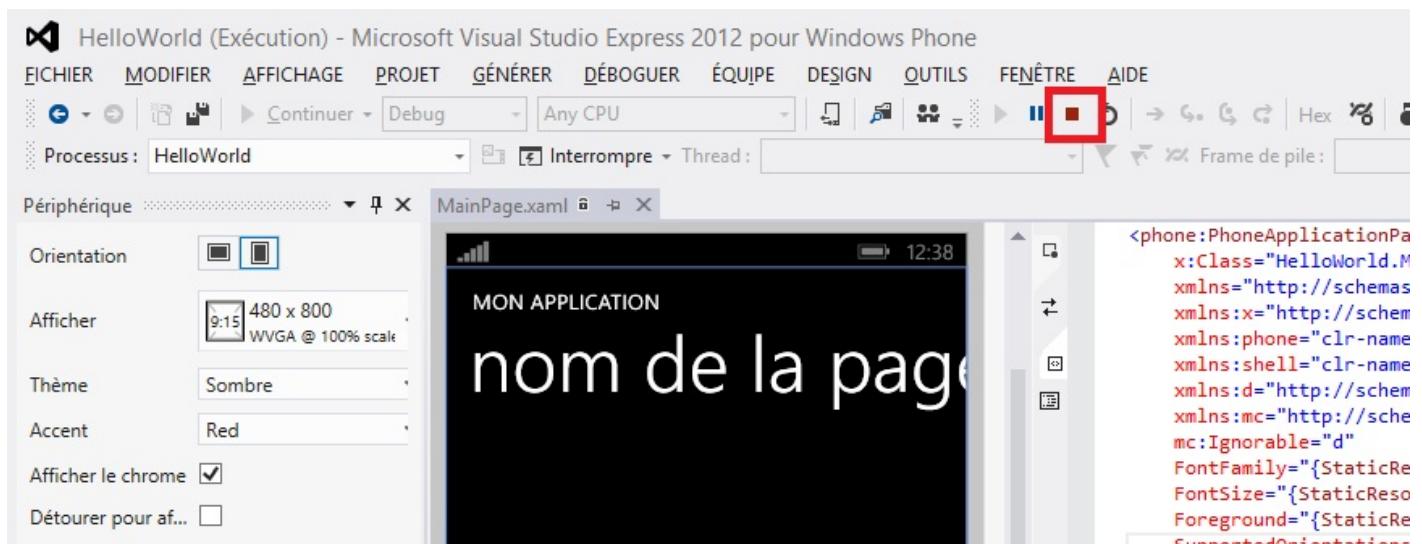
- Fermer l'émulateur
- Minimiser l'émulateur
- Faire pivoter de 90° vers la gauche l'émulateur
- Faire pivoter de 90° vers la droite l'émulateur
- Adapter à la résolution
- Zoomer/dézoomer sur l'émulateur
- Ouvrir les outils supplémentaires



Boutons permettant de faire des actions sur l'émulateur

Remarquons également que des chiffres s'affichent sur le côté droit de l'émulateur. Ce sont des informations sur les performances de l'application, nous y reviendrons en fin de cours.

Enfin, vous pouvez fermer l'application en arrêtant le débogueur en cliquant sur le carré (voir la figure suivante).



Bouton pour arrêter le débogage dans Visual Studio

XAML et code behind

Revenons un peu sur cette page que nous a affiché Visual Studio. Nous pouvons voir que le milieu ressemble à l'émulateur et que le côté droit ressemble à un fichier XML.

Vous ne connaissez pas les fichiers XML ? Si vous voulez en savoir plus, n'hésitez pas à faire un petit tour sur internet, c'est un format très utilisé dans l'informatique !

Pour faire court, le fichier XML est un langage de balise, un peu comme le HTML, où l'on décrit de l'information. Les balises sont des valeurs entourées de < et > qui décrivent la sémantique de la donnée. Par exemple :

Code : XML

```
<prenom>Nicolas</prenom>
```

La balise <prenom> est ce qu'on appelle une balise ouvrante, cela signifie que ce qui se trouve après (en l'occurrence la chaîne « Nicolas ») fait partie de cette balise jusqu'à ce que l'on rencontre la balise fermante </prenom> qui est comme la balise ouvrante à l'exception du / précédent le nom de la balise.

Le XML est un fichier facile à lire par nous autres humains. On en déduit assez facilement que le fichier contient la chaîne « Nicolas » et qu'il s'agit sémantiquement d'un prénom.

Une balise peut contenir des attributs permettant de donner des informations sur la donnée. Les attributs sont entourés de guillemets " et " et font partie de la balise. Par exemple :

Code : XML

```
<client nom="Nicolas" age="30"></client>
```

Ici, la balise client possède un attribut « nom » ayant la valeur « Nicolas » et un attribut « age » ayant la valeur « 30 ». Encore une fois, c'est très facile à lire pour un humain.

Il est possible que la balise n'ait pas de valeur, comme c'est le cas dans l'exemple ci-dessus. On peut dans ce cas-là remplacer la balise ouvrante et la balise fermante par cet équivalent :

Code : XML

```
<client nom="Nicolas" age="30"/>
```

Enfin, et nous allons terminer notre aperçu rapide du XML avec un dernier point. Il est important de noter que le XML peut imbriquer ses balises et qu'il ne peut posséder qu'un seul élément racine, ce qui nous permet d'avoir une hiérarchie de données. Par exemple nous pourrons avoir :

Code : XML

```
<listesDesClients>
  <client type="Particulier">
    <nom>Nicolas</nom>
    <age>30</age>
  </client>
  <client type="Professionnel">
    <nom>Jérémie</nom>
    <age>40</age>
  </client>
</listesDesClients>
```

On voit tout de suite que le fichier décrit une liste de deux clients. Nous en avons un qui est un particulier, qui s'appelle Nicolas et qui a 30 ans alors que l'autre est un professionnel, prénommé Jérémie et qui a 40 ans.

A quoi cela nous sert-il ? A comprendre ce fameux fichier de droite. C'est le **fichier XAML** (prononcez « Zammel »). Le XAML

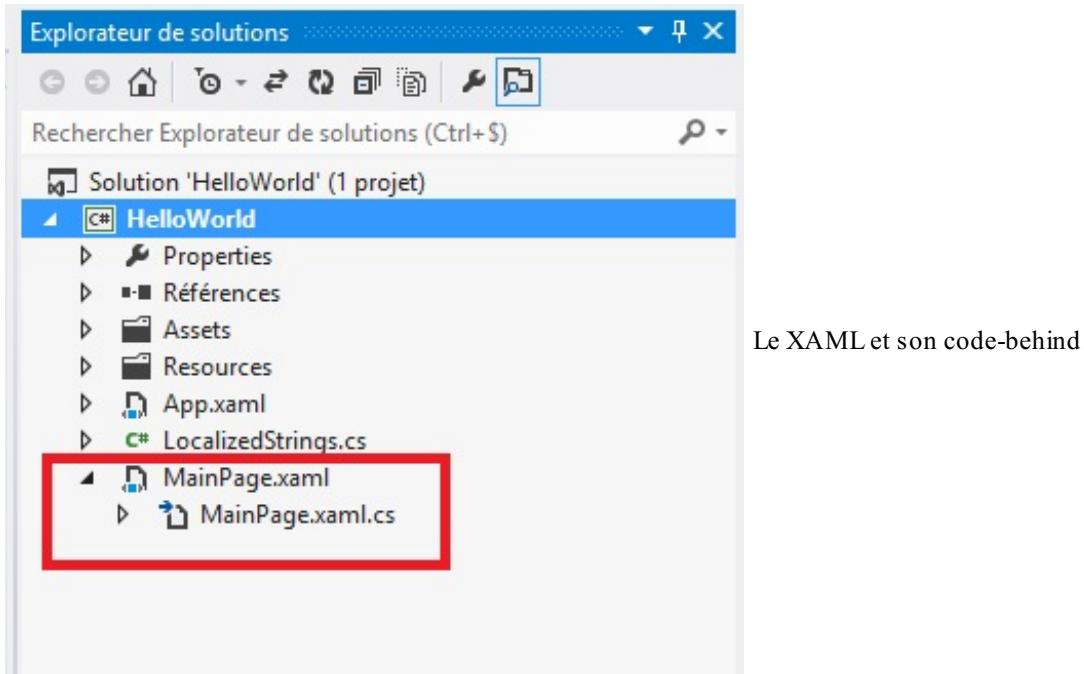
est un langage qui permet de décrire des interfaces et en l'occurrence ici le code XAML (à droite dans Visual Studio) décrit l'interface que nous retrouvons au milieu. Cette zone est la prévisualisation du rendu du code écrit dans la partie droite. On appelle la zone du milieu, le **concepteur** (ou plus souvent le designer en anglais). En fait, le fichier XAML contient des balises qui décrivent ce qui doit s'afficher sur l'écran du téléphone. Nous allons y revenir.

Nous allons donc devoir apprendre un nouveau langage pour pouvoir créer des applications sur Windows Phone : le XAML. Ne vous inquiétez pas, il est assez facile à apprendre et des outils vont nous permettre de simplifier notre apprentissage.



Ok pour le XAML si tu dis que ce n'est pas trop compliqué, mais le C# dans tout ça ?

Eh bien il arrive dans le fichier du même nom que le fichier XAML et il est suffixé par .cs. Nous le retrouvons si nous cliquons sur le petit triangle à côté du fichier XAML qui permet de déplier les fichiers (voir la figure suivante).



Il est juste en dessous, on l'appelle le **code behind**. Le code behind est le code C# qui est associé à l'écran qui va s'afficher à partir du code XAML. Il permet de gérer toute la logique associée au XAML.

Si vous ouvrez ce fichier C#, vous pouvez voir quelques instructions déjà créées en même temps que le XAML. Nous allons également y revenir.



Dans la suite de ce cours, les extraits de code XAML seront indiqués par le site OpenClassrooms comme du XML. C'est seulement pour les besoins de la coloration syntaxique, mais assurez-vous, il s'agira bien de XAML.

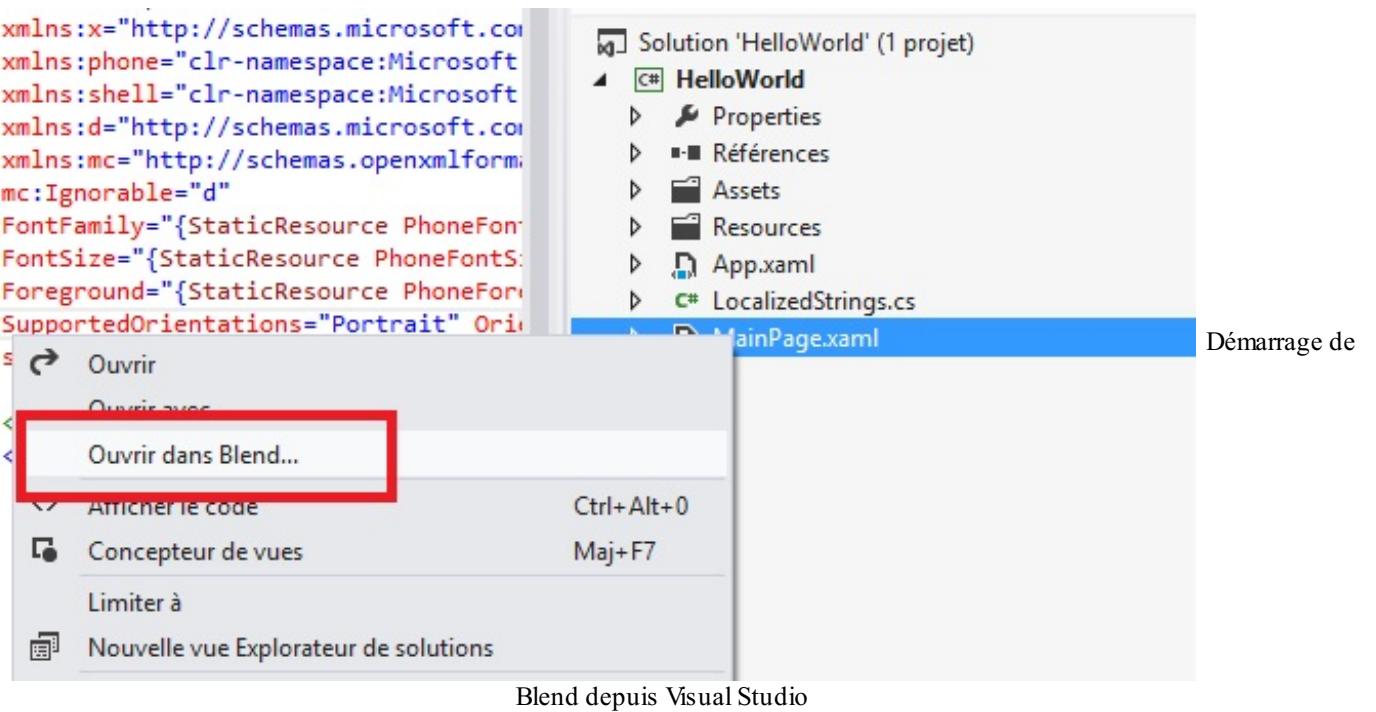
Blend pour le design

Afin de faciliter la réalisation de jolis écrans à destination du téléphone, nous pouvons modifier le XAML. C'est un point que nous verrons plus en détail un peu plus loin. Il est possible de le modifier directement à la main lorsqu'on connaît la syntaxe, mais nous avons aussi à notre disposition un outil dédié au design qui le fait tout seul : Blend.

Microsoft Expression Blend est un outil professionnel de conception d'interfaces utilisateur de Microsoft. Une version gratuite pour Windows Phone a été installée en même temps que Visual Studio Express 2012 pour Windows Phone et permet de travailler sur le design de nos écrans XAML.

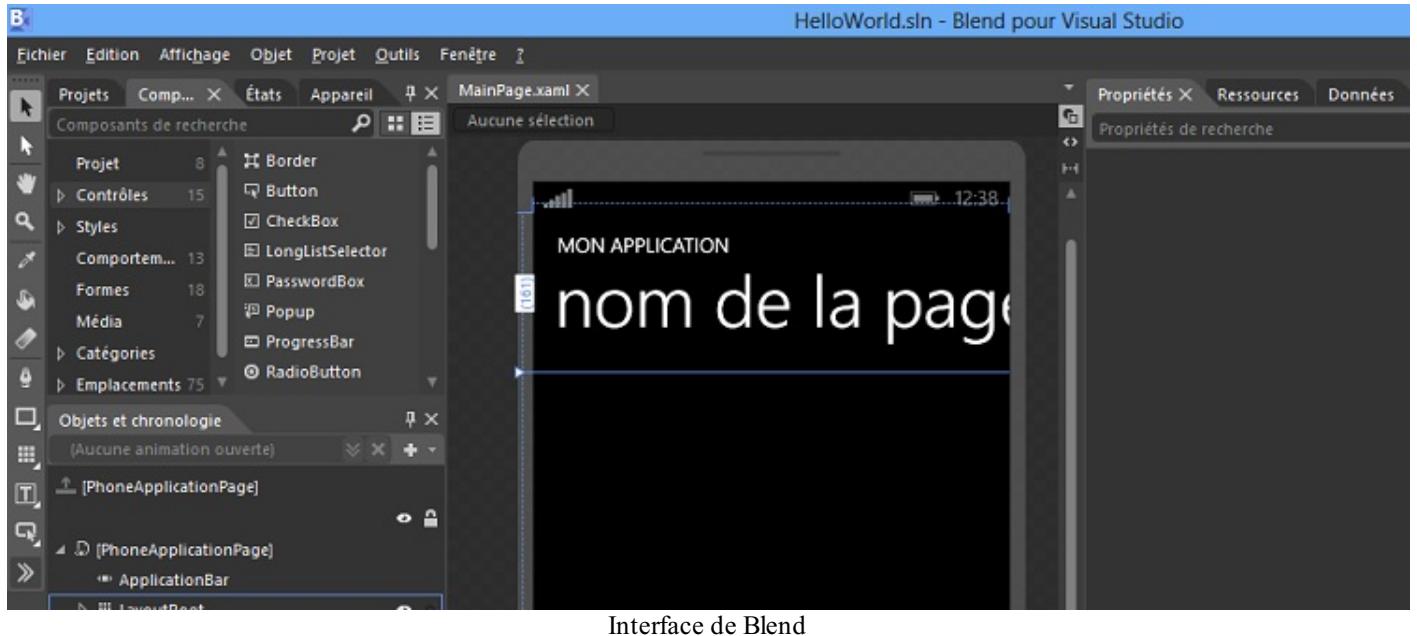
Nous verrons comment il fonctionne mais nous ne nous attarderons pas trop sur son fonctionnement car il mériterait un cours entier.

Ce qui est intéressant c'est qu'il est possible de travailler en même temps sur le même projet dans Visual Studio et dans Blend, les modifications se répercutant de l'un à l'autre. Faisons un clic droit sur le fichier XAML et choisissons de l'ouvrir dans Expression Blend (voir la figure suivante).



Blend depuis Visual Studio

Blend s'ouvre alors et affiche à nouveau le rendu de notre écran (voir la figure suivante).



Interface de Blend

On peut voir également une grosse boîte à outils qui va nous permettre de styler notre application. Nous y reviendrons.

- Pour développer pour Windows Phone, nous avons besoin de Visual Studio et du kit de développement pour Windows Phone.
- Il existe une version totalement gratuite de Visual Studio permettant de réaliser des applications sous Windows Phone.
- Un émulateur accompagne Visual Studio pour tester ses applications en mode développement.
- Blend est l'outil de design permettant de styler son application, dont une version gratuite est fournie avec le kit de développement pour Windows Phone.

Notre première application

Nous avons désormais tous les outils qu'il faut pour commencer à apprendre à réaliser des applications pour Windows Phone. Nous avons pu voir que ces outils fonctionnent et nous avons pu constater un début de résultat grâce à l'émulateur. Mais pour bien comprendre et assimiler toutes les notions, nous avons besoin de plus de concret, de manipuler des éléments et de voir qu'est-ce qui exactement agit sur quoi. Aussi, il est temps de voir comment réaliser une première application avec le classique « Hello World » ! 😊

Cette première application va nous servir de base pour commencer à découvrir ce qu'il faut pour réaliser des applications pour Windows Phone.

Hello World

Revenons donc sur notre écran où nous avons le designer et le XAML. Positionnons-nous sur le code XAML et ajoutons des éléments sans trop comprendre ce que nous allons faire afin de réaliser notre « Hello World ». Nous reviendrons ensuite sur ce que nous avons fait pour expliquer le tout en détail.

Pour commencer, on va modifier la ligne suivante :

Code : XML

```
<TextBlock Text="nom de la page" Margin="9,-7,0,0"  
Style="{StaticResource PhoneTextTitle1Style}"/>
```

et écrire ceci :

Code : XML

```
<TextBlock Text="Hello World" Margin="9,-7,0,0"  
Style="{StaticResource PhoneTextTitle1Style}"/>
```

Nous changeons donc la valeur de l'attribut `Text` de la balise `<TextBlock>`.



J'emploie ici du vocabulaire XML, mais ne le retenez pas car ce n'est pas exactement de ça qu'il s'agit. Nous y reviendrons.

Ensuite, juste après :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```

Donc, à l'intérieur de cette balise `<Grid>`, rajoutez :

Code : XML

```
<StackPanel>  
    <TextBlock Text="Saisir votre nom" HorizontalAlignment="Center"  
/>  
    <TextBox x:Name="Nom" />  
    <Button Content="Valider" Tap="Button_Tap_1" />  
    <TextBlock x:Name="Resultat" Foreground="Red" />  
</StackPanel>
```

Remarquez que lorsque vous avez saisi la ligne :

Code : XML

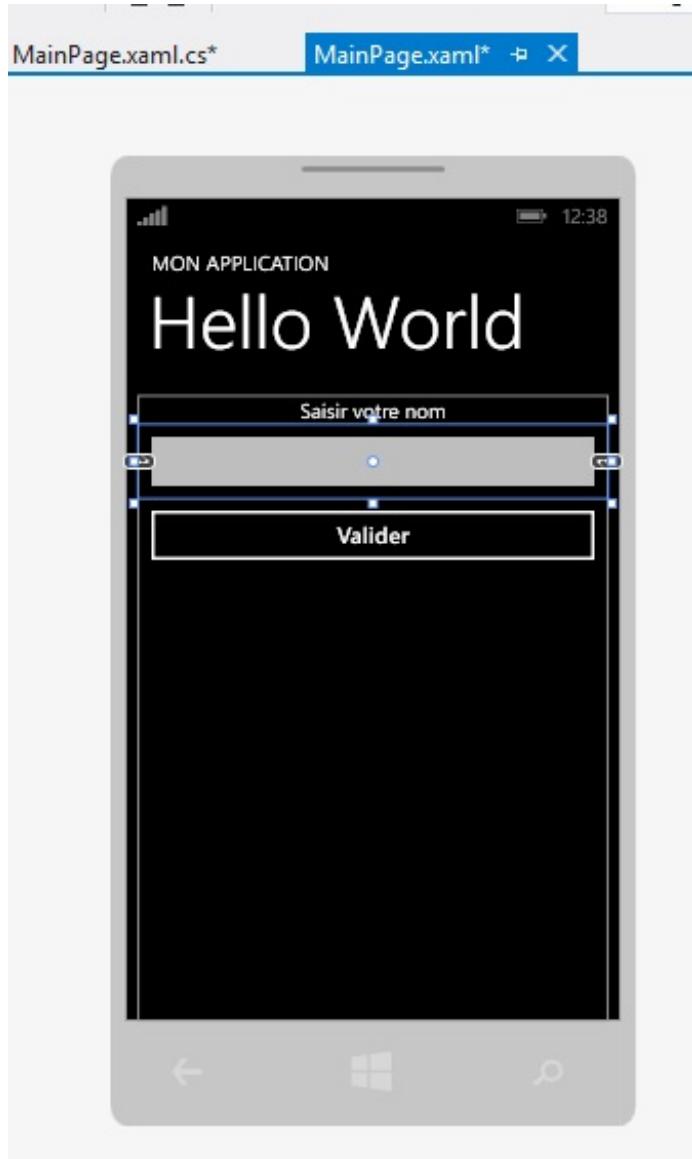
```
<Button Content="Valider" Tap="Button_Tap_1" />
```

au moment de taper : `Tap=""`, Visual Studio Express vous a proposé de générer un nouveau gestionnaire d'événement (voir la figure suivante).



Génération du gestionnaire d'événement depuis le XAML

Vous pouvez le générer en appuyant sur Entrée, cela nous fera gagner du temps plus tard. Sinon, ce n'est pas grave. Vous pouvez constater que le designer s'est mis à jour pour faire apparaître nos modifications (voir la figure suivante).



Le rendu du XAML dans la fenêtre du concepteur

Ouvrez maintenant le fichier de code behind et modifiez la méthode Button_Tap_1 pour avoir :

Code : C#

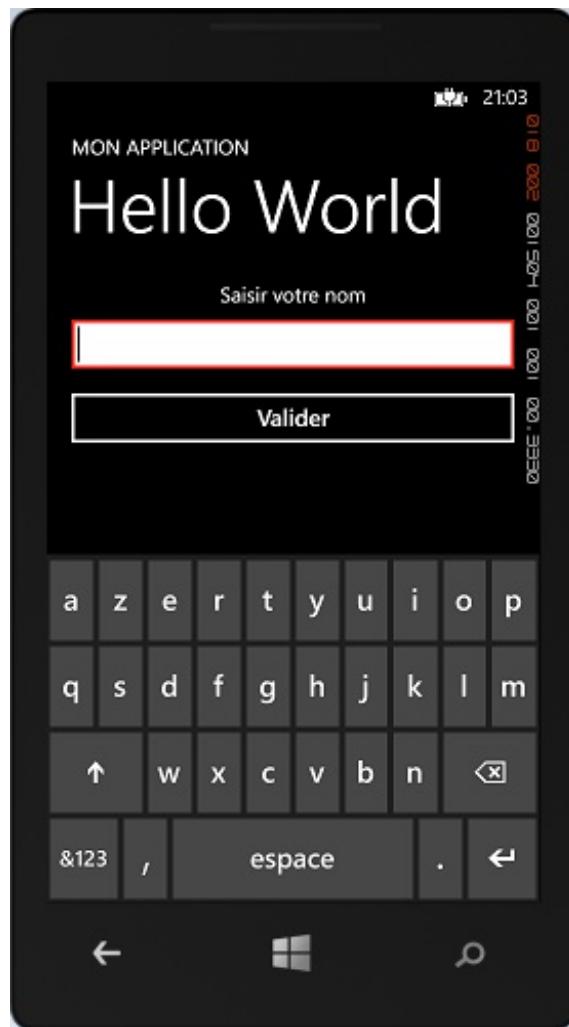
```
private void Button_Tap_1(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    Resultat.Text = "Bonjour " + Nom.Text;  
}
```

Nous pouvons dès à présent démarrer notre application en appuyant sur F5. L'émulateur se lance et nous voyons apparaître les nouvelles informations sur l'écran (voir la figure suivante).



Rendu de l'application dans l'émulateur

La souris va ici permettre de simuler le « toucher » avec le doigt lorsque nous cliquons. Cliquons donc dans la zone de texte et nous voyons apparaître un clavier virtuel à l'intérieur de notre application (voir la figure suivante).



Le clavier virtuel dans l'émulateur

Ce clavier virtuel s'appelle le SIP (pour *Soft Input Panel*) et apparaît automatiquement quand on en a besoin, notamment dans les zones de saisie, nous y reviendrons. Saisissons une valeur dans la zone de texte et cliquons sur le bouton Valider. Nous voyons apparaître le résultat en rouge avec un magnifique message construit (voir la figure suivante).



Affichage de l'Hello World dans l'émulateur

Et voilà, notre Hello World est terminé ! Chouette non ?

Pour quitter l'application, le plus simple est d'arrêter le débogueur de Visual Studio en cliquant sur le carré Stop.

L'interface en XAML

Alors, taper des choses sans rien comprendre, ça va un moment... mais là, il est temps de savoir ce que nous avons fait !

Nous avons dans un premier temps fait des choses dans le XAML. Pour rappel, le XAML sert à décrire le contenu de ce que nous allons voir à l'écran. En fait, un fichier XAML correspond à une page. Une application peut être découpée en plusieurs pages, nous y reviendrons plus tard. Ce que nous avons vu sur l'émulateur est l'affichage de la page MainPage.

Donc, nous avons utilisé le XAML pour décrire le contenu de la page. Il est globalement assez explicite mais ce qu'il faut comprendre c'est que nous avons ajouté des contrôles du framework .NET dans la page. Un contrôle est une classe C# complète qui sait s'afficher, se positionner, traiter des événements de l'utilisateur (comme le clic sur le bouton), etc. Ces contrôles ont des propriétés et peuvent être ajoutés dans le XAML.

Par exemple, prenons la ligne suivante :

Code : XML

```
<TextBlock Text="Saisir votre nom" HorizontalAlignment="Center" />
```

Nous demandons d'ajouter dans la page le contrôle **TextBlock**. Le contrôle **TextBlock** correspond à une zone de texte non modifiable. Nous positionnons sa propriété **Text** à la chaîne de caractères "Saisir votre nom". Ce contrôle sera aligné au centre grâce à la propriété **HorizontalAlignment** positionnée à **Center**. En fait, j'ai dit que nous l'ajoutons dans la page, mais pour être plus précis, nous l'ajoutons dans le contrôle **StackPanel** qui est lui-même contenu dans le contrôle **Grid**, qui est contenu dans la page. Nous verrons plus loin ce que sont ces contrôles.



Ce que nous avons appelé « balise » plus tôt est en fait un contrôle, et ce que nous avons appelé « attribut »



correspond à une propriété de ce contrôle.

Derrière, automatiquement, cette ligne de XAML entraîne la déclaration et l'instanciation d'un objet de type `TextBlock` avec les affectations de propriétés adéquates. Puis ce contrôle est ajouté dans le contrôle `StackPanel`. Tout ceci nous est masqué. Grâce au XAML nous avons simplement décrit l'interface de la page et c'est Visual Studio qui s'est occupé de le transformer en C#.

Parfait ! Moins on en fait et mieux on se porte... et surtout il y a moins de risque d'erreurs.

Et c'est pareil pour tous les autres contrôles de la page, le `TextBlock` qui est une zone de texte non modifiable, le `TextBox` qui est une zone de texte modifiable déclenchant l'affichage du clavier virtuel, le bouton, etc.

Vous l'aurez peut-être deviné, mais c'est pareil pour la page. Elle est déclarée tout en haut du fichier XAML :

Code : XML

```
<phone:PhoneApplicationPage
    x:Class="HelloWorld.MainPage"

    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
    namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
    namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
    compatibility/2006"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    mc:Ignorable="d" d:DesignHeight="768" d:DesignWidth="480"
    shell:SystemTray.IsVisible="True">
```

C'est d'ailleurs le conteneur de base du fichier XAML, celui qui contient tous les autres contrôles. La page est en fait représentée par la classe `PhoneApplicationPage` qui est aussi un objet du framework .NET. Plus précisément, notre page est une classe générée qui dérive de l'objet `PhoneApplicationPage`. Il s'agit de la class `MainPage` située dans l'espace de nom `HelloWorld`, c'est ce que l'on voit dans la propriété :

Code : XML

```
x:Class="HelloWorld.MainPage"
```

On peut s'en rendre compte également dans le code behind de la page où Visual Studio a généré une classe partielle du même nom que le fichier XAML et qui dérive de `PhoneApplicationPage` :

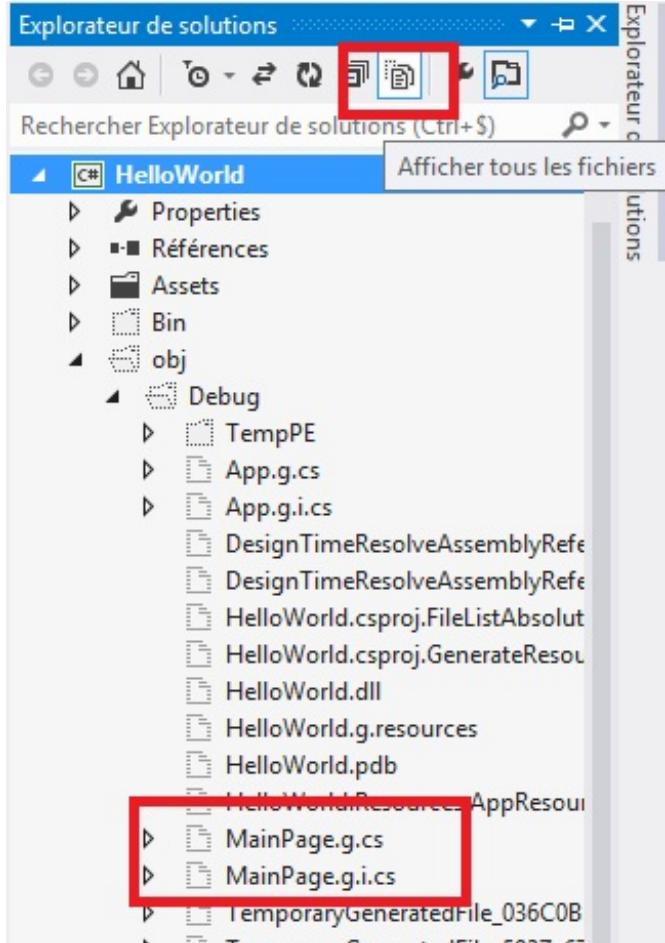
Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    // Constructeur
    public MainPage()
    {
        InitializeComponent();
    }

    private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Resultat.Text = "Bonjour " + Nom.Text;
```

```
    }  
}
```

Pourquoi partielle ? Parce qu'il existe un autre fichier dans votre projet. Ce fichier est caché mais on peut l'afficher en cliquant sur le bouton en haut de l'explorateur de solution (voir la figure suivante).



Affichage des fichiers cachés dans l'explorateur de solutions

Et nous pouvons voir notamment un répertoire obj contenant un répertoire debug contenant le fichier MainPage.g.i.cs. Si vous l'ouvrez, vous pouvez trouver le code suivant :

Code : C#

```
public partial class MainPage :  
    Microsoft.Phone.Controls.PhoneApplicationPage  
{  
    internal System.Windows.Controls.Grid LayoutRoot;  
    internal System.Windows.Controls.StackPanel TitlePanel;  
    internal System.Windows.Controls.Grid ContentPanel;  
    internal System.Windows.Controls.TextBox Nom;  
    internal System.Windows.Controls.TextBlock Resultat;  
  
    private bool _contentLoaded;  
  
    /// <summary>  
    /// InitializeComponent  
    /// </summary>  
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]  
    public void InitializeComponent()  
    {  
        if (_contentLoaded)  
        {  
            return;  
        }
```

```
        }
        _contentLoaded = true;
        System.Windows.Application.LoadComponent(this, new
System.Uri("/HelloWorld;component/MainPage.xaml",
System.UriKind.Relative));
        this.LayoutRoot =
((System.Windows.Controls.Grid)(this.FindName("LayoutRoot")));
        this.TitlePanel =
((System.Windows.Controls.StackPanel)(this.FindName("TitlePanel")));
        this.ContentPanel =
((System.Windows.Controls.Grid)(this.FindName("ContentPanel")));
        this.Nom =
((System.Windows.Controls.TextBox)(this.FindName("Nom")));
        this.Resultat =
((System.Windows.Controls.TextBlock)(this.FindName("Resultat")));
    }
}
```

Il s'agit d'une classe qui est générée lorsqu'on modifie le fichier XAML. Ne modifiez pas ce fichier car il sera re-généré tout le temps. On peut voir qu'il s'agit d'une classe MainPage, du même nom que la propriété `x:Class` de tout à l'heure, qui s'occupe de charger le fichier XAML et qui crée des variables à partir des contrôles qu'il trouvera dedans. Nous voyons notamment qu'il a créé les deux variables suivantes :

Code : C#

```
internal System.Windows.Controls.TextBox Nom;
internal System.Windows.Controls.TextBlock Resultat;
```

Le nom de ces variables correspond aux propriétés `x:Name` des deux contrôles que nous avons créé :

Code : XML

```
<TextBox x:Name="Nom" />
<TextBlock x:Name="Resultat" Foreground="Red" />
```

Ces variables sont initialisées après qu'il ait chargé tout le XAML en faisant une recherche à partir du nom du contrôle. Cela veut dire que nous disposons d'une variable qui permet d'accéder au contrôle de la page, par exemple la variable `Nom` du type `TextBox`. Je vais y revenir.

Nous avons donc :

- Un fichier `MainPage.xaml` qui contient la description des contrôles
- Un fichier généré qui contient une classe partielle qui dérive de `PhoneApplicationPage` et qui charge ce XAML et qui rend accessible nos contrôles via des variables
- Un fichier de code behind qui contient la même classe partielle où nous pourrons écrire la logique de notre code

Remarquez qu'il existe également le fichier `MainPage.g.cs` qui correspond au fichier généré après la compilation. Nous ne nous occuperons plus de ces fichiers générés, ils ne servent plus à rien. Nous les avons regardés pour comprendre comment cela fonctionne.

Le code-behind en C#

Revenons sur le code behind, donc sur le fichier `MainPage.xaml.cs`, nous avons :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
```

```
// Constructeur
public MainPage()
{
    InitializeComponent();
}

private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Resultat.Text = "Bonjour " + Nom.Text;
}
```

On retrouve bien notre classe partielle qui hérite des fonctionnalités de la classe `PhoneApplicationPage`. Regardez à l'intérieur de la méthode `Button_Tap_1`, nous utilisons les fameuses variables que nous n'avons pas déclaré nous-même mais qui ont été générées... Ce sont ces variables qui nous permettent de manipuler nos contrôles et en l'occurrence ici, qui nous permettent de modifier la valeur de la zone de texte non modifiable en concaténant la chaîne « Bonjour » à la valeur de la zone de texte modifiable, accessible via sa propriété `Text`.

Vous aurez compris ici que ce sont les propriétés `Text` des `TextBlock` et `TextBox` qui nous permettent d'accéder au contenu qui est affiché sur la page. Il existe plein d'autres propriétés pour ces contrôles comme la propriété `Foreground` qui permet de modifier la couleur du contrôle, sauf qu'ici nous l'avions positionné grâce au XAML :

Code : XML

```
<TextBlock x:Name="Resultat" Foreground="Red" />
```

Chose que nous aurions également pu faire depuis le code behind :

Code : C#

```
private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Resultat.Foreground = new SolidColorBrush(Colors.Red);
    Resultat.Text = "Bonjour " + Nom.Text;
}
```

Sachez quand même que d'une manière générale, on aura tendance à essayer de mettre le plus de chose possible dans le XAML plutôt que dans le code behind. La propriété `Foreground` ici a tout intérêt à être déclarée dans le XAML.

Le contrôle Grid

Je vais y revenir plus loin un peu plus loin, mais pour que vous ne soyiez pas complètement perdu dans notre Hello World, il faut savoir que la `Grid` est un conteneur.



<*mode bilingue = on*> Vous aurez compris que la `Grid` est en fait une grille...<*mode bilingue = off*>

Après cet effort de traduction intense, nous pouvons dire que la grille sert à contenir et à agencer d'autres contrôles. Dans notre cas, le code suivant :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock Text="MON APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
    <TextBlock Text="Hello World" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock Text="Saisir votre nom" HorizontalAlignment="Center" />
        <TextBox x:Name="Nom" />
        <Button Content="Valider" Tap="Button_Tap_1" />
        <TextBlock x:Name="Resultat" Foreground="Red" />
    </StackPanel>
</Grid>
</Grid>
```

Défini une grille qui contient deux lignes. La première contient un contrôle StackPanel, nous allons en parler juste après. La seconde ligne contient une nouvelle grille sans ligne ni colonne, qui est également composée d'un StackPanel. Nous aurons l'occasion d'en parler plus longuement plus tard donc je m'arrête là pour l'instant sur la grille.

Le contrôle StackPanel

Ici c'est pareil, le contrôle StackPanel est également un conteneur. Je vais y revenir un peu plus loin également mais il permet ici d'aligner les contrôles les uns en dessous des autres. Par exemple, celui que nous avons rajouté contient un TextBlock, un TextBox, un bouton et un autre TextBlock :

Code : XML

```
<StackPanel>
    <TextBlock Text="Saisir votre nom" HorizontalAlignment="Center" />
    <TextBox x:Name="Nom" />
    <Button Content="Valider" Tap="Button_Tap_1" />
    <TextBlock x:Name="Resultat" Foreground="Red" />
</StackPanel>
```

Nous pouvons voir sur le designer que les contrôles sont bien les uns en dessous des autres.

Nous avons donc au final, la page qui contient une grille, qui contient un StackPanel et une grille qui contiennent chacun des contrôles.

Le contrôle TextBox

Le contrôle TextBox est une zone de texte modifiable. Nous l'avons utilisée pour saisir le prénom de l'utilisateur. On déclare ce contrôle ainsi :

Code : XML

```
<TextBox x:Name="Nom" />
```

Lorsque nous cliquons dans la zone de texte, le clavier virtuel apparaît et nous offre la possibilité de saisir une valeur. Nous verrons un peu plus loin qu'il est possible de changer le type du clavier virtuel.

La valeur saisie est récupérée via la propriété Text du contrôle, par exemple ici je récupère la valeur saisie que je concatène à la chaîne Bonjour et que je stocke dans la variable resultat :

Code : C#

```
string resultat = "Bonjour " + Nom.Text;
```

Inversement, je peux pré-remplir la zone de texte avec une valeur en utilisant la propriété `Text`, par exemple depuis le XAML :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBox x:Name="Nom" Text="Nicolas" />
    </StackPanel>
</Grid>
```

Ce qui donne la figure suivante.



La valeur du TextBox s'affiche dans la fenêtre de rendu

La même chose est faisable en code behind, il suffit d'initialiser la propriété de la variable dans le constructeur de la page :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
        Nom.Text = "Nicolas";
    }
}
```

Évidemment, il sera toujours possible de modifier la valeur pré-remplie grâce au clavier virtuel.

Le contrôle TextBlock

Le contrôle TextBlock représente une zone de texte non modifiable. Nous l'avons utilisé pour afficher le résultat de notre Hello World. Il suffit d'utiliser sa propriété Text pour afficher un texte. Par exemple, le XAML suivant :

Code : XML

```
<TextBlock Text="Je suis un texte non modifiable de couleur rouge"  
Foreground="Red" FontSize="25" />
```

affiche la fenêtre de prévisualisation présentée dans la figure suivante.



Le texte s'affiche en rouge dans le TextBlock

Je peux modifier la couleur du texte grâce à la propriété Foreground. C'est la même chose pour la taille du texte, modifiable via la propriété FontSize. Nous pouvons remarquer que le texte que j'ai saisi dépasse de l'écran et que nous ne le voyons pas en entier. Pour corriger ça, j'utilise la propriété TextWrapping que je positionne à Wrap :

Code : XML

```
<TextBlock Text="Je suis un texte non modifiable de couleur rouge"  
Foreground="Red" FontSize="25" TextWrapping="Wrap" />
```

Comme nous l'avons déjà fait, il est possible de modifier la valeur d'un TextBlock en passant par le code-behind :

Code : C#

```
Resultat.Text = "Bonjour " + Nom.Text;
```

Les événements

Il s'agit des événements sur les contrôles. Chaque contrôle est capable de lever une série d'événements lorsque cela est opportun. C'est le cas par exemple du contrôle bouton qui est capable de lever un événement lorsque nous tapotons dessus (ou que nous cliquons avec la souris). Nous l'avons vu dans l'exemple du Hello World, il suffit de déclarer une méthode que l'on associe à l'événement, par exemple :

Code : XML

```
<Button Content="Valider" Tap="Button_Tap_1" />
```

qui permet de faire en sorte que la méthode Button_Tap_1 soit appelée lors du clic sur le bouton. Rappelez-vous, dans notre Hello World, nous avions la méthode suivante :

Code : C#

```
private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Resultat.Text = "Bonjour " + Nom.Text;
}
```

Il est également possible de s'abonner à un événement via le code behind, il suffit d'avoir une variable de type bouton, pour cela donnons un nom à un bouton :

Code : XML

```
<Button x:Name="UnBouton" Content="Cliquez-moi" />
```

Et d'associer une méthode à l'événement de clic :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        UnBouton.Tap += UnBouton_Tap;
    }

    void UnBouton_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        throw new NotImplementedException();
    }
}
```

Il existe beaucoup d'événements de ce genre, par exemple la case à cocher (CheckBox) permet de s'abonner à l'événement qui est déclenché lorsqu'on coche la case :

Code : XML

```
<CheckBox Content="Cochez-moi" Checked="CheckBox_Checked_1" />
```

Avec la méthode :

Code : C#

```
private void CheckBox_Checked_1(object sender, RoutedEventArgs e)
{
}
```

Il existe énormément d'événement sur les contrôles, mais aussi sur la page, citons encore par exemple l'événement qui permet d'être notifié lors de la fin du chargement de la page :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    throw new NotImplementedException();
}
```

Nous aurons l'occasion de voir beaucoup d'autres événements tout au long de ce cours.

Remarquez que les événements sont toujours construits de la même façon. Le premier paramètre est du type `object` et représente le contrôle qui a déclenché l'événement. En l'occurrence, dans l'exemple suivant :

Code : XML

```
<Button Content="Valider" Tap="Button_Tap_1" />
```

Nous pouvons accéder au contrôle de cette façon :

Code : C#

```
private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Button bouton = (Button)sender;
    bouton.Content = "C'est validé !";
}
```

Le second paramètre est, quant à lui, spécifique au type d'événement et peut fournir des informations complémentaires.

Le bouton

Revenons à présent rapidement sur le bouton, nous l'avons vu il n'est pas très compliqué à utiliser. On utilise la propriété Content pour mettre du texte et il est capable de lever un événement lorsqu'on clique dessus, grâce à l'événement Tap. Le bouton possède également un événement Click qui fait la même chose et qui existe encore pour des raisons de compatibilité avec Silverlight.



Préférez cependant l'événement Tap qui est plus performant.

Il est également possible de passer des paramètres à un bouton. Pour un bouton tout seul, ce n'est pas toujours utile, mais dans certaines situations cela peut être primordial.

Dans l'exemple qui suit, j'utilise deux boutons qui ont la même méthode pour traiter l'événement de clic sur le bouton :

Code : XML

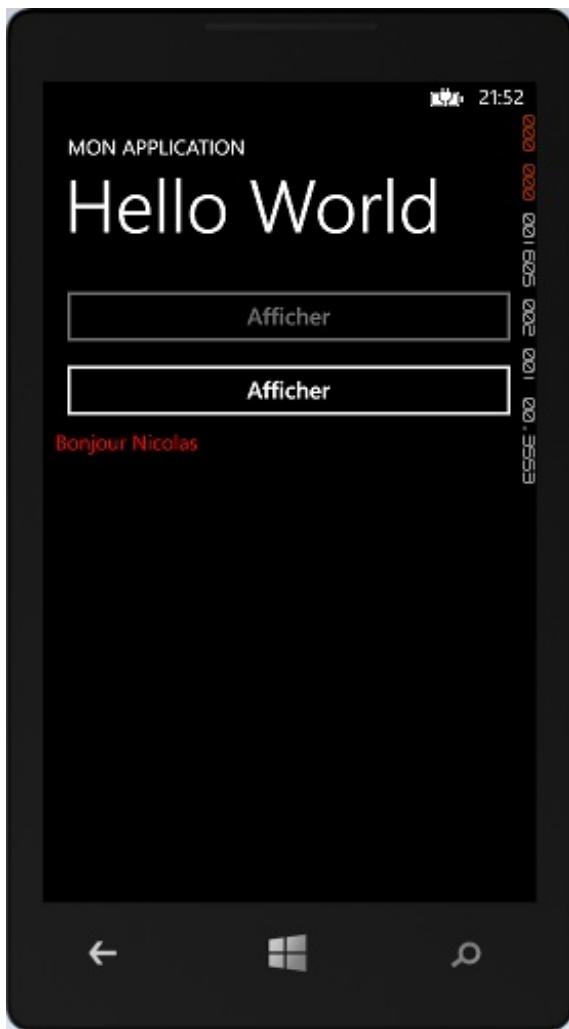
```
<StackPanel>
    <Button Content="Afficher" Tap="Button_Tap"
        CommandParameter="Nicolas" />
    <Button Content="Afficher" Tap="Button_Tap"
        CommandParameter="Jérémie" />
    <TextBlock x:Name="Resultat" Foreground="Red" />
</StackPanel>
```

C'est la propriété CommandParameter qui me permet de passer un paramètre. Je pourrais ensuite l'utiliser dans mon code behind :

Code : C#

```
private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Button bouton = (Button)sender;
    bouton.IsEnabled = false;
    Resultat.Text = "Bonjour " + bouton.CommandParameter;
}
```

J'utilise ainsi le paramètre CommandParameter pour récupérer le prénom de la personne à qui dire bonjour (voir la figure suivante).



Passage d'un paramètre au bouton s'affichant dans l'émulateur

Remarquez au passage l'utilisation de la propriété `IsEnabled` qui permet d'indiquer si un contrôle est activé ou pas. Si un bouton est désactivé, il ne pourra pas être cliqué.

Et Silverlight dans tout ça ?

Vous avez remarqué que j'ai parlé de Silverlight et de XAML. Quelle différence ?

Pour bien comprendre, Silverlight était utilisé pour développer avec les versions 7 de Windows Phone. On utilise par contre le XAML/C# pour développer pour la version 8. En fait, grossièrement c'est la même chose.

XAML est l'évolution de Silverlight. Si vous avez des connaissances en Silverlight, vous vous êtes bien rendu compte que ce qu'on appelle aujourd'hui XAML/C#, c'est pareil.

Il s'agit juste d'un changement de vocabulaire afin d'unifier les développements utilisant du code XAML pour définir l'interface d'une application, qu'elle soit Windows Phone ou Windows ...

Ce qui est valable avec Silverlight l'est aussi avec XAML/C#, et inversement proportionnel.

- Le XAML permet de décrire l'interface de nos pages.
- Le code behind permet d'écrire le code C# de la logique de nos pages.
- On utilise des contrôles dans nos interfaces, comme le bouton ou la zone de texte.
- Les contrôles sont des classes complètes qui savent s'afficher, se positionner ou réagir à des événements utilisateurs, comme le clic sur un bouton.

Les contrôles

Jusqu'à présent nous avons vu peu de contrôles, la zone de texte non modifiable, la zone de saisie modifiable, le bouton... Il existe beaucoup de contrôles disponibles dans les bibliothèques de contrôles XAML pour Windows Phone. Ceux-ci sont facilement visibles grâce à la boîte à outils que l'on retrouve à gauche de l'écran.

Voyons un peu ce qu'il y a autour de ces fameux contrôles.

Généralités sur les contrôles

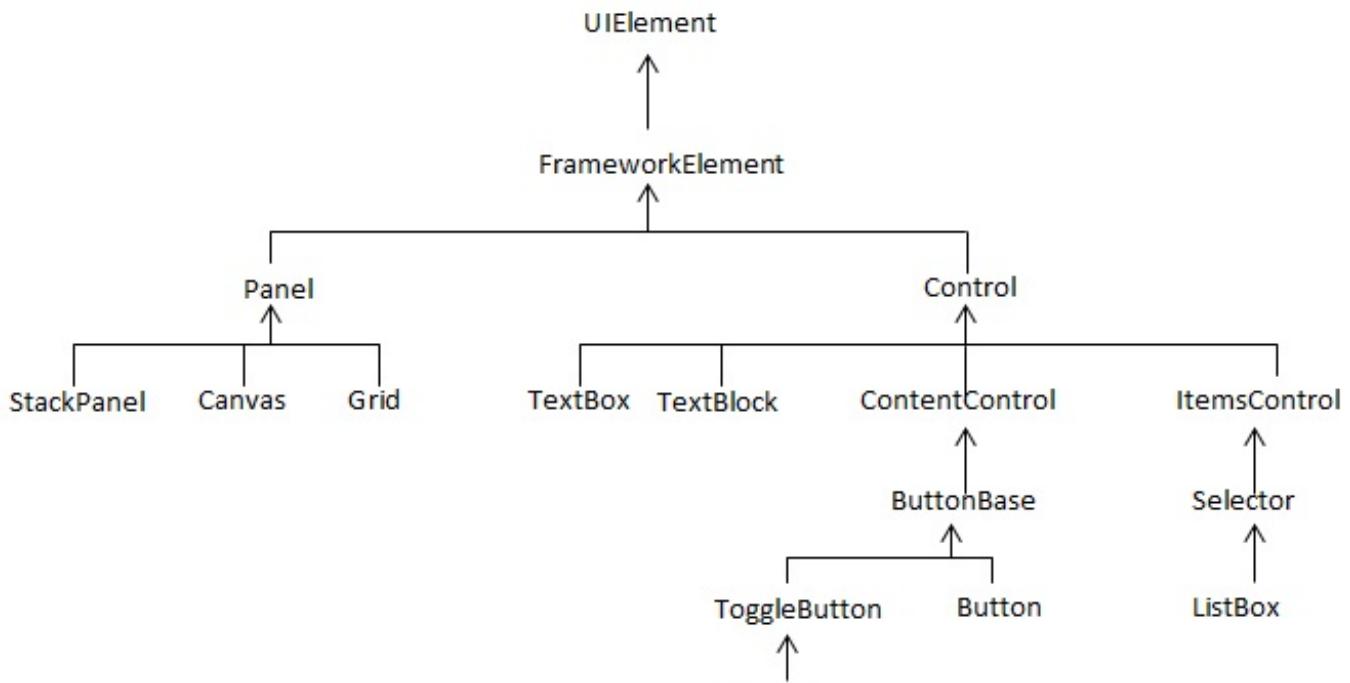
Il y a plusieurs types de contrôles, ceux-ci dérivent tous d'une classe de base abstraite qui s'appelle `UIElement` qui sert à gérer tout ce qui doit pouvoir s'afficher et qui est capable de réagir à une interaction simple de l'utilisateur. Mais la classe `UIElement` ne fait pas grand-chose sans la classe abstraite dérivée `FrameworkElement` qui fournit tous les composants de base pour les objets qui doivent s'afficher sur une page. C'est cette classe également qui gère toute la liaison de données que nous découvrirons un peu plus tard.

C'est donc de cette classe que dérivent deux grandes familles de contrôles : les contrôles à proprement parler et les panneaux. Les panneaux dérivent de la classe abstraite de base `Panel` et servent comme conteneurs permettant de gérer le placement des contrôles. Nous allons y revenir dans un prochain chapitre.

Les contrôles dérivent de la classe abstraite `Control`. Elle sert de classe de base pour tous les éléments qui utilisent un modèle pour définir leur apparence. Nous y reviendrons plus tard, mais une des grandes forces du XAML est d'offrir la possibilité de changer l'apparence d'un contrôle, tout en conservant ses fonctionnalités. Les contrôles qui héritent de la classe `Control` peuvent se répartir en trois grandes catégories.

- Ceux qui dérivent de la classe `ContentControl` et qui permettent de contenir d'autres objets. C'est le cas du bouton par exemple, nous y reviendrons.
- Il y a également les contrôles qui dérivent de la classe `ItemsControl`, qui servent à afficher une liste d'éléments, c'est le cas de la `ListBox` par exemple, nous l'étudierons plus loin.
- Et enfin, il reste les contrôles qui dérivent directement de la classe `Control`, qui ne contiennent pas d'autres contrôles ou qui n'affichent pas de liste d'éléments, comme par exemple le `TextBlock` ou le `TextBox` que nous avons vu.

Pour schématiser, nous pouvons observer ce schéma (incomplet) à la figure suivante.



Hiérarchie de classe pour les contrôles

Une dernière remarque avant de terminer, sur l'utilisation des propriétés. Nous avons vu l'écriture suivante pour par exemple modifier la valeur de la propriété `Text` du contrôle `TextBlock` :

Code : XML

```
<TextBlock Text="Hello world" />
```

Il est également possible d'écrire la propriété Text de cette façon :

Code : XML

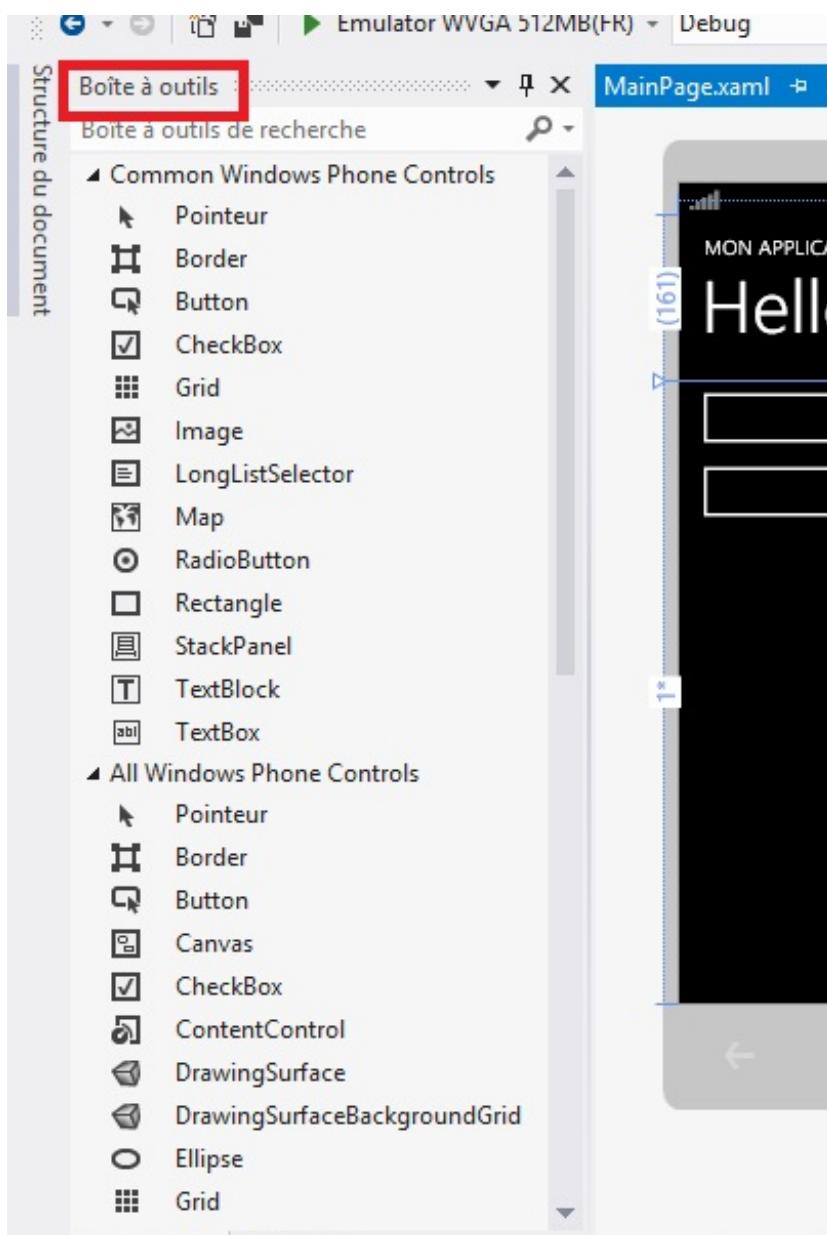
```
<TextBlock>
    <TextBlock.Text>
        Hello world
    </TextBlock.Text>
</TextBlock>
```

Cette écriture nous sera très utile lorsque nous aurons besoin de mettre des choses plus complexes que des chaînes de caractères dans nos propriétés. Nous y reviendrons.

Enfin, une propriété possède généralement une valeur par défaut. C'est pour cela que notre TextBox sait s'afficher même si on ne lui précise que sa propriété Text, elle possède une couleur d'écriture par défaut, une taille d'écriture par défaut, etc.

Utiliser le designer pour ajouter un CheckBox

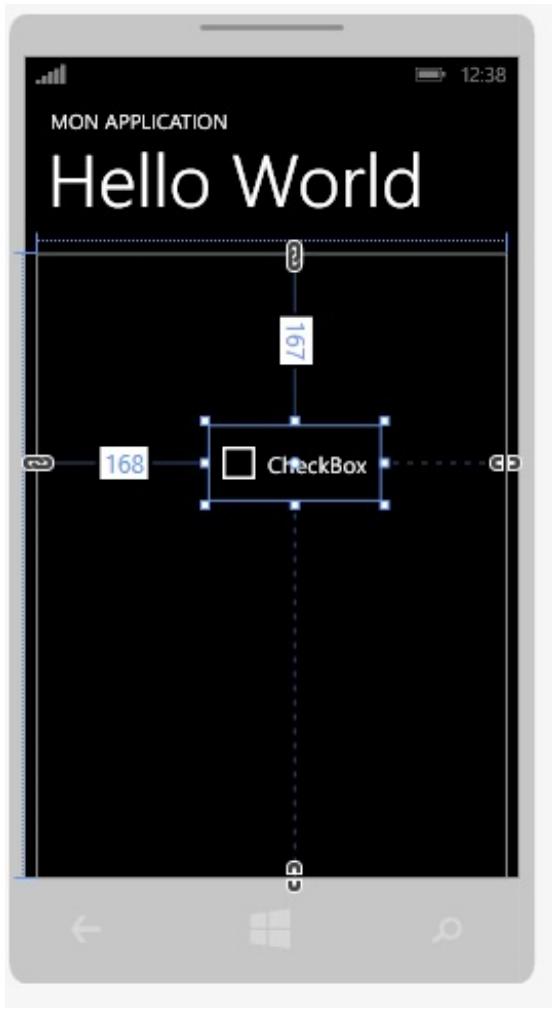
Revenons à notre boîte à outils remplie de contrôle. Elle se trouve à gauche de l'écran, ainsi que vous pouvez le voir sur la figure suivante.



La boîte à outils des contrôles dans Visual Studio

Grâce au designer, vous pouvez faire glisser un contrôle de la boîte à outils dans le rendu de la page. Celui-ci se positionnera automatiquement.

Prenons par exemple le contrôle de case à cocher que nous avons entre-aperçu un peu plus tôt : CheckBox. Sélectionnez-le et faites le glisser sur le rendu de la page (voir la figure suivante).



Ajout d'un contrôle CheckBox à partir du designer

Le designer m'a automatiquement généré le code XAML correspondant :

Code : XML

```
<CheckBox Content="CheckBox" HorizontalAlignment="Left"
Margin="168,167,0,0" VerticalAlignment="Top"/>
```

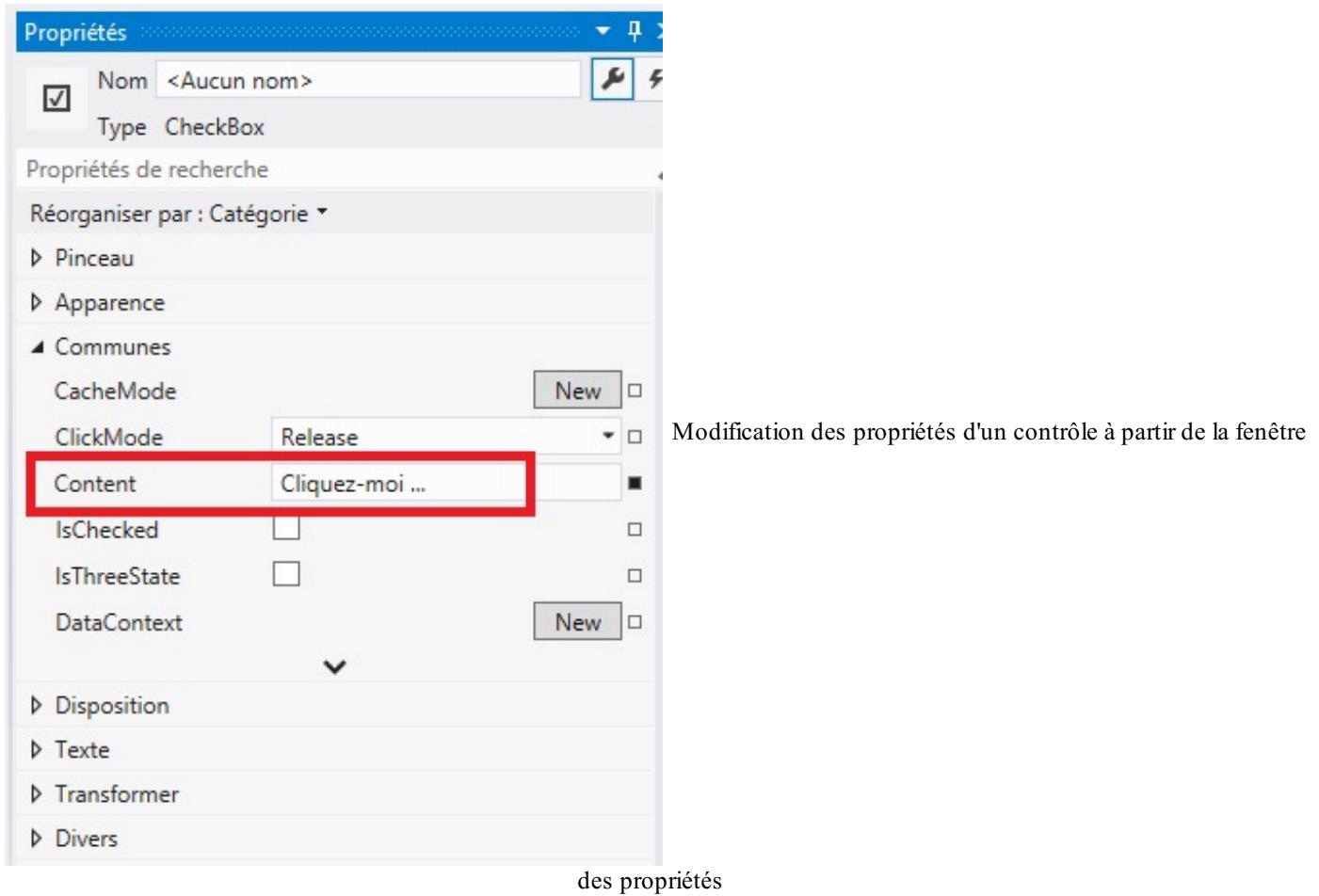
Vous aurez sûrement des valeurs légèrement différentes, notamment au niveau de la propriété Margin. C'est d'ailleurs en utilisant ces valeurs que le designer a pu me positionner le contrôle dans la grille.
Remarquons que si j'avais fait glisser un Canvas et ensuite la case à cocher dans ce Canvas, j'aurais plutôt eu du code comme le suivant :

Code : XML

```
<Canvas HorizontalAlignment="Left" Height="100" Margin="102,125,0,0"
VerticalAlignment="Top" Width="100">
<CheckBox Content="CheckBox" Canvas.Left="56" Canvas.Top="40"/>
</Canvas>
```

Ici, il a utilisé la propriété `Canvas.Top` et `Canvas.Left` pour positionner la case à cocher. Nous allons revenir sur ce positionnement un peu plus loin.

Il est possible de modifier les propriétés de la case à cocher, par exemple son contenu, en allant dans la fenêtre de Propriétés (voir la figure suivante).

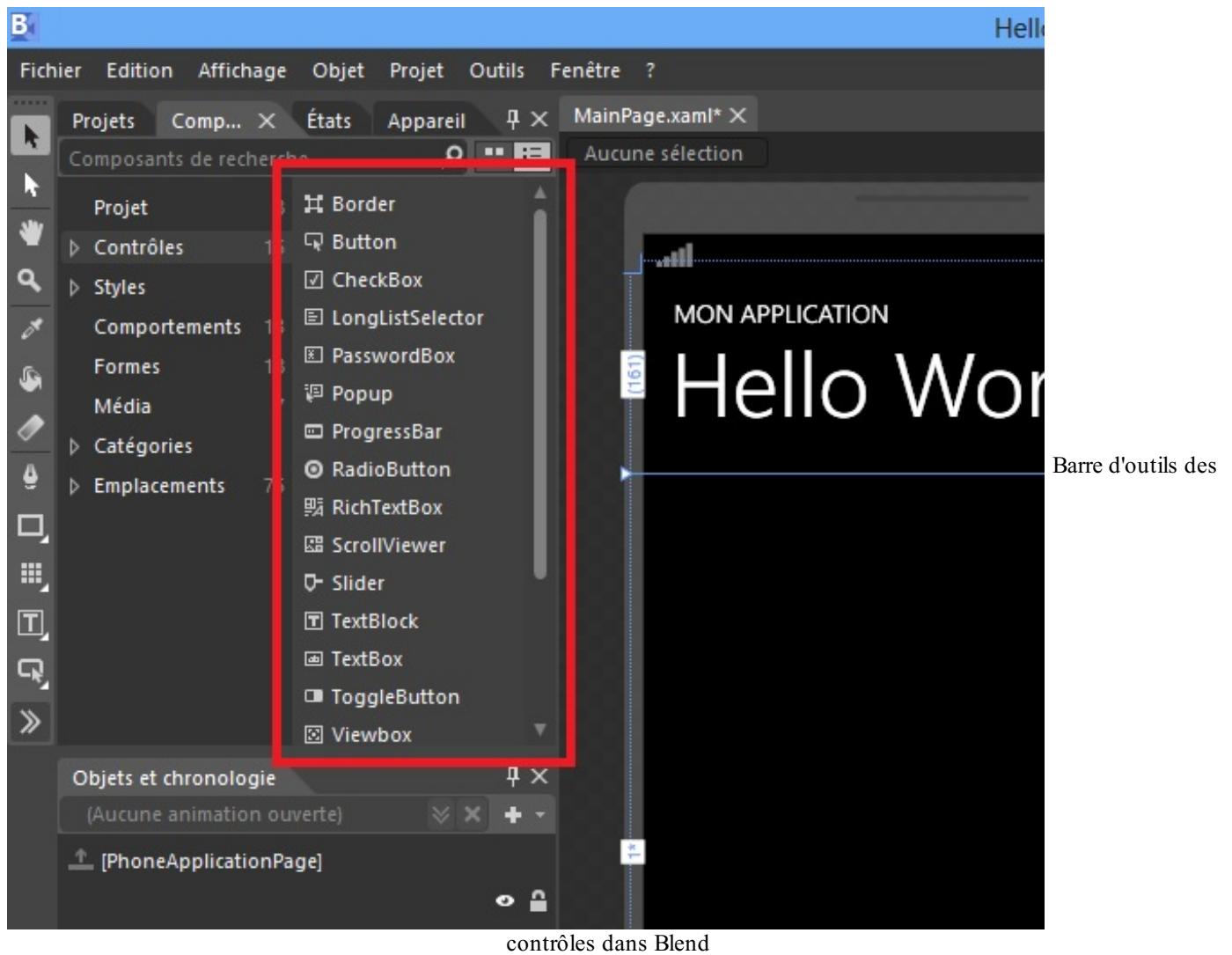


i Si la fenêtre de propriétés n'est pas affichée, vous pouvez la faire apparaître en allant dans le menu Affichage, Fenêtre propriétés.

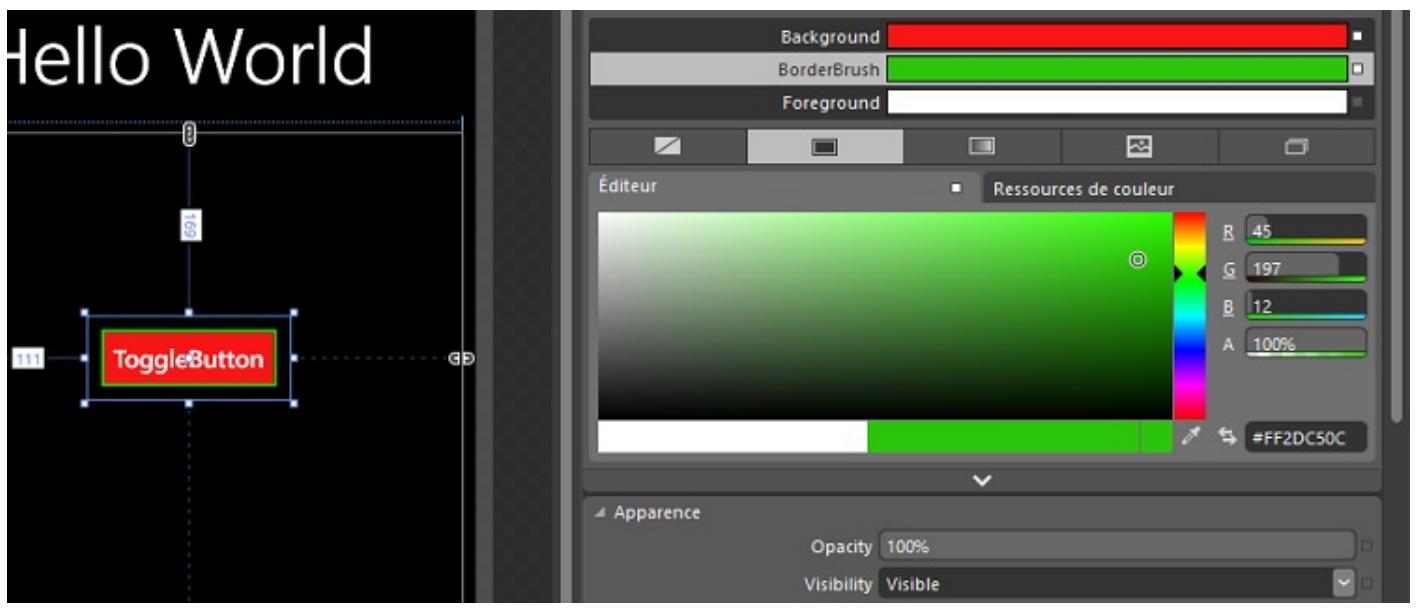
Ici, je change la valeur de la propriété `Content`. Je peux observer les modifications sur le rendu et dans le XAML. Remarquons que le designer est un outil utile pour créer son rendu, par contre il ne remplace pas une bonne connaissance du XAML afin de gérer correctement le positionnement.

Utiliser Expression Blend pour ajouter un ToggleButton

J'aurais aussi pu faire la même chose dans Expression Blend qui est l'outil de design professionnel. J'ai également accès à la boîte à outils (voir la figure suivante).



Et de la même façon, je peux faire glisser un contrôle, disons le ToggleButton, sur ma page. J'ai également accès à ses propriétés afin de les modifier. Ici, par exemple, je modifie la couleur du ToggleButton (voir la figure suivante).



Une fois revenu dans Visual Studio, je peux voir les modifications apportées depuis Blend, avec par exemple dans mon cas :

Code : XML

```
<ToggleButton Content="ToggleButton" HorizontalAlignment="Left"
Margin="111,169,0,0" VerticalAlignment="Top" Background="#FFF71616"
BorderBrush="#FF2DC50C"/>
```

Nous reviendrons sur Blend tout au long de ce cours.

- Il existe tout une hiérarchie des contrôles utilisables dans nos pages.
- Les contrôles sont accessibles depuis la barre d'outils de Visual Studio ou dans expression blend.
- Le designer de Visual Studio et celui de Blend peuvent nous faciliter la tâche dans le design de nos applications.

Le clavier virtuel

Le clavier virtuel est ce petit clavier qui apparaît lorsque l'on clique dans une zone de texte modifiable, que nous avons pu voir dans notre Hello World.

En anglais il se nomme le SIP, pour Soft Input Panel, que l'on traduit par clavier virtuel.
Nous allons voir comment nous en servir.

Afficher le clavier virtuel

Vous vous rappelez de notre Hello World ? Lorsque nous avons cliqué dans le TextBox, nous avons vu apparaître ce fameux clavier virtuel (voir la figure suivante).



Affichage du clavier virtuel

Il n'y a qu'une seule solution pour afficher le clavier virtuel. Il faut que l'utilisateur clique dans une zone de texte modifiable. Et à ce moment-là, le clavier virtuel apparaît en bas de l'écran. Techniquement, il s'affiche quand le contrôle TextBox prend le focus (lorsque l'on clique dans le contrôle) et il disparaît lorsque celui-ci perd le focus (lorsqu'on clique en dehors du contrôle).
Il n'est pas possible de déclencher son affichage par programmation, ni son masquage, à part en manipulant le focus.
Pour afficher un TextBox, on utilisera le XAML suivant :

Code : XML

```
<TextBox x:Name="MonTextBox" />
```

Intercepter les touches du clavier virtuel

Comme déjà dit, il n'est pas possible de manipuler le clavier. Par contre, on peut savoir quand une touche est appuyée en utilisant l'événement KeyDown ou KeyUp du TextBox. Il s'agit d'événements qui sont levés lorsqu'on appuie sur une touche ou lorsqu'on relâche la touche.

Prenons par exemple le code suivant :

Code : XML

```
<StackPanel>
    <TextBox x:Name="MonTextBox" KeyDown="MonTextBox_KeyDown"
    KeyUp="MonTextBox_KeyUp" />
    <TextBlock x:Name="Statut" />
</StackPanel>
```

Et le code behind :

Code : C#

```
private void MonTextBox_KeyDown(object sender, KeyEventArgs e)
{
    Statut.Text = "Touche appuyée : " + e.Key;
}

private void MonTextBox_KeyUp(object sender, KeyEventArgs e)
{
    Statut.Text = "Touche relachée : " + e.Key;
}
```

Nous aurons la figure suivante.



Affichage de la touche relachée dans l'émulateur

Les différents types de clavier

Le clavier que nous avons vu est le clavier par défaut. Nous avons à notre disposition d'autres types de clavier, par exemple un clavier numérique permettant de saisir des numéros de téléphone (voir la figure suivante).



Clavier virtuel de type numérique

Pour choisir le type de clavier à afficher, nous allons utiliser la propriété `InputScope` du contrôle `TextBox`. Par exemple, pour afficher le clavier numérique, je vais utiliser :

Code : XML

```
<TextBox x:Name="MonTextBox" InputScope="Number" />
```

La liste des différents claviers supportés est disponible [ici](#).

Cela permet d'avoir un clavier plus adapté, si l'on doit par exemple permettre de saisir un @ pour un email, ou des caractères spéciaux, etc.

Sur la figure suivante, vous pouvez voir un clavier optimisé pour la saisie d'un email (type `EmailUserName`), avec un arrobas (@) et un « .fr ».



Clavier virtuel optimisé pour la saisie d'adresse email

- Le clavier virtuel s'affiche lorsque l'on clique dans une zone de texte modifiable.
- Il existe plusieurs types de clavier à notre disposition que nous pouvons choisir grâce à la propriété InputScope.

Les conteneurs et le placement

Dans notre Hello World, lorsque nous avons parlé du contrôle `TextBlock`, nous avons dit qu'il faisait partie du contrôle `StackPanel` qui lui-même faisait partie du contrôle `Grid`.

Ces deux contrôles sont en fait des conteneurs de contrôles dont l'objectif est de regrouper des contrôles de différentes façons.

Les contrôles conteneurs vont être très utiles pour organiser le look et l'agencement de nos pages. Il y en a quelques uns indispensables à découvrir qui vont constamment vous servir lors des vos développements. Nous allons à présent les découvrir et voir comment nous en servir.

StackPanel

Le `StackPanel` par exemple, comme son nom peut le suggérer, permet d'empiler les contrôles les uns à la suite des autres. Dans l'exemple suivant, les contrôles sont ajoutés les uns en-dessous des autres :

Code : XML

```
<StackPanel>
    <TextBlock Text="Bonjour à tous" />
    <Button Content="Cliquez-moi" />
    <Image
        Source="http://www.siteduzero.com/images/designs/2/logo_sdz_fr.png?
        normal" />
</StackPanel>
```

Ce qui donne la figure suivante.



Empilement vertical des contrôles grâce au StackPanel

Où nous affichons un texte, un bouton et une image. Nous verrons un peu plus précisément le contrôle `Image` dans le chapitre suivant.

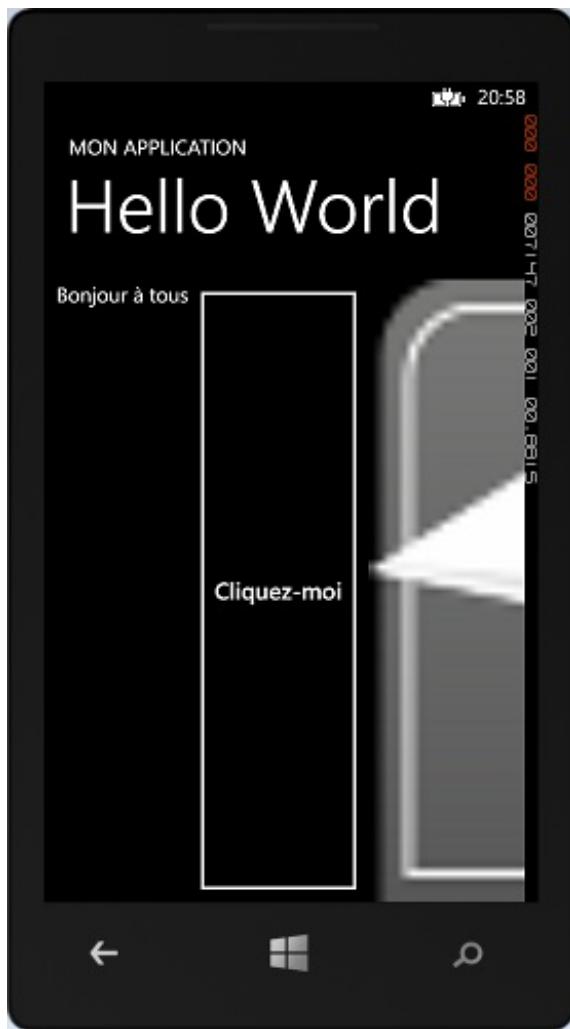
Notons au passage que Visual Studio et l'émulateur peuvent très facilement récupérer des contenus sur internet, sauf si vous utilisez un proxy. Ici par exemple, en utilisant l'URL d'une image, je peux l'afficher sans problème dans mon application, si celle-ci est connectée à internet bien sûr.

Le contrôle `StackPanel` peut aussi empiler les contrôles horizontalement. Pour cela, il suffit de changer la valeur d'une de ses propriétés :

Code : XML

```
<StackPanel Orientation="Horizontal">
    <TextBlock Text="Bonjour à tous" />
    <Button Content="Cliquez-moi" />
    <Image
        Source="http://www.siteduzero.com/images/designs/2/logo_sdz_fr.png?
        normal" />
</StackPanel>
```

Ici, nous avons changé l'orientation de l'empilement pour la mettre en horizontal. Ce qui donne la figure suivante.



Empilement horizontal des contrôles

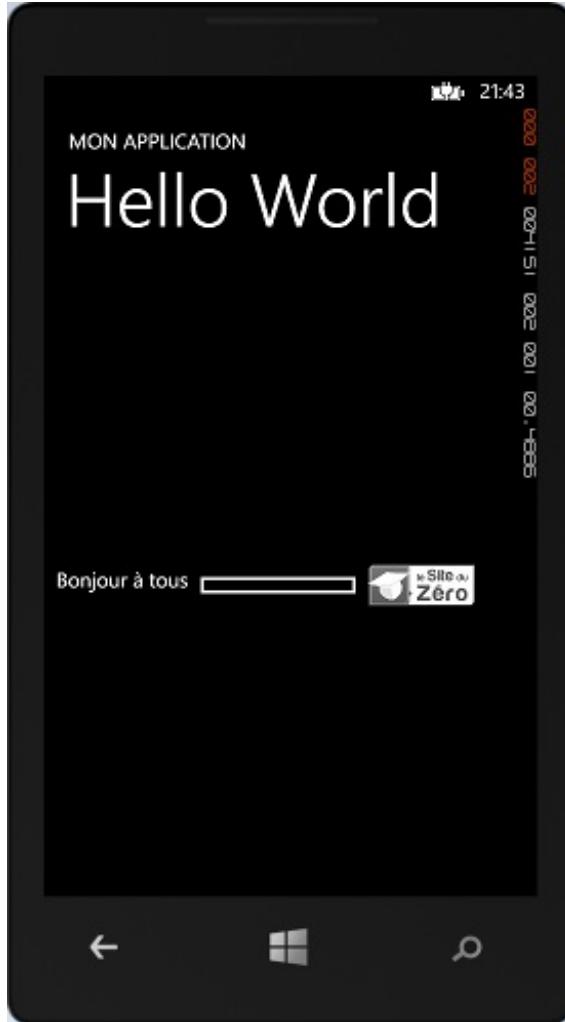
Ce qui ne rend pas très bien ici... Pour l'améliorer, nous pouvons ajouter d'autres propriétés à nos contrôles, notamment les réduire en hauteur ou en largeur grâce aux propriétés `Height` ou `Width`. Par exemple :

Code : XML

```
<StackPanel Orientation="Horizontal" Height="40">
    <TextBlock Text="Bonjour à tous" />
```

```
<Button Content="Cliquez-moi" />
<Image
Source="http://www.siteduzero.com/images/designs/2/logo_sdz_fr.png?
normal" />
</StackPanel>
```

Ici, j'ai rajouté une hauteur pour le contrôle StackPanel, en fixant la propriété Height à 40 pixels, ce qui fait que tous les sous-contrôles se sont adaptés à cette hauteur. Nous aurons donc la figure suivante.



Les contrôles ont la taille fixée à 40 pixels

Par défaut, le contrôle StackPanel essaie d'occuper le maximum de place disponible dans la grille dont il fait partie. Comme nous avons forcé la hauteur, le StackPanel va alors se centrer. Il est possible d'aligner le StackPanel en haut grâce à la propriété VerticalAlignment :

Code : XML

```
<StackPanel Orientation="Horizontal" Height="40"
VerticalAlignment="Top">
<TextBlock Text="Bonjour à tous" />
<Button Content="Cliquez-moi" />
<Image
Source="http://www.siteduzero.com/images/designs/2/logo_sdz_fr.png?
normal" />
</StackPanel>
```

Ce qui donne la figure suivante.



Alignement vertical en haut du StackPanel

Nous allons revenir sur l'alignement un peu plus loin. Voilà pour ce petit tour du StackPanel !

ScrollViewer

Il existe d'autres conteneurs, voyons par exemple le ScrollViewer.

Il nous sert à rendre accessible des contrôles qui pourraient être masqués par un écran trop petit. Prenons par exemple ce code XAML :

Code : XML

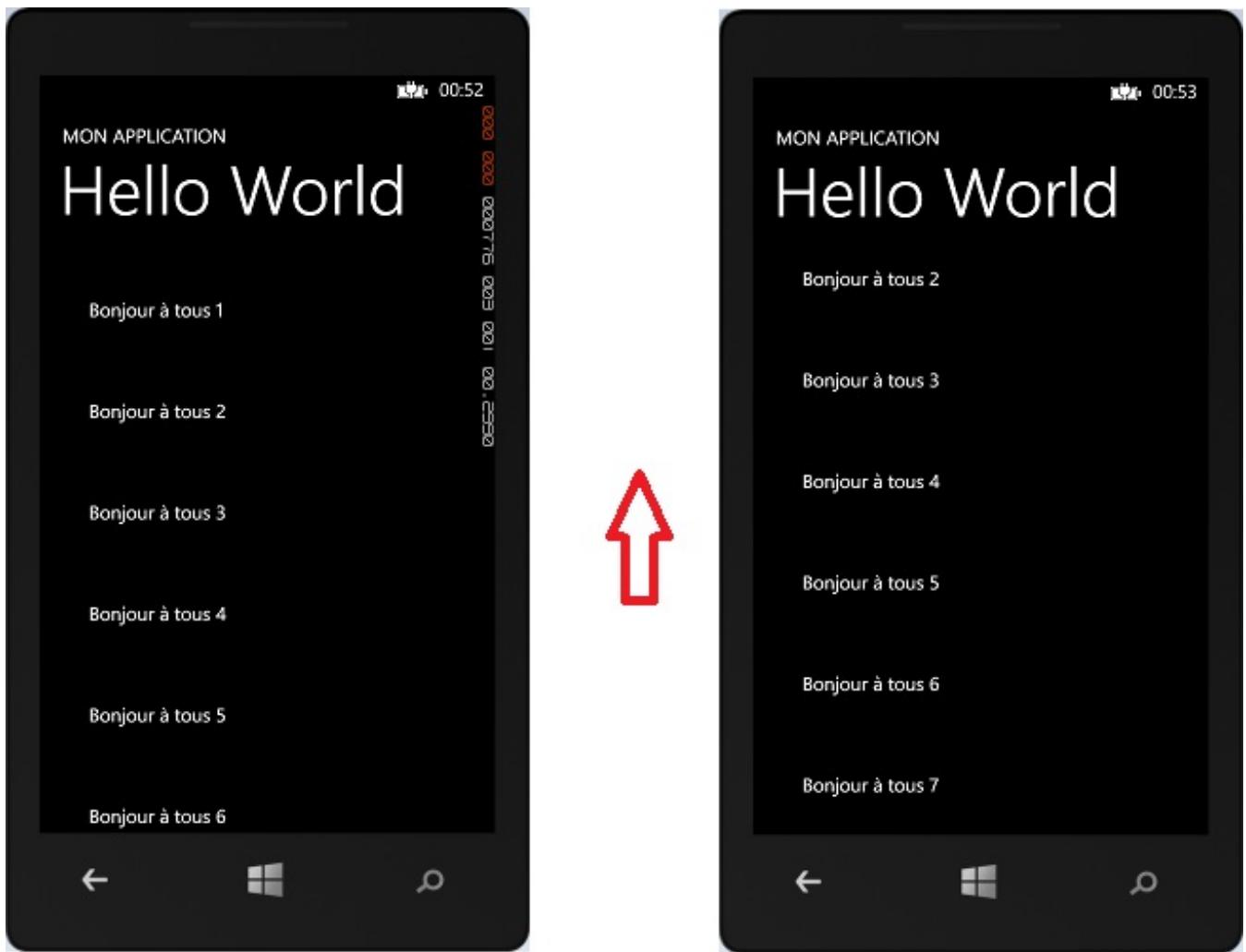
```
<ScrollViewer>
    <StackPanel>
        <TextBlock Text="Bonjour à tous 1" Margin="40" />
        <TextBlock Text="Bonjour à tous 2" Margin="40" />
        <TextBlock Text="Bonjour à tous 3" Margin="40" />
        <TextBlock Text="Bonjour à tous 4" Margin="40" />
        <TextBlock Text="Bonjour à tous 5" Margin="40" />
        <TextBlock Text="Bonjour à tous 6" Margin="40" />
        <TextBlock Text="Bonjour à tous 7" Margin="40" />
    </StackPanel>
</ScrollViewer>
```

Nous créons 7 contrôles TextBlock, contenant une petite phrase, qui doivent se mettre les uns en-dessous des autres. Vous aurez deviné que la propriété Margin permet de définir une marge autour du contrôle, j'y reviendrai. Si nous regardons le résultat, nous pouvons constater qu'il nous manque un TextBlock (voir la figure suivante).



Il manque un contrôle à l'écran

Vous vous en doutez, il s'affiche trop bas et nous ne pouvons pas le voir sur l'écran car il y a trop de choses. Le ScrollViewer va nous permettre de résoudre ce problème. Ce contrôle gère une espèce de défilement, comme lorsque nous avons un ascenseur dans nos pages web. Ce qui fait qu'il sera possible de naviguer de haut en bas sur notre émulateur en cliquant sur l'écran et en maintenant le clic tout en bougeant la souris de haut en bas (voir la figure suivante).



ScrollViewer permet de faire défiler l'écran

Vous pouvez également vous amuser à faire défiler le ScrollViewer horizontalement, mais il vous faudra changer une propriété :

Code : XML

```
<ScrollViewer HorizontalScrollBarVisibility="Auto">
    <StackPanel Orientation="Horizontal">
        ...
    </StackPanel>
</ScrollViewer>
```

Grid

Parlons à présent du contrôle Grid. C'est un contrôle très utilisé qui va permettre de positionner d'autres contrôles dans une grille. Une grille peut être définie par des colonnes et des lignes. Il sera alors possible d'indiquer dans quelle colonne ou à quelle ligne se positionne un contrôle. Par exemple, avec le code suivant :

Code : XML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="O" FontSize="50" Grid.Row="0" Grid.Column="0" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="1" Grid.Column="0" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="2" Grid.Column="0" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="0" Grid.Column="1" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="1" Grid.Column="1" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="2" Grid.Column="1" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="0" Grid.Column="2" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="1" Grid.Column="2" ...>
    <TextBlock Text="O" FontSize="50" Grid.Row="2" Grid.Column="2" ...>
</Grid>
```

```
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="O" FontSize="50" Grid.Row="0" Grid.Column="1" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="O" FontSize="50" Grid.Row="0" Grid.Column="2" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="X" FontSize="50" Grid.Row="1" Grid.Column="0" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="O" FontSize="50" Grid.Row="1" Grid.Column="1" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="X" FontSize="50" Grid.Row="1" Grid.Column="2" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="O" FontSize="50" Grid.Row="2" Grid.Column="0" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="X" FontSize="50" Grid.Row="2" Grid.Column="1" />
        HorizontalAlignment="Center" VerticalAlignment="Center" />
        <TextBlock Text="X" FontSize="50" Grid.Row="2" Grid.Column="2" />
    
```

Je définis une grille composée de 3 lignes sur 3 colonnes. Dans chaque case je pose un `TextBlock` avec une valeur qui me simule un jeu de morpion. Ce qu'il est important de remarquer ici c'est que je définis le nombre de colonnes grâce à `ColumnDefinition` :

Code : XML

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

De la même façon, je définis le nombre de lignes grâce à `RowDefinition` :

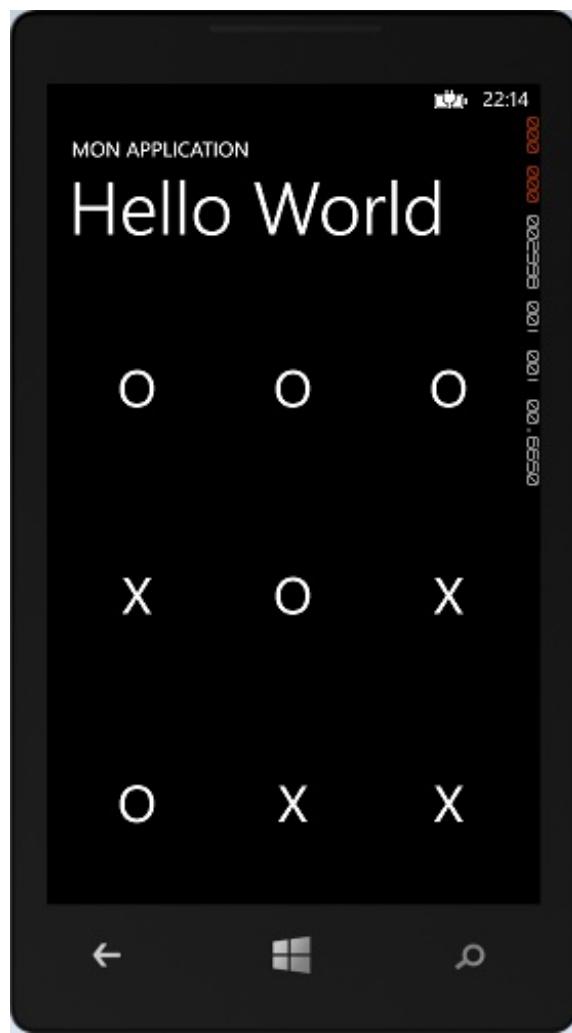
Code : XML

```
<Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

Il y a donc 3 colonnes et 3 lignes. Chaque colonne a une largeur d'un tiers de l'écran. Chaque ligne a une hauteur d'un tiers de l'écran. Je vais y revenir juste après.

Pour indiquer qu'un contrôle est à la ligne 1 de la colonne 2, j'utiliserai les propriétés `Grid.Row` et `Grid.Column` avec les valeurs 1 et 2. (À noter qu'on commence à numérotter à partir de 0, classiquement).

Ce qui donnera la figure suivante.



Une grille de 3x3

Pratique non ?

Nous pouvons voir aussi que dans la définition d'une ligne, nous positionnons la propriété `Height`. C'est ici que nous indiquerons la hauteur de chaque ligne. C'est la même chose pour la largeur des colonnes, cela se fait avec la propriété `Width` sur chaque `ColumnDefinition`.

Ainsi, en utilisant l'étoile, nous avons dit que nous voulions que le XAML s'occupe de répartir toute la place disponible. Il y a trois étoiles, chaque ligne et colonne a donc un tiers de la place pour s'afficher.

D'autres valeurs sont possibles. Il est par exemple possible de forcer la taille à une valeur précise. Par exemple, si je modifie l'exemple précédent pour avoir :

Code : XML

```
<Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="200" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="100" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="50" />
</Grid.ColumnDefinitions>
```

J'indiquerai ici que la première colonne aura une taille fixe de 100, la troisième une taille fixe de 50 et la deuxième prendra la taille restante.

De la même façon, pour les lignes, la deuxième est forcée à 200 et les deux autres se répartiront la taille restante, à savoir la moitié chacune.

J'en profite pour vous rappeler qu'un téléphone Windows Phone 7.5 a une résolution de 480 en largeur et de 800 en hauteur et qu'un téléphone Windows Phone 8 possède trois résolutions :

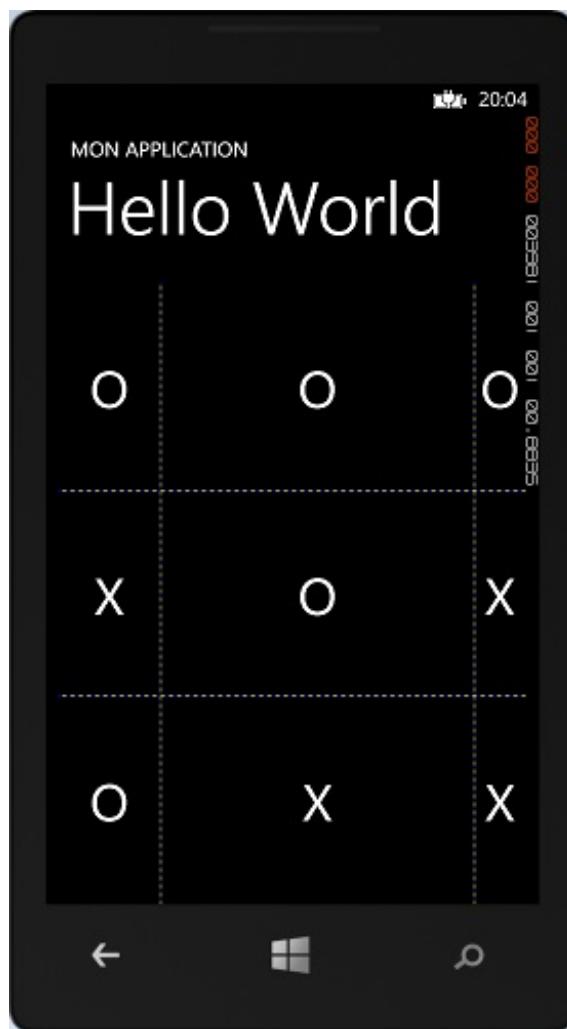
- WVGA (800x480 pixels), comme Windows Phone 7.5
- WXVGA (1280x768)
- "True 720p" (1280x720)

Ainsi, sur un téléphone en WVGA la deuxième colonne aura une taille de $480 - 100 - 50 = 330$. Ce qui donne une grille plutôt disgracieuse, mais étant donné que chaque contrôle est aligné au centre, cela ne se verra pas trop. Pour bien le mettre en valeur, il est possible de rajouter une propriété à la grille lui indiquant que nous souhaitons voir les lignes. Bien souvent, cette propriété ne servira qu'à des fins de débogages. Il suffit d'indiquer :

Code : XML

```
<Grid ShowGridLines="True">
```

Par contre, les lignes s'affichent uniquement dans l'émulateur car le designer montre déjà ce que ça donne (voir la figure suivante).



Affichage des lignes de la grille

Il est bien sûr possible de faire en sorte qu'un contrôle s'étende sur plusieurs colonnes ou sur plusieurs lignes, à ce moment-là, on utilisera la propriété `Grid.ColumnSpan` ou `Grid.RowSpan` pour indiquer le nombre de colonnes ou lignes que l'on doit fusionner. Par exemple :

Code : XML

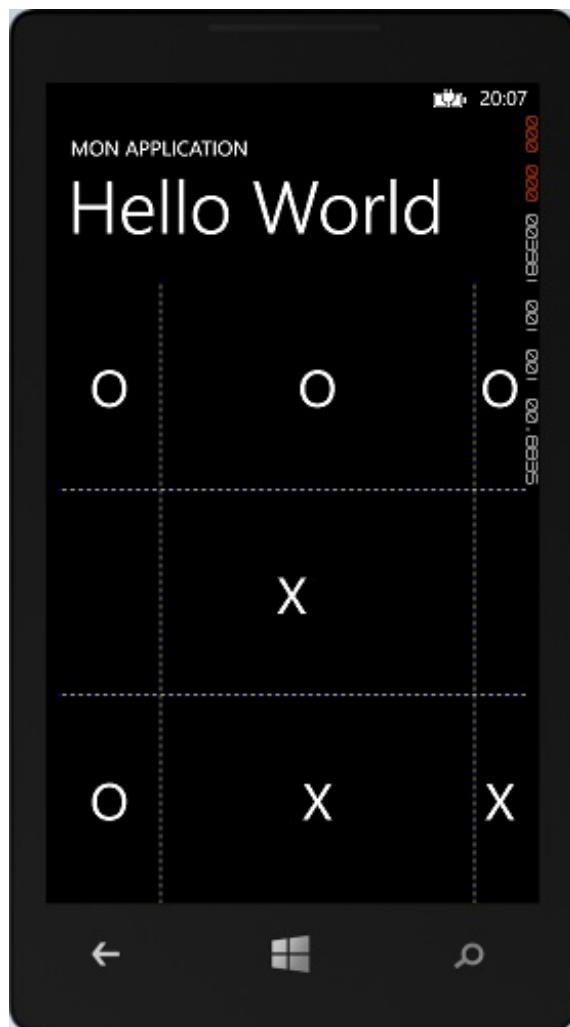
```
<TextBlock Text="X" FontSize="50" Grid.Row="1" Grid.Column="0"
HorizontalAlignment="Center" VerticalAlignment="Center"
Grid.ColumnSpan="3" />
```

à la place de :

Code : XML

```
<TextBlock Text="X" FontSize="50" Grid.Row="1" Grid.Column="0"
HorizontalAlignment="Center" VerticalAlignment="Center" />
<TextBlock Text="O" FontSize="50" Grid.Row="1" Grid.Column="1"
HorizontalAlignment="Center" VerticalAlignment="Center" />
<TextBlock Text="X" FontSize="50" Grid.Row="1" Grid.Column="2"
HorizontalAlignment="Center" VerticalAlignment="Center" />
```

Et nous avons donc la figure suivante.



Une grille avec une case s'étirant sur 3 colonnes

Avant de terminer sur les lignes et les colonnes, il est important de savoir qu'il existe une autre valeur pour définir la largeur ou la hauteur, à savoir la valeur `Auto`. Elle permet de dire que c'est la largeur ou la hauteur des contrôles qui vont définir la hauteur d'une ligne ou d'une colonne.

Remarquez que par défaut, un contrôle s'affichera à la ligne 0 et à la colonne 0 tant que son `Grid.Row` ou son `Grid.Column` n'est pas défini. Ainsi la ligne suivante :

Code : XML

```
<TextBlock Text="X" FontSize="50" Grid.Row="0" Grid.Column="0" />
```

est équivalente à celle-ci :

Code : XML

```
<TextBlock Text="X" FontSize="50" />
```

Voilà pour ce petit tour de ce contrôle si pratique qu'est la grille.

Canvas

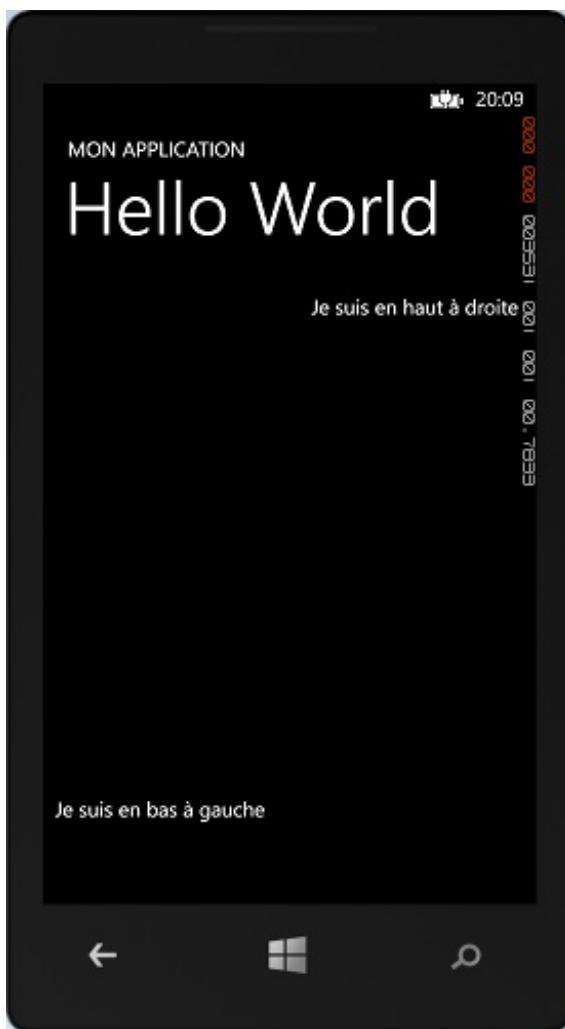
Nous finirons notre aperçu des conteneurs avec le Canvas. Au contraire des autres conteneurs qui calculent eux même la position des contrôles, ici c'est le développeur qui indique l'emplacement d'un contrôle, de manière relative à la position du Canvas. De plus le Canvas ne calculera pas automatiquement sa hauteur et sa largeur en analysant ses enfants, contrairement aux autres conteneurs. Ainsi si on met dans un StackPanel un Canvas suivi d'un bouton, le bouton sera affiché par-dessus le Canvas, car ce dernier aura une hauteur de 0 bien qu'il possède des enfants.

Ainsi, l'exemple suivant :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Canvas>
        <TextBlock Text="Je suis en bas à gauche" Canvas.Top="500"
/>
        <TextBlock Text="Je suis en haut à droite" Canvas.Left="250"
Canvas.Top="10"/>
    </Canvas>
</Grid>
```

affichera la figure suivante.



Positionnement absolu avec le Canvas

Nous nous servons des propriétés `Canvas.Top` et `Canvas.Left` pour indiquer la position du contrôle relativement au Canvas.

C'est sans doute le conteneur qui permet le placement le plus simple à comprendre, par contre ce n'est pas forcément le plus efficace, surtout pour s'adapter à plusieurs résolutions ou lorsque nous retournerons l'écran. J'en parlerai un peu plus loin. Remarquons qu'une page doit absolument commencer par avoir un conteneur comme contrôle racine de tous les autres contrôles. C'est ce que génère par défaut Visual Studio lorsqu'on crée une nouvelle page. Il y met en l'occurrence un contrôle Grid.

Alignement

L'alignement permet de définir comment est aligné un contrôle par rapport à son contenant, en général un panneau. Il existe plusieurs valeurs pour cette propriété :

- Stretch (étiré) qui est la valeur par défaut
- Left (gauche)
- Right (droite)
- Center (centre)

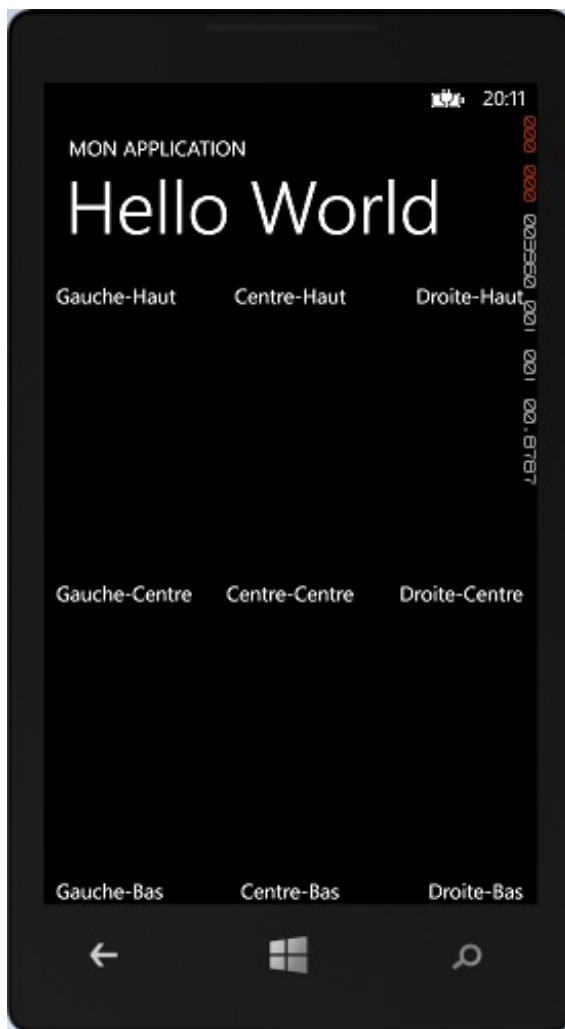
Ainsi le code XAML suivant :

Code : XML

```
<Grid>
    <TextBlock Text="Gauche-Haut" HorizontalAlignment="Left"
    VerticalAlignment="Top" />
    <TextBlock Text="Centre-Haut" HorizontalAlignment="Center"
    VerticalAlignment="Top" />
    <TextBlock Text="Droite-Haut" HorizontalAlignment="Right"
```

```
VerticalAlignment="Top" />
    <TextBlock Text="Gauche-Centre" HorizontalAlignment="Left"
    VerticalAlignment="Center" />
    <TextBlock Text="Centre-Centre" HorizontalAlignment="Center"
    VerticalAlignment="Center" />
    <TextBlock Text="Droite-Centre" HorizontalAlignment="Right"
    VerticalAlignment="Center" />
    <TextBlock Text="Gauche-Bas" HorizontalAlignment="Left"
    VerticalAlignment="Bottom" />
    <TextBlock Text="Centre-Bas" HorizontalAlignment="Center"
    VerticalAlignment="Bottom" />
    <TextBlock Text="Droite-Bas" HorizontalAlignment="Right"
    VerticalAlignment="Bottom" />
</Grid>
```

produira le résultat que vous pouvez voir à la figure suivante.



Les différents alignements

Lorsqu'on utilise la valeur Stretch, les valeurs des propriétés Width et Height peuvent annuler l'effet de l'étirement. On peut voir cet effet avec le code suivant :

Code : XML

```
<Grid>
    <Button Content="Etiré en largeur" Height="100"
    VerticalAlignment="Top" />
    <Button Content="Etiré en hauteur" Width="300"
    HorizontalAlignment="Left" />
</Grid>
```

Qui nous donne la figure suivante.



L'étirement est annulé par les propriétés Height et Width

Bien sûr, un bouton avec que du Stretch remplirait ici tout l'écran.



Les propriétés d'alignements n'ont pas d'impact dans un Canvas.

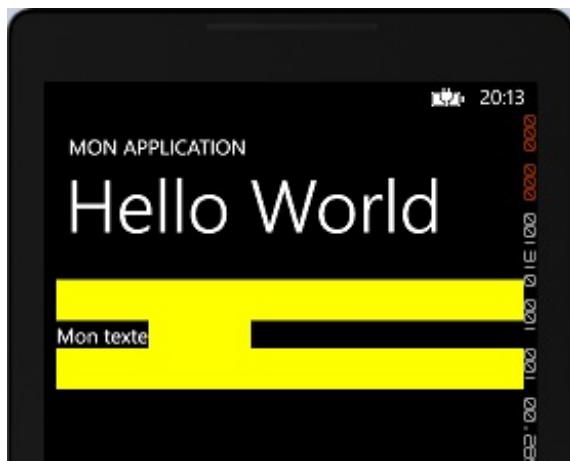
Marges et espacement

Avant de terminer, je vais revenir rapidement sur les marges. Je les ai rapidement évoquées tout à l'heure. Pour mieux les comprendre, regardons cet exemple :

Code : XML

```
<StackPanel>
    <Rectangle Height="40" Fill="Yellow" />
    <StackPanel Orientation="Horizontal">
        <TextBlock Text="Mon texte" />
        <Rectangle Width="100" Fill="Yellow" />
    </StackPanel>
    <Rectangle Height="40" Fill="Yellow" />
</StackPanel>
```

Il donne la figure suivante.



Un TextBlock sans marge

En rajoutant une marge au TextBlock, nous pouvons voir concrètement se décaler le texte :

Code : XML

```
<TextBlock Text="Mon texte" Margin="50"/>
```

Qui donne la figure suivante.



Une marge de 50 autour du TextBlock

En fait, la marge précédente rajoute une marge de 50 à gauche, en haut, à droite et en bas. Ce qui est l'équivalent de :

Code : XML

```
<TextBlock Text="Mon texte" Margin="50 50 50 50"/>
```

Il est tout à fait possible de choisir de mettre des marges différentes pour, respectivement, la marge à gauche, en haut, à droite et en bas :

Code : XML

```
<TextBlock Text="Mon texte" Margin="0 10 270 100"/>
```

Qui donne la figure suivante.



La marge peut être de taille différente en haut, en bas, à gauche ou à droite
du contrôle

Bref, les marges aident à positionner le contrôle à l'emplacement voulu, très utile pour avoir un peu d'espace dans un StackPanel.

Remarquez que nous avons aperçu dans ces exemples le contrôle `Rectangle` qui permet, vous vous en doutez, de dessiner un rectangle. Nous l'étudierons un peu plus loin.

- Les conteneurs contiennent des contrôles et nous sont utiles pour les positionner sur la page.
- Chaque page doit posséder un unique conteneur racine.
- Les propriétés d'alignement, de marge et d'espacement nous permettent d'affiner nos positionnements dans les conteneurs.

Ajouter du style

Nous avons vu qu'on pouvait modifier les couleurs, la taille de l'écriture... grâce à la fenêtre des propriétés d'un contrôle. Cela modifie les propriétés des contrôles et affecte leur rendu. C'est très bien. Mais imaginons que nous voulions changer les couleurs et l'écriture de plusieurs contrôles, il va falloir reproduire ceci sur tous les contrôles, ce qui d'un coup est plutôt moins bien ! 😞

C'est là qu'intervient le **style**. Il correspond à l'identification de plusieurs propriétés par un nom, que l'on peut appliquer facilement à plusieurs contrôles.

Afficher des images

Pour commencer, nous allons reparler du contrôle `Image`. Ce n'est pas vraiment un style à proprement parler, mais il va être très utile pour rendre nos pages un peu plus jolies. Nous l'avons rapidement utilisé en montrant qu'il était très simple d'afficher une image présente sur internet simplement en indiquant l'URL de celle-ci.

Il est également très facile d'afficher des images à nous, embarquées dans l'application. Pour cela, j'utilise une petite image toute bête, représentant un cercle rouge (voir la figure suivante).



Un cercle rouge sur fond blanc

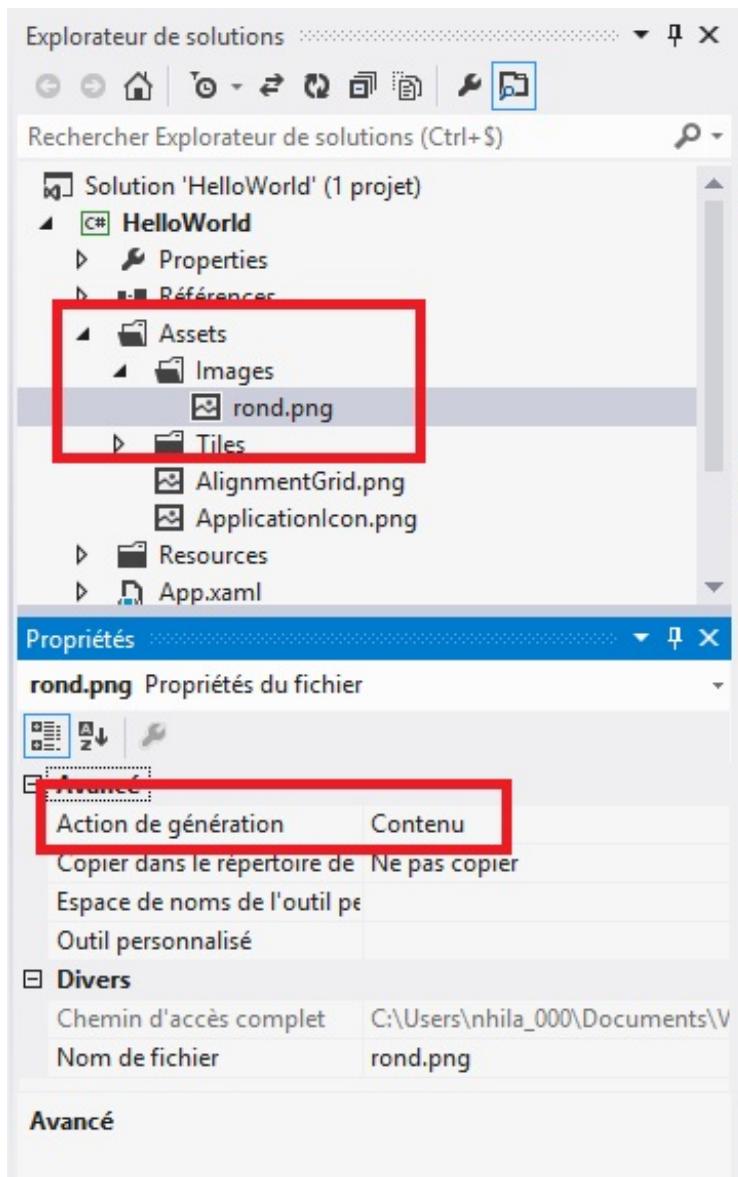
Pour suivre cet exemple avec moi, je vous conseille de télécharger cette image, en cliquant [ici](#).

Ajoutons donc cette image à la solution. Pour cela, je vais commencer par créer un nouveau répertoire `Images` sous le répertoire `Assets` qui a été ajouté lors de la création de la solution. Ensuite, nous allons ajouter un élément existant en faisant un clic droit sur le projet. Je sélectionne l'image qui s'ajoute automatiquement à la solution.



J'ai créé un répertoire pour que mes images soient mieux rangées et pour illustrer le chemin d'accès de celles-ci, mais ce n'est pas du tout une étape obligatoire.

Ici, il faut faire attention à ce que dans les propriétés de l'image, l'action de génération soit à *Contenu*, ce qui est le paramètre par défaut pour les projets ciblant Windows Phone 8, mais pas Windows Phone 7 où c'est l'action de génération *Resource* qui est le paramètre par défaut. *Contenu* permet d'indiquer que l'image sera un fichier à part, non intégrée à l'assembly, nous y reviendrons à la fin de la partie (voir la figure suivante).



L'image doit avoir son action de génération à Contenu

Nous pourrons alors très simplement afficher l'image en nous basant sur l'URL relative de l'image dans la solution :

Code : XML

```
<Image x:Name="MonImage" Source="/Assets/Images/rond.png" Width="60"  
Height="60" />
```

À noter que cela peut aussi se faire grâce au code behind. Pour cela, supprimons la propriété Source du XAML :

Code : XML

```
<Image x:Name="MonImage" Width="60" Height="60" />
```

Et chargeons l'image dans le code de cette façon :

Code : C#

```
MonImage.Source = new BitmapImage(new Uri("/Assets/Images/rond.png",  
UriKind.Relative));
```

Remarque : pour utiliser la classe `BitmapImage`, il faut ajouter le using suivant :

Code : C#

```
using System.Windows.Media.Imaging;
```

Cela semble moins pratique, mais je vous l'ai présenté car nous utiliserons cette méthode un petit peu plus loin. D'une manière générale, il sera toujours plus pertinent de passer par le XAML que par le code !



Il n'est pas possible d'afficher des images GIF dans ce contrôle lorsqu'on développe pour Windows Phone 7.5, seuls les formats JPG et PNG sont supportés. Par contre, le GIF est utilisable pour des projets Windows Phone 8 mais ne s'anime pas.

Les ressources

Les ressources sont un mécanisme de XAML qui permet de réutiliser facilement des objets ou des valeurs. Chaque classe qui dérive de `FrameworkElement` dispose d'une propriété `Resources`, qui est en fait un dictionnaire de ressources. Chaque contrôle peut donc avoir son propre dictionnaire de ressources mais en général, on définit les ressources soit au niveau de la page, soit au niveau de l'application.

Par exemple, pour définir une ressource au niveau de la page, nous utiliserons la syntaxe suivante :

Code : XML

```
<phone:PhoneApplicationPage  
    x:Class="HelloWorld.MainPage"  
  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    [... plein de choses ...]  
    shell:SystemTray.IsVisible="True">  
  
    <phone:PhoneApplicationPage.Resources>  
        <SolidColorBrush x:Key="BrushRouge" Color="Red"/>  
    </phone:PhoneApplicationPage.Resources>  
  
    <[... plein de choses dans la page ...]>  
</phone:PhoneApplicationPage>
```

Ici, j'ai créé un objet `SolidColorBrush`, qui sert à peindre une zone d'une couleur unie, dont la couleur est Rouge dans les ressources de ma page. Il est obligatoire qu'une ressource possède un nom, ici je l'ai nommé `BrushRouge`. Je vais désormais pouvoir utiliser cet objet avec des contrôles, ce qui donne :

Code : XML

```
<StackPanel>  
    <TextBlock Text="Bonjour ma ressource"  
    Foreground="{StaticResource BrushRouge}" />  
    <Button Content="Cliquez-moi, je suis rouge"  
    Foreground="{StaticResource BrushRouge}" />  
</StackPanel>
```

Et nous aurons la figure suivante.



Utilisation d'une ressource de type pinceau rouge

Alors, qu'y-a-t-il derrière ces ressources ?

La première chose que l'on peut voir c'est la syntaxe particulière à l'intérieur de la propriété ForeGround :

Code : XML

```
Foreground="{StaticResource BrushRouge}"
```

Des accolades avec le mot-clé `StaticResource`... Cela signifie qu'à l'exécution de l'application, le moteur va aller chercher la ressource associée au nom `BrushRouge` et il va la mettre dans la propriété `Foreground` de notre contrôle.



On appelle la syntaxe entre accolades une « extension de balisage XAML », en anglais : extension markup.

Ce moteur commence par chercher la ressource dans les ressources de la page et s'il ne la trouve pas, il ira chercher dans le dictionnaire de ressources de l'application. Nous avons positionné notre ressource dans la page, c'est donc celle-ci qu'il utilise en premier.

Remarquez que le dictionnaire de ressources, c'est simplement une collection d'objets associés à un nom. S'il est défini dans la page, alors il sera accessible pour tous les contrôles de la page. S'il est défini au niveau de l'application, alors il sera utilisable partout dans l'application.

Vous aurez pu constater qu'ici, notre principal intérêt d'utiliser une ressource est de pouvoir changer la couleur de tous les contrôles en une seule fois.

Nous pouvons mettre n'importe quel objet dans les ressources. Nous y avons mis un `SolidColorBrush` afin que cela se voit, mais il est possible d'y mettre un peu tout et n'importe quoi. Pour illustrer ce point, nous allons utiliser le dictionnaire de

ressource de l'application et y stocker une chaîne de caractère. Ouvrez donc le fichier App.xaml où se trouve le dictionnaire de ressources. Nous pouvons ajouter notre chaîne de caractères dans la section <Application.Resources> déjà existante pour avoir :

Code : XML

```
<Application.Resources>
    <system:String x:Key="TitreApplication">Hello
    World</system:String>
</Application.Resources>
```

Dans le projet créé par défaut pour Windows Phone 8, il y a déjà une ligne dans les ressources de l'application :

 **<local:LocalizedStrings xmlns:local="clr-namespace:HelloWorld"**
x:Key="LocalizedStrings"/> qui fait globalement la même chose, sauf que l'objet mis en ressource est une instance de la classe LocalizedStrings qui se trouve à la racine du projet.

Vous serez obligés de rajouter l'espace de nom suivant en haut du fichier App.xaml :

Code : XML

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

dans les propriétés de l'application de manière à avoir :

Code : XML

```
<Application
    x:Class="HelloWorld.App"

    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
    namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
    namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:system="clr-namespace:System;assembly=mscorlib">
```

Pourquoi ? Parce que la classe String n'est pas connue de l'application. Il faut lui indiquer où elle se trouve, en indiquant son espace de nom, un peu comme un **using C#**.

Pour cela on utilise la syntaxe précédente pour dire que l'espace de nom que j'ai nommé « system » correspondra à l'espace de nom System de l'assembly mscorelib.

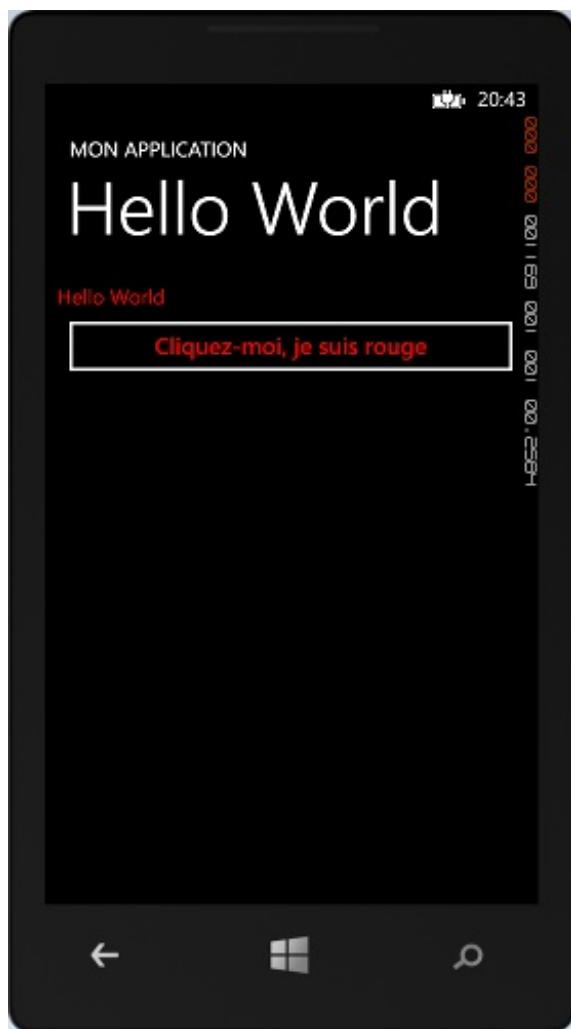
Pour utiliser ma classe String, il faudra que je la préfixe de « system: ».

Bref, revenons à notre ressource de type String. Je vais pouvoir l'utiliser depuis n'importe quelle page vu qu'elle est définie dans le dictionnaire de ressources de l'application, par exemple dans ma page principale :

Code : XML

```
<TextBlock Text="{StaticResource TitreApplication}"
Foreground="{StaticResource BrushRouge}" />
```

Et nous aurons donc la figure suivante.



Utilisation d'une ressource de type chaîne de caractère

i Le fichier App.xaml est à l'application ce que le fichier Mainpage.xaml est à la page MainPage. Il est accompagné de son code behind App.xaml.cs et on peut voir que la classe App dérive de la classe Application. Nous y reviendrons mais c'est dans cette classe que nous pourrons gérer tout ce qui rapporte à l'application. C'est le cas par exemple du dictionnaire de ressources, mais c'est également là que nous pourrons gérer les erreurs applicatives non interceptées dans le code et plein d'autres choses que nous découvrirons au fur et à mesure.

Les styles

Le **style** correspond à l'identification de plusieurs propriétés par un nom, que l'on peut appliquer facilement à plusieurs contrôles.

Un style trouve donc tout à fait naturellement sa place dans les dictionnaires de ressources que nous avons déjà vus. Un style est, comme une ressource, caractérisé par un nom et cible un type de contrôle. Par exemple, observons le style suivant :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
    <Style x:Key="StyleTexte" TargetType="TextBlock">
        <Setter Property="Foreground" Value="Green" />
        <Setter Property="FontSize" Value="35" />
        <Setter Property="FontFamily" Value="Comic Sans MS" />
        <Setter Property="Margin" Value="0 20 0 20" />
        <Setter Property="HorizontalAlignment" Value="Center" />
    </Style>
</phone:PhoneApplicationPage.Resources>
```

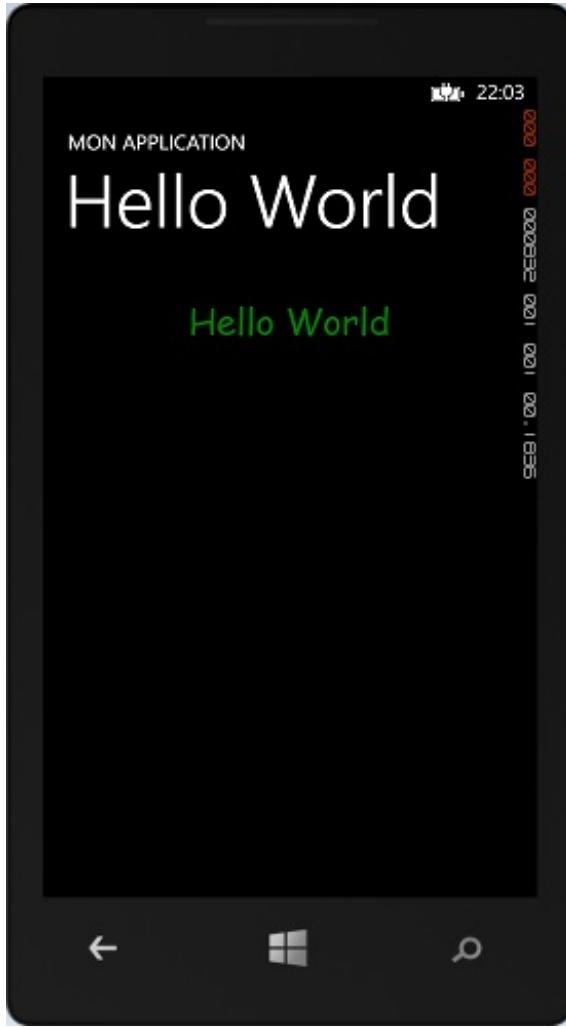
On remarque l'élément important `TargetType="TextBlock"` qui me permet d'indiquer que le style s'applique aux contrôles `TextBlock`. La lecture du style nous renseigne sur ce qu'il fait. Nous pouvons remarquer que la propriété `Foreground` aura la valeur `Green`, que la propriété `FontSize` aura la valeur `35`, etc.

Pour que notre contrôle bénéficie de ce style, nous pourrons utiliser encore la syntaxe suivante :

Code : XML

```
<TextBlock Text="{StaticResource TitreApplication}"  
Style="{StaticResource StyleTexte}"/>
```

Ce qui nous donnera la figure suivante.



Le style appliqué au TextBlock

Ah, mais ça me rappelle quelque chose, on n'a pas déjà vu des styles ?

Et si, lorsque nous créons une nouvelle application, Visual Studio nous crée le squelette d'une page dans le fichier MainPage.xaml et nous avons notamment le titre de la page défini de cette façon :

Code : XML

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">  
    <TextBlock Text="MON APPLICATION" Style="{StaticResource  
    PhoneTextNormalStyle}" Margin="12,0"/>  
    <TextBlock Text="Hello World" Margin="9,-7,0,0"  
    Style="{StaticResource PhoneTextTitle1Style}"/>  
</StackPanel>
```

Vous pouvez désormais comprendre que ces deux TextBlock utilisent les styles PhoneTextNormalStyle et PhoneTextTitle1Style. Ce ne sont pas des styles que nous avons créés. Il s'agit de styles systèmes, présents directement dans le système d'exploitation et que nous pouvons utiliser comme bon nous semble.

Le style est un élément qui sera très souvent utilisé dans nos applications. Définir le XAML associé à ces styles est un peu

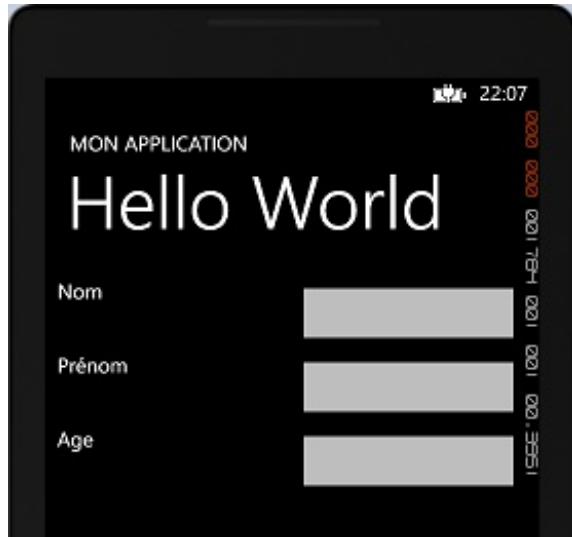
rébarbatif. Heureusement, Blend peut nous aider dans la création de style...

Prenons par exemple le code XAML suivant :

Code : XML

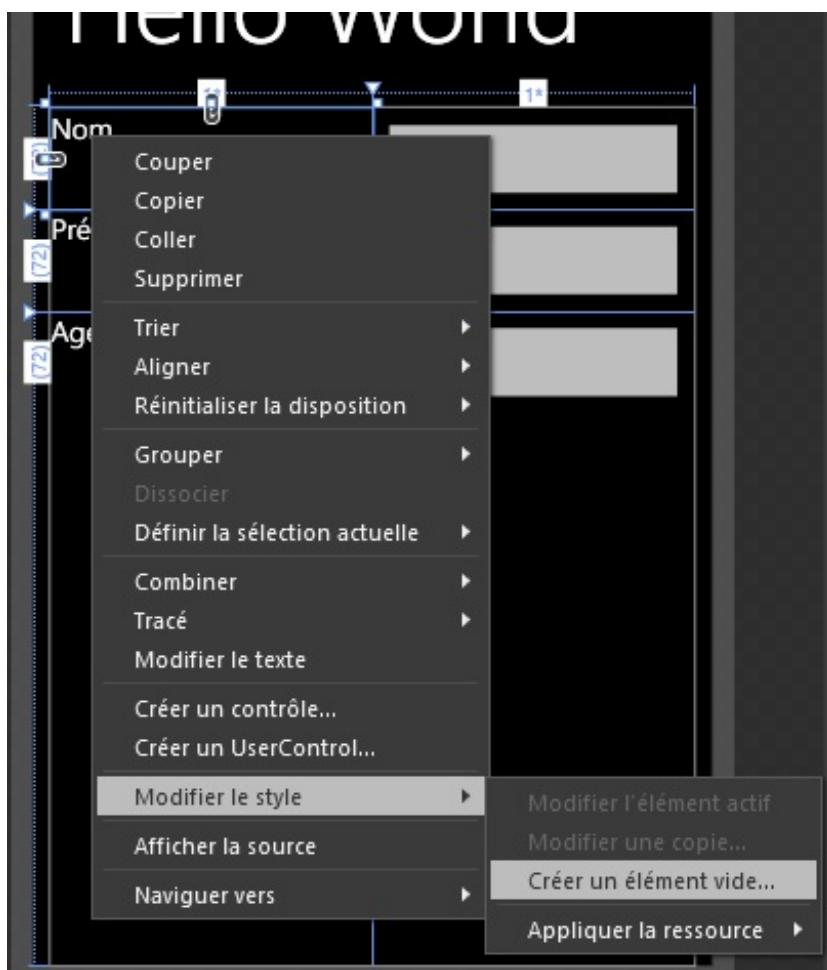
```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>
    <TextBlock Text="Nom" Grid.Column="0" Grid.Row="0" />
    <TextBox Grid.Row="0" Grid.Column="1" />
    <TextBlock Text="Prénom" Grid.Column="0" Grid.Row="1" />
    <TextBox Grid.Row="1" Grid.Column="1" />
    <TextBlock Text="Age" Grid.Column="0" Grid.Row="2" />
    <TextBox Grid.Row="2" Grid.Column="1" />
</Grid>
```

Qui donne la figure suivante.



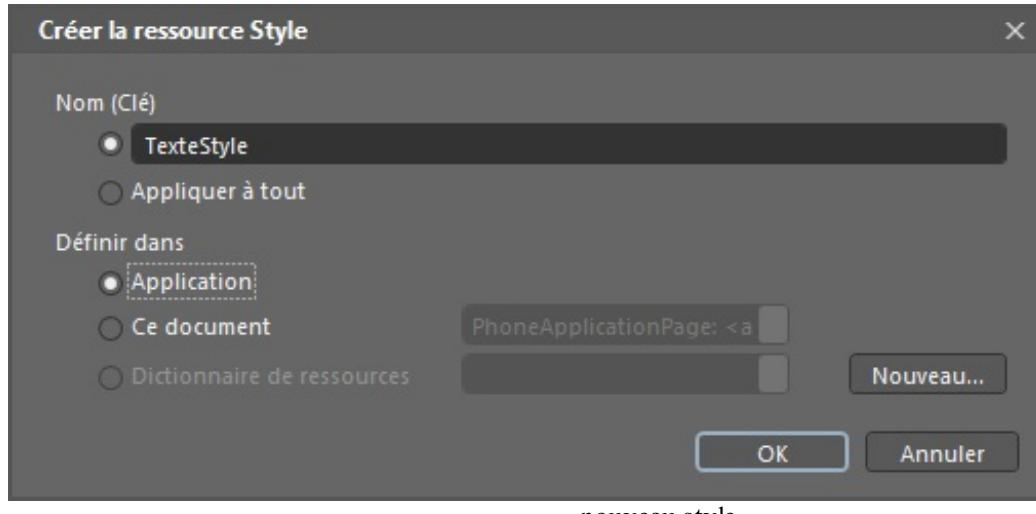
Aperçu d'un formulaire construit en XAML

Si nous passons dans Blend, nous pouvons facilement créer un style en faisant un clic droit sur un `TextBlock` et en choisissant de modifier le style, puis de créer un nouveau style en choisissant de créer un élément vide (voir la figure suivante).



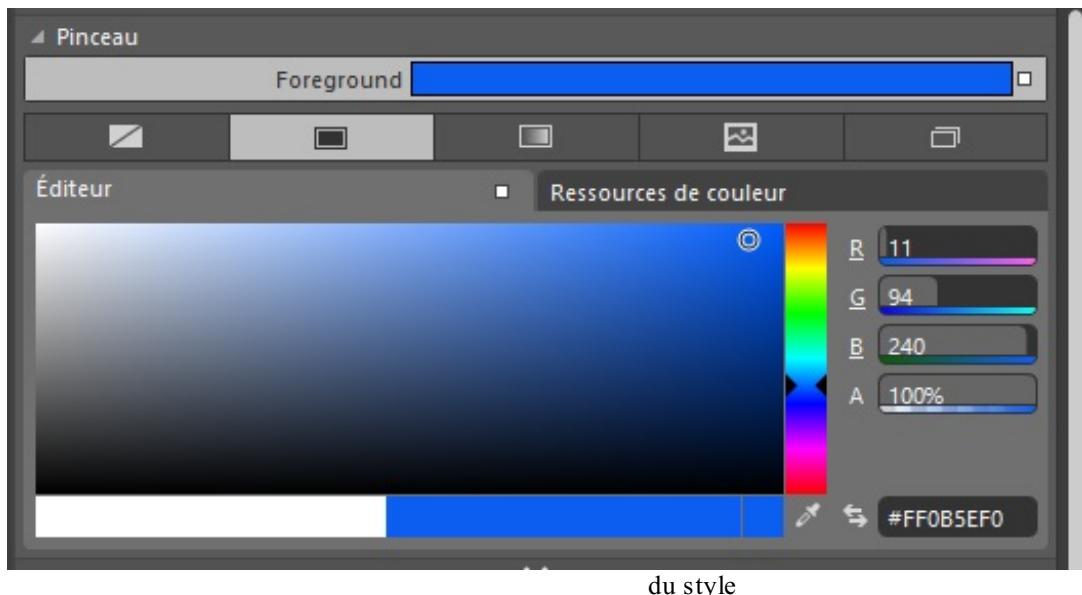
Modification du style dans Blend

Blend nous ouvre une nouvelle fenêtre où nous pouvons créer un nouveau style (voir la figure suivante).



Nous devons fournir un nom au style puis nous pouvons indiquer quelle est la portée du style, soit toute l'application (ce que j'ai choisi), soit la page courante, soit un dictionnaire de ressources déjà existant.

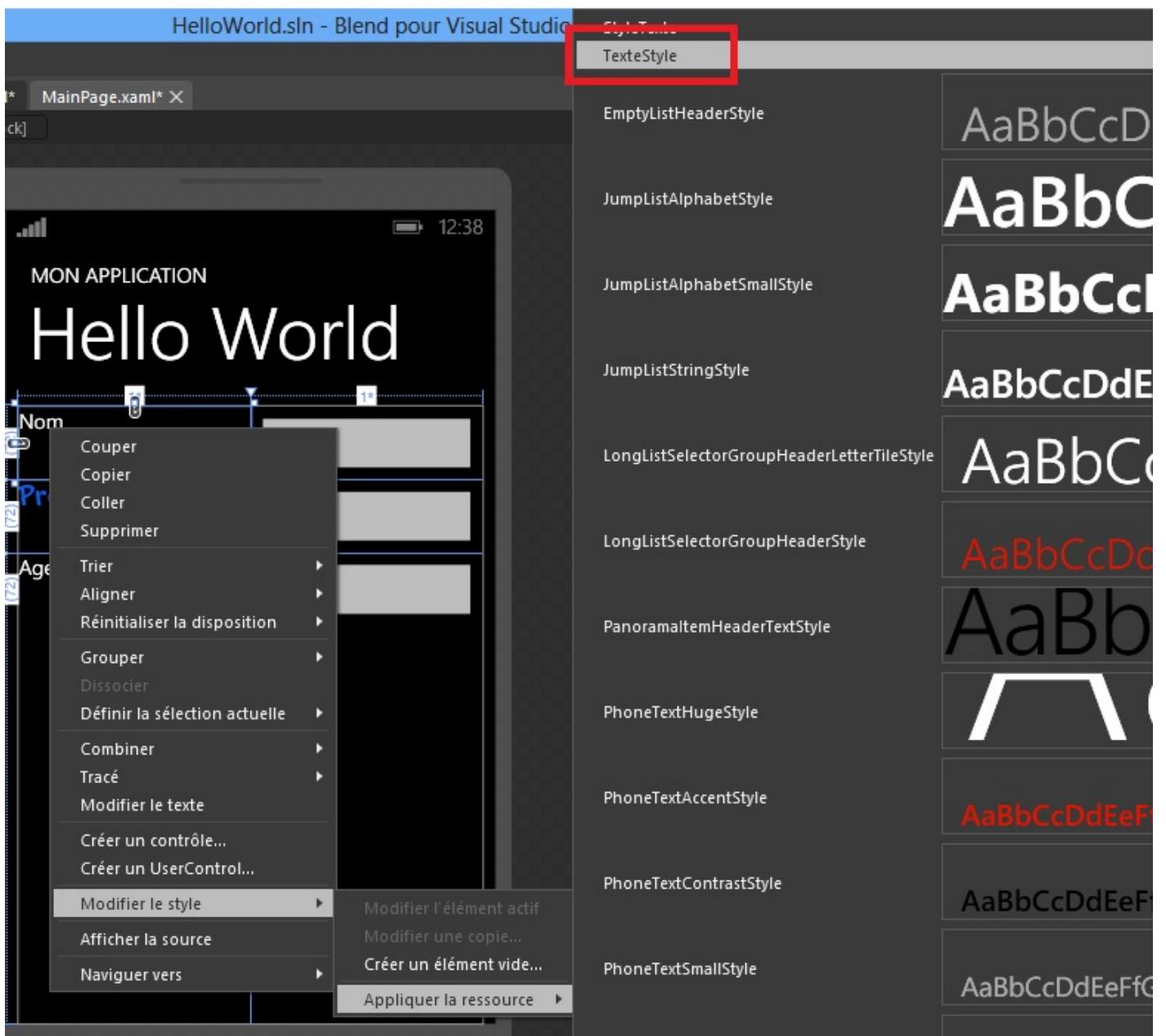
Le style est créé dans le fichier App.xaml, comme nous l'avons déjà vu, qui est le fichier de lancement de l'application. Je peux aller modifier les propriétés du style, par exemple la couleur (voir la figure suivante).



Modification de la couleur

du style

Une fois le style terminé, je peux retourner dans ma page pour appliquer ce style aux autres contrôles. Pour cela, j'utilise le bouton droit sur le contrôle, **Modifier le style**, **Appliquer la ressource**, et je peux retrouver mon style tout en haut (voir la figure suivante).



Notre nouveau style fait partie de la liste des styles

On remarque au passage qu'il existe plein de styles déjà tout prêts, ce sont des styles systèmes comme ceux que nous avons vu un peu plus haut et dont nous pouvons allégerement nous servir !

De retour dans le XAML, je peux constater qu'une propriété a été rajoutée à mes `TextBlock` :

Code : XML

```
<TextBlock Text="Prénom" Grid.Column="0" Grid.Row="1"  
Style="{StaticResource TexteStyle}" />
```

C'est la propriété `Style` bien évidemment, qui va permettre d'indiquer que l'on applique le style `TexteStyle`. Celui-ci est défini dans le XAML du fichier `App.xaml` :

Code : XML

```
<Style x:Key="TexteStyle" TargetType="TextBlock">  
    <Setter Property="Foreground" Value="#FF0B5EF0"/>  
    <Setter Property="FontFamily" Value="Andy"/>  
    <Setter Property="FontSize" Value="32"/>
```

```
</Style>
```

Ce qui correspond aux valeurs que j'ai modifiées. Et voilà, plutôt simple non ?

Remarquons avant de terminer que les styles peuvent hériter entre eux, ce qui permet de compléter ou de remplacer certaines valeurs. Prenons par exemple le XAML suivant :

Code : XML

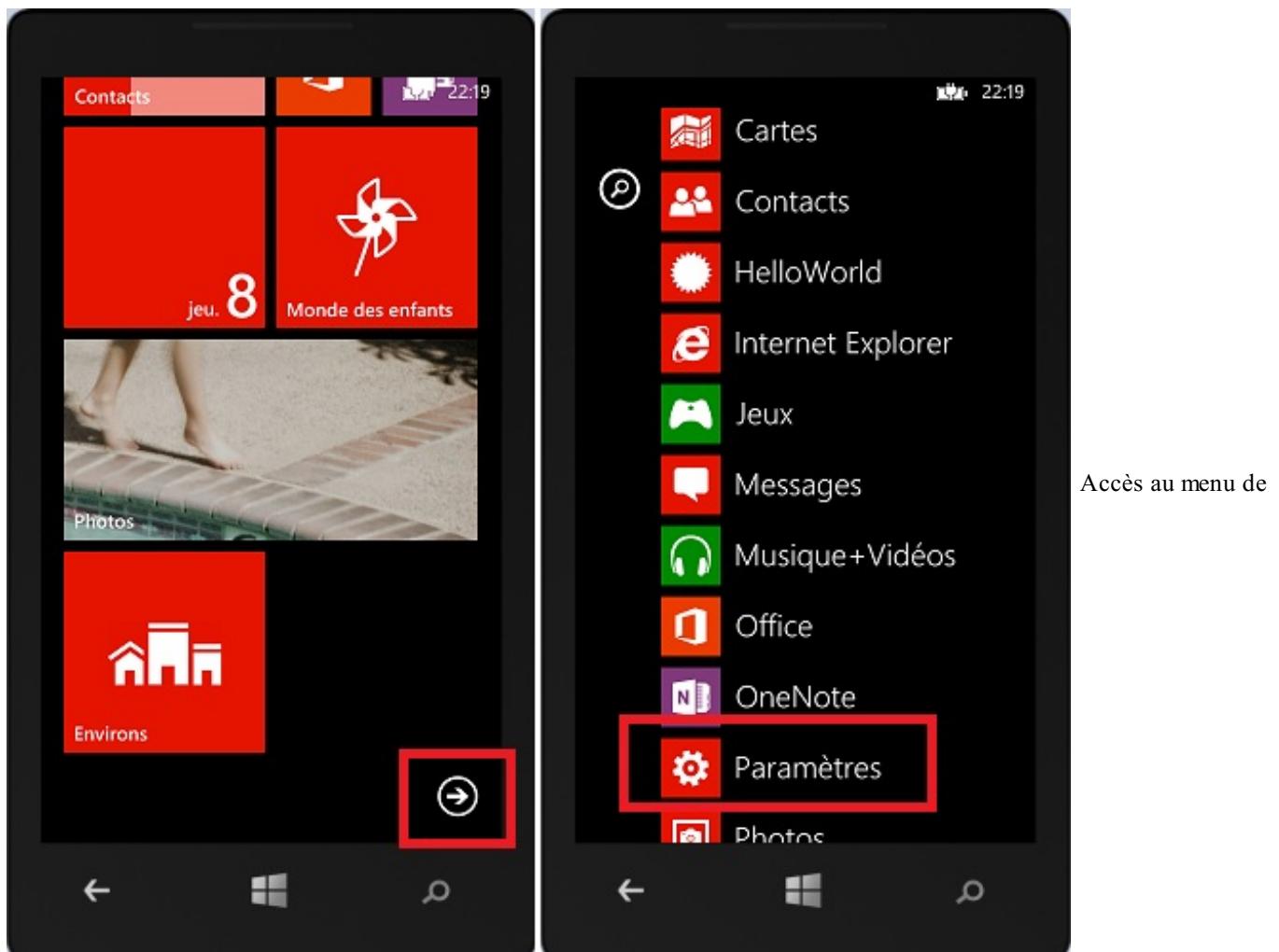
```
<Style x:Key="TexteStyle" TargetType="TextBlock">
    <Setter Property="Foreground" Value="#FF0B5EF0"/>
    <Setter Property="FontFamily" Value="Andy"/>
    <Setter Property="FontSize" Value="32"/>
</Style>
<Style x:Key="TexteStyle2" TargetType="TextBlock"
BasedOn="{StaticResource TexteStyle}">
    <Setter Property="FontSize" Value="45"/>
    <Setter Property="HorizontalAlignment" Value="Center" />
</Style>
```

Le deuxième style hérite du premier grâce à la propriété `BaseOn`.

Notez que les styles sont encore plus puissants que ça, nous aurons l'occasion de voir d'autres utilisations plus loin dans le cours.

Les thèmes

Si vous avez joué avec l'émulateur ou avec vos Windows Phone, vous avez pu vous rendre compte que Windows Phone disposait de plusieurs thèmes. Ouvrez votre émulateur et faites glisser l'écran sur la droite ou cliquez sur la flèche en bas de l'écran d'accueil, cliquez ensuite sur Paramètres (voir la figure suivante).



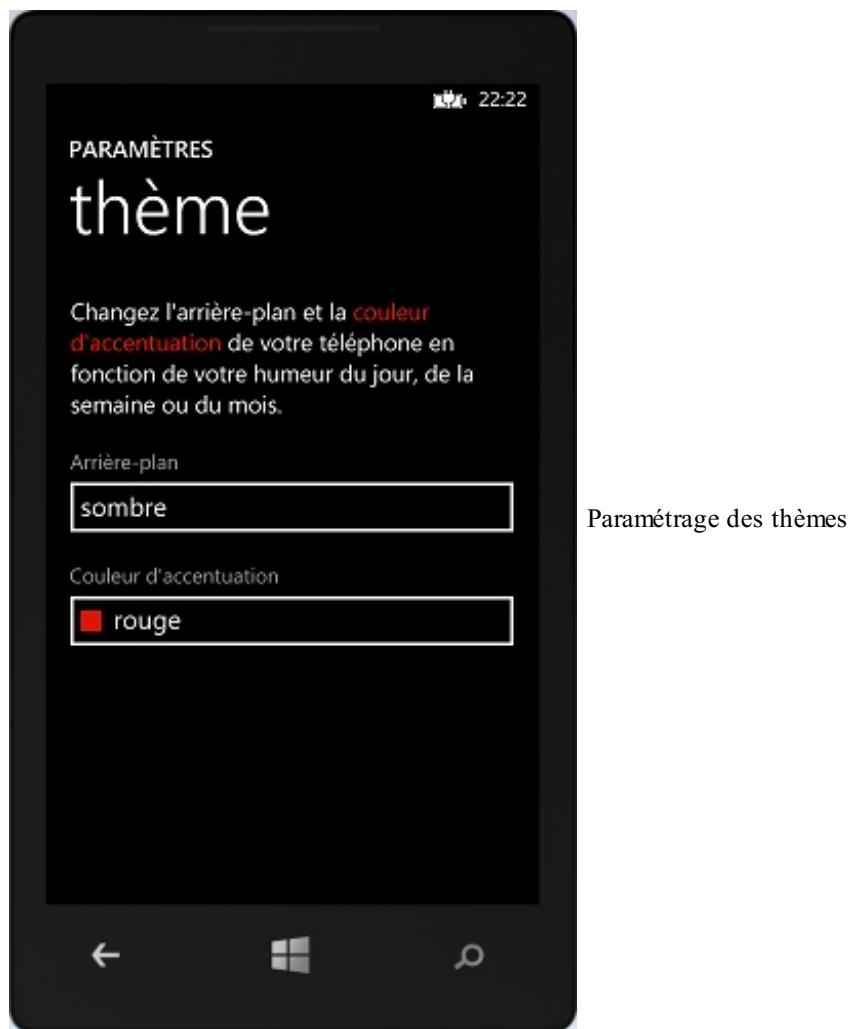
Accès au menu de

paramétrage de l'émulateur

Puis sur thème (voir la figure suivante).



On obtient cet écran (voir la figure suivante).



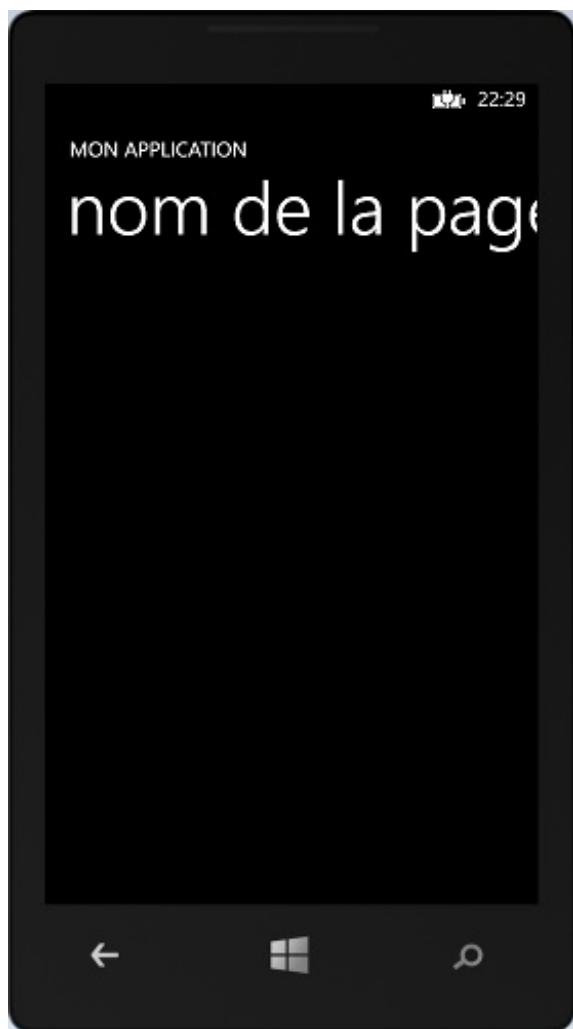
Il est possible de choisir le thème, soit sombre soit clair puis la couleur d'accentuation (voir la figure suivante).



Modification de la couleur d'accentuation

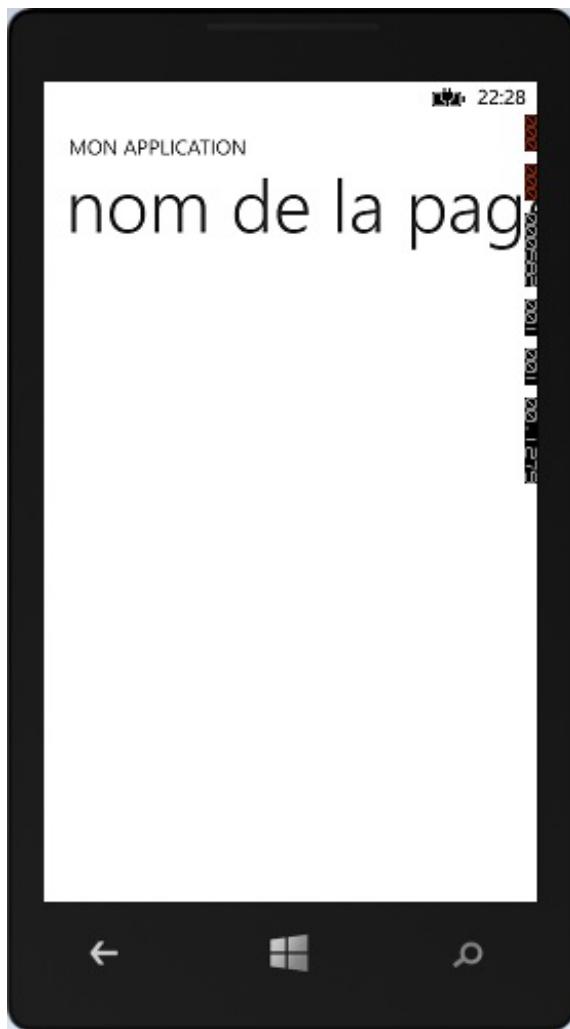
Qu'est-ce que ça veut dire ? Eh bien cela veut dire qu'on ne peut pas faire n'importe quoi avec les couleurs et surtout pas n'importe comment. Par exemple, si j'écris du texte de couleur blanche, cela passera très bien avec mon thème sombre, mais cela deviendra invisible avec un thème clair.

Les contrôles de Windows Phone savent gérer ces différents thèmes sans aucun problème grâce aux styles systèmes qui savent s'adapter aux différents thèmes, comme par exemple les styles `PhoneTextNormalStyle` et `PhoneTextTitle1Style`. Ainsi, si vous lancez votre application fraîchement créée en mode sombre, vous aurez les titres suivants (voir la figure suivante).



Le titre est blanc sur fond noir en mode sombre

Alors qu'en mode clair, vous aurez la figure suivante.



Le titre est noir sur fond blanc en mode clair

Les couleurs sont différentes, c'est le style qui gère les différents thèmes. Il est possible de détecter le thème de l'application afin d'adapter nos designs, par exemple en changeant une image ou en changeant une couleur, etc.

Pour ce faire, on peut utiliser la technique suivante :

Code : C#

```
Visibility darkBackgroundVisibility =  
    (Visibility)Application.Current.Resources["PhoneDarkThemeVisibility"];  
if (darkBackgroundVisibility == Visibility.Visible)  
{  
    // le thème est sombre  
}  
else  
{  
    // le thème est clair  
}
```

De même, on peut récupérer la couleur d'accentuation choisie afin de l'utiliser sur nos pages, par exemple :

Code : C#

```
Color couleur =  
    (Color)Application.Current.Resources["PhoneAccentColor"];  
MonTextBox.Foreground = new SolidColorBrush(couleur);
```

Changer l'apparence de son contrôle

Il n'y a pas que les styles qui permettent de changer l'apparence d'un contrôle. Rappelez-vous, nous avons dit que certains

contrôles dérivaient de la classe `ContentControl`. Il s'agit de contrôles qui contiennent d'autres objets. C'est le cas du bouton par exemple. Il est possible de modifier son apparence sans changer ses fonctionnalités. C'est une des grandes forces du XAML. Il suffit de redéfinir la propriété `Content` du bouton... Jusqu'à présent, un bouton c'était ce XAML (à l'intérieur d'un `StackPanel`) :

Code : XML

```
<Button Content="Cliquez-moi !" />
```

Qui donnait la figure suivante.



Nous avons mis une chaîne de caractères dans la propriété `Content`. Cette propriété est de type `object`, il est donc possible d'y mettre n'importe quoi. En l'occurrence, on peut y mettre un `TextBlock` qui donnera le même résultat visuel :

Code : XML

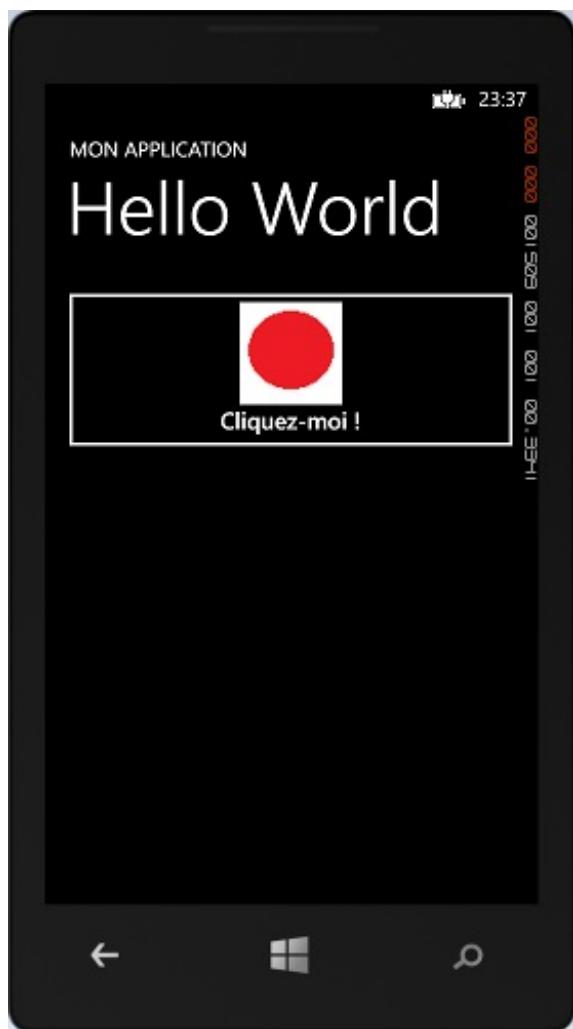
```
<Button>
    <Button.Content>
        <TextBlock Text="Cliquez-moi" />
    </Button.Content>
</Button>
```

Si on peut mettre un `TextBlock`, on peut mettre n'importe quoi et c'est ça qui est formidable. Par exemple, on peut facilement mettre une image. Reprenons notre rond rouge du début du chapitre, puis utilisez le XAML suivant :

Code : XML

```
<Button>
    <Button.Content>
        <StackPanel>
            <Image Source="/Assets/Images/rond.png" Width="100"
Height="100" />
            <TextBlock Text="Cliquez-moi !" />
        </StackPanel>
    </Button.Content>
</Button>
```

Nous obtenons un bouton tout à fait fonctionnel possédant une image et un texte (voir la figure suivante).



Un bouton avec une image

Nous n'avons rien eu d'autre à faire que de modifier l'objet `Content` et ce bouton continue à se comporter comme un bouton classique.

Remarquons avant de terminer qu'il est possible de pousser la personnalisation encore plus loin grâce aux modèles (en anglais template) que nous verrons plus loin.

- Les styles sont un élément très puissant nous permettant de modifier l'apparence de nos contrôles.
- On peut changer l'apparence des contrôles de type `ContentControl` pour créer un autre look tout en conservant la fonctionnalité du contrôle.
- Il faut faire attention aux différents thèmes d'une application et toujours vérifier que ce qu'on souhaite afficher soit bien visible en fonction des thèmes et de la couleur d'accentuation.

TP1 : Création du jeu du plus ou du moins

Bienvenue dans ce premier TP ! Vous avez pu découvrir dans les chapitres précédents les premières bases du XAML permettant la construction d'applications Windows Phone. Il est grand temps de mettre en pratique ce que nous avons appris. C'est ici l'occasion pour vous de tester vos connaissances et de valider ce que vous appris en réalisant cet exercice.

Pour l'occasion, nous allons réaliser un petit jeu, le classique jeu du plus ou du moins.

Instructions pour réaliser le TP

Voici donc un petit TP sous forme de création d'un jeu simple qui va vous permettre de vous entraîner. L'idée est de réaliser le jeu classique du plus ou du moins... Je vous rappelle les règles. L'ordinateur calcule un nombre aléatoire et nous devons le deviner. À chaque saisie, il nous indique si le nombre saisi est plus grand ou plus petit que le nombre à trouver, ainsi que le nombre de coups. Une fois trouvé, il nous indique que nous avons gagné.

Nous allons donc pouvoir utiliser nos connaissances en XAML pour créer une interface graphique permettant de réaliser ce jeu. Nous aurons bien sûr besoin d'un `TextBox` pour obtenir la saisie de l'utilisateur. Vous pouvez ensuite utiliser un `TextBlock` pour donner les instructions, qui pourront être de la couleur d'accentuation. De même, vous utiliserez un autre `TextBlock` pour afficher le nombre de coups. Vous pourrez utiliser un bouton afin de vérifier le résultat et un autre bouton pour recommencer une partie.

Pour rappel, vous pouvez obtenir un nombre aléatoire en instanciant un objet `Random` et en appelant la méthode `Next` :

Code : C#

```
Random random = new Random();
int valeurSecrete = random.Next(1, 500);
```

Vous pouvez choisir les bornes que vous voulez, mais de 1 à 500 me paraît pas trop mal.

N'oubliez pas de gérer le cas où l'utilisateur saisit n'importe quoi. Nous ne voudrions pas que notre premier jeu sur Windows Phone ait un bug qui fasse planter l'application !

C'est à vous de jouer. Bon courage.

Correction

Alors, comment était ce TP ? Pas trop difficile, non ?

Alors, voyons ma correction. Il y a plusieurs façons de réaliser ce TP ainsi qu'une infinité de mise en page possible. J'ai choisi un look très simple, mais n'hésitez pas à faire parler votre créativité.

Commençons par le XAML :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP du jeu du plus ou du moins" Style="{StaticResource PhoneTextTitle2Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="Veuillez saisir une valeur (entre 0 et 500)" Style="{StaticResource PhoneTextNormalStyle}" HorizontalAlignment="Center" />
            <TextBox x:Name="Valeur" InputScope="Number" />
            <Button Content="Vérifier" Tap="Button_Tap_1" />
            <TextBlock x:Name="Indications" Height="50" TextWrapping="Wrap" />
        </StackPanel>
    </Grid>
</Grid>
```

```
<TextBlock x:Name="NombreDeCoups" Height="50"  
TextWrapping="Wrap" Style="{StaticResource PhoneTextNormalStyle}" />  
</StackPanel>  
</Grid>  
<Button Content="Rejouer" Tap="Button_Tap_2" Grid.Row="2"/>  
</Grid>
```

Il s'agit de disposer mes contrôles de manière à obtenir ce rendu (voir la figure suivante).



Rendu du TP du jeu du plus ou du moins dans l'émulateur

Ce qu'il est important de voir ici c'est que tous mes `TextBlock` possèdent un style qui sait gérer les thèmes, sauf celui pour les indications car c'est par code que je vais positionner la couleur.

Remarquez également que le `TextBox` affichera un clavier numérique grâce à l'`InputScope` qui vaut `Number`.

Passons à présent au code behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage  
{  
    private Random random;  
    private int valeurSecrete;  
    private int nbCoups;  
  
    public MainPage()  
    {  
        InitializeComponent();  
  
        random = new Random();
```

```
valeurSecrete = random.Next(1, 500);
nbCoups = 0;
Color couleur =
(Color)Application.Current.Resources["PhoneAccentColor"];
Indications.Foreground = new SolidColorBrush(couleur);
}

private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    int num;
    if (int.TryParse(Valeur.Text, out num))
    {
        if (valeurSecrete == num)
        {
            Indications.Text = "Gagné !!";
        }
        else
        {
            nbCoups++;
            if (valeurSecrete < num)
                Indications.Text = "Trop grand ...";
            else
                Indications.Text = "Trop petit ...";
            if (nbCoups == 1)
                NombreDeCoups.Text = nbCoups + " coup";
            else
                NombreDeCoups.Text = nbCoups + " coups";
        }
    }
    else
        Indications.Text = "Veuillez saisir un entier ...";
}

private void Button_Tap_2(object sender,
System.Windows.Input.GestureEventArgs e)
{
    valeurSecrete = random.Next(1, 500);
    nbCoups = 0;
    Indications.Text = string.Empty;
    NombreDeCoups.Text = string.Empty;
    Valeur.Text = string.Empty;
}
```

La classe `Color` et la classe `SolidColorBrush` nécessitent l'import suivant :

Code : C#

```
using System.Windows.Media;
```

Le jeu en lui-même ne devrait pas avoir posé trop de problèmes. L'algorithme est classique, on commence par déterminer un nombre aléatoire puis à chaque demande de vérification, on transforme la valeur saisie en entier, afin de vérifier que l'utilisateur n'a pas saisi n'importe quoi. Avec le clavier numérique, les erreurs sont limitées, mais elles sont encore possibles car on demande des entiers et l'utilisateur a la possibilité de saisir des nombres à virgule. Puis on compare et on indique le résultat (voir la figure suivante).



Une partie en cours de jeu ...

Et voilà pour notre premier TP. Vous avez pu voir comme il est finalement assez simple de créer des petits programmes sur nos téléphones grâce au XAML et au C#.

Dessiner avec le XAML

En plus des contrôles, le XAML possède les formes (en anglais Shape). Elles permettent de dessiner différentes formes sur nos pages. Voyons à présent comment cela fonctionne.

Dessin 2D

Il existe plusieurs types de formes. Elles sont représentées par des classes qui dérivent toutes d'une classe abstraite de base : **Shape**. **Shape** est un élément affichable sur une page dans la mesure où elle dérive, comme les contrôles, de **FrameworkElement** et de **UIElement**. Nous avons à notre disposition :

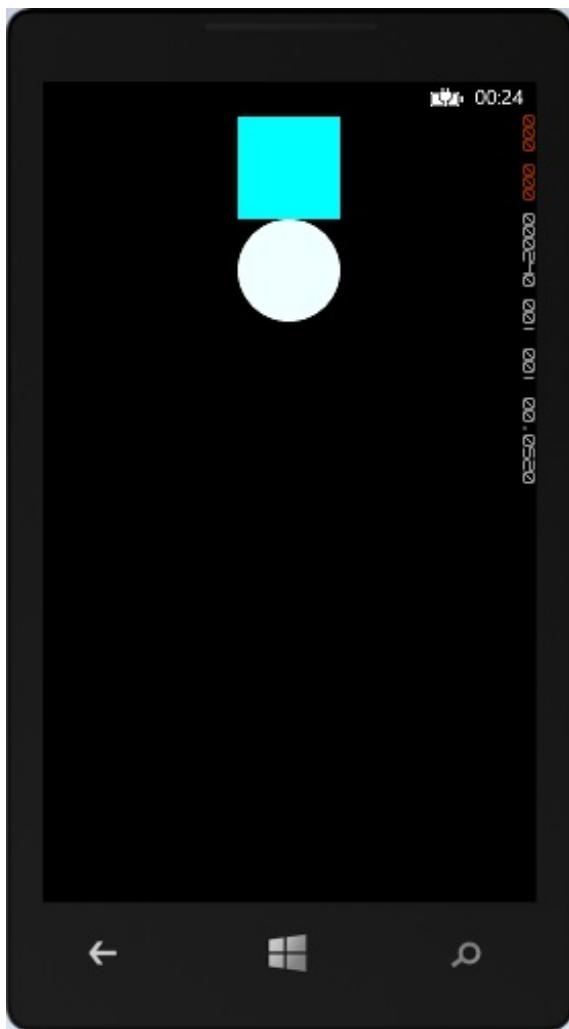
- Les ellipses et cercles via la classe **Ellipse**
- Les lignes, via la classe **Line**
- Plusieurs lignes ou courbes connectées, via la classe **Path**. **Path** pouvant être traduit en tracé, il s'agit d'un tracé de lignes ou de courbes.
- Des lignes connectées via la classe **PolyLine**
- Les polygones, via la classe **Polygon**. La différence avec le **PolyLine** est que la forme se termine en reliant le dernier trait au premier.
- Des rectangles via la classe **Rectangle**

Si vous vous rappelez, nous avons utilisé la classe **Rectangle** dans un précédent chapitre pour illustrer les marges. Dessinons par exemple un carré et un cercle. Pour cela, je peux utiliser les classes **Rectangle** et **Ellipse** :

Code : XML

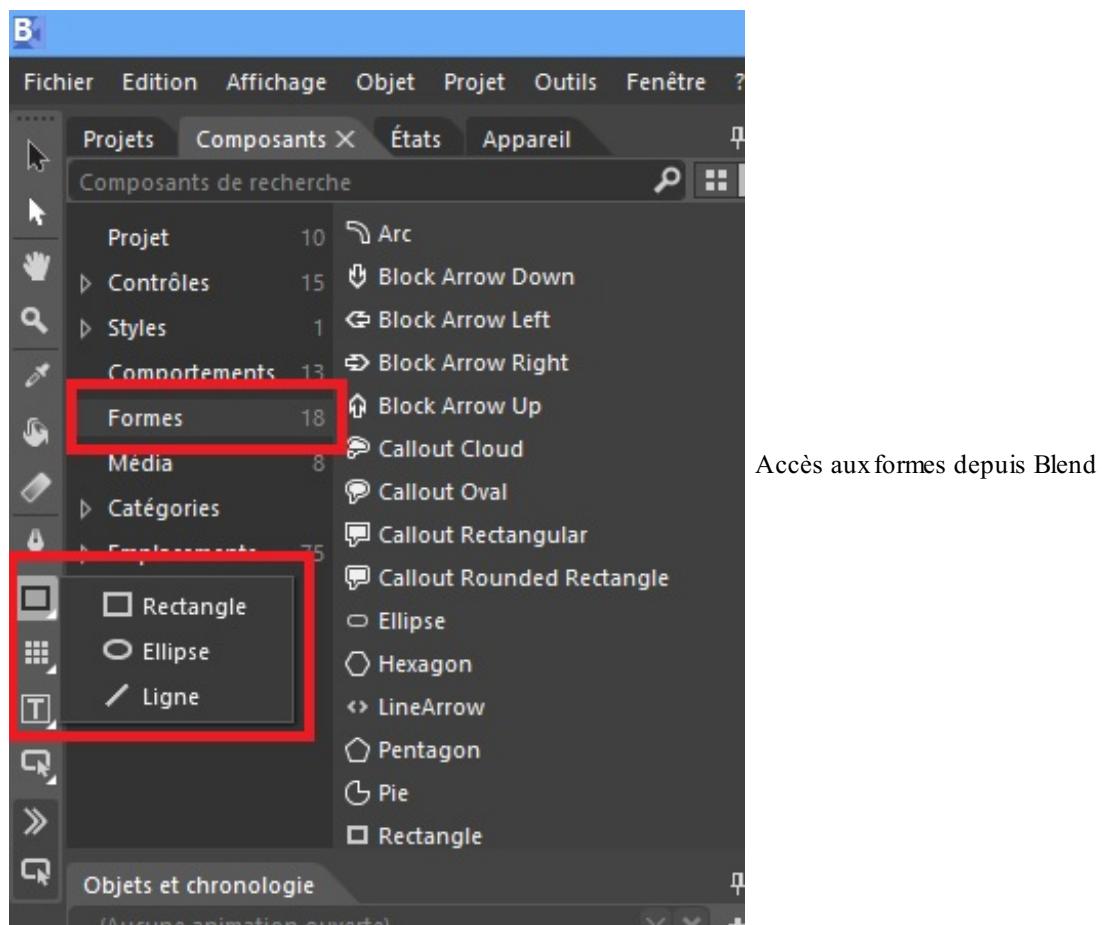
```
<StackPanel>
    <Rectangle Width="100" Height="100" Fill="Aqua" />
    <Ellipse Height="100" Width="100" Fill="Azure" />
</StackPanel>
```

Ce qui nous donne la figure suivante.



Affichage d'un rectangle et d'une ellipse grâce à leurs contrôles respectifs

Remarquons que la propriété `Fill` permet de colorer les formes. Nous allons y revenir. Mais le plus simple est encore d'utiliser Blend pour ce genre de choses.
Vous avez accès aux formes soit dans l'onglet des composants, soit en cliquant sur le rectangle (voir la figure suivante).

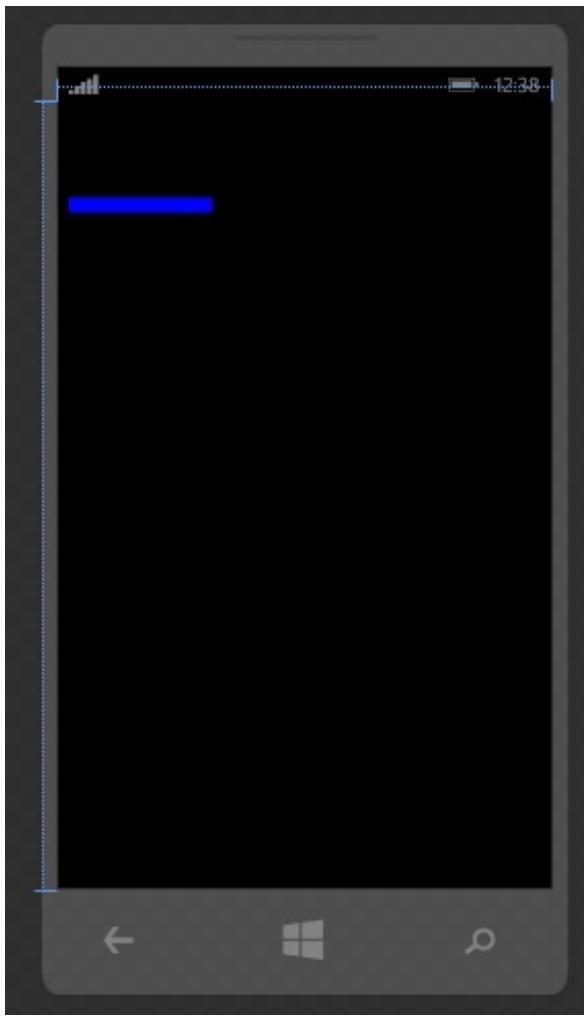


Blend est votre meilleur allié pour dessiner sur vos pages. N'oubliez pas qu'il est capable d'exploiter le XAML que vous avez saisi à la main dans Visual Studio, par exemple :

Code : XML

```
<Canvas>
    <Line X1="10" Y1="100" X2="150" Y2="100" Stroke="Blue"
        StrokeThickness="15"/>
</Canvas>
```

qui va nous permettre de tracer une ligne bleue horizontale d'épaisseur 15. Nous la voyons apparaître dans Blend (voir la figure suivante).



Affichage d'une ligne bleue dans Blend

Je vais m'arrêter là pour les exemples de formes car la documentation en ligne possède des exemples qui sont plutôt simples à comprendre. Vous allez d'ailleurs voir dans le prochain chapitre un exemple de polygone.

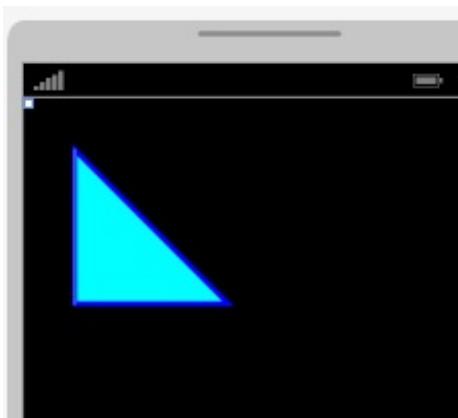
Pinceaux

Les pinceaux vont nous permettre de colorier nos formes. Nous avons rapidement vu tout à l'heure que nous pouvions colorier nos formes grâce à la propriété `Fill`. Par exemple, le XAML suivant :

Code : XML

```
<Canvas>
    <Polygon Points="50,50 200, 200 50,200" Fill="Aqua"
Stroke="Blue" StrokeThickness="5" />
</Canvas>
```

dessine un triangle rectangle de couleur Aqua dont le trait est bleu, d'épaisseur 5 (voir la figure suivante).



Le triangle est coloré grâce au pinceau Aqua

En fait, Aqua et Blue sont des objets dérivés de la classe `Brush`, en l'occurrence ici il s'agit d'une `SolidColorBrush`. Comme on l'a déjà vu, on peut donc écrire notre précédent pinceau de cette façon :

Code : XML

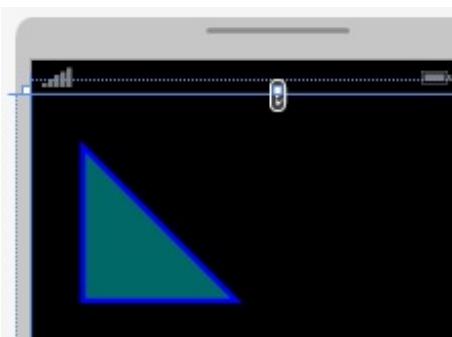
```
<Polygon Points="50,50 200, 200 50,200" Stroke="Blue"
StrokeThickness="5">
  <Polygon.Fill>
    <SolidColorBrush Color="Aqua" />
  </Polygon.Fill>
</Polygon>
```

Ce qui nous offre un meilleur contrôle sur le pinceau. Nous pouvons par exemple changer l'opacité et la passer de 1 (valeur par défaut) à 0.4 par exemple :

Code : XML

```
<Polygon Points="50,50 200, 200 50,200" Stroke="Blue"
StrokeThickness="5">
  <Polygon.Fill>
    <SolidColorBrush Color="Aqua" Opacity="0.4" />
  </Polygon.Fill>
</Polygon>
```

Et nous pouvons voir que la couleur est un peu plus transparente (voir la figure suivante).



L'opacité joue sur la transparence du contrôle



Toutes les propriétés commençant par `Stroke` se rapportent au trait de la forme. Par exemple, `Stroke` permet de modifier la couleur du trait, `StrokeThickness` permet de modifier l'épaisseur du trait ou encore `StrokeDash` que nous n'avons pas vu qui permet de modifier l'apparence d'un trait (pointillés, flèches aux extrémités, ...).

Vous vous en doutez, il existe d'autres pinceaux que le pinceau uni. Nous avons également à notre disposition :

- Un gradient linéaire, via la classe `LinearGradientBrush`
- Un gradient radial, via la classe `RadialGradientBrush`
- Une image, via la classe `ImageBrush`
- Une vidéo, via la classe `VideoBrush`

Utilisons par exemple une `ImageBrush` pour afficher la mascotte d'OpenClassrooms dans notre triangle (voir la figure suivante).



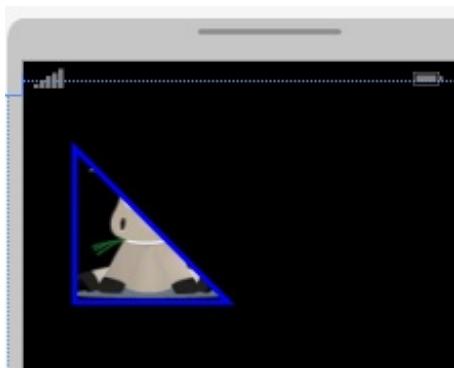
Zozor, la mascotte

Nous aurons le XAML suivant :

Code : XML

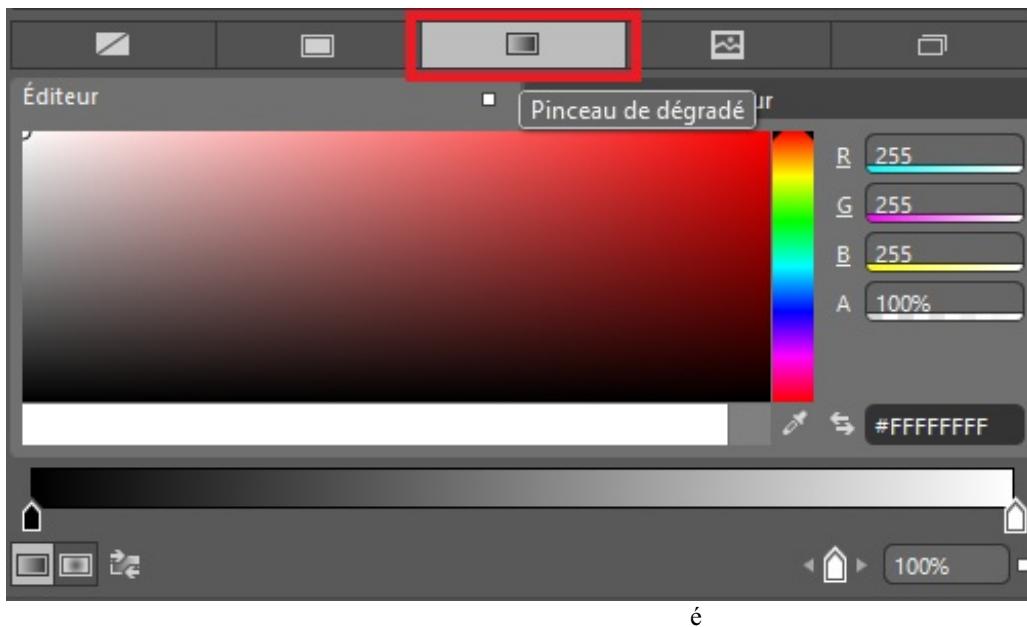
```
<Polygon Points="50,50 200, 200 50,200" Stroke="Blue" StrokeThickness="5">
    <Polygon.Fill>
        <ImageBrush
            ImageSource="http://uploads.siteduzero.com/files/337001_338000/337519.png"
        />
    </Polygon.Fill>
</Polygon>
```

Qui donnera la figure suivante.



Le triangle avec un pinceau utilisant l'image de Zozor

Et voilà comment utiliser une image comme pinceau. Sauf que ce triangle rectangle ne lui rend vraiment pas honneur... ! Pour faire un dégradé, le mieux est d'utiliser Blend. Reprenons notre triangle rectangle et cliquez à droite sur le pinceau de dégradé (voir la figure suivante).



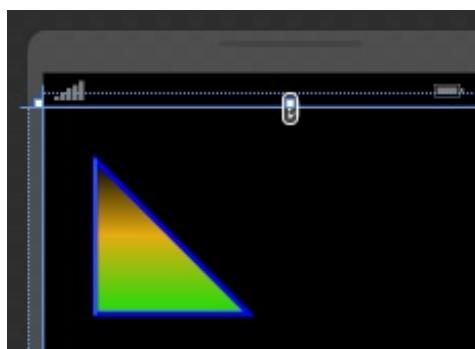
Création d'un pinceau de dégradé

Il ne reste plus qu'à choisir les couleurs de votre dégradé. Il faut vous servir de la bande en bas pour définir les différentes couleurs du dégradé (voir la figure suivante).



Choix du dégradé

Et nous aurons un mâââagnifique triangle dégradé (voir la figure suivante) !



Le triangle avec le pinceau dégradé

Notons que le XAML généré est le suivant :

Code : XML

```
<Polygon Points="50,50 200, 200 50,200" Stroke="Blue"
```

```
StrokeThickness="5">
    <Polygon.Fill>
        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
            <GradientStop Color="Black" Offset="0"/>
            <GradientStop Color="#FF1FDC0C" Offset="1"/>
            <GradientStop Color="#FFE8AD11" Offset="0.488"/>
        </LinearGradientBrush>
    </Polygon.Fill>
</Polygon>
```

Voilà pour ce petit tour des pinceaux.

Les transformations

Le XAML possède un système de transformations qui permet d'agir sur les contrôles. Il existe plusieurs types de transformations dites affines car elles conservent la structure originale du contrôle. Il est par exemple possible d'effectuer :

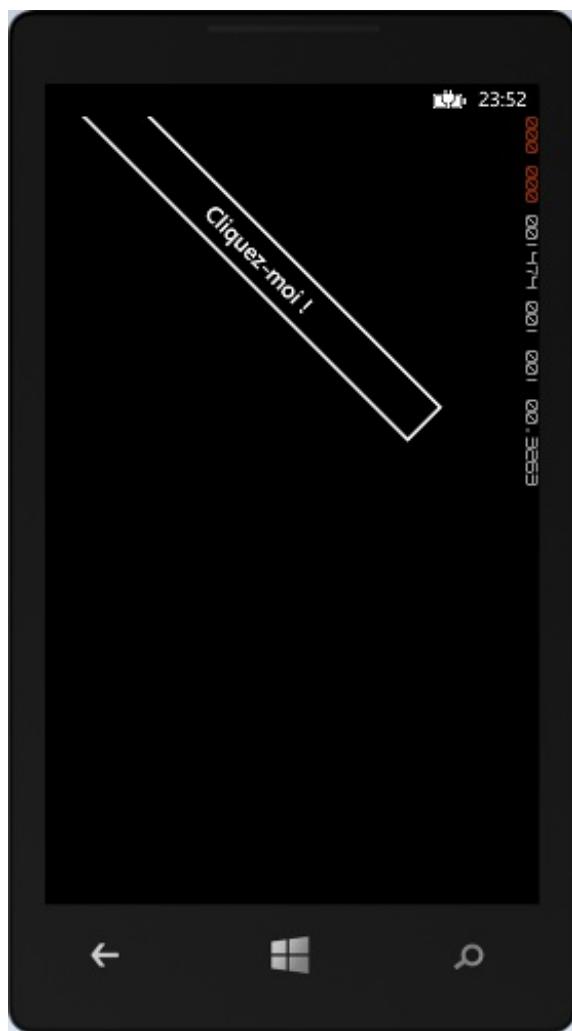
- une rotation grâce à la classe `RotateTransform`
- une translation grâce à la classe `TranslateTransform`
- une mise à l'échelle grâce à la classe `ScaleTransform`
- une inclinaison grâce à la classe `SkewTransform`
- Une transformation matricielle grâce à la classe `MatrixTransform`

Par exemple, pour faire pivoter un bouton de 45°, je peux utiliser le code suivant :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <StackPanel>
        <Button Content="Cliquez-moi !">
            <Button.RenderTransform>
                <RotateTransform x:Name="Rotation" Angle="45"
CenterX="100" CenterY="50" />
            </Button.RenderTransform>
        </Button>
    </StackPanel>
</Grid>
```

Ce qui nous donne la figure suivante.



Rotation d'un contrôle de 45°

Il suffit de renseigner la propriété `RenderTransform` du contrôle, sachant que cette propriété fait partie de la classe `UIElement` qui est la classe mère de tous les contrôles affichables. Dans cette propriété, on met la classe `RotateTransform` en lui précisant notamment l'angle de rotation et les coordonnées du centre de rotation.
Illustrons encore une transformation grâce à la classe `ScaleTransform` pour effectuer un grossissement d'un `TextBlock` :

Code : XML

```
<TextBlock Text="Hello world" />
<TextBlock Text="Hello world">
    <TextBlock.RenderTransform>
        <ScaleTransform ScaleX="3" ScaleY="10" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Qui donne la figure suivante.



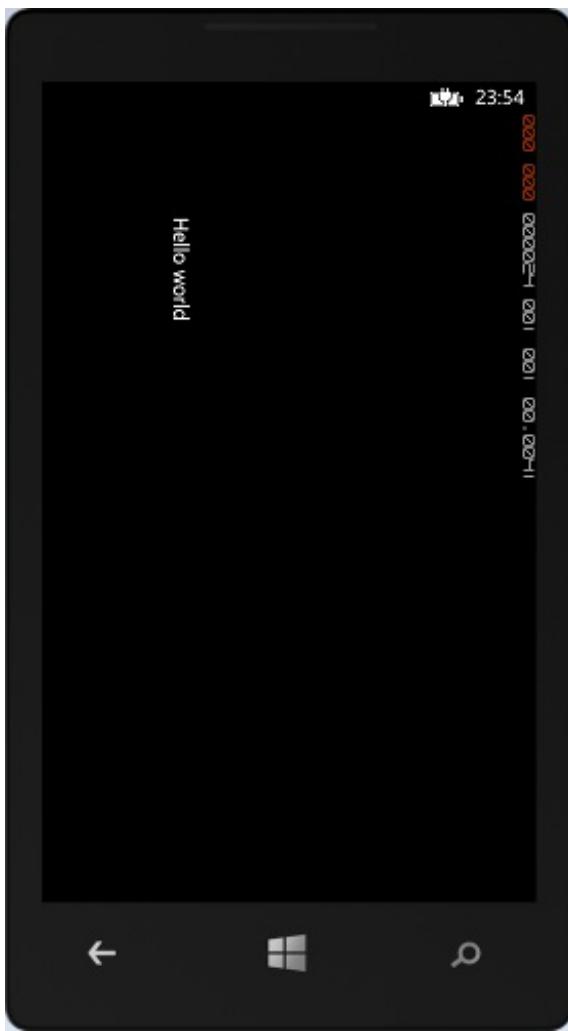
Mise à l'échelle du contrôle

Ces transformations peuvent se combiner grâce à la classe `TransformGroup`, par exemple ici je combine une rotation avec une translation :

Code : XML

```
<TextBlock Text="Hello world">
    <TextBlock.RenderTransform>
        <TransformGroup>
            <RotateTransform Angle="90" />
            <TranslateTransform X="150" Y="100" />
        </TransformGroup>
    </TextBlock.RenderTransform>
</TextBlock>
```

Et nous aurons la figure suivante.



Rotation combinée à une translation

Sachant qu'il est possible de faire la même chose avec une transformation composite, grâce à la classe `CompositeTransform`. Elle s'utilise ainsi :

Code : XML

```
<TextBlock Text="Hello world">
    <TextBlock.RenderTransform>
        <CompositeTransform TranslateX="150" TranslateY="100"
Rotation="90" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Voilà pour les transformations. En soi elles ne sont pas toujours très utiles, mais elles révèlent toutes leurs puissances grâce aux animations que nous découvrirons dans le chapitre suivant.

- Le XAML possède plein de formes que nous pouvons utiliser pour dessiner dans nos applications, comme le trait, l'ellipse, le rectangle, etc.
- À chaque forme peut être appliquée une couleur de remplissage ou de traits grâce aux pinceaux
- Il est également possible de faire subir des transformations à un contrôle comme une rotation ou une translation.

Créer des animations

Des contrôles, du dessin ... nous sommes presque prêts à réaliser des jolies interfaces en laissant parler notre créativité. Mais tout cela manque un peu de dynamique, de trucs qui bougent et nous en mettent plein la vue.

Le XAML nous a entendu ! Grâce à lui, il est très facile de créer des animations. Elles vont nous servir à mettre en valeur certains éléments, ou réaliser un effet de transition en rajoutant du mouvement et de l'interactivité. Bref, de quoi innover un peu et embellir vos applications.

Nous allons découvrir dans ce chapitre comment tout cela fonctionne et comment réaliser nos propres animations directement en manipulant le XAML, ou encore grâce à l'outil professionnel de design : Expression Blend. Soyez prêt à ce que ça bouge 😊

Principe généraux des animations

Une animation consiste à faire varier les propriétés d'un contrôle dans un temps précis. Par exemple, si je veux faire bouger un contrôle dans un Canvas, je vais pouvoir faire varier les propriétés `Canvas.Top` et `Canvas.Left`. De la même façon, si je veux faire disparaître un élément avec ce que l'on appelle communément l'effet « fade », je vais pouvoir faire varier la propriété d'opacité d'un contrôle.

Pour cela, le XAML possède plusieurs classes qui vont nous être utiles. Des classes permettant de faire varier une propriété de type couleur, une propriété de type double et une propriété de type Point qui sont respectivement les classes `ColorAnimation`, `DoubleAnimation` et `PointAnimation`.



Il n'est possible d'animer que ces trois types de valeur.

Pour fonctionner, elles ont besoin d'une autre classe qui s'occupe de contrôler les animations afin d'indiquer leurs cibles et la planification de l'animation. Il s'agit de la classe `Storyboard` dont le nom explicite rappelle un peu les projets de montage audio ou vidéo. C'est le même principe, c'est elle qui va cadencer les différentes animations.

Création d'une animation simple (XAML)

Pour illustrer les animations, je vais vous montrer l'effet de disparition (« fade ») appliqué à un contrôle. Créons donc un contrôle, par exemple un `StackPanel` contenant un bouton et un `TextBlock`. J'ai besoin également d'un autre bouton qui va me permettre de déclencher l'animation :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="LeStackPanel">
        <Button Content="Cliquez-moi !" />
        <TextBlock Text="Je vais bientôt disparaître ..." />
    </StackPanel>
    <StackPanel Grid.Row="1">
        <Button Content="Démarrer l'animation" Tap="Button_Tap" />
    </StackPanel>
</Grid>
```

Nous allons maintenant créer notre Storyboard. Celui-ci doit se trouver en ressources. Comme on l'a déjà vu, vous pouvez le mettre en ressources de l'application, de la page ou bien en ressources d'un contrôle parent. Mettons-le dans les ressources de la grille :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.Resources>
```

```
<Storyboard x:Name="MonStoryboard">
</Storyboard>
</Grid.Resources>

<StackPanel x:Name="LeStackPanel">
    <Button Content="Cliquez-moi !" />
    <TextBlock Text="Je vais bientôt disparaître ..." />
</StackPanel>
<StackPanel Grid.Row="1">
    <Button Content="Démarrer l'animation" Tap="Button_Tap" />
</StackPanel>
</Grid>
```

Ce Storyboard doit avoir un nom afin d'être manipulé par le clic sur le bouton. Il faut maintenant définir l'animation :

Code : XML

```
<Storyboard x:Name="MonStoryboard">
    <DoubleAnimation From="1.0" To="0.0" Duration="0:0:2"
        Storyboard.TargetName="LeStackPanel"
        Storyboard.TargetProperty="Opacity"/>
</Storyboard>
```

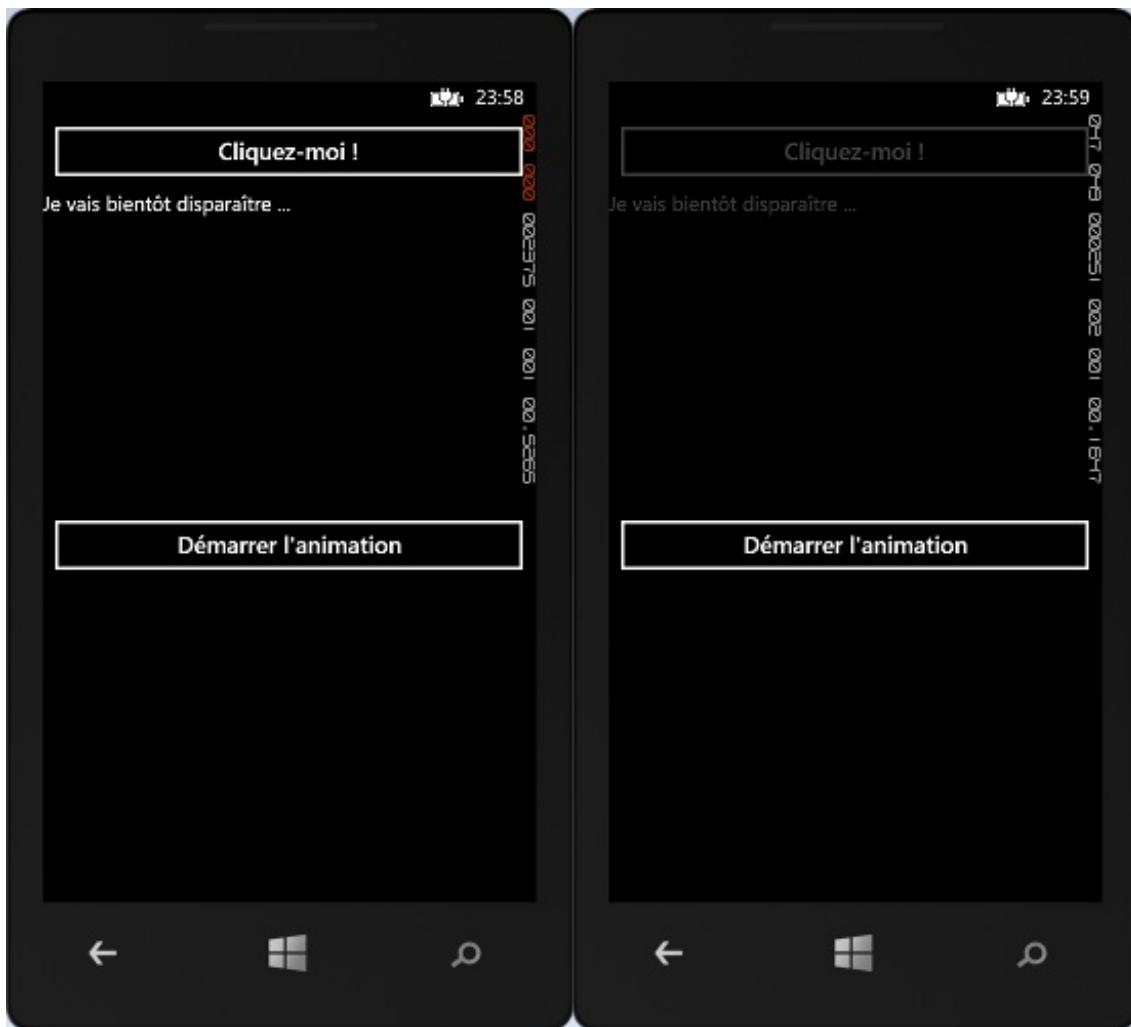
Il s'agit d'une animation de type double où nous allons animer la propriété `Opacity` pour la faire aller de la valeur 1 (visible) à la valeur 0 (invisible) pendant une durée de deux secondes, ciblant notre contrôle nommé `LeStackPanel`.

Il faut maintenant déclencher l'animation lors du clic sur le bouton :

Code : C#

```
private void Button_Tap(object sender,
    System.Windows.Input.GestureEventArgs e)
{
    MonStoryboard.Begin();
}
```

Difficile de vous faire une copie d'écran du résultat mais n'hésitez pas à essayer par vous-même (voir la figure suivante).



fait disparaître le contrôle

Il est possible de faire en sorte que l'animation se joue en boucle et de manière indéfinie. Il suffit de rajouter les propriétés AutoReverse et RepeatBehavior. Par exemple, ici je vais animer un bouton de manière à ce qu'il se déplace de gauche à droite et de droite à gauche indéfiniment.

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.Resources>
        <Storyboard x:Name="MonStoryboard">
            <DoubleAnimation From="0" To="200" Duration="0:0:3"
                AutoReverse="True"
                RepeatBehavior="Forever"
                Storyboard.TargetName="MonBouton"
                Storyboard.TargetProperty="(Canvas.Left)"/>
        </Storyboard>
    </Grid.Resources>

    <Canvas Width="480" Height="500" x:Name="LeCanvas">
        <Button x:Name="MonBouton" Content="Cliquez-moi !" />
    </Canvas>
    <StackPanel Grid.Row="1">
        <Button Content="Démarrer l'animation" Tap="Button_Tap" />
    </StackPanel>
</Grid>
```



J'en profite pour indiquer que pour animer une propriété complexe, il faut la saisir entre parenthèses. Je reviendrai sur ce type de propriété plus loin dans le cours.

Nous pouvons contrôler plus finement une animation. Jusqu'à présent, nous avons utilisé la méthode `Begin()` pour démarrer une animation. Vous pouvez également utiliser la méthode `Stop()` pour arrêter une animation, la méthode `Pause()` pour la mettre en pause et la méthode `Resume()` pour la reprendre. Vous pouvez également faire des animations de transformations. Il suffit de combiner l'utilisation des transformations et d'une `DoubleAnimation`. Par exemple, ici je vais faire tourner mon bouton de 90 degrés et le faire revenir à sa position initiale :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.Resources>
        <Storyboard x:Name="MonStoryboard">
            <DoubleAnimation From="0" To="90" Duration="0:0:1"
                AutoReverse="True"
                Storyboard.TargetName="Rotation"
                Storyboard.TargetProperty="Angle"/>
        </Storyboard>
    </Grid.Resources>

    <StackPanel>
        <Button x:Name="MonBouton" Content="Cliquez-moi !">
            <Button.RenderTransform>
                <RotateTransform x:Name="Rotation" Angle="0" />
            </Button.RenderTransform>
        </Button>
    </StackPanel>
    <StackPanel Grid.Row="1">
        <Button Content="Démarrer l'animation" Tap="Button_Tap" />
    </StackPanel>
</Grid>
```

Il suffit de cibler la propriété `Angle` de l'objet `RotateTransform`.

Si vous voulez qu'une animation démarre à partir d'un certain temps, vous pouvez rajouter la propriété `BeginTime` au `Storyboard` :

Code : XML

```
<Storyboard x:Name="MonStoryboard" BeginTime="0:0:2">
    <DoubleAnimation From="0" To="90" Duration="0:0:1"
        AutoReverse="True"
        Storyboard.TargetName="Rotation"
        Storyboard.TargetProperty="Angle"/>
</Storyboard>
```

Par exemple ici, l'animation va durer une seconde et démarrera deux secondes après son déclenchement via la méthode `Begin()`.

On peut contrôler plus finement une animation grâce aux animations dites « Key Frame » qui permettent d'indiquer différents moments clés d'une animation. Il est possible ainsi de spécifier la valeur de la propriété animée à un moment T. On utilisera les trois types d'animations suivantes :

- `DoubleAnimationUsingKeyFrames`
- `ColorAnimationUsingKeyFrames`
- `PointAnimationUsingKeyFrames`

Chacune de ces animations peut être de trois types : Linear, Discret et Spline.

L'animation linéaire se rapproche des animations que nous avons vues précédemment dans la mesure où entre les moments clés, l'animation passera par toutes les valeurs séparant les deux valeurs des moments clés.

On pourrait illustrer ceci en simulant un « secouage » de bouton afin d'attirer l'attention de l'utilisateur. Le « secouage » va consister à faire une rotation de X degrés dans le sens horaire, puis revenir à la position initiale, puis faire la rotation de X degrés dans le sens anti horaire et enfin revenir à la position initiale. Il y a donc cinq moments clés dans cette animation :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.Resources>
        <Storyboard x:Name="MonStoryboard" >
            <DoubleAnimationUsingKeyFrames
                Storyboard.TargetName="Rotation" Storyboard.TargetProperty="Angle"
                RepeatBehavior="5x">
                <LinearDoubleKeyFrame Value="0" KeyTime="00:00:00" />
                <LinearDoubleKeyFrame Value="5"
                    KeyTime="00:00:00.1" />
                <LinearDoubleKeyFrame Value="0"
                    KeyTime="00:00:00.2" />
                <LinearDoubleKeyFrame Value="-5"
                    KeyTime="00:00:00.3" />
                <LinearDoubleKeyFrame Value="0"
                    KeyTime="00:00:00.4" />
            </DoubleAnimationUsingKeyFrames>
        </Storyboard>
    </Grid.Resources>

    <StackPanel>
        <Button x:Name="MonBouton" Content="Cliquez-moi !" Width="200" Height="100">
            <Button.RenderTransform>
                <RotateTransform x:Name="Rotation" Angle="0" CenterX="100" CenterY="50" />
            </Button.RenderTransform>
        </Button>
    </StackPanel>
    <StackPanel Grid.Row="1">
        <Button Content="Démarrer l'animation" Tap="Button_Tap" />
    </StackPanel>
</Grid>
```

Étant donné que nous animons un angle, nous utiliserons la classe DoubleAnimationUsingKeyFrames. Vu que nous voulons des transitions linéaires pour une animation de type double, nous pourrons utiliser la classe LinearDoubleKeyFrame pour indiquer nos moments clés. Ainsi, j'indique qu'au moment 0, l'angle sera de 0 degrés. Une fraction de seconde plus tard, l'angle sera de 5 degrés. Au bout de deux fractions de seconde, l'angle sera à nouveau à 0 degrés. Une fraction de seconde plus tard, l'angle sera de -5 degrés et enfin, une fraction de seconde plus tard, l'angle reviendra à sa position initiale.

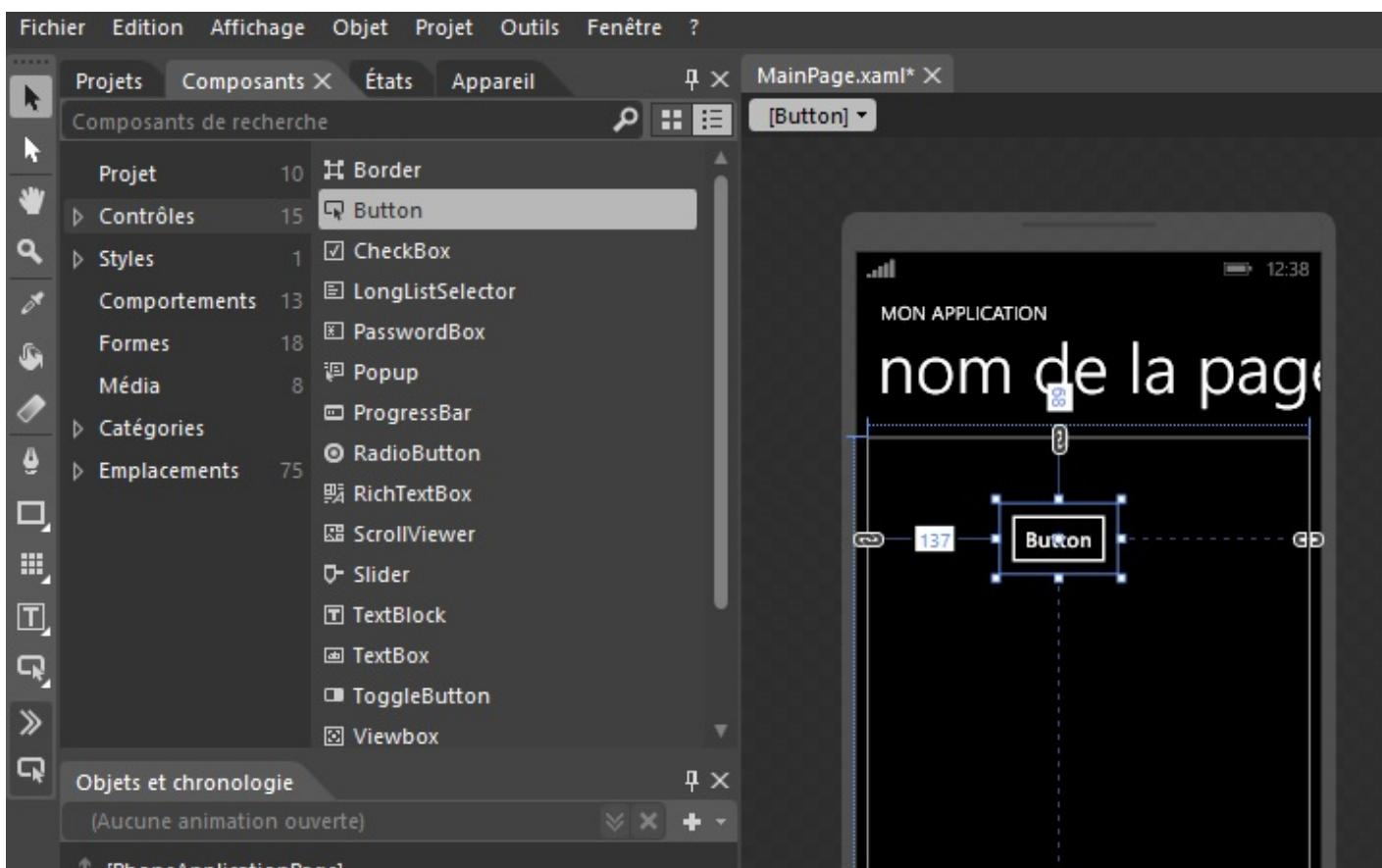
À noter que cette animation sera jouée 5 fois grâce à la propriété RepeatBehavior="5x".

Il y aurait encore beaucoup de choses à dire sur ce genre d'animations, mais nous allons à présent découvrir comment réaliser des animations grâce à blend.

Création d'une animation complexe (Blend)

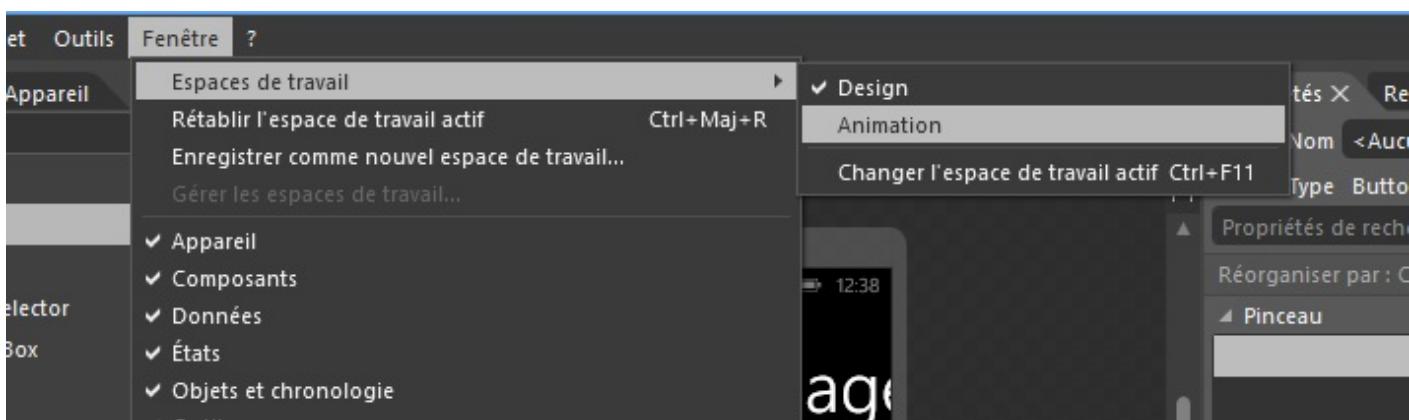
Expression Blend, en sa qualité de logiciel de design professionnel facilite la création d'animations. Nous allons créer une petite animation inutile pour illustrer son fonctionnement. Repartez d'une page toute neuve et ouvrez-là dans Expression Blend. Pour rappel, cliquez-droit sur le fichier XAML et choisissez Ouvrir dans expression blend.

Nous allons à présent ajouter un bouton. Sélectionnez le bouton dans la boîte à outils et faites le glisser sur la surface de la page (voir la figure suivante).



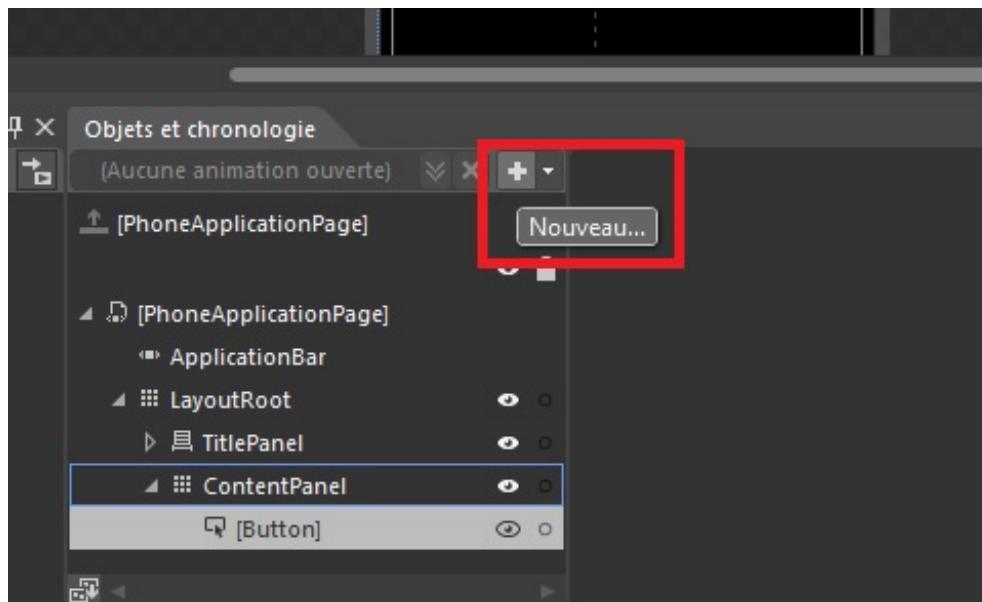
Ajout d'un bouton à partir de Blend

Allez maintenant dans le menu Fenêtre et choisissez espace de travail puis animation afin de passer dans la vue dédiée à l'animation (voir la figure suivante).



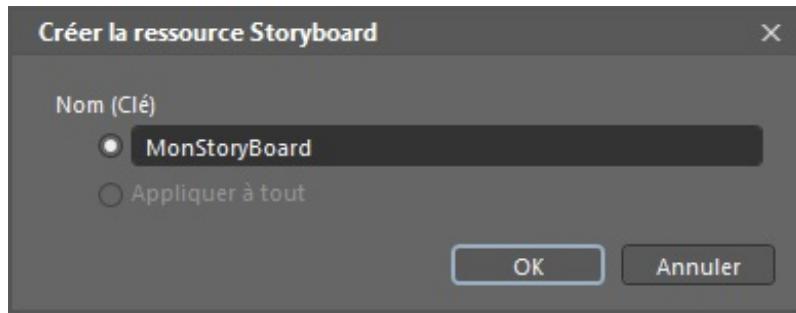
Changement de l'espace de travail

En bas, dans l'onglet Objets et chronologie, cliquez sur le plus pour créer une nouvelle animation (voir la figure suivante).



Création d'une nouvelle animation

Donnez un nom à votre Storyboard, comme indiqué à la figure suivante.



Nommage du storyboard

Il apparaît ensuite en bas à droite la ligne de temps qui va nous permettre de définir des images clés (voir la figure suivante).



La ligne de temps du storyboard

Déplacez le trait jaune qui est sous le chiffre zéro pour le placer sous le chiffre un, en le sélectionnant par le haut de la ligne. Cela nous permet de définir une image clé à la première seconde de l'animation. Nous allons déplacer le bouton vers le bas à droite. Cela signifiera que pendant cette seconde, l'animation fera le trajet de la position 0 à la position 1 correspondant au déplacement du bouton que nous avons réalisé.

Pour voir comment rend l'animation, cliquez sur le petit bouton de lecture en haut de la ligne de temps (voir la figure suivante).



Je ne peux pas vous illustrer le résultat, mais vous devriez voir votre rectangle se déplacer de haut en bas à droite. Essayez ! N'hésitez pas à réduire le zoom si vous ne voyez pas tout l'écran du designer. Et voilà, un début d'animation plutôt simple à faire !

Sauvegardez votre fichier et repassez dans Visual Studio. Vous pouvez voir que le fichier XAML, après rechargement, contient désormais un code qui ressemble au suivant :

Code : XML

```

<phone:PhoneApplicationPage.Resources>
    <Storyboard x:Name="MonStoryboard">
        <DoubleAnimation Duration="0:0:1" To="166"
Storyboard.TargetProperty=" (UIElement.RenderTransform) . (CompositeTransform.TranslateX)" Storyboard.TargetName="button" d:IsOptimized="True"/>
        <DoubleAnimation Duration="0:0:1" To="26"
Storyboard.TargetProperty=" (UIElement.RenderTransform) . (CompositeTransform.TranslateX)" Storyboard.TargetName="button" d:IsOptimized="True"/>
    </Storyboard>
</phone:PhoneApplicationPage.Resources>

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock Text="MON APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" Margin="12,0"/>
        <TextBlock Text="nom de la page" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Button x:Name="button" Content="Button" HorizontalAlignment="Left" Margin="137,68,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.5,0.5">
            <Button.RenderTransform>
                <CompositeTransform/>
            </Button.RenderTransform>
        </Button>
    </Grid>
</Grid>

```

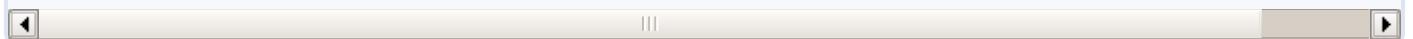
On peut voir qu'il nous a mis une `CompositeTransform` dans le bouton avec une translation sur l'axe des X et sur l'axe des

Y de une seconde.

Remarquez la syntaxe particulière qu'a utilisé Blend :

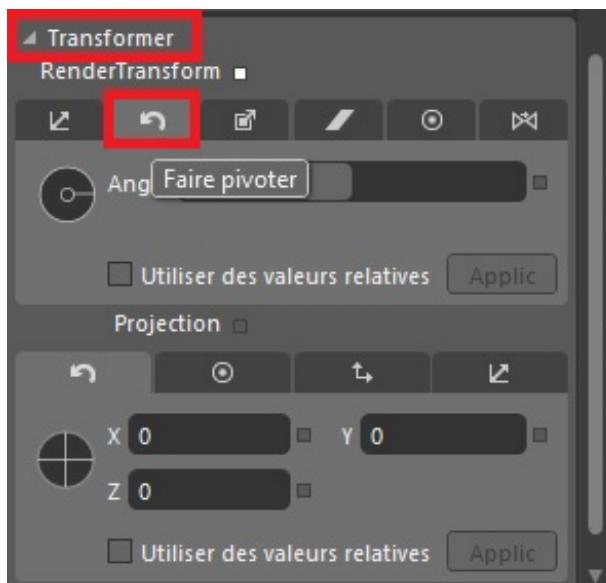
Code : XML

```
<DoubleAnimation Duration="0:0:1" To="166"
Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.TranslationX)"
Storyboard.TargetName="button" d:IsOptimized="True"/>
```



et notamment sur la propriété TargetProperty. Alors que dans mon exemple, j'avais donné un nom à la transformation pour animer une propriété de cette transformation, ici Blend a choisi d'animer une propriété relative du bouton, nommé button. Il dit qu'il va animer la propriété TranslateX de l'objet CompositeTransform faisant partie du RenderTransform correspondant au bouton, sachant que la propriété RenderTransform fait partie de la classe de base UIElement.

Revenons à Expression Blend pour rajouter une rotation. Plaçons donc notre ligne de temps sur la deuxième seconde. Je déplace mon bouton en bas à gauche afin de réaliser une translation à laquelle je vais combiner une rotation. Aller dans la fenêtre de propriétés du bouton et aller tout en bas pour cliquer sur transformer, et choisir le deuxième onglet pour faire pivoter (voir la figure suivante).



Rotation d'un contrôle via Blend

Vous pouvez désormais choisir un angle, disons 40°. Vous pouvez vérifier que la translation se fait en même temps que la rotation en appuyant sur le bouton de lecture.

Terminons enfin notre mini boucle en déplaçant la ligne de temps sur la troisième seconde et en faisant revenir le bouton à la position première et en réglant l'angle sur 360°.

Et voilà, nous avons terminé. Enfin... presque. L'animation est prête mais rien ne permet de la déclencher. Il existe une solution pour le faire avec Expression Blend, via les comportements que l'on trouve plus souvent sous le terme anglais de *Behavior*. J'ai choisi de ne pas en parler dans ce cours car cela nécessiterait pas mal d'explications qui ne nous serviront pas particulièrement pour la suite.

Nous allons donc retourner dans Visual Studio pour démarrer manuellement l'animation, par exemple lors du clic sur le bouton. Rajoutons donc l'événement clic directement dans notre bouton :

Code : XML

```
<Button x:Name="button" Content="Button" Height="77"
Margin="98,60,165,0" VerticalAlignment="Top"
RenderTransformOrigin="0.5,0.5" Tap="Button_Tap">
<Button.RenderTransform>
<CompositeTransform/>
</Button.RenderTransform>
</Button>
```

Avec dans le code behind :

Code : C#

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    MonStoryboard.Begin();  
}
```

Et voilà. Notre animation est terminée !

Projections 3D

Chaque élément affichable avec le XAML peut subir une projection 3D. Cela consiste à donner à une surface 2D une perspective 3D afin de réaliser un effet visuel. Plutôt qu'un long discours, un petit exemple qui parlera de lui-même. Prenons par exemple une image, l'image de la couverture de mon livre sur le C# :

Code : XML

```
<Image  
Source="http://uploads.siteduzero.com/files/365001_366000/365350.jpg"/>
```

Qui donne la figure suivante.



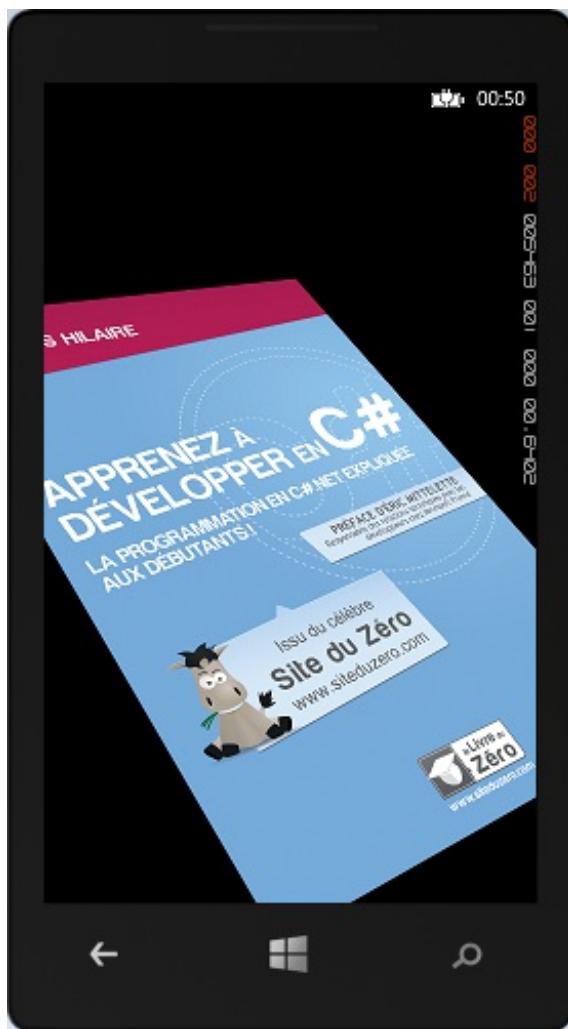
Image de la couverture du livre pour apprendre le C# dans l'émulateur

Pour lui faire subir un effet de perspective, nous pouvons utiliser le XAML suivant :

Code : XML

```
<Image  
Source="http://uploads.siteduzero.com/files/365001_366000/365350.jpg">  
    <Image.Projection>  
        <PlaneProjection RotationX="-35" RotationY="-35"  
        RotationZ="15" />  
    </Image.Projection>  
</Image>
```

qui lui fera subir une rotation de -35° autour de l'axe des X, de -35° autour de l'axe des Y et de 15° autour de l'axe des Z, ce qui donnera la figure suivante.



L'image avec une projection 3D

Plutôt sympa comme effet non ? Nous avons utilisé [la classe PlaneProjection](#) pour le réaliser.

Il existe une autre classe permettant de faire des projections suivant une matrice 3D, il s'agit de [la classe Matrix3DProjection](#) mais je pense que vous ne vous servirez que de la projection plane.

Alors, c'est très joli comme ça, mais combiné à une animation, c'est encore mieux.
Prenons le XAML suivant :

Code : XML

```
<phone:PhoneApplicationPage.Resources>  
    <Storyboard x:Name="Sb">
```

```

<DoubleAnimation Storyboard.TargetName="Projection"
Storyboard.TargetProperty="RotationZ"
From="0" To="360" Duration="0:0:5" />
<DoubleAnimation Storyboard.TargetName="Projection"
Storyboard.TargetProperty="RotationY"
From="0" To="360" Duration="0:0:5" />
</Storyboard>
</phone:PhoneApplicationPage.Resources>
<StackPanel>
    <Image
Source="http://uploads.siteduzero.com/files/365001_366000/365350.jpg">
        <Image.Projection>
            <PlaneProjection x:Name="Projection" RotationX="0"
RotationY="0" RotationZ="0" />
        </Image.Projection>
    </Image>
</StackPanel>

```

Vous vous doutez que je vais animer la rotation sur l'axe des Y et sur l'axe des Z de 0 à 360 degrés pendant une durée de 5 secondes ...

Difficile de vous montrer le résultat, mais je ne peux que vous encourager à tester chez vous (voir la figure suivante).



Animation d'une projection 3D

Vous n'aurez bien sûr pas oublié de démarrer l'animation, par exemple depuis l'événement de chargement de page :

Code : C#

```

public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

void MainPage_Loaded(object sender, RoutedEventArgs e)
{
}

```

```
Sb.Begin();  
}
```

- Le XAML possède un framework complexe d'animation.
- Blend se révèle un atout de qualité dans la réalisation d'animations complexes.
- Les projections 3D permettent d'ajouter un effet de perspective 3D dont le rendu est plutôt intéressant.

Voilà pour cette introduction à XAML ou à Silverlight pour Windows Phone. Je n'ai bien sûr pas pu tout présenter et je m'excuse d'être passé rapidement sur certains points afin que ce cours garde une taille raisonnable et éviter de vous endormir avec plein de petits détails. N'hésitez pas à creuser les points qui vous intéressent, la documentation MSDN est plutôt bien fournie sur le sujet.

Vous avez cependant toutes les bases vous permettant de démarrer la réalisation d'applications pour vos Windows Phone. Alors, n'hésitez pas à vous entraîner sur ce que vous avez appris et ensuite à découvrir la suite du cours où nous allons plonger petit à petit dans les spécificités de Windows Phone ...

Partie 2 : Un mobile orienté données

Maintenant que l'échauffement est terminé et que nous commençons à avoir des bonnes notions pour démarrer avec le XAML, il est temps de passer la vitesse supérieure. Dans cette partie, nous allons voir comment travailler avec les données dans nos applications pour Windows Phone et surtout comment utiliser la liaison de données. C'est une étape indispensable pour pouvoir réaliser des applications d'envergures. Nous allons aussi voir d'autres choses, comme le très intéressant et très utile contrôle ListBox, comment gérer les différents états visuels, mais attaquons nous tout de suite à une spécificité des applications Windows Phone : la navigation et les différents états d'une application.

Une application à plusieurs pages, la navigation

Pour l'instant, notre application est simple, avec une unique page. Il est bien rare qu'une application n'ait qu'une seule page... C'est comme pour un site internet, imaginons que nous réalisons une application mobile pour commander des produits, nous aurons une page contenant la liste des produits par rayon, une page pour afficher la description d'un produit, une page pour commander...

Nous allons donc voir qu'il est possible de naviguer facilement entre les pages de notre application grâce au service de navigation de Windows Phone.

Naviguer entre les pages

Avant de pouvoir naviguer entre des multiples pages, il faut effectivement avoir plusieurs pages ! Nous allons illustrer cette navigation en prenant pour exemple le site OpenClassrooms... enfin, en beaucoup beaucoup moins bien.

Première fonctionnalité, il faut pouvoir se loguer afin d'atteindre la page des cours. Nous allons donc avoir deux pages, une qui permet de se loguer, et une qui permet d'afficher la liste des cours.

Commençons par la page pour se loguer et vu qu'elle existe, utilisons la page MainPage.xaml :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="Démonstration de la navigation" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Page de Login" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="Saisir votre login" HorizontalAlignment="Center" />
            <TextBox x:Name="Login"/>
            <Button Content="Se connecter" Tap="Button_Tap" />
        </StackPanel>
    </Grid>
</Grid>
```



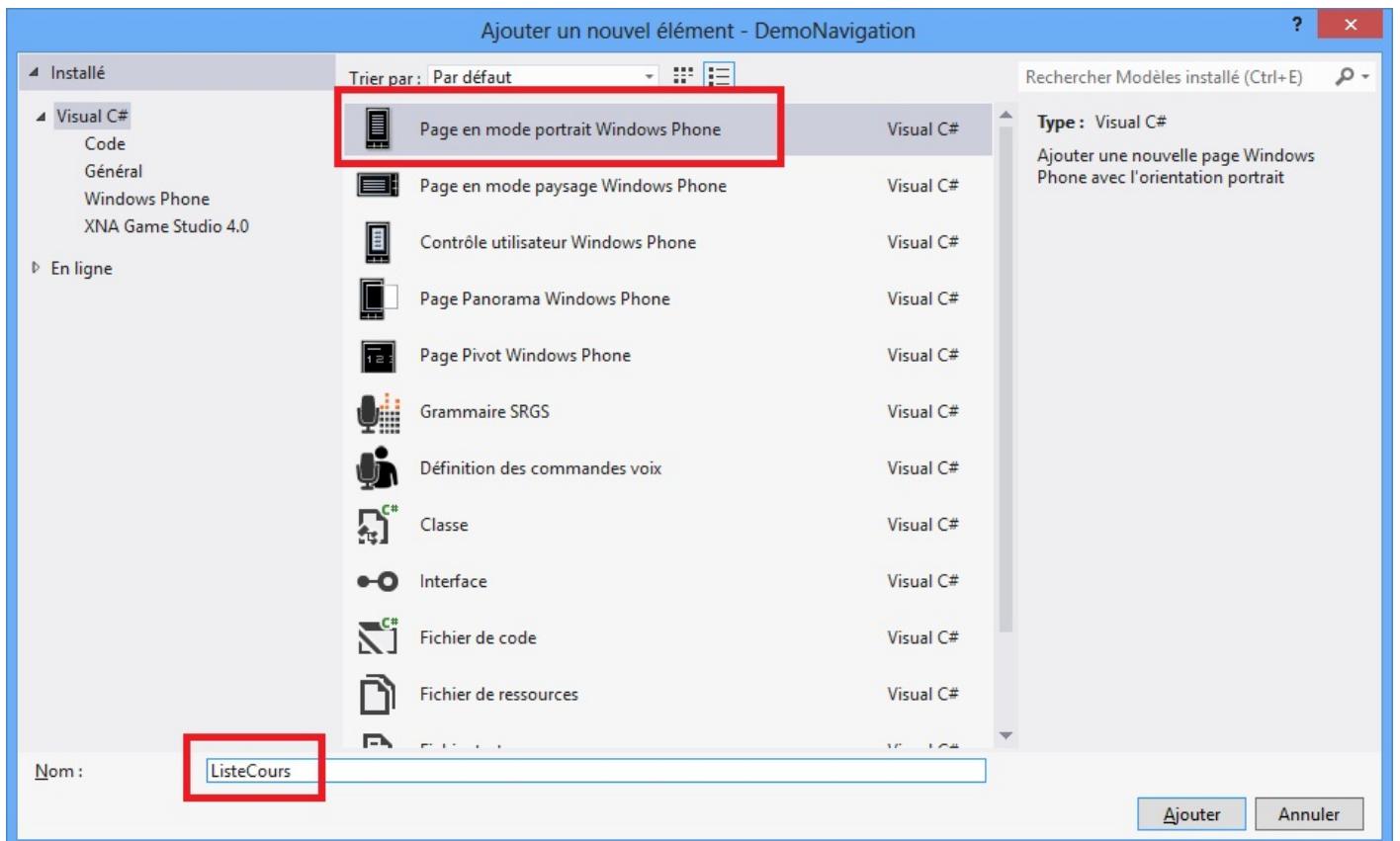
N'oubliez pas de générer à chaque fois les événements des contrôles, si ce n'est pas déjà fait. Dans l'exemple précédent : Button_Tap.

Pour que cela soit plus simple, nous utilisons uniquement un login pour nous connecter. Si nous affichons la page dans l'émulateur, nous avons la figure suivante.



Affichage de la page de login

Nous allons maintenant créer une deuxième page permettant d'afficher la liste des cours. Créons donc une autre page que nous nommons `ListeCours.xaml`. Pour cela, nous faisons un clic droit sur le projet et choisissons d'ajouter un nouvel élément. Il suffit de choisir le modèle de fichier `Page` en mode portrait `Windows Phone` et de lui donner le bon nom (voir la figure suivante).



Ajout d'une nouvelle page XAML dans le projet

Dans cette page, nous allons afficher simplement bonjour et que la page est en construction. Pour cela, un XAML très minimaliste :

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="Démonstration de la navigation" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Page des cours" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="200" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock x:Name="Bonjour" Text="Bonjour" HorizontalAlignment="Center" />
        <TextBlock Grid.Row="1" Text="Cette page est en construction ..." />
    </Grid>
</Grid>

```

Retournons dans la méthode de clic sur le bouton de la première page. Nous allons utiliser le code suivant :

Code : C#

```
private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    if (!string.IsNullOrEmpty(Login.Text))
        NavigationService.Navigate(new Uri("/ListeCours.xaml",
UriKind.Relative));
}
```

Nous utilisons le service de navigation et notamment sa méthode `Navigate` pour accéder à la page `ListeCours.xaml`, si le login n'est pas vide.

Grâce à cette méthode, nous pouvons aller facilement sur la page en construction. Remarquons que si nous appuyons sur le bouton en bas à gauche du téléphone permettant de faire un retour arrière, alors nous revenons à la page précédente. Si nous cliquons à nouveau sur le retour arrière, alors nous quittons l'application car il n'y a pas de page précédente.

Bon, c'est très bien tout ça, mais si on pouvait afficher un bonjour personnalisé, ça serait pas plus mal, avec par exemple le login saisi juste avant ...

Il y a plusieurs solutions pour faire cela. Une des solutions consiste à utiliser la query string. Elle permet de passer des informations complémentaires à une page, un peu comme pour les pages web. Pour cela, on utilise la syntaxe suivante :

Code : HTML

```
Page.xaml?parametre1=valeur1&parametre2=valeur2
```

Modifions donc notre méthode pour avoir :

Code : C#

```
private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    if (!string.IsNullOrEmpty(Login.Text))
        NavigationService.Navigate(new Uri("/ListeCours.xaml?login=" +
Login.Text, UriKind.Relative));
}
```

Désormais, la page `ListeCours` sera appelée avec le paramètre `login` qui vaudra la valeur saisie dans la `TextBox`.

Pour récupérer cette valeur, rendez-vous dans le code behind de la seconde page où nous allons substituer la méthode appelée lorsqu'on navigue sur la page, il s'agit de la méthode `OnNavigatedTo`, cette méthode faisant partie de la classe `PhoneApplicationPage`. Nous aurons donc le code behind suivant :

Code : C#

```
public partial class ListeCours : PhoneApplicationPage
{
    public ListeCours()
    {
        InitializeComponent();
    }

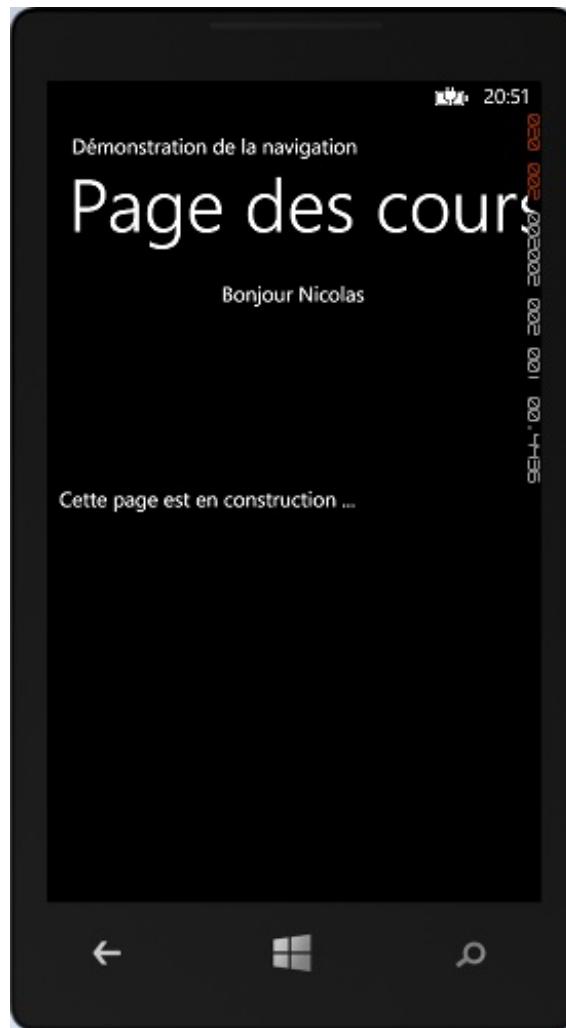
    protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        base.OnNavigatedTo(e);
    }
}
```

C'est à cet endroit que nous allons extraire la valeur du paramètre avec le code suivant :

Code : C#

```
protected override void  
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)  
{  
    string login;  
    if (NavigationContext.QueryString.TryGetValue("login", out  
login))  
    {  
        Bonjour.Text += " " + login;  
    }  
    base.OnNavigatedTo(e);  
}
```

On utilise la méthode TryGetValue en lui passant le nom du paramètre. Cette méthode fait partie de l'objet `QueryString` du contexte de navigation accessible via `NavigationContext`. Ce qui nous donne la figure suivante.



Affichage de la seconde page

Une autre solution pour passer des informations de page en page serait d'utiliser le dictionnaire d'état de l'application afin de communiquer un contexte à la page vers laquelle nous allons naviguer. Il s'agit d'un objet accessible de partout où nous pouvons stocker des informations et les lier à une clé. Cela donne :

Code : C#

```
private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    if (!string.IsNullOrEmpty(Login.Text))
    {
        PhoneApplicationService.Current.State["login"] = Login.Text;
        NavigationService.Navigate(new Uri("/ListeCours.xaml",
UriKind.Relative));
    }
}
```

Et pour récupérer la valeur dans la deuxième page, nous ferons :

Code : C#

```
protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    string login =
(string)PhoneApplicationService.Current.State["login"];
    Bonjour.Text += " " + login;
    base.OnNavigatedTo(e);
}
```

L'utilisation du dictionnaire d'état est très pratique pour faire transiter un objet complexe qui sera difficilement représentable dans des paramètres de query string.

 Attention : le dictionnaire d'état ne doit contenir que des informations sérialisables.

Voilà pour ce premier aperçu du service de navigation. Remarquez que le XAML possède également un contrôle qui permet de naviguer entre les pages, comme le `NavigationService`. Il s'agit du `contrôle HyperlinkButton`. Il suffira de renseigner sa propriété `NavigateUri`. Complétons notre page `ListeCours` pour rajouter en bas un `HyperLinkButton` qui renverra vers une page `Contact.xaml` :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
<Grid.RowDefinitions>
    <RowDefinition Height="200" />
    <RowDefinition Height="*" />
    <RowDefinition Height="auto" />
</Grid.RowDefinitions>
    <TextBlock x:Name="Bonjour" Text="Bonjour"
HorizontalAlignment="Center" />
    <TextBlock Grid.Row="1" Text="Cette page est en construction
..." />
    <HyperlinkButton Grid.Row="2" Content="Nous contacter"
NavigateUri="/Contact.xaml" />
</Grid>
```

Puis créons une page `Contact.xaml` :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
<Grid.RowDefinitions>
```

```
<RowDefinition Height="Auto"/>
<RowDefinition Height="*" />
</Grid.RowDefinitions>

<StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="Démonstration de
la navigation" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="Nous contacter"
Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock Text="Il n'y a rien pour l'instant ..." />
        <Button Content="Revenir à la page précédente"
Tap="Button_Tap" />
    </StackPanel>
</Grid>
</Grid>
```

Ainsi, lorsque nous démarrerons l'application et après nous être logués, nous pouvons voir le bouton « nous contacter » en bas de la page (voir la figure suivante).



Utilisation du contrôle HyperLinkButton pour la navigation

Un clic dessus nous amène à la page de contact (voir la figure suivante).



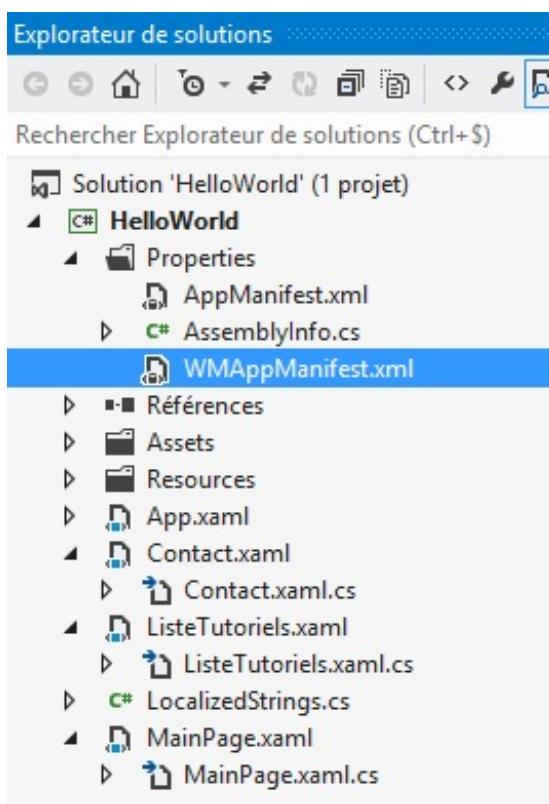
Affichage de la page de contact

Et voilà, la navigation est rendue très simple avec ce contrôle, nous naviguons entre les pages de notre application en n'ayant presque rien fait, à part ajouter un contrôle `HyperlinkButton`. Il sait gérer facilement une navigation avec des liens entre des pages. C'est la forme de navigation la plus simple.

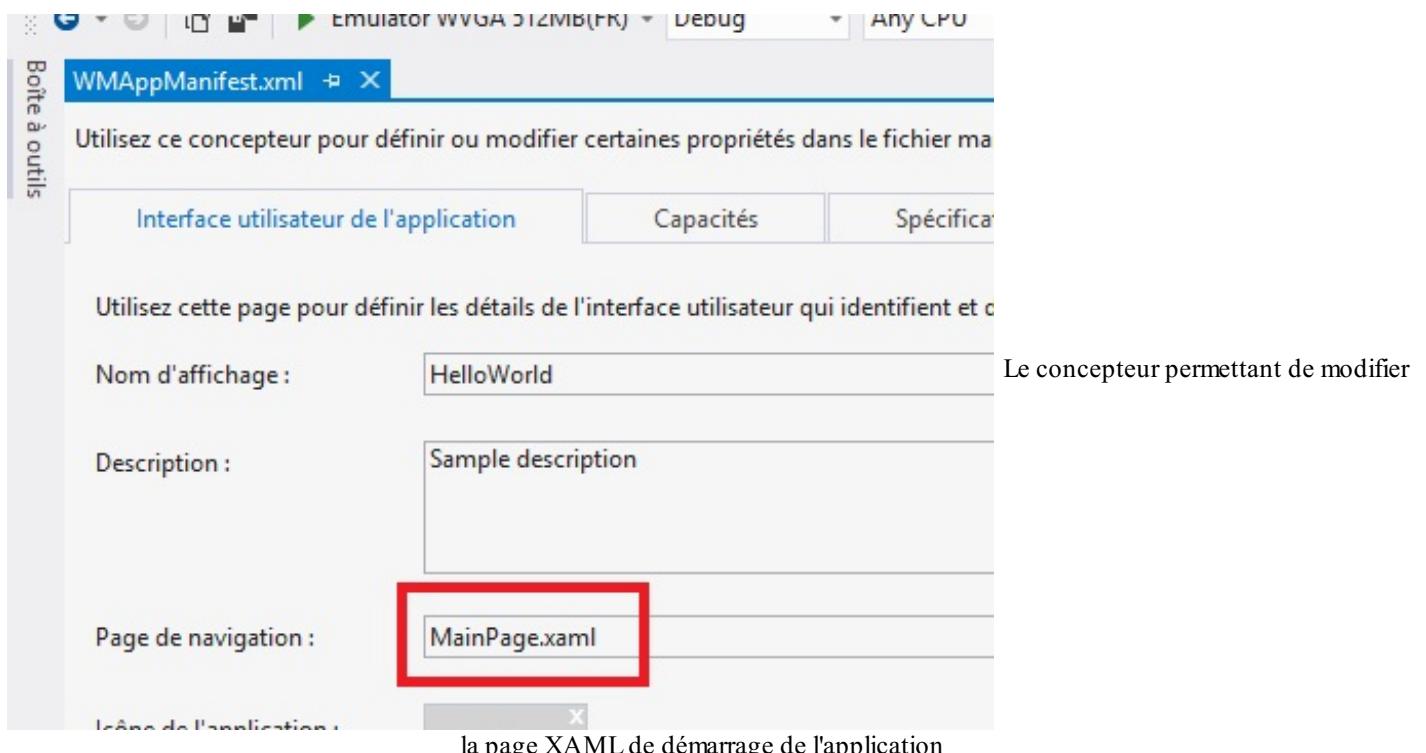
Nous avons pu voir ainsi deux façons différentes de naviguer entre les pages, via le contrôle `HyperlinkButton` et via le `NavigationService`. Puis nous avons vu deux façons différentes de passer des informations entre les pages, via la query string et via le dictionnaire d'état de l'application.



On remarque que la première page à s'afficher lorsqu'on démarre une application est la page `MainPage.xaml`. Ceci est configurable en allant modifier le fichier `WMAppManifest.xml` qui se trouve dans l'explorateur de solutions, sous `properties` (voir la figure suivante).



Double-cliquez dessus et une nouvelle page s'ouvre permettant de saisir une autre page de démarrage (voir la figure suivante).



Gérer le bouton de retour arrière

Et pour revenir en arrière ? Nous l'avons vu, il faut cliquer sur le bouton de retour arrière qui fait nécessairement partie d'un téléphone Windows Phone. Mais il est également possible de déclencher ce retour arrière grâce au service de navigation. C'est à cela que va servir le bouton que j'ai rajouté dans la page Contact.xaml. Observons l'événement associé au clic :

Code : C#

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{
```

```
        NavigationService.GoBack();  
    }
```

Très simple, il suffit de déclencher le retour arrière avec la méthode `GoBack()` du service de navigation. Notez qu'il peut être utile dans certaines situations de tester si un retour arrière est effectivement possible. Cela se fait avec la propriété `CanGoBack` :

Code : C#

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    if (NavigationService.CanGoBack)  
        NavigationService.GoBack();  
}
```

Il est également possible de savoir si l'utilisateur a appuyé sur le fameux bouton de retour arrière. À ce moment-là, on passera dans la méthode `OnBackKeyPress`. Pour pouvoir faire quelque chose lors de ce clic, on pourra substituer cette méthode dans notre classe :

Code : C#

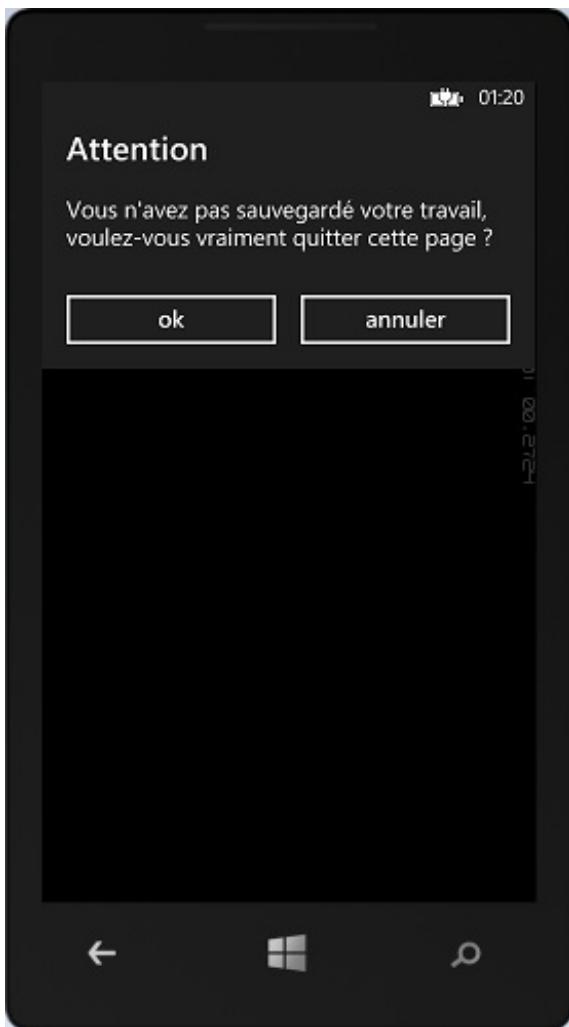
```
protected override void  
OnBackKeyPress(System.ComponentModel.CancelEventArgs e)  
{  
    base.OnBackKeyPress(e);  
}
```

Il est possible ici de faire ce que l'on veut, comme afficher un message de confirmation demandant si on veut réellement quitter cette page, ou sauvegarder des infos, etc. On pourra annuler l'action de retour arrière en modifiant la propriété `Cancel` de l'objet `CancelEventArgs` à `true`, si par exemple l'utilisateur ne souhaite finalement pas revenir en arrière. On peut également choisir de rediriger vers une autre page si c'est pertinent :

Code : C#

```
protected override void  
OnBackKeyPress(System.ComponentModel.CancelEventArgs e)  
{  
    if (MessageBox.Show("Vous n'avez pas sauvegardé votre travail,  
voulez-vous vraiment quitter cette page ?", "Attention",  
MessageBoxButton.OKCancel) == MessageBoxResult.Cancel)  
    {  
        e.Cancel = true;  
    }  
    base.OnBackKeyPress(e);  
}
```

Qui donne la figure suivante.



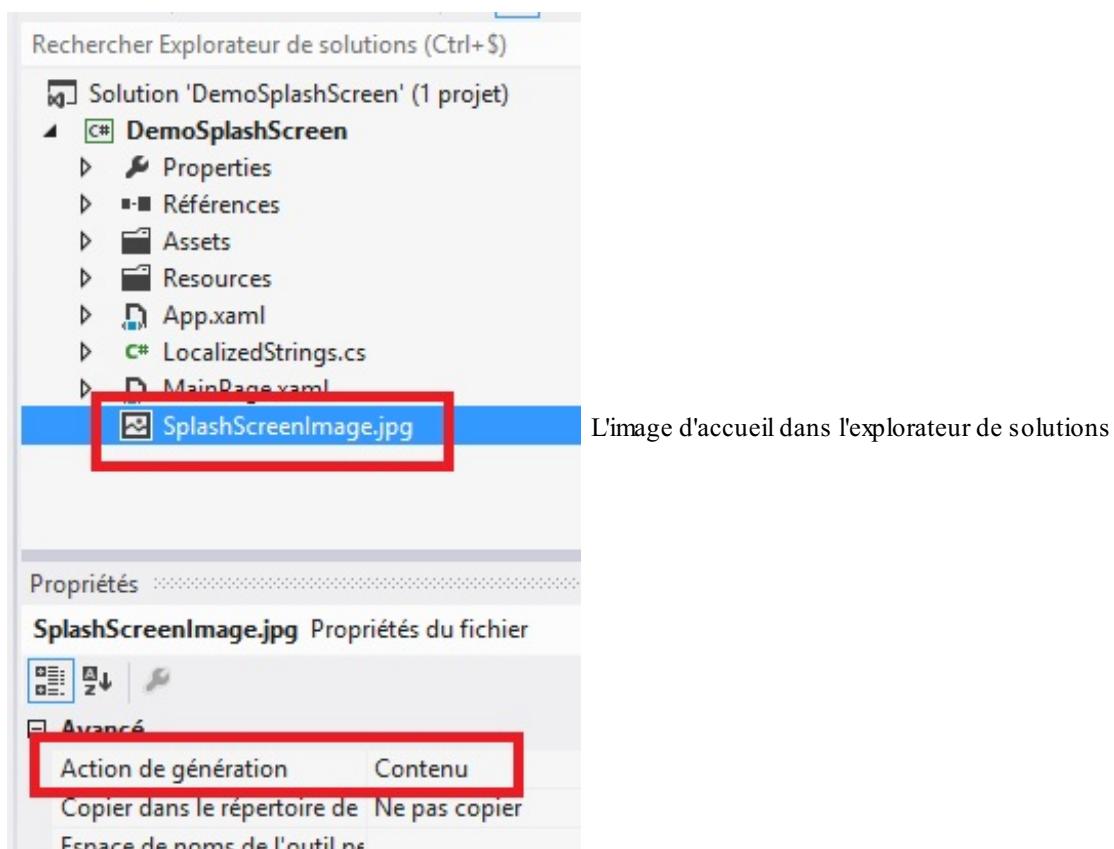
Affichage d'une confirmation avant de quitter la page

Et voilà pour les bases de la navigation. D'une manière générale, il est de bon ton de garder une mécanique de navigation fluide et cohérente. Il faut privilégier la navigation intuitive pour que l'utilisateur ne soit pas perdu dans un labyrinthe d'écran...

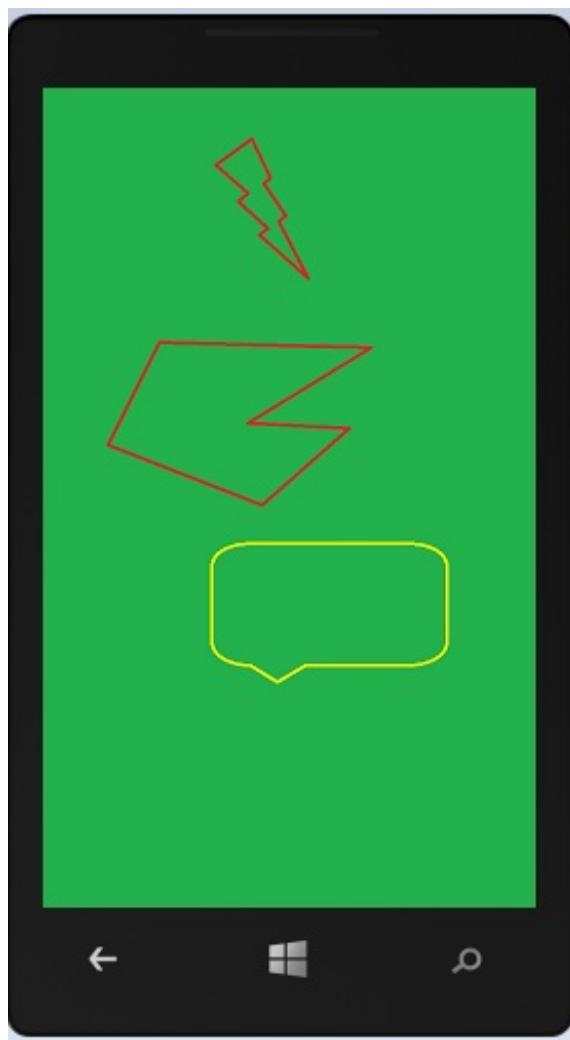
Ajouter une image d'accueil (splash screen)

Il est maintenant d'un usage commun de faire en sorte qu'au démarrage de son application il y ait une image pour faire patienter l'utilisateur pendant que tout se charge. On appelle cela en général de son nom anglais : *Splash screen*, que l'on peut traduire en image d'accueil. On y trouve souvent un petit logo de l'application ou de l'entreprise qui l'a réalisé, pourquoi pas le numéro de version,... bref, des choses pour faire patienter l'utilisateur et lui dire que l'application va bientôt démarrer.

Avec Windows Phone, il est très facile de réaliser ce genre d'image d'accueil, il suffit de rajouter (si elle n'est pas déjà présente) une image s'appelant `SplashScreenImage.jpg` à la racine du projet. Elle doit avoir son action de génération à Contenu (voir la figure suivante).



Pour les applications Windows Phone 8, elle doit avoir la taille 768x1280 pixels alors que pour les applications Windows Phone 7.X elle devra être de 480x800 (voir la figure suivante).



Affichage de l'écran d'accueil dans l'émulateur

Vous noterez au passage mon talent de dessinateur et ma grande force à exploiter toute la puissance des formes de Paint.

Le tombstonning

Avec Windows Phone 7 est apparu un changement radical dans la façon dont sont traitées les applications. Fini le multitâche comme on le connaît auparavant avec un Windows classique, il n'y a désormais qu'une application qui s'exécute à la fois. Cela veut dire que si je suis dans une application, que j'appuie sur le bouton de menu et que j'ouvre une nouvelle application, je n'ai pas deux applications qui tournent en parallèle, mais une seule. Ma première application ne s'est pas fermée non plus, elle est passée dans un mode « suspendu », voire « désactivé » en fonction du contexte. Ainsi, si je quitte ma seconde application en appuyant par exemple sur le bouton de retour arrière, alors ma première application qui était en mode suspendu ou désactivé, se réactive et repasse dans le mode en cours d'exécution.

Ce fonctionnement est conservé pour Windows Phone 8 et a été également étendu pour les applications Windows 8. Une application peut donc soit être :

- En cours d'exécution
- Suspendue
- Désactivée
- Non démarrée

Lorsque l'application passe en mode suspendu, par exemple lorsque j'appuie sur le bouton de menu, elle reste intacte en mémoire. Cela veut dire qu'un retour arrière pour retourner à l'application sera très rapide et vous retrouverez votre application comme elle se trouvait précédemment.

Enfin, ça, c'est la théorie. En vrai, c'est un peu plus compliqué que ça. C'est le système d'exploitation qui s'occupe de gérer les états des applications en fonction notamment de la mémoire disponible. En effet, Windows Phone peut choisir de désactiver une application suspendue si l'application en cours d'exécution a besoin de mémoire. De la même façon, l'application peut avoir simplement été suspendue et se réactiver de manière optimale si le système d'exploitation n'a pas eu besoin de mémoire. Une application désactivée a été terminée par le système d'exploitation, bien souvent parce qu'il avait besoin de mémoire. Quelques informations restent cependant disponibles et si l'utilisateur revient sur l'application, celle-ci est alors redémarrée depuis zéro mais ces informations sont accessibles afin de permettre de restaurer l'état de l'application.

Tout ceci implique que l'on ne peut jamais garantir que l'utilisateur va fermer correctement une application ou que celle-ci va se terminer proprement, c'est même d'ailleurs rarement le cas. Il peut y avoir plein de scénarios possibles. Par exemple votre utilisateur est en train de saisir des informations dans votre application, il change d'application pour aller lire un mail, voir la météo, puis plus tard il revient à votre application ; entre temps il a reçu un coup de téléphone, recharge son téléphone ... Qu'est devenue notre application ? Comment apporter à l'utilisateur tout le confort d'utilisation possible et lui garantir qu'il n'a pas perdu toute sa saisie ?

Heureusement, lorsque l'application change d'état, nous pouvons être prévenus grâce à des événements. Ce sont des événements applicatifs que l'on retrouve dans le fichier d'application que nous avons déjà vu : App.xaml. Par contre, ici nous allons nous intéresser à son code behind : App.xaml.cs et notamment aux événements déjà générés pour nous. Ouvrez-le et vous pouvez voir :

Code : C#

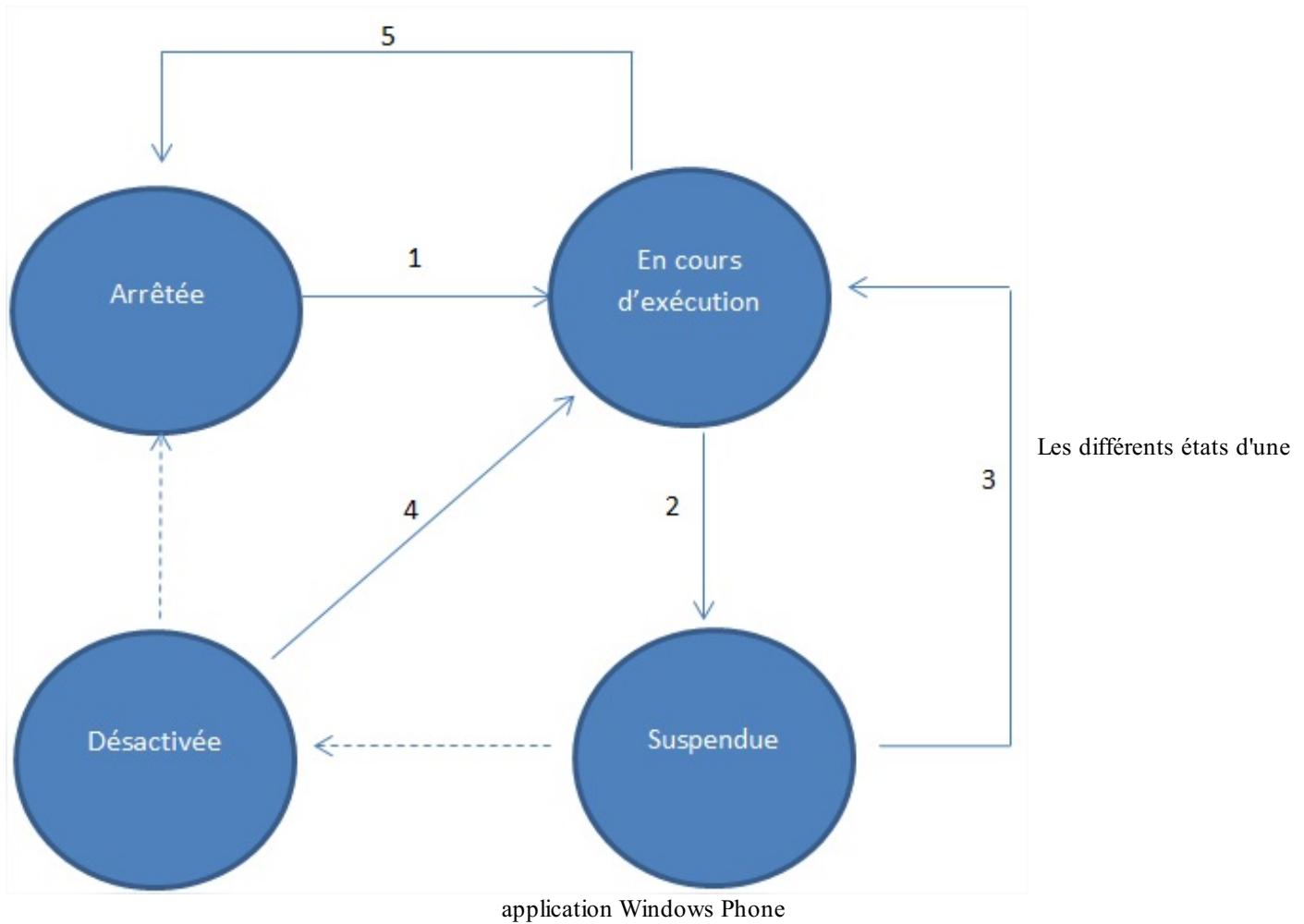
```
// Code à exécuter lorsque l'application démarre (par exemple, à
// partir de Démarrer)
// Ce code ne s'exécute pas lorsque l'application est réactivée
private void Application_Launching(object sender, LaunchingEventArgs
e)
{
}

// Code à exécuter lorsque l'application est activée (affichée au
// premier plan)
// Ce code ne s'exécute pas lorsque l'application est démarrée pour
// la première fois
private void Application_Activated(object sender, ActivatedEventArgs
e)
{
}

// Code à exécuter lorsque l'application est désactivée (envoyée à
// l'arrière-plan)
// Ce code ne s'exécute pas lors de la fermeture de l'application
private void Application_Deactivated(object sender,
DeactivatedEventArgs e)
{
}

// Code à exécuter lors de la fermeture de l'application (par
// exemple, lorsque l'utilisateur clique sur Précédent)
// Ce code ne s'exécute pas lorsque l'application est désactivée
private void Application_Closing(object sender, ClosingEventArgs e)
{}
```

Les commentaires générés parlent d'eux-mêmes. Ces méthodes sont donc l'endroit idéal pour faire des sauvegardes de contexte. Voici à la figure suivante un schéma récapitulatif des différents états.



| | Action | Événement applicatif |
|---|---------------------------|----------------------|
| 1 | Démarrage | Launching |
| 2 | Désactivation | Deactivated |
| 3 | Reprise rapide | Activated |
| 4 | Reprise lente | Activated |
| 5 | Fermeture | Closing |
| 6 | Fermeture forcée par l'OS | - |



L'événement **Closing** n'est pas levé lorsque c'est le système d'exploitation qui choisit de terminer l'application, par manque de ressources par exemple.

Imaginons que je sois en train de saisir un long texte dans mon application, que mon téléphone sonne et que je doive partir d'urgence. L'application va commencer par passer en mode suspendu. Si je retourne dans mon application rapidement en appuyant sur le retour arrière, alors je vais retrouver mon texte intact. Par contre, si celle-ci est désactivée par le système d'exploitation, alors je peux dire adieu à ma saisie. Et là, je risque de maudire cette application et ne plus jamais l'utiliser. Et c'est encore pire si c'est moi qui relance depuis zéro l'application depuis le menu, je ne pourrais même pas maudire le système d'exploitation.

Une solution est de sauvegarder ce texte au fur et à mesure de sa saisie. Comme ça, si jamais l'application est désactivée, je pourrai alors profiter des événements applicatifs pour enregistrer ce texte dans la mémoire du téléphone, afin de le repositionner si jamais l'utilisateur ré-ouvre l'application. Mais avant cela, essayons de bien comprendre le processus d'activation/désactivation et modifions `MainPage.xaml` pour avoir ceci :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBox />
    </StackPanel>
</Grid>
```

Et dans le code-behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private bool _estNouvelleInstance = false;
    private string laPage; // info à conserver

    public MainPage()
    {
        InitializeComponent();
        _estNouvelleInstance = true;
    }

    protected override void
    OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
    {
        if (e.NavigationMode !=
System.Windows.Navigation.NavigationMode.Back)
        {
            // l'appli est désactivée, la page est quittée
            State["MainPage"] = laPage;
        }
        else
        {
            // on quitte l'appli en appuyant sur back
        }
    }

    protected override void
    OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        if (_estNouvelleInstance)
        {
            if (State.Count > 0)
            {
                // application a été désactivée par l'OS, on peut
                accéder au dictionnaire d'état
                // pour restaurer les infos
                laPage = (string)State["MainPage"];
            }
            else
            {
                // équivalent à un démarrage de l'appli
                laPage = "MainPage";
            }
        }
        else
        {
            // la page est gardée en mémoire, pas besoin d'aller
            lire le dictionnaire d'état
        }
        _estNouvelleInstance = false;
    }
}
```

J'espère que les commentaires sont assez explicites. Ce qu'il faut retenir c'est que si nous démarrons l'application en appuyant

sur F5, que nous saisissons un texte dans le TextBox et que nous appuyons sur le bouton menu, alors l'application est suspendue (voir la figure suivante).

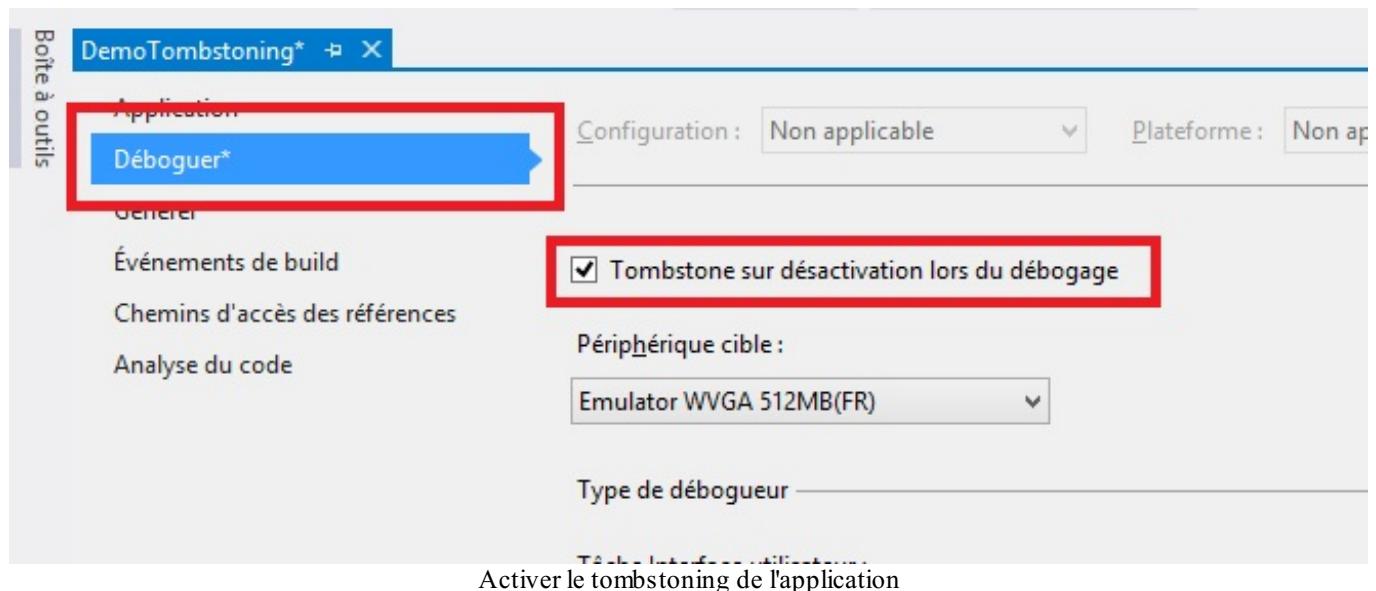


Appui sur le bouton de menu pour que l'application soit suspendue

Nous sommes alors passés dans la méthode `OnNavigatedTo` et nous sommes dans le cas où `_estNouvelleInstance` vaut vrai et que le dictionnaire d'état est vide (démarrage de l'application). Lorsqu'on clique sur le bouton de menu, alors nous passons dans la méthode `OnNavigatedFrom` qui indique que la page est désactivée. Vous arrivez alors sur la page d'accueil de votre émulateur (ou de votre Windows Phone). Un petit clic sur le bouton de retour vous ramène sur notre application avec la zone de texte qui correspond à ce que nous avons saisi. On repasse dans la méthode `OnNavigatedTo` avec `_estNouvelleInstance` qui vaut faux. Ceci prouve bien que l'application est intacte en mémoire et que nous ne sommes pas repassés dans le constructeur de la classe. Il n'y a rien à faire car l'application est exactement la même qu'avant son changement d'état.

Il ne reste plus qu'à cliquer à nouveau sur la flèche de retour pour fermer l'application et ainsi repasser dans la méthode `OnNavigatedFrom` mais cette fois-ci dans le `else`, quand `e.NavigationMode` vaut `NavigationMode.Back`. Remarquez que le débogueur est toujours en route et qu'il faut l'arrêter. Voilà pour l'état suspendu.

Pour simuler l'état désactivé, il faut aller dans les propriétés du projet en faisant un clic droit dessus, puis propriétés. On arrive sur l'écran des propriétés du projet, cliquez sur déboguer et vous pouvez alors cocher la case qui permet de forcer à passer dans l'état désactivé (Tombstone) lorsque l'on suspend l'application (voir la figure suivante).



Maintenant, lorsque vous allez appuyer sur F5, saisir un texte dans la TextBox, puis appuyer sur le bouton de menu, l'application sera désactivée, comme si c'était le système d'exploitation qui le faisait pour libérer de la mémoire. Revenez en arrière pour réveiller l'application, nous pouvons constater que le TextBox est vide. L'état de l'application n'est pas conservé et nous passons cette fois-ci dans la méthode `OnNavigatedTo` mais lorsque le dictionnaire d'état n'est pas vide. Il contient en l'occurrence ce que nous avons associé à la chaîne « MainPage ». Nous allons donc pouvoir nous servir du dictionnaire d'état pour restaurer l'état de l'application lorsque celle-ci est désactivée.



N'oubliez pas qu'il n'y a rien à faire lorsque l'application est suspendue.

Allez modifier le XAML pour donner un nom à notre TextBox :

Code : XML

```
<TextBox x:Name="LeTextBox" />
```

Puis modifiez le code behind, tout en conservant le squelette de `MainPage`, pour avoir ceci :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private bool _estNouvelleInstance = false;

    public MainPage()
    {
        InitializeComponent();
        _estNouvelleInstance = true;
    }

    protected override void
    OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
    {
        if (e.NavigationMode !=
System.Windows.Navigation.NavigationMode.Back)
        {
            // l'applic est désactivée, la page est quittée
            State["MonTexte"] = LeTextBox.Text;
        }
    }
}
```

```

        // on quitte l'appli en appuyant sur back
    }

    protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    if (_estNouvelleInstance)
    {
        if (State.Count > 0)
        {
            // application a été désactivée par l'OS, on peut
            accéder au dictionnaire d'état pour restaurer les infos
            LeTextBox.Text = (string)State["MonTexte"];
        }
        else
        {
            // équivalent à un démarrage de l'appli
        }
    }
    else
    {
        // la page est gardée en mémoire, pas besoin d'aller
        lire le dictionnaire d'état
    }
    _estNouvelleInstance = false;
}
}

```

Et voilà. L'état de la zone de texte est restauré grâce à :

Code : C#

```
LeTextBox.Text = (string)State["MonTexte"];
```

Qui est exécuté lorsque l'application est désactivée.

Vous me direz qu'on s'embête peut-être un peu pour rien. Ne pourrait-on pas remplacer la méthode `OnNavigatedTo` par :

Code : C#

```

protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    if (State.ContainsKey("MonTexte"))
        LeTextBox.Text = (string)State["MonTexte"];
}

```

Car finalement, peu importe si on est en mode démarré, suspendu ou désactivé, tout ce qui nous intéresse c'est que le `TextBox` soit rempli si jamais il l'a été auparavant.

Je vous dirais oui, mais... Ici, l'appel au dictionnaire d'état est assez rapide, mais imaginons que nous ayons besoin d'aller lire une information sur internet (ce que nous apprendrons à faire très bientôt), ou effectuer un calcul complexe, ou quoi que ce soit, ce n'est pas la peine de le faire si nous possédons déjà l'information. Cela fluidifie le redémarrage de l'application, évite de consommer des datas pour rien, etc. Veillez toujours à ne pas faire des choses inutiles et gardez à l'esprit qu'un téléphone a des ressources limitées.



Et les événements applicatifs ?

Ils servent aussi à ça. Lorsque l'application est suspendue ou désactivée, nous avons vu que l'événement application

Deactivated était levé. C'est également l'emplacement idéal pour faire persister des informations dans le dictionnaire d'état :

Code : C#

```
private void Application_Deactivated(object sender,
DeactivatedEventArgs e)
{
    PhoneApplicationService.Current.State["DateDernierLancement"] =
DateTime.Now;
}
```

De la même façon, lorsque l'application est réactivée, que ce soit en reprise rapide (depuis l'état suspendu) ou en reprise lente (depuis l'état désactivée), l'événement applicatif Activated est levé. C'est également un endroit idéal pour préparer à nouveau l'état de notre application :

Code : C#

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    if
(PhoneApplicationService.Current.State.ContainsKey("DateDernierLancement"))
    {
        DateTime dernierLancement =
(DateTime)PhoneApplicationService.Current.State["DateDernierLancement"];
        if (DateTime.Now - dernierLancement > TimeSpan.FromMinutes(30))
        {
            // si ça fait plus de trente minutes que l'application est
désactivée, mon information n'est peut-être plus à jour
            TelechargerLesNouvellesDonnees();
        }
    }
}
```



Remarquez que le dictionnaire d'état est accessible via la propriété `PhoneApplicationService.Current.State["..."]` depuis n'importe quel emplacement et via la propriété `State["..."]` depuis une page (qui dérive de `PhoneApplicationPage`). C'est la même chose.

Mais bon, il y a encore un petit problème ! Étant donné que nous ne maîtrisons pas le passage en désactivé, encore moins la fermeture forcée de l'application par le système d'exploitation, ou encore le redémarrage manuel de l'application par l'utilisateur (s'il relance l'application depuis la page d'accueil), nous risquons de perdre à nouveau toutes les informations que notre utilisateur a saisie mais cette fois-ci, pas parce que nous avons mal géré la désactivation, mais parce que l'application s'est terminée !

La solution est d'utiliser la même mécanique mais en faisant persister les informations dans la mémoire du téléphone dédiée à l'application. Il s'agit du **répertoire local** (en anglais *Local folder*), également connu sous son ancien nom :

`IsolatedStorage`. Voyez cela un peu comme des fichiers sur un disque dur où nous pouvons enregistrer ce que bon nous semble. Puisque cette information persiste entre les démarrages successifs de l'application, il sera possible d'enregistrer l'état de notre application afin par exemple que l'utilisateur ne perde pas toute sa saisie.

Je ne vais pas détailler le code qui permet d'enregistrer des informations dans la mémoire du téléphone, car j'y reviendrai dans un prochain chapitre.

Avant de terminer, sachez qu'il existe un autre état, l'état `Obscured`, que l'on peut traduire par « obscurci ». Il s'agit d'un état où une partie de l'écran est masqué, par exemple lorsqu'on reçoit un SMS, un appel, une notification, etc. L'application reste dans un état où elle est en cours d'exécution, mais l'application peut devenir difficile à utiliser. Imaginons que votre utilisateur soit en plein compte à rebours final, prêt à vaincre le boss final et qu'il reçoive un SMS juste au moment où il va gagner et qu'à cause de cela il reçoit la flèche fatale juste en plein milieu du cœur..., il va maudire votre application, à juste titre !

Pour éviter cela, il est possible d'être notifié lorsque l'application passe en mode `Obscured`, ce qui nous laisse par exemple l'opportunité de faire une pause dans notre compte à rebours final ... Pour cela, rendez-vous dans le constructeur de la classe `App` pour vous abonner aux événements `Obscured` et `Unobscured` :

Code : C#

```
public App()
{
    // plein de choses ...

    RootFrame.Obscured += RootFrame_Obscured;
    RootFrame.Unobsured += RootFrame_Unobsured;
}

private void RootFrame_Unobsured(object sender, EventArgs e)
{
    // L'application quitte le mode Obscured
}

private void RootFrame_Obscured(object sender, ObscuredEventArgs e)
{
    // L'application passe en mode Obscured
    bool estVerouille = e.IsLocked;
}
```

Remarquez que le paramètre de type ObscuredEventArgs permet de savoir si l'écran est verrouillé ou non.

- Il est possible de naviguer de page en page dans une application grâce au service de navigation.
- Le contrôle HyperlinkButton permet de démarrer une navigation très simplement.
- Il est possible de faire transiter des informations de contexte entre les pages ; on pourra utiliser la query string ou le dictionnaire d'état.
- Pour déclencher un retour arrière par code, on utilisera la méthode GoBack() du NavigationService.
- On peut ajouter facilement une image d'accueil à une application Windows Phone en utilisant une image JPEG, nommée SplashScreenImage.jpg.
- Il est très important de gérer correctement les différents états par lesquels peut passer une application afin de fournir une expérience d'utilisation la plus optimale.

TP 2 : Créer une animation de transition entre les pages

Maintenant que nous avons vu comment naviguer entre les pages et que nous savons faire des animations, il est temps de nous entraîner dans un contexte combiné. Le but bien sûr est de vérifier si vous avez bien compris ces précédents chapitres et de vous exercer.

Ce TP va nous permettre de créer une animation de transition entre des pages et par la même occasion embellir nos applications avec tout notre savoir 😊.

Instructions pour réaliser le TP

Nous allons donc réaliser une animation de transition entre des pages, histoire que nos navigations soient un peu plus jolies. Vous allez créer une application avec deux pages. Vous pouvez mettre ce que vous voulez sur les pages, une image, du texte..., mais il faudra que la première page possède un bouton permettant de déclencher la navigation vers la seconde page. L'animation fera glisser le contenu de la page vers le bas jusqu'à sortir de l'écran tout en réduisant l'opacité jusqu'à la disparition. L'animation ne sera pas trop longue, disons ½ seconde, voire 1 seconde (ce qui est déjà long !). L'affichage de la seconde page subira aussi une transition. L'animation fera apparaître le contenu de la page comme si elle apparaissait d'en haut et l'opacité augmentera pour devenir complètement visible. Bref, l'inverse de la première transition.

Vous vous le sentez ? Alors, à vous de jouer.

Si vous vous le sentez un peu moins, je vais vous donner quelques indications pour réaliser ce TP sereinement.

La première chose à faire est d'animer le contenu de la page. Dans tous nos exemples, nous avons utilisé un conteneur racine (bien souvent une Grid), qui contient tous les éléments de la page. Il suffit de faire porter l'animation sur ce contrôle pour faire tout disparaître.

Ensuite, même si cela fonctionne, il est plus propre d'attendre la fin de l'animation pour déclencher la navigation. Il faut donc s'abonner à l'événement de fin d'animation et à ce moment-là déclencher la navigation.

Enfin, pour démarrer la seconde animation, il faudra le faire depuis la méthode qui est appelée lorsqu'on arrive sur la seconde page.

Voilà, vous savez tout.

Correction

Passons à la correction, maintenant que tout le monde a réalisé ce défi haut la main.

Il s'agit dans un premier temps de créer deux pages différentes. Vous avez pu y mettre ce que vous vouliez, il fallait juste un moyen de pouvoir naviguer sur une autre page.

La première chose à faire est donc de créer l'animation qui va permettre de faire disparaître élégamment la première page. Il s'agit d'une animation qui cible le conteneur de premier niveau de notre page, dans mon cas une Grid. Étant donné que je vais avoir besoin de faire une translation, je vais définir une classe TranslateTransform dans la propriété RenderTransform de ma grille :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RenderTransform>
        <TranslateTransform x:Name="Translation" />
    </Grid.RenderTransform>
    [...reste du code supprimé pour plus de lisibilité...]
    <Button Content="Aller à la page 2" Tap="Button_Tap" />
</Grid>
```

Mon Storyboard sera déclaré dans les ressources de ma page :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
    <Storyboard x:Name="CachePage">
        <DoubleAnimation Storyboard.TargetName="LayoutRoot"
            Storyboard.TargetProperty="Opacity"
            From="1" To="0" Duration="0:0:0.5" />
        <DoubleAnimation Storyboard.TargetName="Translation"
            Storyboard.TargetProperty="Y"
            From="0" To="800" Duration="0:0:0.5" />
```

```
</Storyboard>
</phone:PhoneApplicationPage.Resources>
```

L'animation consiste à faire varier la propriété `Opacity` de la grille et la propriété `Y` de l'objet de translation. Puis il faut démarrer cette animation lors du clic sur le bouton sachant qu'auparavant, je vais m'abonner à l'événement de fin d'animation afin de pouvoir démarrer la navigation à ce moment-là :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
        CachePage.Completed += CachePage_Completed;
    }

    private void CachePage_Completed(object sender, EventArgs e)
    {
        NavigationService.Navigate(new Uri("/Page2.xaml",
        UriKind.Relative));
    }

    private void Button_Tap(object sender,
    System.Windows.Input.GestureEventArgs e)
    {
        CachePage.Begin();
    }
}
```

L'événement de fin d'animation est évidemment l'événement `Completed`, c'est dans la méthode associée que je déclenchera la navigation via la méthode `Navigate` du `NavigationService`.

Passons maintenant à la deuxième page. Le principe est le même, voici le XAML qui nous intéresse :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
    <Storyboard x:Name="AffichePage">
        <DoubleAnimation Storyboard.TargetName="LayoutRoot"
        Storyboard.TargetProperty="Opacity"
            From="0" To="1" Duration="0:0:0.5" />
        <DoubleAnimation Storyboard.TargetName="Translation"
        Storyboard.TargetProperty="Y"
            From="-800" To="0" Duration="0:0:0.5" />
    </Storyboard>
</phone:PhoneApplicationPage.Resources>

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RenderTransform>
        <TranslateTransform x:Name="Translation" />
    </Grid.RenderTransform>
    [...code supprimé pour plus de lisibilité...]
</Grid>
```

Cette fois-ci on incrémente l'opacité et on passe de -800 à 0 pour les valeurs de la translation de Y. Coté code behind nous aurons :

Code : C#

```
public partial class Page2 : PhoneApplicationPage
{
    public Page2()
    {
        InitializeComponent();
    }

    protected override void
    OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        AffichePage.Begin();
        base.OnNavigatedTo(e);
    }
}
```

On redéfinit la méthode qui est appelée lorsqu'on navigue sur la page afin de démarrer l'animation.

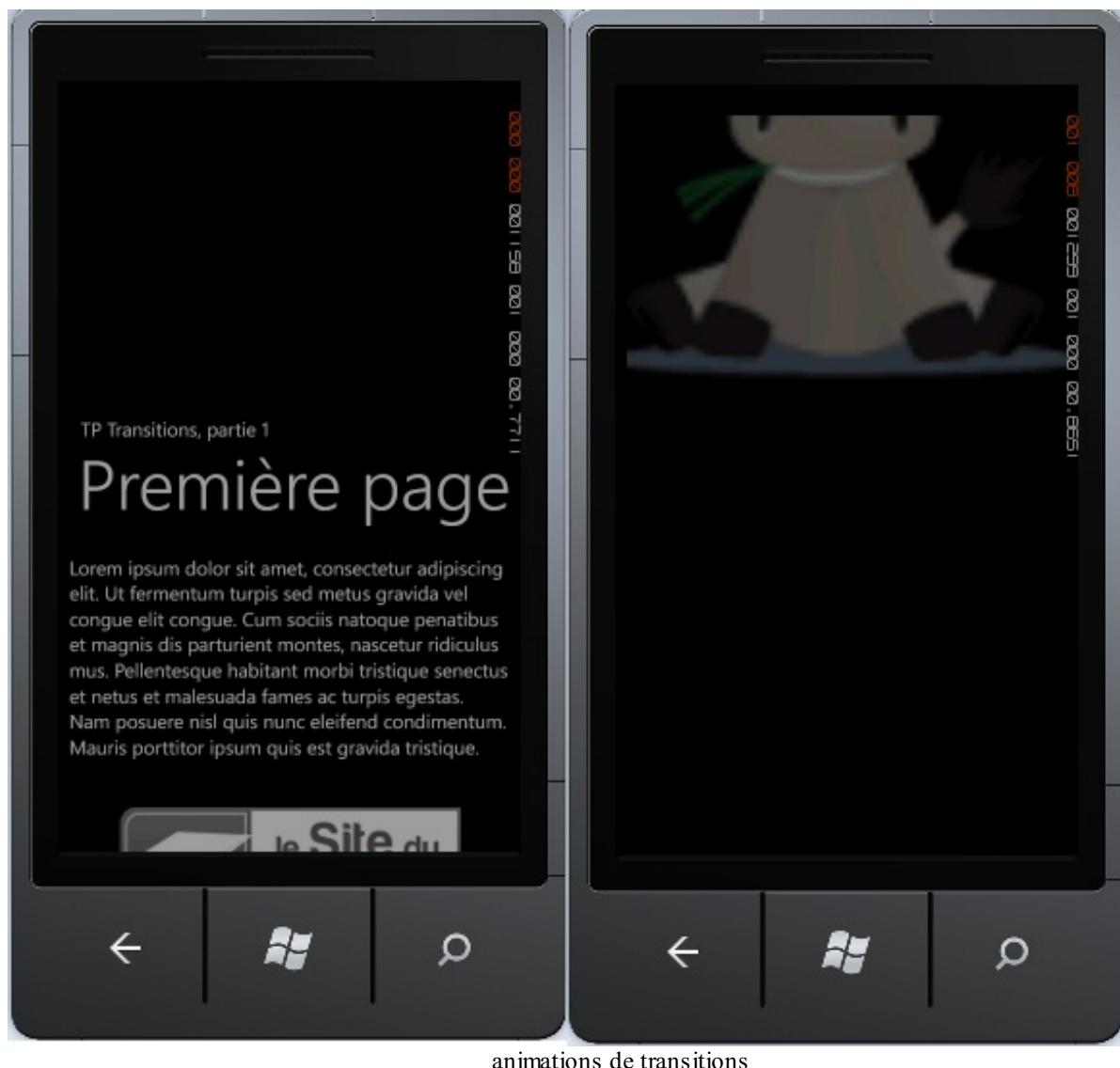
Et voilà !

Quoique... ce n'est pas tout à fait complet en l'état car si vous revenez sur la page précédente en appuyant sur le bouton de retour arrière, vous vous rendrez compte que la page est vide. Eh oui, nous avons fait disparaître la page lors de l'animation de transition ! Nous devons donc arrêter cette fameuse animation lorsque nous revenons sur la page avec :

Code : C#

```
protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    CachePage.Stop();
    base.OnNavigatedTo(e);
}
```

Et voilà, présentées comme j'ai pu, nos deux animations (voir la figure suivante).



Rendu des

animations de transitions

Remarquez que j'ai choisi la valeur 800 pour l'animation car j'ai considéré que la plupart des résolutions de téléphones étaient du 800. C'est évidemment une mauvaise idée car je ne connais pas le comportement sur les téléphones qui ont une autre résolution. Il aurait été judicieux de fixer la valeur à la résolution de l'écran, sauf que nous ne savons pas encore comment faire ... 😊

Avant de terminer ce TP, notez que le toolkit pour Windows Phone, que nous découvrirons plus loin, possède tout un lot de transitions prêtes à l'emploi.

Les propriétés de dépendances et propriétés attachées

Il est temps de vous dire la vérité sur les propriétés. Jusqu'à présent, j'ai fait comme si toutes les propriétés que nous avons vues étaient des propriétés classiques au sens C#. Elles sont en fait plus évoluées que ça.

Les propriétés de dépendances

De leurs noms anglais « dependency properties », ces propriétés sont plus complexes que la propriété de base de C# que nous connaissons, à savoir :

Code : C#

```
public class MaClasse
{
    public int MaPropriete { get; set; }
}
```

Avec une telle classe, il est possible d'utiliser cette propriété ainsi :

Code : C#

```
MaClasse maClasse = new MaClasse();
maClasse.MaPropriete = 6;
int valeur = maClasse.MaPropriete;
```

En XAML, nous avons dit que le code suivant :

Code : XML

```
<TextBlock Text="Je suis un texte" Foreground="Red" />
```

correspondait au code C# suivant :

Code : C#

```
TextBlock monTextBlock = new TextBlock();
monTextBlock.Text = "Je suis un texte";
monTextBlock.Foreground = new SolidColorBrush(Colors.Red);
```

Ici, on ne le voit pas mais ces deux propriétés sont en fait des propriétés de dépendances. Le vrai appel qui s'opère derrière est équivalent à :

Code : C#

```
TextBlock monTextBlock = new TextBlock();
monTextBlock.SetValue(TextBlock.TextProperty, "Je suis un texte");
monTextBlock.SetValue(TextBlock.ForegroundProperty, new
SolidColorBrush(Colors.Red));
```

La méthode `SetValue` est héritée de la classe de base `DependencyObject` dont héritent tous les `UIElement`. La propriété de dépendance étend les fonctionnalités d'une propriété classique C#. C'est la base notamment du système de

liaison de données que nous découvrirons plus loin. Cela permet de traiter la valeur d'une propriété en fonction du contexte de l'objet et d'être potentiellement déterminée à partir de plusieurs sources de données. La propriété de dépendance peut également avoir une valeur par défaut et des informations de description.



Ce qu'il faut retenir c'est que la propriété de dépendance est une propriété évoluée gérée par le moteur XAML qui va permettre de gérer les styles, les animations, la liaison de données, ...

Pour obtenir la valeur d'une propriété de dépendance, on pourra utiliser :

Code : C#

```
string text = (string)monTextBlock.GetValue(TextBlock.TextProperty);
```

Les propriétés attachées

Le mécanisme de propriété attachée permet de rajouter des propriétés à un contexte donné. Prenons par exemple le XAML suivant :

Code : XML

```
<Canvas>
    <TextBlock Text="Je suis un texte" Canvas.Top="150"
    Canvas.Left="80" />
</Canvas>
```

Il est possible d'indiquer la position du TextBlock dans le Canvas. Or, le TextBlock ne possède pas de propriété Canvas.Top ou Canvas.Left. Il s'agit de propriétés attachées qui vont permettre d'indiquer des informations pour le TextBlock, dans le contexte de son conteneur, le Canvas. C'est bien le Canvas qui va se servir de ces propriétés pour placer correctement le TextBlock.

Le même principe s'applique à la grille par exemple, si vous vous rappelez, afin d'indiquer dans quelle ligne ou colonne se place un contrôle :

Code : XML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Grid.Column="0" Text="Je suis un texte" />
    <TextBlock Grid.Column="1" Text="Je suis un autre texte" />
</Grid>
```

On utilise ici la propriété attachée Grid.Column pour indiquer à la grille à quel endroit il faut placer nos TextBlock.

Allez, j'arrête de vous embêter avec ces propriétés évoluées. J'en ai rapidement parlé pour que vous sachiez que quand je parle de propriétés, je parle en fait de quelque chose d'un peu plus poussé que ce que nous connaissons déjà.

Ce qu'il faut retenir c'est qu'il y a un système complexe derrière qui est presque invisible pour une utilisation de débutant. Si le sujet vous intéresse, n'hésitez pas à poser des questions ou à aller chercher des informations sur internet.

Sachez enfin qu'il est possible de créer nos propres propriétés de dépendances et nos propres propriétés attachées, mais nous sortons du cadre débutant et je ne le montrerai pas dans ce cours.

- Les propriétés de dépendances offrent un mécanisme plus complet que la propriété classique C#.
- Ces propriétés sont utilisées par le moteur XAML pour gérer les styles, la liaison de données, etc.
- En tant que débutant, nous n'avons pas vraiment besoin de savoir ce qu'il se cache là dessous.

Où est mon application ?

Nous avons commencé à créer des applications, mais... que sont-elles réellement ? Lorsque nous créons des programmes en C# pour Windows, nous obtenons des fichiers exécutables dont l'extension est .exe. Mais pour un téléphone, comment ça marche ? Et lorsque j'ajoute des images dans ma solution, où finissent-elles ?

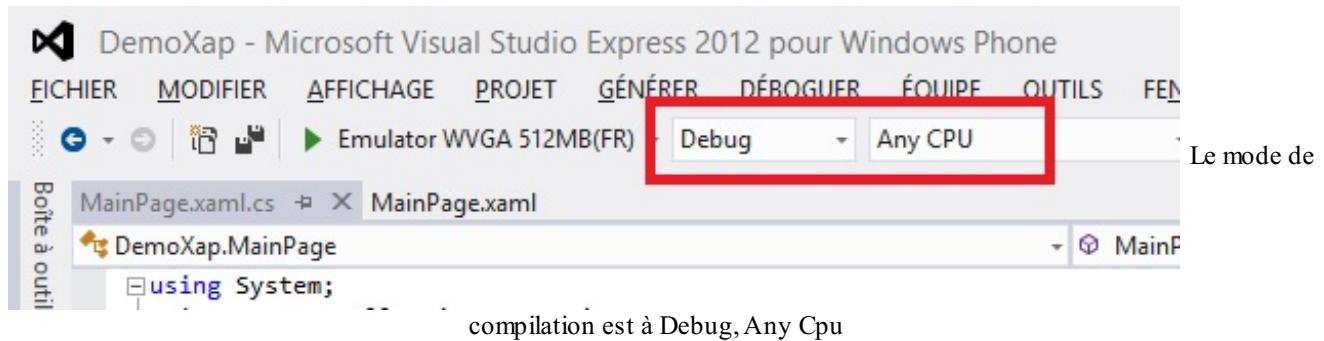
Voyons à présent les subtilités de la construction d'applications pour Windows Phone.

Le .XAP

Pour illustrer ce chapitre, créons un nouveau projet, que nous nommerons par exemple DemoXap.

Lorsque nous compilons notre application pour Windows Phone, celle-ci se génère par défaut dans un sous répertoire de notre projet : DemoXap\Bin\Debug, Debug étant le mode de compilation par défaut lorsque nous créons une solution. Nous verrons dans la dernière partie comment passer le mode de compilation en Release.

Toujours est-il que dans ce répertoire, il va s'y générer plein de choses, mais une seule nous intéresse vraiment ici, c'est le fichier qui porte le nom du projet et dont l'extension est .xap. Dans mon cas, il s'appelle DemoXap_Debug_AnyCPU.xap car mon mode de compilation est Debug, Any Cpu (voir la figure suivante).



Qu'est-ce donc que ce fichier ? En fait, c'est une archive au format compressé (zip) qui va contenir tous les fichiers dont notre application va avoir besoin. Si vous l'ouvrez avec votre décompresseur préféré, vous pourrez voir que cette archive contient les fichiers suivants :

- Assets
 - Tiles
 - FlipCycleTileLarge.png
 - FlipCycleTileMedium.png
 - FlipCycleTileSmall.png
 - IconicTileMediumLarge.png
 - IconicTileSmall.png
 - AlignmentGrid.png
 - ApplicationIcon.png
- AppManifest.xaml
- DemoXap.dll
- WMAppManifest.xml

Plein de choses que nous retrouvons dans notre solution. En fait, tout ce qui est du XAML et du code a été compilé dans l'assembly DemoXap.dll, les fichiers de Manifest sont laissés dans l'archive car ce sont eux qui donnent les instructions concernant la configuration de l'application. Et ensuite, il y a les quelques images.

Ajoutons un nouveau projet à notre solution (clic droit sur la solution > Ajouter > Nouveau projet), de type bibliothèque de classes Windows Phone, que nous nommons MaBibliotheque. Ciblez la plate-forme 8.0 et faites une référence à cette assembly depuis votre application DemoXap.

Compilez et vous pourrez retrouver cette assembly dans le .xap généré. Nous retrouvons donc toutes les assemblies dont le projet a besoin dans ce .xap.

Ajoutez à présent un nouvel élément déjà existant à votre projet et allez chercher une image par exemple. Elle s'ajoute dans la solution et si vous compilez et que vous ouvrez le fichier .xap, alors cette image apparaît dedans. C'est parce que votre image est par défaut ajoutée avec l'action de génération à « Contenu ». Si vous changez cette action de génération et que vous mettez « Resource », alors cette fois-ci, elle n'apparaît plus dans le .xap.

En fait, un fichier inclus en tant que Resource est compilé à l'intérieur de son assembly. Si vous avez été attentifs aux fichiers, vous aurez pu constater que l'assembly (DemoXap.dll) est beaucoup plus grosse lorsque l'image est compilée en tant que

ressource. Tandis que si elle est compilée en tant que contenu, alors celle-ci fait partie du .xap.

Affichage d'images en ressources

Ceci implique des contraintes. Si nous voulons afficher par code une image, nous avons vu que nous pouvions compiler l'image en tant que contenu et utiliser le code suivant :

Code : XML

```
<Image x:Name="MonImage" />
```

et

Code : C#

```
MonImage.Source = new BitmapImage(new Uri("/monimage.png",  
UriKind.Relative));
```

Ceci s'explique simplement. Étant donné que l'image est inclue dans le .xap, il va pouvoir aller la chercher tranquillement à l'emplacement /monimage.png, donc à la racine du package.

Essayez désormais de changer l'action de génération à ressource, et vous verrez que l'image ne s'affiche plus. En effet, l'image n'est plus à cet emplacement mais compilée à l'intérieur de l'assembly.

Il est quand même possible d'accéder à son contenu, mais cela demande d'aller lire à l'intérieur de l'assembly, ce que l'on peut faire de cette façon :

Code : C#

```
MonImage.Source = new BitmapImage(new  
Uri("/DemoXap;component/monimage.png", UriKind.Relative));
```

Bien sûr, si l'image n'est pas placée à la racine, mais dans un sous répertoire, il faudra indiquer le sous répertoire dans l'URL de l'image.

Remarquez que d'une manière générale, il vaudra mieux positionner le plus souvent possible l'action de génération à contenu afin de réduire la taille de l'assembly et ainsi augmenter les performances de chargement de celle-ci. Si les images sont en ressources, elles se chargeront plus vite. Si elles sont en contenu, alors c'est l'application qui se chargera plus vite. De même, pour des raisons de performances, vous aurez intérêt à utiliser des jpg partout sauf si vous avez besoin de transparence.

Accéder au flux des ressources

Il est possible d'accéder en lecture aux ressources. Cela peut être intéressant par exemple pour lire un fichier texte, xml ou autre. Le fichier doit bien sûr avoir l'action de génération à Ressource. Puis vous pouvez utiliser le code suivant :

Code : C#

```
StreamResourceInfo sr = Application.GetResourceStream(new  
Uri("/DemoXap;component/MonFichier.txt", UriKind.Relative));  
StreamReader s = new StreamReader(sr.Stream);  
MonTextBlock.Text = s.ReadToEnd();  
s.Close();
```

- Une application Windows Phone est générée sous la forme d'une archive dont l'extension est .xap.
- On peut embarquer des éléments dans notre assembly en positionnant l'action de génération à *Resource*.
- Un élément compilé avec l'action de génération à *Contenu* sera disponible directement dans le .xap.

ListBox

La **ListBox** est un élément incontournable dans la création d'applications pour Windows Phone. Elle permet un puissant affichage d'une liste d'élément. Voyons tout de suite de quoi il s'agit car vous allez vous en servir très souvent !

Un contrôle majeur

Utilisons notre designer préféré pour rajouter une **ListBox** dans notre page et nommons-là **ListeDesTaches**, ce qui donne le XAML suivant :

Code : XML

```
<ListBox x:Name="ListeDesTaches">
</ListBox>
```

Une **ListBox** permet d'afficher des éléments sous la forme d'une liste. Pour ajouter des éléments, on peut utiliser le XAML suivant :

Code : XML

```
<ListBox x:Name="ListeDesTaches">
    <ListBoxItem Content="Arroser les plantes" />
    <ListBoxItem Content="Tondre le gazon" />
    <ListBoxItem Content="Planter les tomates" />
</ListBox>
```

Ce qui donne la figure suivante.



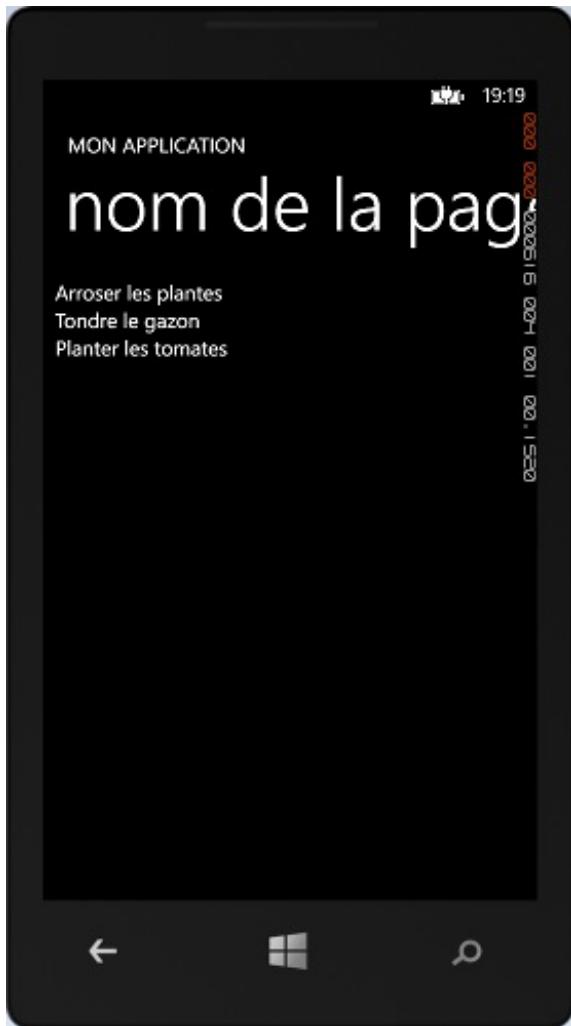
Une **ListBox** contenant 3 éléments positionnés par XAML

Il est cependant plutôt rare d'énumérer les éléments d'une `ListBox` directement dans le XAML. En général, ces éléments viennent d'une source dynamique construite depuis le code behind. Pour cela, il suffit d'alimenter la propriété `ItemSource` avec un objet implémentant `IEnumerable`, par exemple une liste. Supprimons les `ListBoxItem` de notre `ListBox` et utilisons ce code behind :

Code : C#

```
List<string> chosesAFaire = new List<string>
{
    "Arroser les plantes",
    "Tondre le gazon",
    "Planter les tomates"
};
ListeDesTaches.ItemsSource = chosesAFaire;
```

Il ne reste plus qu'à démarrer l'application. Nous pouvons voir que la `ListBox` s'est automatiquement remplie avec nos valeurs (voir la figure suivante).



La `ListBox` est remplie depuis le code-behind

Et ceci sans rien faire de plus. Ce qu'il est important de remarquer, c'est que si nous ajoutons beaucoup d'éléments à notre liste, alors celle-ci gère automatiquement un ascenseur pour pouvoir faire défiler la liste. La `ListBox` est donc une espèce de `ScrollView` qui contient une liste d'éléments dans un `StackPanel`. Tout ceci est géré nativement par la `ListBox`.

Nous avons quand même rencontré un petit truc étrange. Dans le XAML, nous avons mis la `ListBox`, mais la liste est vide, rien ne s'affiche dans le designer. Ce qui n'est pas très pratique pour créer notre page. C'est logique car l'alimentation de la `ListBox` est faite dans le constructeur, c'est-à-dire lorsque nous démarrons notre application. Nous verrons plus loin comment y remédier.

L'autre souci, c'est que si vous essayez de mettre des choses un peu plus complexes qu'une chaîne de caractère dans la ListBox, par exemple un objet :

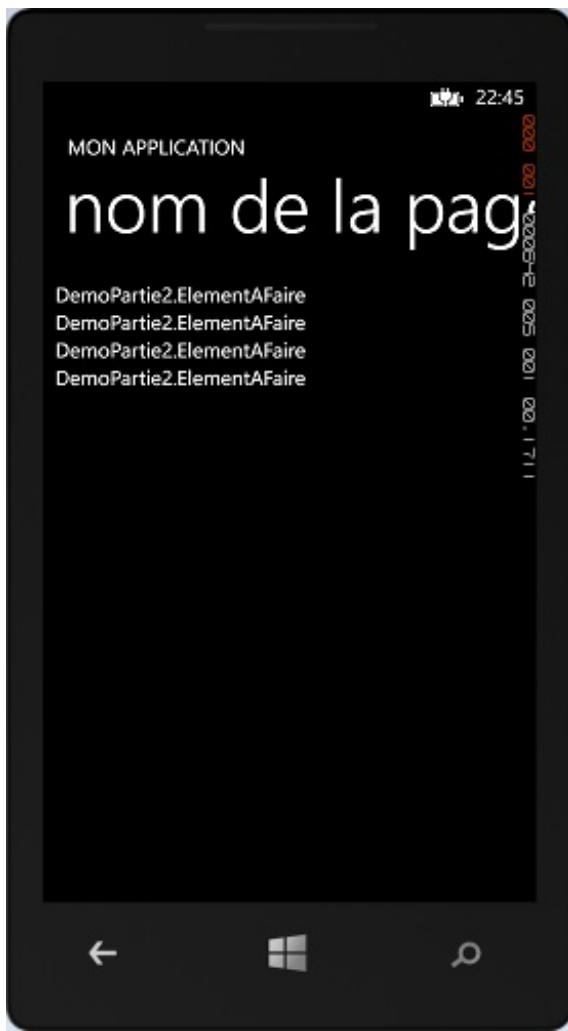
Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        List<ElementAFaire> chosesAFaire = new List<ElementAFaire>
        {
            new ElementAFaire { Priorite = 1, Description = "Arroser les plantes" },
            new ElementAFaire { Priorite = 2, Description = "Tondre le gazon" },
            new ElementAFaire { Priorite = 1, Description = "Planter les tomates" },
            new ElementAFaire { Priorite = 3, Description = "Laver la voiture" },
        };
        ListeDesTaches.ItemsSource = chosesAFaire.OrderBy(e => e.Priorite);
    }
}

public class ElementAFaire
{
    public int Priorite { get; set; }
    public string Description { get; set; }
}
```

vous aurez l'affichage suivant (voir la figure suivante).



C'est la représentation de l'objet qui s'affiche ici dans la ListBox

La ListBox affiche la représentation de l'objet, c'est-à-dire le résultat de sa méthode `ToString()`. Ce qui est un peu moche ici.

Vous me direz, il suffit de substituer la méthode `ToString()` avec quelque chose comme ça :

Code : C#

```
public class ElementAFaire
{
    public int Priorite { get; set; }
    public string Description { get; set; }

    public override string ToString()
    {
        return Priorite + " - " + Description;
    }
}
```

Et l'affaire est réglée ! Et je vous répondrai oui, parfait. Sauf que cela ne fonctionne que parce que nous affichons du texte ! Et si nous devions afficher du texte et une image ? Ou du texte et un bouton ?

Gérer les modèles

C'est là qu'interviennent les modèles, plus couramment appelés en anglais : template.

Ils permettent de personnaliser le rendu de son contrôle. Le contrôle garde toute sa logique mais peut nous confier le soin de gérer l'affichage, si nous le souhaitons. C'est justement ce que nous voulons faire.



L'utilisation de template est différente de la redéfinition de la propriété Content que nous avons déjà fait dans la partie précédente. D'une manière générale, on redéfinit la propriété Content pour redéfinir le look général d'un contrôle alors qu'on utilise les templates pour redéfinir l'apparence de plusieurs éléments d'une collection.

Nous allons donc redéfinir l'affichage de chaque élément de la liste. Pour cela, plutôt que d'afficher un simple texte, nous allons en profiter pour afficher une image pour la priorité et le texte de la description. Si la priorité est égale à 1, alors nous afficherons un rond rouge, sinon un rond vert (voir la figure suivante).



Créez un répertoire `Images` sous le répertoire `Assets` par exemple et ajoutez les deux images en tant que `Contenu`, comme nous l'avons déjà fait. Vous pouvez les télécharger la rouge [ici](#), et la verte [là](#).

La première chose à faire est de définir le modèle des éléments de la `ListBox`. Cela se fait avec le code suivant :

Code : XML

```
<ListBox x:Name="ListeDesTaches">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <Image Source="{Binding Image}" Width="30"
Height="30" />
                <TextBlock Text="{Binding Description}" Margin="20 0
0 0" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

On redéfinit la propriété `ItemTemplate`, c'est-à-dire le modèle d'un élément de la liste. Puis à l'intérieur de la balise `DataTemplate`, on peut rajouter nos propres contrôles. Ici, il y a un conteneur, le `StackPanel`, qui contient une image et une zone de texte en lecture seule. Le `DataTemplate` doit toujours contenir un seul contrôle.

Vu que vous avez l'œil, vous avez remarqué des extensions de balisage XAML à l'intérieur du contrôle `Image` et du contrôle `TextBlock`. Je vais y revenir dans le prochain chapitre.

En attendant, nous allons modifier légèrement le code behind de cette façon :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        List<ElementAFaire> chosesAFaire = new List<ElementAFaire>
        {
            new ElementAFaire { Priorite = 1, Description = "Arroser
les plantes" },
            new ElementAFaire { Priorite = 2, Description = "Tondre
le gazon" },
            new ElementAFaire { Priorite = 1, Description = "Planter
les tomates" },
            new ElementAFaire { Priorite = 3, Description = "Laver
la voiture" },
        };

        ListeDesTaches.ItemsSource = chosesAFaire.OrderBy(e =>
e.Priorite).Select(e => new ElementAFaireBinding { Description =
e.Description, Image = ObtientImage(e.Priorite) });
    }

    private BitmapImage ObtientImage(int priorite)
```

```
{  
    if (priorite <= 1)  
        return new BitmapImage(new  
Uri("/Assets/Images/vert.png", UriKind.Relative));  
    return new BitmapImage(new Uri("/Assets/Images/rouge.png",  
UriKind.Relative));  
}
```

Pour rappel, la classe `BitmapImage` se trouve dans l'espace de nom `System.Windows.Media.Imaging`. Vérifiez également que le l'URL passée à la classe `BitmapImage` correspond à l'emplacement où vous avez ajouté les images.

Nous aurons besoin de la nouvelle classe suivante :

Code : C#

```
public class ElementAFaireBinding  
{  
    public BitmapImage Image { get; set; }  
    public string Description { get; set; }  
}
```

Le principe est de construire des éléments énumérables à partir de notre liste. Il s'agit d'y mettre un nouvel objet qui possède une propriété `Description` et une propriété `Image` qui contient un objet `BitmapImage` construit à partir de la valeur de la priorité de la tâche.

Il est important de constater que la classe contient des propriétés qui ont les mêmes noms que ce qu'on a écrit dans l'extension de balisage vue plus haut.

Code : XML

```
<Image Source="{Binding Image}" Width="30" Height="30" />  
<TextBlock Text="{Binding Description}" Margin="20 0 0 0" />
```

J'y reviendrai plus tard, mais nous avons ici fait ce qu'on appelle un binding, que l'on peut traduire par une liaison de données. Nous indiquons que nous souhaitons mettre la valeur de la propriété `Image` de l'élément courant dans la propriété `Source` de l'`Image` et la valeur de la propriété `Description` de l'élément courant dans la propriété `Text` du `TextBlock`. Rappelez-vous, l'élément courant est justement un objet spécial qui contient ces propriétés.

Si nous exécutons le code, nous obtenons donc la figure suivante.



Les images s'affichent dans la ListBox grâce au modèle

Magique ! Le seul défaut viendrait de mes images qui ne sont pas transparentes...

i Remarquons qu'il est obligatoire de créer une nouvelle classe contenant les propriétés `Description` et `Image`. Il n'aurait pas été possible d'utiliser un type anonyme car le type anonyme n'est pas `public` mais `internal`. Dans ce cas, il aurait fallu rajouter une instruction particulière permettant de dire que les classes qui font la liaison de données ont le droit de voir les classes internes. Ce qui est beaucoup plus compliqué que ce que nous avons fait !

Sachez qu'il existe beaucoup de contrôles qui utilisent ce même mécanisme de modèle, nous aurons l'occasion d'en voir d'autres. C'est une fonctionnalité très puissante qui nous laisse beaucoup de contrôle sur le rendu de nos données.

Sélection d'un élément

La `ListBox` gère également un autre point intéressant : savoir quel élément est sélectionné. Cela se fait en toute logique grâce à un événement. Pour que cela soit plus simple, enlevons nos templates et modifions le XAML pour avoir :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="100" />
    </Grid.RowDefinitions>
    <ListBox x:Name="ListeDesTaches"
        SelectionChanged="ListeDesTaches_SelectionChanged">
        </ListBox>
        <TextBlock x:Name="Selection" Grid.Row="1" />
    </Grid>
```

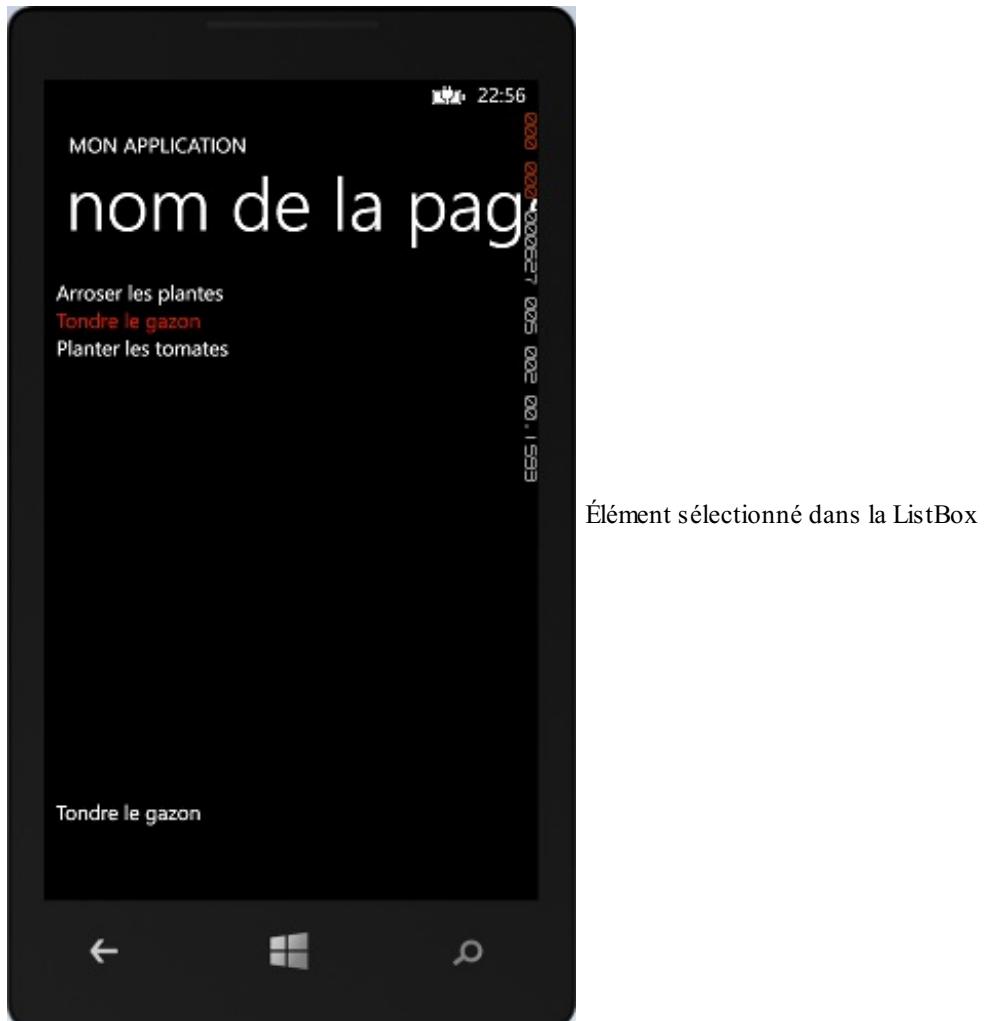
Notre grille a donc deux lignes, la première contenant la `ListBox` et la seconde un `TextBlock`. Remarquons l'événement `SelectionChanged` qui est associé à la méthode `ListeDesTaches_SelectionChanged`. Dans le code behind, nous aurons :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
        List<string> chosesAFaire = new List<string>
        {
            "Arroser les plantes",
            "Tondre le gazon",
            "Planter les tomates"
        };
        ListeDesTaches.ItemsSource = chosesAFaire;
    }

    private void ListeDesTaches_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        if (e.AddedItems.Count > 0)
            Selection.Text = e.AddedItems[0].ToString();
    }
}
```

Ce qui va nous permettre d'afficher dans le `TextBlock` la valeur de ce que nous avons choisi (voir la figure suivante).



Nous voyons au passage que la sélection est mise en valeur automatiquement dans la `ListBox`.

Remarquez qu'étant donné que notre `ListBox` a un nom et donc une variable pour la manipuler, il est également d'obtenir la valeur sélectionnée grâce à l'instruction suivante :

Code : C#

```
private void ListeDesTaches_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    Selection.Text = ListeDesTaches.SelectedItem.ToString();
}
```

Ce qui est d'ailleurs plus propre.

L'objet `SelectedItem` est du type `object` et sera du type de ce que nous avons mis dans la propriété `ItemsSource`. Étant donné que nous avons mis une liste de chaîne de caractères, `SelectedItem` sera une chaîne, nous pouvons donc faire :

Code : C#

```
private void ListeDesTaches_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    Selection.Text = (string)ListeDesTaches.SelectedItem;
}
```



Dans l'exemple précédent, avec les templates, `SelectedItem` aurait bien sûr été du type `ElementAffaireBinding`.

De la même façon, l'index de l'élément sélectionné sera accessible grâce à la propriété `SelectedIndex` et sera du type entier. Ce qui permet par exemple de présélectionner une valeur dans notre `ListBox` au chargement de celle-ci. Ainsi, pour sélectionner le deuxième élément, je pourrais faire :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    List<string> chosesAFaire = new List<string>
    {
        "Arroser les plantes",
        "Tondre le gazon",
        "Planter les tomates"
    };
    ListeDesTaches.ItemsSource = chosesAFaire;
    ListeDesTaches.SelectedIndex = 1;
}
```

Ou encore :

Code : C#

```
public MainPage()
{
    InitializeComponent();
```

```

List<string> chosesAFaire = new List<string>
{
    "Arroser les plantes",
    "Tondre le gazon",
    "Planter les tomates"
};
ListeDesTaches.ItemsSource = chosesAFaire;
ListeDesTaches.SelectedValue = chosesAFaire[1];
}

```

sachant que le fait d'initialiser la sélection d'un élément par code déclenche l'événement de changement de sélection ; ce qui nous arrange pour que notre `TextBlock` soit rempli. Pour éviter ceci, il faudrait associer la méthode à l'événement de changement de sélection après avoir sélectionné l'élément. Cela revient à enlever la définition de l'événement dans le XAML :

Code : XML

```

<ListBox x:Name="ListeDesTaches">
</ListBox>

```

Et à s'abonner à l'événement depuis le code behind, après avoir initialisé l'élément sélectionné :

Code : C#

```

public MainPage()
{
    InitializeComponent();
    List<string> chosesAFaire = new List<string>
    {
        "Arroser les plantes",
        "Tondre le gazon",
        "Planter les tomates"
    };
    ListeDesTaches.ItemsSource = chosesAFaire;
    ListeDesTaches.SelectedValue = chosesAFaire[1];
    ListeDesTaches.SelectionChanged +=
        ListeDesTaches_SelectionChanged;
}

```

Pour finir sur la sélection d'un élément, il faut savoir que la `ListBox` peut permettre de sélectionner plusieurs éléments en changeant sa propriété `SelectionMode` et en la passant à `Multiple` :

Code : XML

```

<ListBox x:Name="ListeDesTaches"
SelectionChanged="ListeDesTaches_SelectionChanged"
SelectionMode="Multiple">
</ListBox>

```

Et nous pourrons récupérer les différentes valeurs sélectionnées grâce à la collection `SelectedItems` :

Code : C#

```

private void ListeDesTaches_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    string selection = string.Empty;
}

```

```
foreach (string choix in ListeDesTaches.SelectedItems)
{
    selection += choix + ";";
}
Selection.Text = selection;
```

Très bien tout ça. 😊

- La ListBox est un contrôle très utile qui permet d'afficher une liste d'éléments très facilement.
- Il est possible de personnaliser efficacement le rendu de chaque élément d'une ListBox grâce au système de modèles.
- La ListBox est un contrôle complet qui possède toute une gestion de l'élément sélectionné.

La manipulation des données (DataBinding & Converters)

S'il y a vraiment un chapitre où il faut être attentif, c'est bien celui-là. La liaison de données (ou databinding en anglais) est une notion indispensable et incontournable pour toute personne souhaitant réaliser des applications XAML sérieuses. Nous allons voir dans ce chapitre de quoi il s'agit et comment le mettre en place.

Principe du Databinding

Le databinding se traduit en français par « liaison de données ». Il s'agit de la possibilité de lier un contrôle à des données. Le principe consiste à indiquer à un contrôle où il peut trouver sa valeur et celui-ci se débrouille pour l'afficher. Nous l'avons entreaperçu dans le chapitre précédent avec la `ListBox`, il est temps de creuser un peu son fonctionnement.

Techniquement, le moteur utilise un objet de type `Binding` qui associe une source de données à un élément de destination, d'où l'emploi du mot raccourci `binding` pour représenter la liaison de données.

Le binding permet de positionner automatiquement des valeurs aux propriétés des contrôles en fonction du contenu de la source de données. En effet, il est très fréquent de mettre des valeurs dans des `TextBox`, dans des `TextBlock` ou dans des `ListBox`, comme nous l'avons fait. Le binding est là pour faciliter tout ce qui peut être automatisable et risque d'erreurs. De plus, si la source de données change, il est possible de faire en sorte que le contrôle soit automatiquement mis à jour.

Inversement, si des modifications sont faites depuis l'interface, alors on peut être notifié automatiquement des changements.

Le binding des données

Pour illustrer le fonctionnement le plus simple du binding, nous allons lier une zone de texte modifiable (`TextBox`) à une propriété d'une classe. Puisque le `TextBox` travaille avec du texte, il faut créer une propriété de type `string` sur une classe. Cette classe sera le contexte de données du contrôle. Créons donc la nouvelle classe suivante :

Code : C#

```
public class Contexte
{
    public string Valeur { get; set; }
}
```

Et une instance de cette classe dans notre code behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private Contexte contexte;

    public MainPage()
    {
        InitializeComponent();

        contexte = new Contexte { Valeur = "Nicolas" };
        DataContext = contexte;
    }
}
```

Nous remarquons à la fin du constructeur que la propriété `DataContext` de la page est initialisée avec notre contexte de données, étape obligatoire permettant de lier la page au contexte de données.

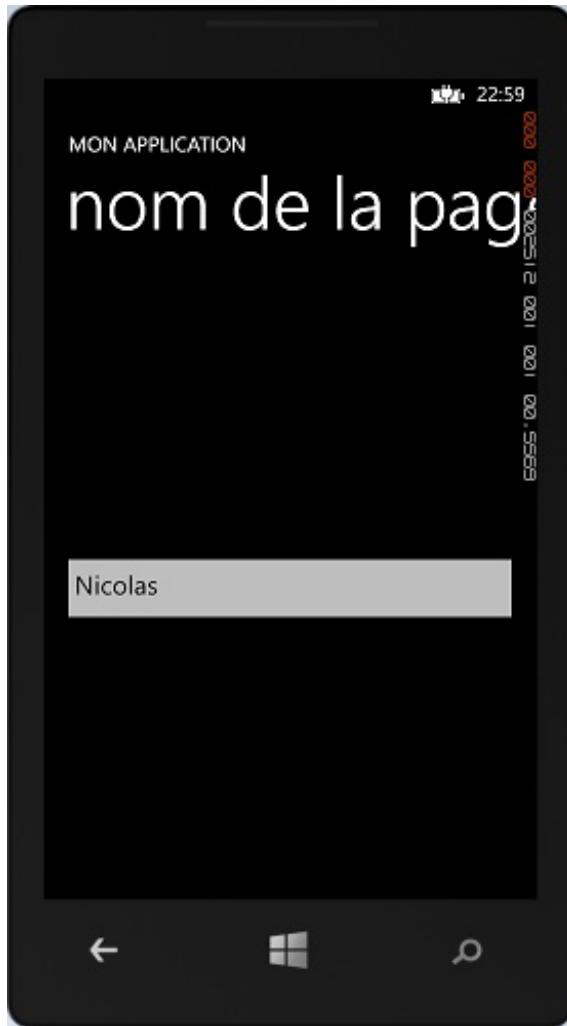
Chaque objet `FrameworkElement` possède une propriété `DataContext` et chaque élément enfant d'un autre élément hérite de son contexte de données implicitement. C'est pour cela qu'ici on initialise notre contexte de données au niveau de la page afin que tous les éléments contenus dans la page héritent de ce contexte de données.

Il ne reste plus qu'à ajouter un contrôle `TextBox` qui sera lié à cette propriété :

Code : XML

```
<TextBox Text="{Binding Valeur}" Height="80"/>
```

Cela se fait grâce à l'expression de balisage {Binding}. Lorsque nous exécutons notre application, nous pouvons voir que la TextBox s'est correctement remplie avec la chaîne de caractères Nicolas (voir la figure suivante).



La valeur du TextBox est liée à la propriété Valeur

Et tout ça automatiquement, sans avoir besoin de positionner la valeur de la propriété Text depuis le code behind.

Qu'est-ce qu'a fait le moteur de binding ? Il est allé voir dans son contexte (propriété DataContext) puis il est allé prendre le contenu de la propriété Valeur de ce contexte pour le mettre dans la propriété Text du TextBox, c'est-à-dire la chaîne « Nicolas ».

Il faut faire attention car dans le XAML nous écrivons du texte, si nous orthographions mal Valeur, par exemple en oubliant le « u » :

Code : XML

```
<TextBox Text="{Binding Valer}" Height="80"/>
```

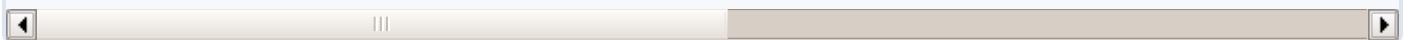
Alors la liaison de données n'aura pas lieu car la propriété est introuvable sur l'objet Contexte. Ce qui est vrai ! « Valer » n'existe pas. Il n'y a pas de vérification à la compilation, c'est donc au moment de l'exécution que nous remarquerons l'absence du binding.

La seule information que nous aurons, c'est dans la fenêtre de sortie du débogueur, où nous aurons :

Code : Autre

```
System.Windows.Data Error: BindingExpression path error: 'Valer' property not fou
```

```
(HashCode=54897010). BindingExpression: Path='Vale' DataItem='DemoPartie2.Contex  
(Name='') ; target property is 'Text' (type 'System.String')..
```



indiquant que la propriété n'a pas été trouvée.



Attention, le binding ne fonctionne qu'avec une propriété ayant la visibilité public.

Il est également possible de définir un binding par code behind, pour cela enlevez l'expression de balisage dans le XAML et donnez un nom à votre contrôle :

Code : XML

```
<TextBox x:Name="MonTextBox" Height="80"/>
```

puis utilisez le code behind suivant :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private Contexte contexte;

    public MainPage()
    {
        InitializeComponent();

        MonTextBox.SetBinding(TextBox.TextProperty, new
Binding("Valeur"));
        contexte = new Contexte { Valeur = "Nicolas" };
        DataContext = contexte;
    }
}
```



La classe Binding fait partie de l'espace de nom `System.Windows.Data` que vous devrez ajouter avec un `using`.

On en profite pour constater que le binding se fait bien avec une propriété de dépendance, ici `TextBox.TextProperty`.

Le binding est très pratique pour qu'un contrôle se remplisse avec la bonne valeur.

Il est possible de modifier la valeur affichée dans la zone de texte très facilement en modifiant la valeur du contexte depuis le code. Pour cela, changeons le XAML pour ajouter un bouton qui va nous permettre de déclencher ce changement de valeur :

Code : XML

```
<StackPanel>
    <TextBox Text="{Binding Valeur}" Height="80"/>
    <Button Content="Changer valeur" Tap="Button_Tap" />
</StackPanel>
```

Et dans l'événement de Tap, faisons :

Code : C#

```
private void button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    if (contexte.Valeur == "Nicolas")
        contexte.Valeur = "Jérémie";
    else
        contexte.Valeur = "Nicolas";
}
```

Par contre, il va manquer quelque chose. Un moyen de dire à la page « hé, j'ai modifié un truc, il faut que tu regardes si tu es impacté ». Ça, c'est le rôle de l'interface `INotifyPropertyChanged`. Notre classe de contexte doit implémenter cette interface et faire en sorte que quand on modifie la propriété, elle lève l'événement qui va permettre à l'interface de se mettre à jour. Notre classe de contexte va donc devenir :

Code : C#

```
public class Contexte : INotifyPropertyChanged
{
    private string valeur;
    public string Valeur
    {
        get
        {
            return valeur;
        }
        set
        {
            if (value == valeur)
                return;
            valeur = value;
            NotifyPropertyChanged("Valeur");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }
}
```



N'oubliez pas de rajouter le `using System.ComponentModel;`



L'implémentation de l'interface `INotifyPropertyChanged` est très classique dans les applications XAML, aussi cette façon de faire est souvent factorisée dans des classes mères. Vous le verrez dans le chapitre suivant.

Ici, lorsque nous affectons une valeur à la propriété, la méthode `NotifyPropertyChanged` est appelée en passant en paramètre le nom de la propriété de la classe qu'il faut rafraîchir sur la page. Attention, c'est une erreur classique de ne pas avoir le bon nom de propriété en paramètres, faites-y attention.

Notez qu'avec la version 8.0 du SDK (en fait, grâce au framework 4.5), il est possible d'utiliser une autre solution pour

implémenter `INotifyPropertyChanged` sans avoir l'inconvénient de devoir passer une chaîne de caractère en paramètre. Il suffit d'utiliser l'attribut de méthode `CallerMemberName`, qui permet d'obtenir le nom de la propriété (ou méthode) qui a appelé notre méthode, en l'occurrence il s'agira justement du nom de la propriété qu'on aurait passé en paramètre :

Code : C#

```
public class Contexte : INotifyPropertyChanged
{
    private string valeur;
    public string Valeur
    {
        get { return valeur; }
        set { NotifyPropertyChanged(ref valeur, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }
}
```

La syntaxe est plus élégante et il n'y a pas de risque de mal orthographier le nom de la propriété. Par contre, je suis désolé pour ceux qui suivent le cours avec la version 7.1 du SDK, il faudra continuer à utiliser la solution que j'ai présentée juste avant.

Note, il faudra inclure l'espace de nom :

Code : C#

```
using System.Runtime.CompilerServices;
```

Relançons l'application, nous pouvons voir que le clic sur le bouton entraîne bien le changement de valeur dans la `TextBox`.



Ok, c'est bien beau tout ça, mais n'est-ce pas un peu compliqué par rapport à ce qu'on a déjà fait, à savoir modifier directement la valeur de la propriété `Text` ?

Effectivement, dans ce cas-là, on pourrait juger que c'est sortir l'artillerie lourde pour pas grand-chose. Cependant c'est une bonne pratique dans la mesure où on automatise le processus de mise à jour de la propriété. Vous aurez remarqué que l'on ne manipule plus directement le contrôle mais une classe qui n'a rien à voir avec le `TextBox`. Et quand il y a plusieurs valeurs à mettre à jour d'un coup, c'est d'autant plus facile. De plus, nous pouvons faire encore mieux avec ce binding grâce à la bidirectionnalité de la liaison de données. Par exemple, modifions le XAML pour rajouter encore un bouton :

Code : XML

```
<StackPanel>
    <TextBox Text="{Binding Valeur, Mode=TwoWay}" Height="80"/>
```

```
<Button Content="Changer valeur" Tap="Button_Tap" />
<Button Content="Afficher valeur" Tap="Button_Tap_1" />
</StackPanel>
```

La méthode associée à ce nouveau clic affichera la valeur du contexte :

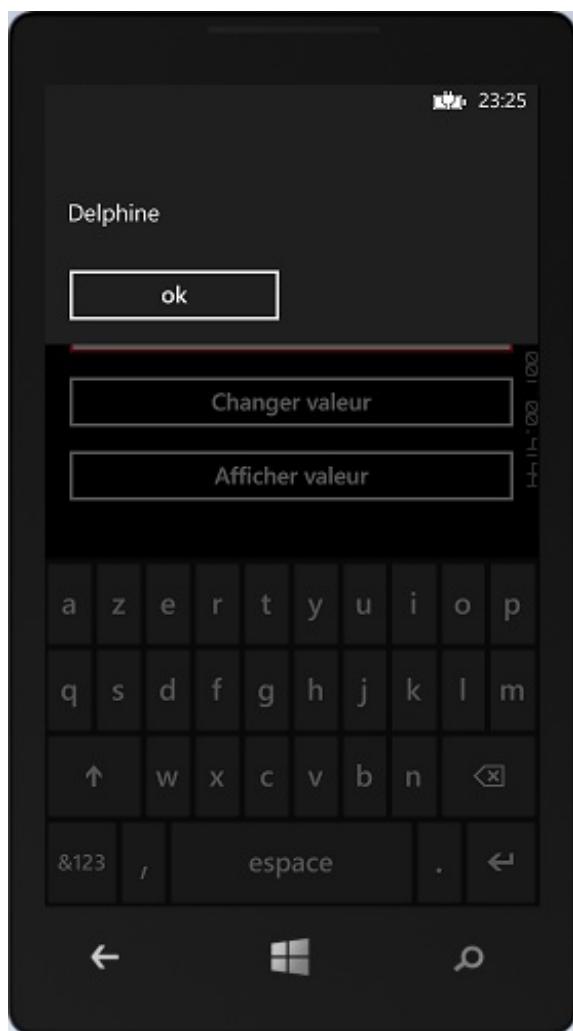
Code : C#

```
private void button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    MessageBox.Show(contexte.Valeur);
}
```

On utilise pour cela la méthode `MessageBox.Show` qui affiche une petite boîte de dialogue minimaliste.

Les lecteurs attentifs auront remarqué que j'ai enrichi le binding sur la `Valeur` en rajoutant un `Mode=TwoWay`. Ceci permet d'indiquer que le binding s'effectue dans les deux sens. C'est-à-dire que si je modifie la propriété de la classe de contexte, alors l'interface est mise à jour. Inversement, si je modifie la valeur de la `TextBox` avec le clavier virtuel, alors la propriété de la classe est également mise à jour.

C'est à cela que va servir notre deuxième bouton. Démarrer l'application, modifiez la valeur du champ avec le clavier virtuel et cliquez sur le bouton permettant d'afficher la valeur (voir la figure suivante).



La valeur est affichée grâce à la liaison de données bidirectionnelle

La valeur est bien récupérée. Vous pouvez faire le test en enlevant le mode `TwoWay`, vous verrez que vous ne récupérerez pas la bonne valeur.

Plutôt pas mal non ?

Maintenant que nous avons un peu mieux compris le principe du binding, il est temps de préciser un point important. Pour illustrer le fonctionnement du binding, j'ai créé une classe puis j'ai créé une variable à l'intérieur de cette classe contenant une instance de cette classe. Puis j'ai relié cette classe au contexte de données de la page. En général, on utilise ce découpage dans une application utilisant le patron de conception MVVM (*Model-View-ViewModel*). Je parlerai de ce design pattern dans le prochain chapitre.

Remarquez que l'on voit souvent la construction où c'est la classe de la page qui sert de contexte de données de la page. Cela veut dire qu'on peut modifier l'exemple précédent pour que ça soit la classe MainPage qui implémente l'interface INotifyPropertyChanged, ce qui donne :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    private string valeur;
    public string Valeur
    {
        get { return valeur; }
        set { NotifyPropertyChanged(ref valeur, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    public MainPage()
    {
        InitializeComponent();

        Valeur = "Nicolas";
        DataContext = this;
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        if (Valeur == "Nicolas")
            Valeur = "Jérémie";
        else
            Valeur = "Nicolas";
    }

    private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        MessageBox.Show(Valeur);
    }
}
```

La classe Contexte n'a plus de raison d'être. Tout est porté par la classe représentant la page. On affecte donc l'objet this à la

propriété `DataContext` de la page. Cette construction est peut-être un peu plus perturbante d'un point de vue architecture où on a tendance à mélanger les responsabilités dans la classe mais elle a l'avantage de simplifier pas mal le travail. Personnellement j'emploie très rarement ce genre de construction (j'utilise plutôt un dérivé de la première solution), mais je l'utiliserais de temps en temps dans ce cours pour simplifier le code.

Notre `ListBox` fonctionne également avec le binding. Il suffit d'utiliser l'expression de balisage avec la propriété `ItemsSource`:

Code : XML

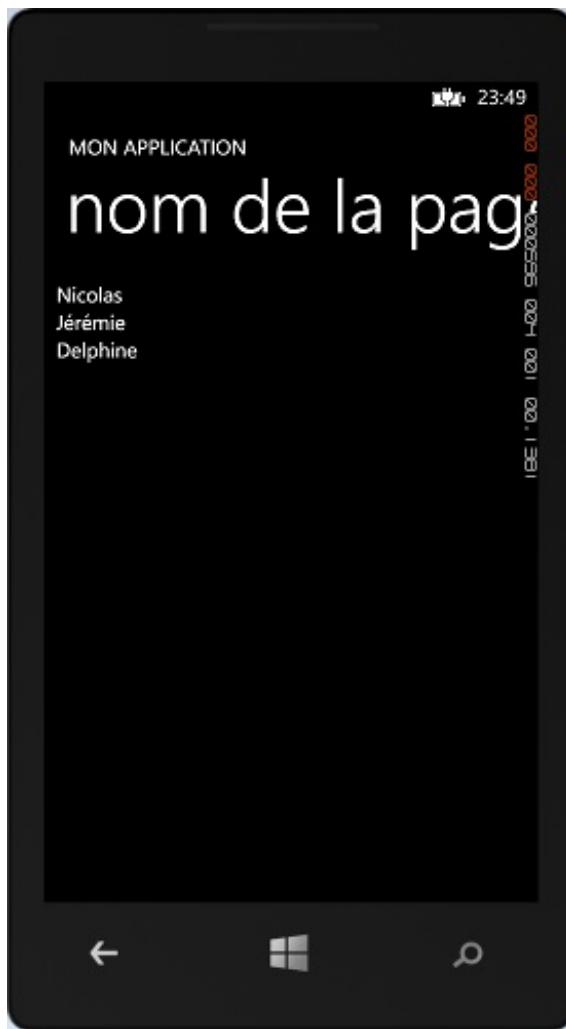
```
<ListBox ItemsSource="{Binding Prenoms}" />
```

Nous aurons bien sûr défini la propriété `Prenoms` dans notre contexte :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,  
INotifyPropertyChanged  
{  
    private List<string> prenoms;  
    public List<string> Prenoms  
    {  
        get { return prenoms; }  
        set { NotifyPropertyChanged(ref prenoms, value); }  
    }  
  
    public event PropertyChangedEventHandler PropertyChanged;  
  
    public void NotifyPropertyChanged(string nomPropriete)  
    {  
        if (PropertyChanged != null)  
            PropertyChanged(this, new  
PropertyChangedEventArgs(nomPropriete));  
    }  
  
    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,  
[CallerMemberName] string nomPropriete = null)  
    {  
        if (object.Equals(variable, valeur)) return false;  
  
        variable = valeur;  
        NotifyPropertyChanged(nomPropriete);  
        return true;  
    }  
  
    public MainPage()  
    {  
        InitializeComponent();  
  
        Prenoms = new List<string> { "Nicolas", "Jérémie",  
"Delphine" };  
        DataContext = this;  
    }  
}
```

Ce qui donne la figure suivante.



Liaison de données avec une ListBox

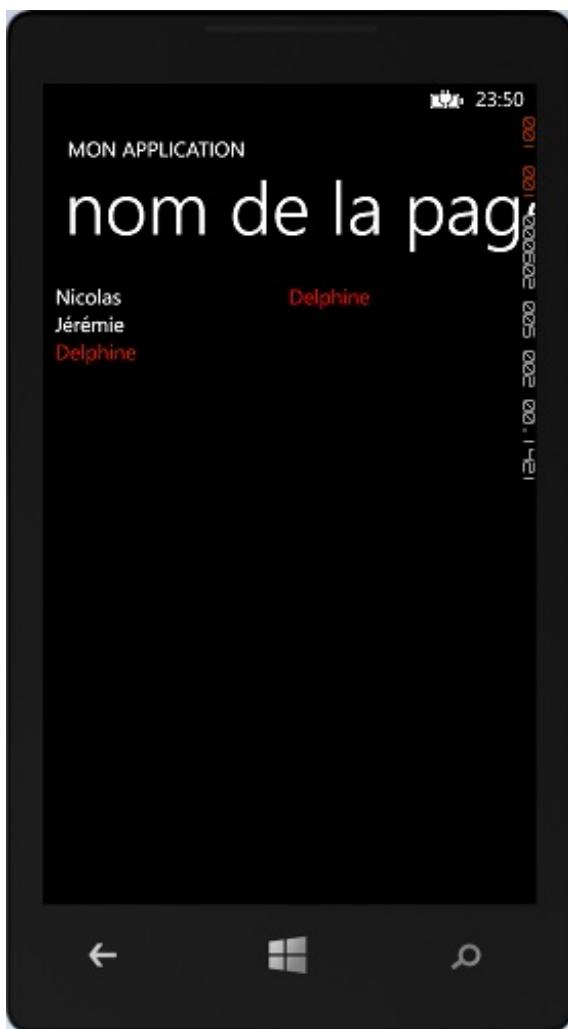
Le binding est encore plus puissant que ça, voyons encore un point intéressant. Il s'agit de la capacité de lier une propriété d'un contrôle à la propriété d'un autre contrôle.

Par exemple, mettons un `TextBlock` en plus de notre `ListBox` :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <ListBox x:Name="ListBoxPrenoms" ItemsSource="{Binding Prenoms}" />
    <TextBlock Grid.Column="1" Text="{Binding ElementName=ListBoxPrenoms, Path=SelectedItem}" Foreground="Red" />
</Grid>
```

Regardons l'expression de binding du `TextBlock`, nous indiquons que nous voulons lier la valeur du `TextBlock` à la propriété `SelectedItem` du contrôle nommé `ListBoxPrenoms`. Ici cela voudra dire que lorsque nous sélectionnerons un élément dans la `ListBox`, alors celui-ci sera automatiquement affiché dans le `TextBlock`, sans avoir rien d'autre à faire :



Liaison de données entre propriétés de contrôles

Tout simplement. 😊

Voilà pour cet aperçu du binding. Nous n'en avons pas vu toutes les subtilités mais ce que nous avons étudié ici vous sera grandement utile et bien souvent suffisant dans vos futures applications Windows Phone !

Binding et mode design

Vous vous rappelez notre `ListBox` quelques chapitres avant ? Nous avions créé une `ListBox` avec une liste de choses à faire. Cette liste de choses à faire était alimentée par la propriété `ItemsSource` dans le constructeur de la page. Le problème c'est que notre `ListBox` était vide en mode design. Du coup, pas facile pour faire du style, pour mettre d'autres contrôles, etc. Le binding va nous permettre de résoudre ce problème. Prenons le code XAML suivant :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <ListBox ItemsSource="{Binding ListeDesTaches}" >
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <Image Source="{Binding Image}" Width="30"
Height="30" />
                    <TextBlock Text="{Binding Description}"
Margin="20 0 0 0" />
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Grid>
```

Avec dans le code behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    private IEnumerable<ElementAFaireBinding> listeDesTaches;
    public IEnumerable<ElementAFaireBinding> ListeDesTaches
    {
        get { return listeDesTaches; }
        set { NotifyPropertyChanged(ref listeDesTaches, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    public MainPage()
    {
        InitializeComponent();

        List<ElementAFaire> chosesAFaire = new List<ElementAFaire>
        {
            new ElementAFaire { Priorite = 1, Description = "Arroser
les plantes" },
            new ElementAFaire { Priorite = 2, Description = "Tondre
le gazon" },
            new ElementAFaire { Priorite = 1, Description = "Planter
les tomates" },
            new ElementAFaire { Priorite = 3, Description = "Laver
la voiture" },
        };

        ListeDesTaches = chosesAFaire.OrderBy(e =>
e.Priorite).Select(e => new ElementAFaireBinding { Description =
e.Description, Image = ObtientImage(e.Priorite) });

        DataContext = this;
    }

    private BitmapImage ObtientImage(int priorite)
    {
        if (priorite <= 1)
            return new BitmapImage(new
Uri("/Assets/Images/vert.png", UriKind.Relative));
        return new BitmapImage(new Uri("/Assets/Images/rouge.png",
UriKind.Relative));
    }

    public class ElementAFaire
    {
        public int Priorite { get; set; }
```

```
    public string Description { get; set; }
}
public class ElementAFaireBinding
{
    public BitmapImage Image { get; set; }
    public string Description { get; set; }
}
```

Pour l'instant, rien n'a changé, la `ListBox` est toujours vide en mode design. Sauf que nous avons également la possibilité de lier le mode design à un contexte de design. Créons donc une nouvelle classe :

Code : C#

```
public class MainPageDesign
{
    public IEnumerable<ElementAFaireBinding> ListeDesTaches
    {
        get
        {
            return new List<ElementAFaireBinding>
            {
                new ElementAFaireBinding { Image = new
                    BitmapImage(new Uri("/Assets/Images/vert.png", UriKind.Relative)),
                    Description = "Arroser les plantes"},
                new ElementAFaireBinding { Image = new
                    BitmapImage(new Uri("/Assets/Images/rouge.png", UriKind.Relative)),
                    Description = "Tondre le gazon"},
                new ElementAFaireBinding { Image = new
                    BitmapImage(new Uri("/Assets/Images/rouge.png", UriKind.Relative)),
                    Description = "Planter les tomates"},
                new ElementAFaireBinding { Image = new
                    BitmapImage(new Uri("/Assets/Images/vert.png", UriKind.Relative)),
                    Description = "Laver la voiture"},
            };
        }
    }
}
```

Cette classe ne fait que renvoyer une propriété `ListeDesTaches` avec des valeurs de design. Compilez et rajoutez maintenant dans le XAML l'instruction suivante avant le conteneur de plus haut niveau :

Code : XML

```
<d:DesignProperties.DataContext>
    <design:MainPageDesign />
</d:DesignProperties.DataContext>
```

Ceci permet de dire que le contexte de design est à aller chercher dans la classe `MainPageDesign`.

Attention, la classe `MainPageDesign` n'est pas connue de la page ! Il faut lui indiquer où elle se trouve, en indiquant son espace de nom, un peu comme un `using C#`. Cette propriété se rajoute dans les propriétés de la page, `<phone:PhoneApplicationPage>` :

Code : XML

```
<phone:PhoneApplicationPage
    x:Class="DemoPartie2.MainPage"
    [... plein de choses ...]
    xmlns:design="clr-namespace:DemoPartie2"
    shell:SystemTray.isVisible="True">
```

Avec cette écriture, je lui dis que le raccourci « design » correspond à l'espace de nom DemoPartie2.

Nous commençons à voir apparaître des choses dans le designer de Visual Studio (voir la figure suivante).



Le designer affiche les données de design grâce à la liaison de données

Avouez que c'est beaucoup plus pratique pour réaliser le design de sa page. 😊



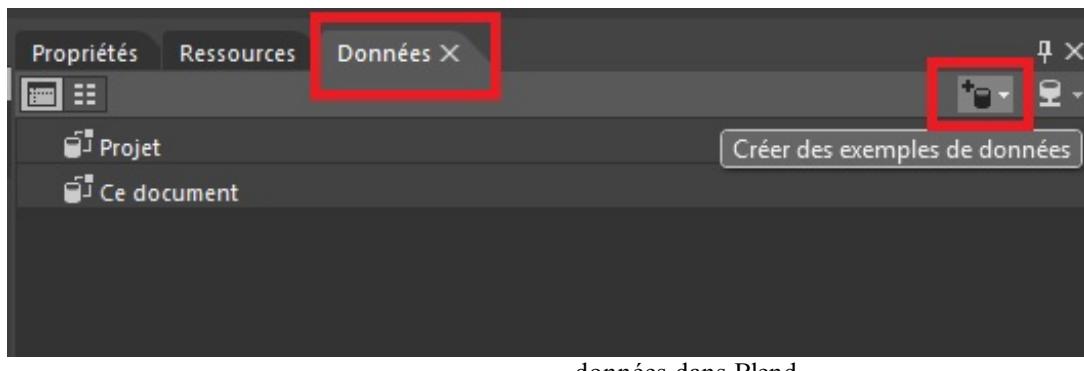
Dans la mesure où le contexte de la page et le contexte de design doivent contenir des propriétés communes, c'est une bonne idée de faire en sorte qu'ils implémentent tous les deux une interface qui contienne les propriétés à implémenter des deux côtés...

Avant de terminer, il faut savoir que Blend est également capable de nous générer des données de design sans que l'on ait forcément besoin de créer une classe spécifique. Pour illustrer ceci, repartez d'une nouvelle page vide. Puis ouvrez la page dans Expression Blend.



Si vous suivez ce cours pas à pas, vous vous trouvez sûrement dans l'espace de travail animation. Revenez à l'espace de travail design (menu Fenêtre > Espaces de travail > Design).

Cliquez sur l'onglet données, puis sur l'icône tout à droite créer des exemples de données (voir la figure suivante).

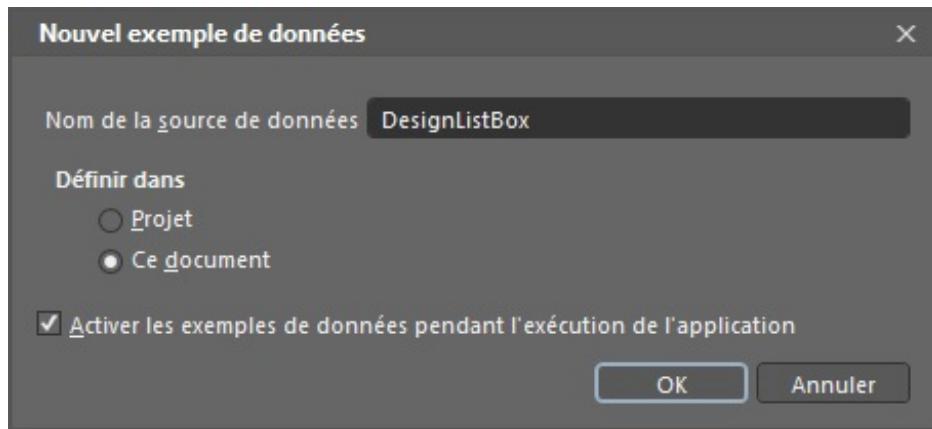


données dans Blend

Création des exemples de

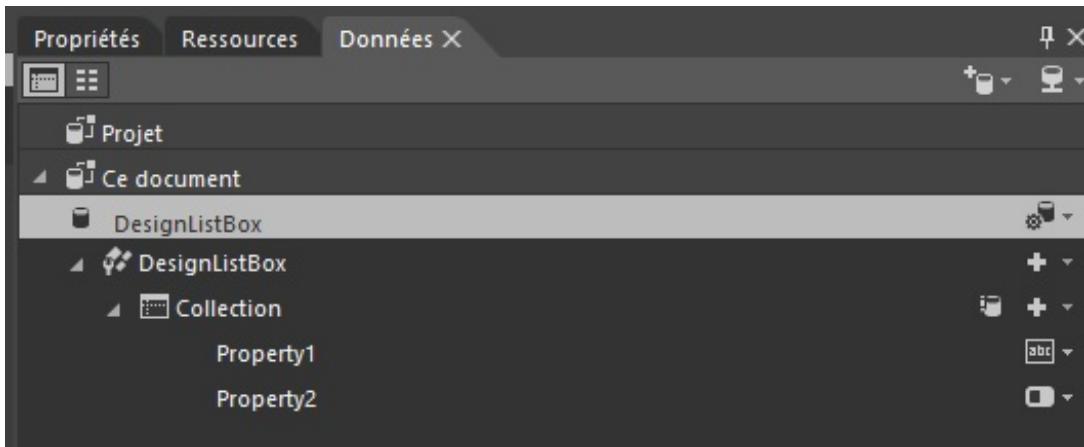
Cliquez sur nouvel exemple de données dans le menu proposé.

Indiquez un nom pour la source de données et choisissez de la définir dans ce document uniquement (voir la figure suivante).



Création de la source de données

Nous obtenons notre source de données, comme vous pouvez le voir à la figure suivante.

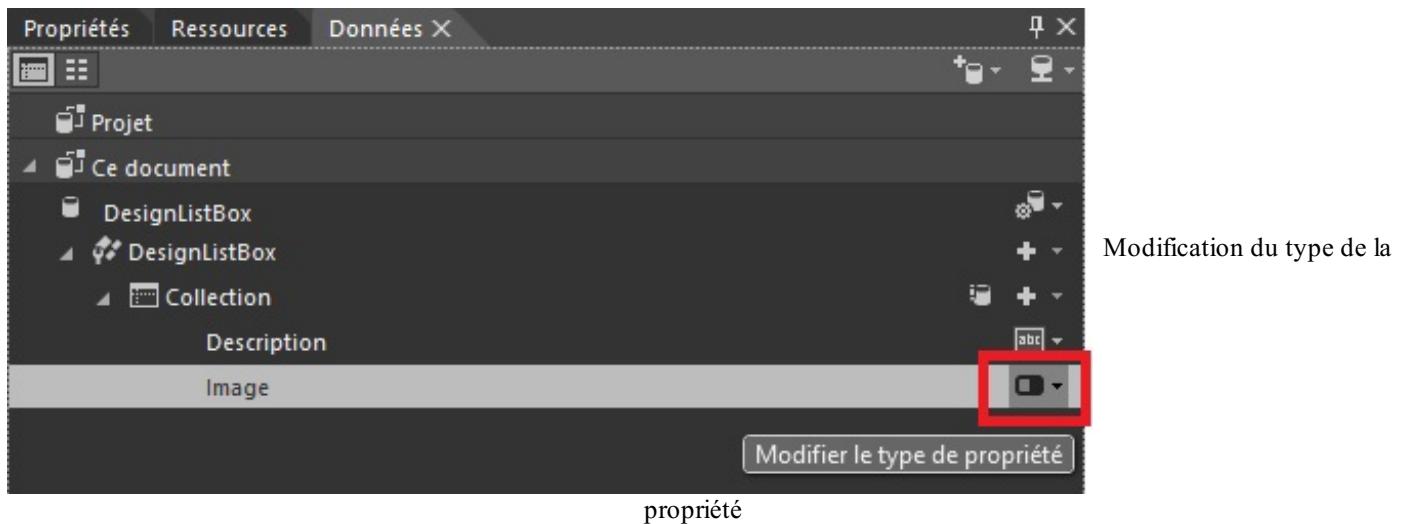


visibles dans l'écran de données

La source de données est

Elle est composée d'une collection d'objets contenant 2 propriétés. `Property1` est de type chaîne de caractères et `Property2` est de type booléen (on peut le voir en cliquant sur les petits boutons à droite).

Renommez la première en `Description` et la seconde en `Image`, puis cliquez sur l'icône à droite pour changer le type de la propriété de la seconde, comme sur la figure suivante.



propriété

Modification du type de la

Et choisissez le type Image (voir la figure suivante).



désormais Image

Le type de la données est

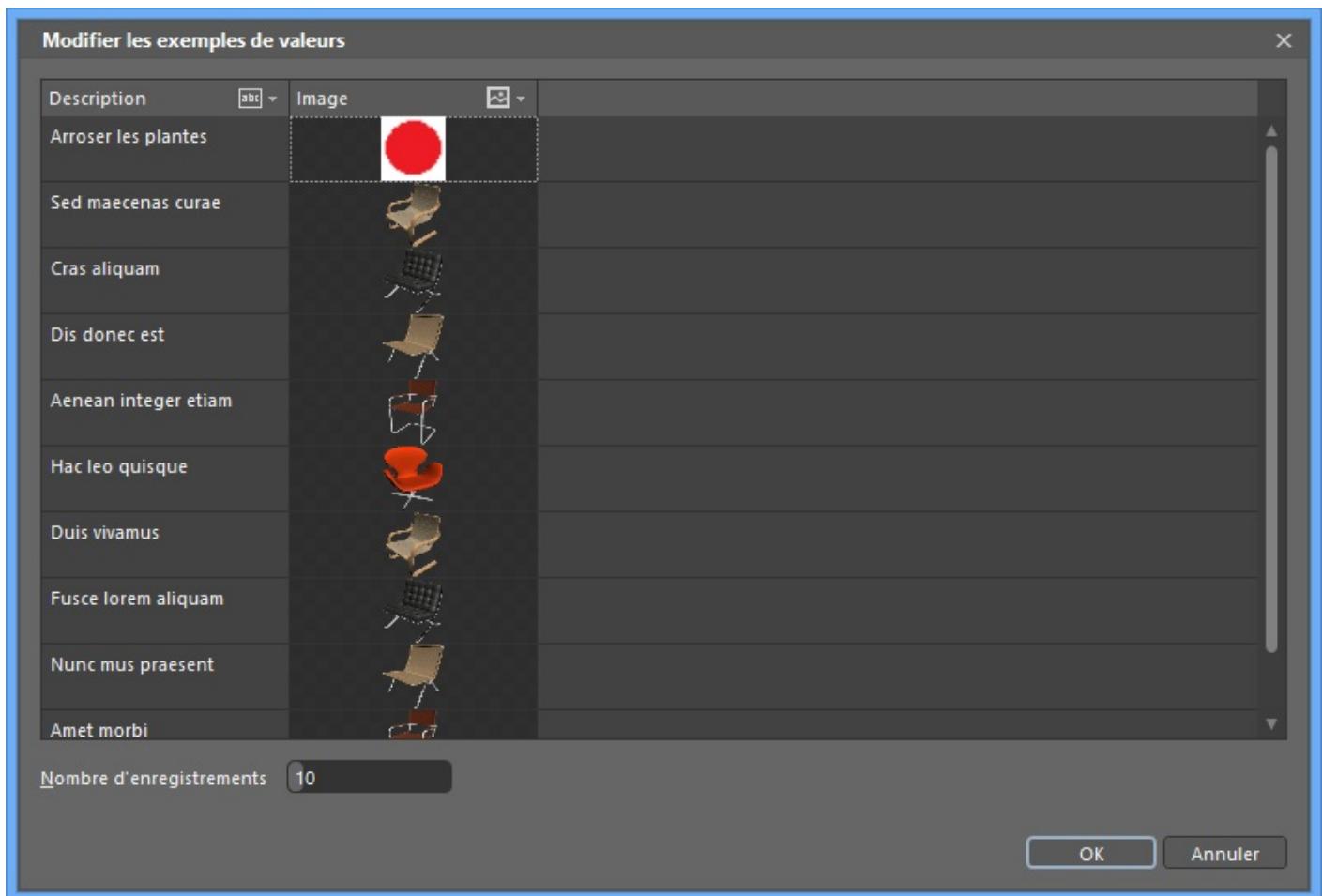
Nous avons créé ici un jeu de données, stockées sous la forme d'un fichier XAML.
Il est possible de modifier les données en cliquant sur le bouton modifier les exemples de valeurs (voir la figure suivante).



de valeurs

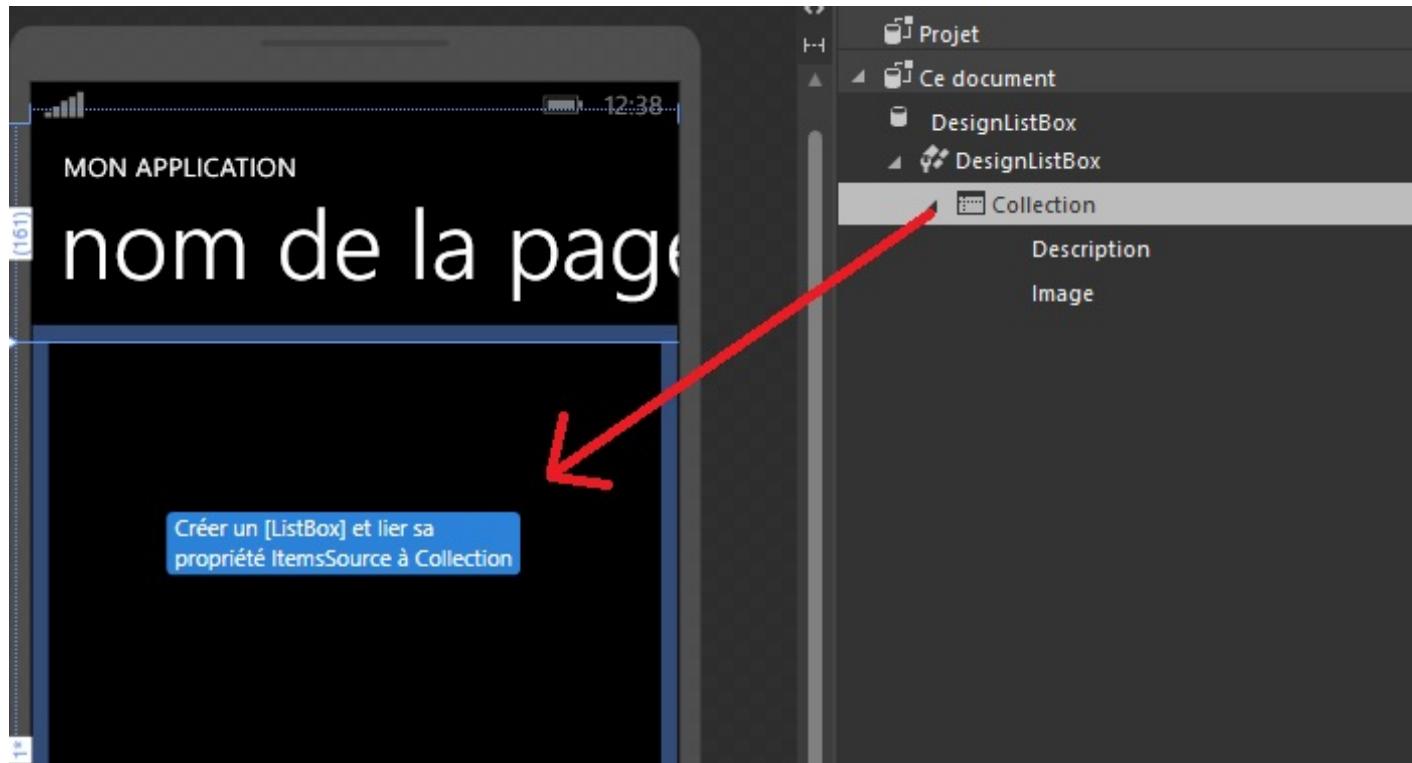
Modification des exemples

Nous obtenons une fenêtre de ce style (voir la figure suivante).



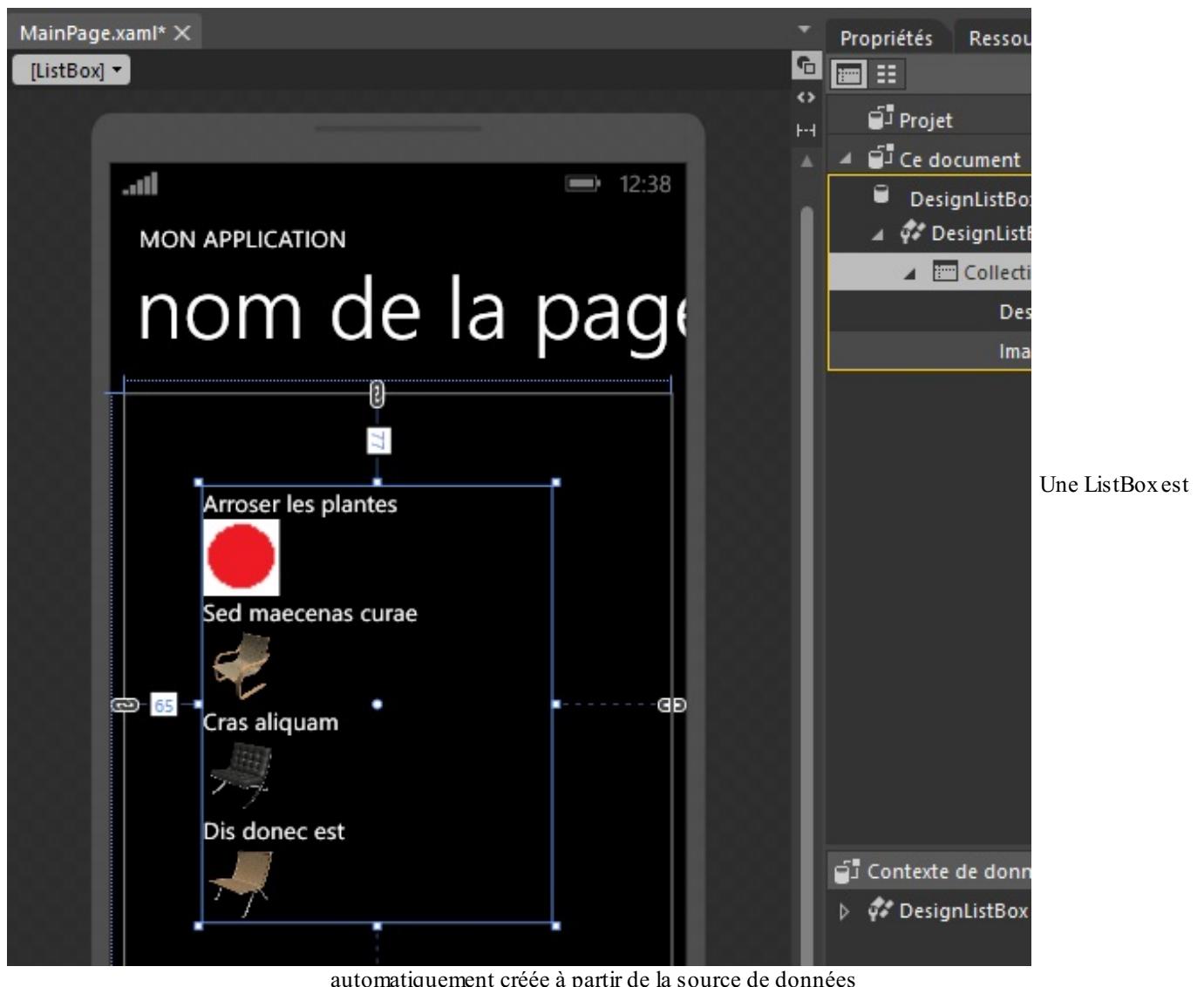
Fenêtre de modification des exemples de valeurs

On peut y mettre nos propres valeurs. Ici, j'ai changé le premier élément pour lui indiquer la valeur que je voulais et l'image que je souhaitais, mais il est également possible d'indiquer un répertoire pour sélectionner les images.
Maintenant, il est temps d'utiliser nos données. Sélectionnez la collection et faites-la glisser dans le fenêtre de design, comme indiqué sur la figure suivante.



La source de données est glissée dans le designer

Il vous crée automatiquement une `ListBox` avec les nouvelles données de la source de données (voir la figure suivante).



automatiquement créée à partir de la source de données

Utiliser l'ObservableCollection

Avant de terminer sur la liaison de données, reprenons un exemple simplifié de notre liste de tâches. Avec le XAML suivant :

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="100"/>
    </Grid.RowDefinitions>
    <ListBox ItemsSource="{Binding ListeDesTaches}" >
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="{Binding Priorite}" Margin="20 0 0 0" />
                    <TextBlock Text="{Binding Description}" Margin="20 0 0 0" />
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
    <Button Content="Ajouter un élément" Tap="Button_Tap" Grid.Row="1" />
</Grid>

```

Où nous affichons notre liste des tâches avec la valeur de la priorité et la description dans des `TextBlock`. Nous disposons également d'un bouton en bas pour rajouter un nouvel élément.

Le code behind sera :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    private List<ElementAFaire> listeDesTaches;
    public List<ElementAFaire> ListeDesTaches
    {
        get { return listeDesTaches; }
        set { NotifyPropertyChanged(ref listeDesTaches, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    public MainPage()
    {
        InitializeComponent();

        List<ElementAFaire> chosesAFaire = new List<ElementAFaire>
        {
            new ElementAFaire { Priorite = 1, Description = "Arroser les plantes" },
            new ElementAFaire { Priorite = 2, Description = "Tondre le gazon" },
            new ElementAFaire { Priorite = 1, Description = "Planter les tomates" },
            new ElementAFaire { Priorite = 3, Description = "Laver la voiture" },
        };

        ListeDesTaches = chosesAFaire.OrderBy(e =>
e.Priorite).ToList();

        DataContext = this;
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        ListeDesTaches.Add(new ElementAFaire { Priorite = 1,
Description = "Faire marcher ce binding !" });
    }
}

public class ElementAFaire
```

```
{  
    public int Priorite { get; set; }  
    public string Description { get; set; }  
}
```

La différence avec la version précédente est que nous utilisons une `List<ElementAFaire>` comme type d'objet lié à la source de données de la `ListBox`. Nous pouvons également voir que dans l'événement de clic sur le bouton, nous ajoutons un nouvel élément à la liste des tâches, en utilisant la méthode `Add()` de la classe `List<>`. Si nous exécutons notre application et que nous cliquons sur le bouton, un élément est rajouté à la liste, sauf que rien n'est visible dans notre `ListBox`. Problème !

Ah oui, c'est vrai, nous n'avons pas informé la page que la `ListBox` devait se mettre à jour. Pour ce faire, il faudrait modifier l'événement de clic sur le bouton de cette façon :

Code : C#

```
List<ElementAFaire> nouvelleListe = new  
List<ElementAFaire>(ListeDesTaches);  
nouvelleListe.Add(new ElementAFaire { Priorite = 1, Description =  
"Faire marcher ce binding ! "});  
ListeDesTaches = nouvelleListe;
```

C'est-à-dire créer une copie de la liste, ajouter un nouvel élément et affecter cette nouvelle liste à la propriété `ListeDesTaches`. Ce qui devient peu naturel ...

C'est parce que la liste n'implémente pas `INotifyCollectionChanged` qui permet d'envoyer des événements sur l'ajout ou la suppression d'un élément dans une liste. Heureusement il existe une autre classe dans le framework .NET qui implémente déjà ce comportement, il s'agit de la classe `ObservableCollection`. Il s'agit d'une liste évoluée prenant en charge les mécanismes de notification automatiquement lorsque nous faisons un ajout à la collection, lorsque nous supprimons un élément, etc. Changeons donc le type de notre propriété de liaison :

Code : C#

```
private ObservableCollection<ElementAFaire> listeDesTaches;  
public ObservableCollection<ElementAFaire> ListeDesTaches  
{  
    get { return listeDesTaches; }  
    set { NotifyPropertyChanged(ref listeDesTaches, value); }  
}
```

 Remarque : vous devez importer l'espace de nom `System.Collections.ObjectModel`.

Dans le constructeur, il faudra changer l'initialisation de la liste :

Code : C#

```
ListeDesTaches = new  
ObservableCollection<ElementAFaire>(choosesAFaire.OrderBy(e =>  
e.Priorite));
```

Et désormais, lors du clic, il suffira de faire :

Code : C#

```
ListeDesTaches.Add(new ElementAFaire { Priorite = 1, Description =
    "Faire marcher ce binding !" });
```

Ce qui est quand même beaucoup plus simple.

Plutôt pratique cette `ObservableCollection`. Elle nous simplifie énormément la tâche lorsqu'il s'agit de faire des opérations sur une collection et qu'un contrôle doit être notifié de ce changement. C'est le complément idéal pour toute `ListBox` qui se respecte. De plus, avec l'`ObservableCollection`, notre `ListBox` ne s'est pas complètement rafraîchie, elle a simplement ajouté un élément. Avec la méthode précédente, c'est toute la liste qui se met à jour d'un coup, ce qui pénalise un peu les performances.

Alors pourquoi je ne l'ai pas utilisé avant ? Parce que je considère qu'il est important de comprendre ce que l'on a fait. Le binding fonctionne avec tout ce qui est énumérable, comme la `List<>` ou n'importe quoi implémentant `IEnumerable<>`. C'est ce que j'ai illustré au début du chapitre. Lorsqu'on a besoin uniquement de remplir un contrôle et qu'il ne va pas se mettre à jour, ou pas directement, utiliser une liste ou un `IEnumerable` est le plus simple et le plus performant. Cela permet également de ne pas avoir besoin d'instancier une `ObservableCollection`.

Si bien sûr, il y a beaucoup d'opération sur la liste, suppression, mise à jour, ajouter, ... il sera beaucoup plus pertinent d'utiliser une `ObservableCollection`. Mais il faut faire attention à l'utiliser correctement...

Imaginons par exemple que je veuille mettre à jour toutes mes priorités... Comme je suis en avance, je rajoute un bouton me permettant d'augmenter la priorité de 1 pour chaque élément :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="150" />
    </Grid.RowDefinitions>
    <ListBox ItemsSource="{Binding ListeDesTaches}" >
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Text="{Binding Priorite}" Margin="20
0 0 0" />
                    <TextBlock Text="{Binding Description}" Margin="20
0 0 0" />
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
    <StackPanel Grid.Row="1">
        <Button Content="Ajouter un élément" Tap="Button_Tap" />
        <Button Content="Augmenter les priorités" Tap="Button_Tap_1"
/>
    </StackPanel>
</Grid>
```

Et dans la méthode du clic, je peux faire :

Code : C#

```
private void Button_Tap_1(object sender, RoutedEventArgs e)
{
    foreach (ElementAFaire element in ListeDesTaches)
    {
        element.Priorite++;
    }
}
```

Sauf qu'après un clic sur notre bouton, on se rend compte que l'ObservableCollection est mise à jour mais pas la ListBox... Aarrggghhh ! Alors que notre ObservableCollection était censée résoudre tous nos problèmes de notification ...

C'est là où il est important d'avoir compris ce qu'on faisait réellement ...

Ici, ce n'est pas la collection que l'on a modifiée (pas d'ajout, pas de suppression, ...), mais bien l'objet contenu dans la collection. Il doit donc implémenter `INotifyPropertyChanged`, ce qui donne :

Code : C#

```
public class ElementAFaire : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private int priorite;
    public int Priorite
    {
        get { return priorite; }
        set { NotifyPropertyChanged(ref priorite, value); }
    }
    public string Description { get; set; }
}
```

Il faut également notifier du changement lors de l'accès à la propriété `Priorite`. En toute logique, il faudrait également le faire sur la propriété `Description`, mais vu que nous ne nous en servons pas ici, je vous fais grâce de ce changement (voir la figure suivante).



La collection est mise à jour grâce à l'implémentation de l'interface

INotifyPropertyChanged

L'ObservableCollection est donc une classe puissante mais qui peut nous jouer quelques tours si son fonctionnement n'est pas bien maîtrisé.

Les converters

Parfois, lorsque nous faisons des liaisons de données, la source de données ne correspond pas exactement à ce que nous souhaitons afficher ; la preuve juste au-dessus. Nous voulions afficher une image dans une ListBox mais nous n'avions à notre disposition qu'un chiffre représentant une priorité. Pour y remédier, nous avions construit un objet spécial avec directement les bonnes valeurs via la classe ElementAFaireBinding :

Code : C#

```
List<ElementAFaire> chosesAFaire = new List<ElementAFaire>
{
    new ElementAFaire { Priorite = 1, Description = "Arroser les
    plantes" },
    new ElementAFaire { Priorite = 2, Description = "Tondre le
    gazon" },
    new ElementAFaire { Priorite = 1, Description = "Planter les
    tomates" },
    new ElementAFaire { Priorite = 3, Description = "Laver la
    voiture" },
};

listeDesTaches.ItemsSource = chosesAFaire.OrderBy(e =>
e.Priorite).Select(e => new ElementAFaireBinding { Description =
e.Description, Image = ObtientImage(e.Priorite) });
```

C'est une bonne façon de faire mais il existe une autre solution qui consiste à appliquer un convertisseur lors de la liaison de

données. Appelés « converters » en anglais, ils font en sorte de transformer une donnée en une autre, adaptée à ce que l'on souhaite lier.

Un exemple sera plus clair qu'un long discours. Prenons par exemple le cas où l'on souhaite masquer une zone de l'écran en fonction d'une valeur. L'affichage / masquage d'un contrôle, c'est le rôle de la propriété `Visibility` qui a la valeur `Visible` par défaut ; pour être invisible, il faut que la valeur soit à `Collapsed` :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock Text="Je suis visible" Visibility="Collapsed" />
        <Button Content="Masquer/Afficher" Tap="Button_Tap" />
    </StackPanel>
</Grid>
```

Ces deux valeurs font partie de l'énumération `Visibility`.

Sauf que pour nous, il est beaucoup plus logique de travailler avec un booléen : s'il est vrai, le contrôle est visible, sinon il est invisible.

C'est là que va servir le converter, il va permettre de transformer true en `Visible` et false en `Collapsed`. Pour créer un tel converter, nous allons ajouter une nouvelle classe : `VisibilityConverter`. Cette classe doit implémenter l'interface `IValueConverter` qui force à implémenter une méthode de conversion de bool vers `Visibility` et inversement une méthode qui transforme `Visibility` en bool. Voici donc une telle classe :

Code : C#

```
public class VisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (bool)value ? Visibility.Visible :
Visibility.Collapsed;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        Visibility visibility = (Visibility)value;
        return (visibility == Visibility.Visible);
    }
}
```



`IValueConverter` fait partie de l'espace de nom `System.Windows.Data`, `CultureInfo` fait partie de l'espace de nom `System.Globalization` et `Visibility` de `System.Windows`.

Le code n'est pas très compliqué. La seule chose peut-être nouvelle pour certains est l'utilisation de l'opérateur ternaire « ? ». L'écriture suivante :

Code : C#

```
Visibility v = visibility ? Visibility.Visible :
Visibility.Collapsed;
```

permet de remplacer :

Code : C#

```
Visibility v;
if (visibility)
    v = Visibility.Visible;
else
    v = Visibility.Collapsed;
```

Il évalue le booléen (ou la condition) à gauche du point d'interrogation. Si le résultat est vrai, alors il prend le premier opérande, sinon il prend le second, situé après les deux points.

Bref, une fois ceci fait, il va falloir déclarer le convertisseur dans notre page XAML. Cela se fait dans les ressources et ici, je le mets dans les ressources de la page :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
    <converter:VisibilityConverter x:Key="VisibilityConverter" />
</phone:PhoneApplicationPage.Resources>
```

Ce code doit être présent au même niveau que le conteneur de base de la page, c'est-à-dire à l'intérieur de la balise `<phone:PhoneApplicationPage>`.

Attention, la classe convertisseur n'est pas connue de la page, il faut ajouter un espace de nom dans les propriétés de la page : `xmlns:converter="clr-namespace:DemoPartie2"`

Il ne reste plus qu'à créer une propriété pour le binding dans notre classe :

Code : C#

```
private bool textBlockVisible;
public bool TextBlockVisible
{
    get { return textBlockVisible; }
    set { NotifyPropertyChanged(ref textBlockVisible, value); }
}
```

Et à modifier sa valeur dans l'événement de clic sur le bouton :

Code : C#

```
private void Button_Tap(object sender, RoutedEventArgs e)
{
    TextBlockVisible = !TextBlockVisible;
}
```

Maintenant, nous devons indiquer le binding dans le XAML et que nous souhaitons utiliser un convertisseur, cela se fait avec :

Code : XML

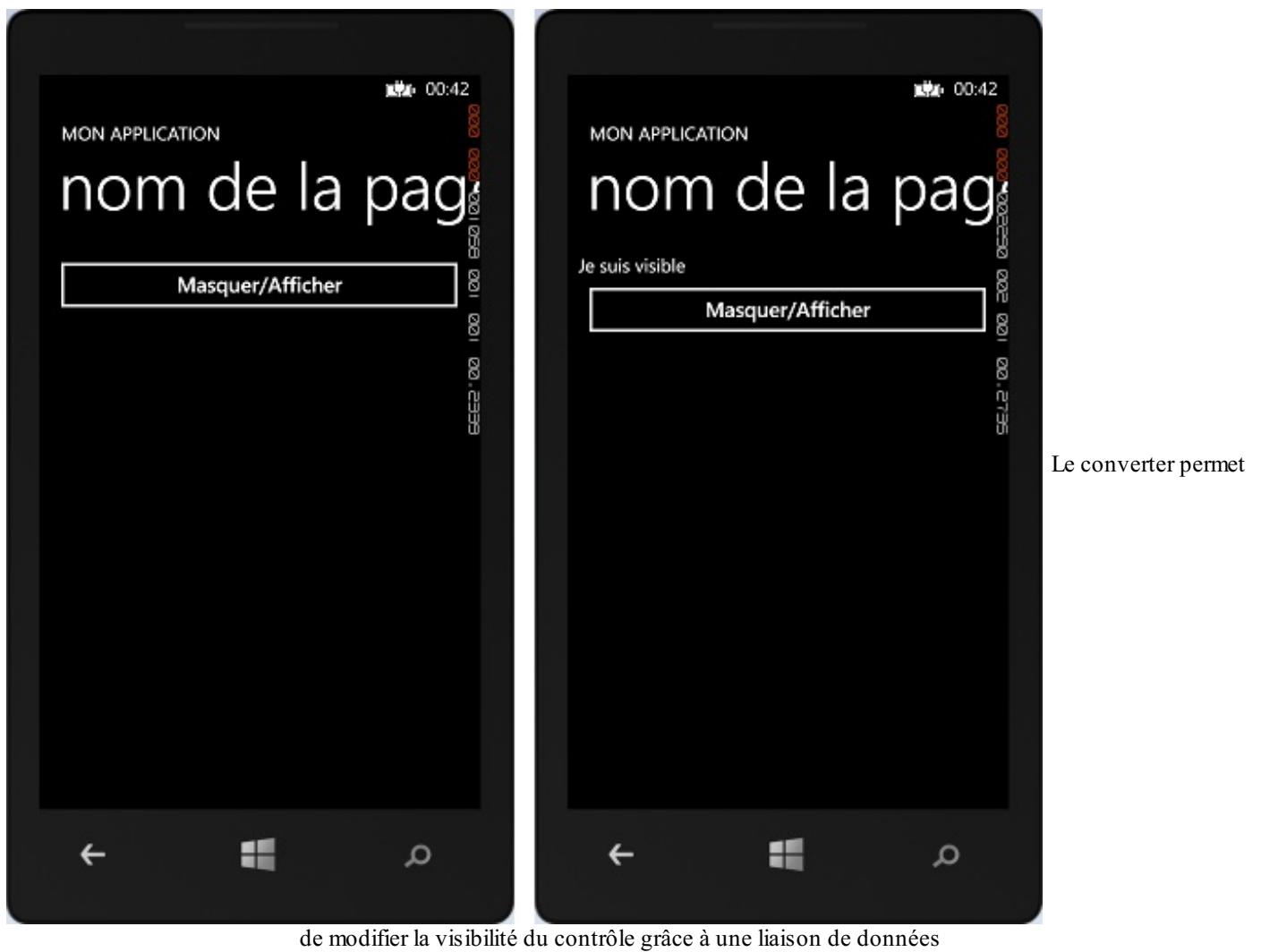
```
<TextBlock Text="Je suis visible" Visibility="{Binding
TextBlockVisible, Converter={StaticResource VisibilityConverter}}" />
```

Nous lui indiquons ici que le convertisseur est accessible en ressources par son nom : `VisibilityConverter`. N'oublions pas de donner la valeur initiale du booléen, par exemple dans le constructeur, ainsi que d'alimenter le `DataContext` :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    TextBlockVisible = false;
    DataContext = this;
}
```

Et voilà, le fait de changer la valeur du booléen influe bien sur la visibilité du contrôle, comme vous pouvez le constater à la figure suivante.



Le convertisseur permet

de modifier la visibilité du contrôle grâce à une liaison de données

Il est assez courant d'écrire des convertisseurs pour se simplifier la tâche, par contre cela rajoute un temps de traitement qui en fonction des cas peut être important. Si c'est possible, préférez les cas où la donnée est correctement préparée dès le début, pensez que les smartphones n'ont pas autant de puissance que votre machine de développement...

- La liaison de données, binding en anglais, est un élément fondamental des applications XAML et permet d'associer une source de données à un contrôle.
- On utilise l'extension de balisage `{Binding}` pour déclarer une liaison de données depuis une propriété d'un contrôle dans le XAML.
- Il est possible d'améliorer le mode design de ses pages grâce au binding.
- L'`ObservableCollection` est une liste évoluée qui gère automatiquement les notifications en cas de changement dans la

liste.

- Les converters sont un mécanisme qui permet de transformer une donnée en une autre au moment du binding.

MVVM

Passons maintenant à MVVM...

Si vous êtes débutants en XAML, je ne vous cache pas que ce chapitre risque d'être difficile à appréhender. Il s'agit de concepts avancés qu'il n'est pas nécessaire de maîtriser immédiatement. Au contraire, d'une manière générale, il faut déjà pas mal de pratique avant de pouvoir utiliser les concepts présentés dans ce chapitre.

Mais n'hésitez pas à le lire quand même et à y revenir plus tard, cela vous sera toujours utile.

Très à la mode, MVVM est un patron de conception (design pattern en anglais) qui s'est construit au fur et à mesure que les développeurs créaient des applications utilisant le XAML.

MVVM signifie Model-View-ViewModel, nous allons détailler son principe et son fonctionnement dans ce chapitre.

Principe du patron de conception

La première chose à savoir est qu'est-ce qu'un patron de conception ? Très connu sous son appellation anglaise, « design pattern », un patron de conception constitue une solution éprouvée et reconnue comme une bonne pratique à un problème récurrent dans la conception d'applications informatiques. En général, il décrit une modélisation de classes utilisées pour résoudre un problème précis. Il existe beaucoup de patrons de conceptions, comme le populaire MVC (Modèle-Vue-Contrôleur), très utilisé dans la réalisation de sites web.

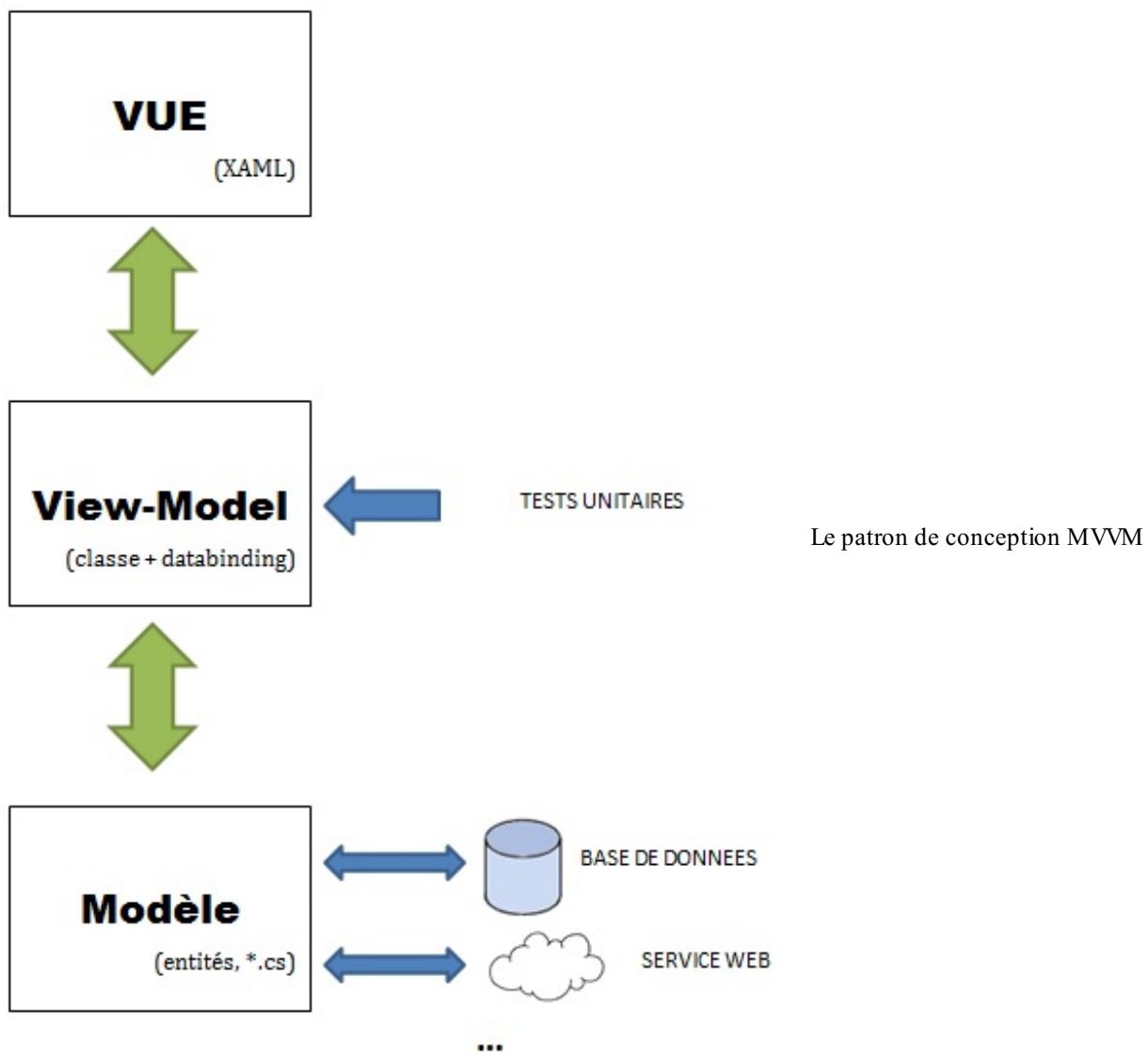
Ici, le patron de conception MVVM est particulièrement adapté à la réalisation d'applications utilisant le XAML, comme les applications pour Windows Phone, mais également les applications Silverlight, Windows 8 ou WPF. Il permet d'architecturer efficacement une application afin d'en faciliter la maintenabilité et la testabilité.

Certains auront tendance à considérer MVVM comme un ensemble de bonnes pratiques plutôt qu'un vrai patron de conception. Ce qui est important, c'est qu'en le comprenant vous allez améliorer votre productivité dans la réalisation d'applications conséquentes. Voyons à présent de quoi il s'agit.

MVVM signifie *Model-View-ViewModel*.

- *Model*, en français « le modèle », correspond aux données. Il s'agit en général de plusieurs classes qui permettent d'accéder aux données, comme une classe Client, une classe Commande, etc. Peu importe la façon dont on remplit ces données (base de données, service web,...), c'est ce modèle qui est manipulé pour accéder aux données.
- *View*, en français « la vue », correspond à tout ce qui sera affiché, comme la page, les boutons, etc. En pratique, il s'agit du fichier .xaml.
- *View Model*, que l'on peut traduire en « modèle de vue », c'est la colle entre le modèle et la vue. Il s'agit d'une classe qui fournit une abstraction de la vue. Ce modèle de vue, que j'appellerai désormais view-model, s'appuie sur la puissance du binding pour mettre à disposition de la vue les données du modèle. Il s'occupe également de gérer les commandes que nous verrons un peu plus loin.

Voici à la figure suivante un schéma représentant ce patron de conception.



Le but de MVVM est de faire en sorte que la vue n'effectue aucun traitement, elle ne doit faire qu'afficher les données présentées par le view-model. C'est le view-model qui a en charge de faire les traitements et d'accéder au modèle.

Et si on se tentait une petite métaphore pour essayer de comprendre un peu mieux ?
Essayons de représenter MVVM à travers un jeu, disons une machine à sous d'un casino.

- Mon modèle correspondra aux différentes valeurs internes des images de la machine à sous, dont le fameux 7 qui fait gagner le gros lot.
- Ma vue correspondra à la carcasse de la boîte à sous et surtout aux images qui s'affichent. Il s'agira de tout ce que l'on voit.
- Mon view-model correspondra aux engrenages qui relient les images à la machine à sous et qui transforment une valeur interne en image affichable sur la machine à sous.

Sans ces engrenages, mes images ne peuvent pas s'accrocher au cadre de la machine à sous et tout se casse la figure, on ne voit rien sur la vue.

Lorsque ces engrenages sont présents, on peut voir les données liées à la vue (grâce au binding). Et je peux agir sur mon modèle par l'intermédiaire de commandes, en l'occurrence le levier de la machine à sous.

Je tire sur le levier, une commande du view-model est activée, les images tournent, le modèle se met à jour (les valeurs internes ont changées) et la vue est mise à jour automatiquement. Je peux voir que les trois sont alignés. JACKPOT.

Plus compréhensible ? N'hésitez pas à me proposer d'autres métaphores en commentaires pour expliquer MVVM.

Première mise en place de MVVM

Nous avons commencé à pratiquer un peu MVVM au chapitre précédent, en fournissant un contexte dans une classe séparée. Ce contexte est le view-model, il prépare les données afin qu'elles soient affichables par la vue. Si on veut être un peu plus précis et utiliser un langage plus proche des patrons de conception, on pourrait dire que le view-model « adapte » le modèle pour la

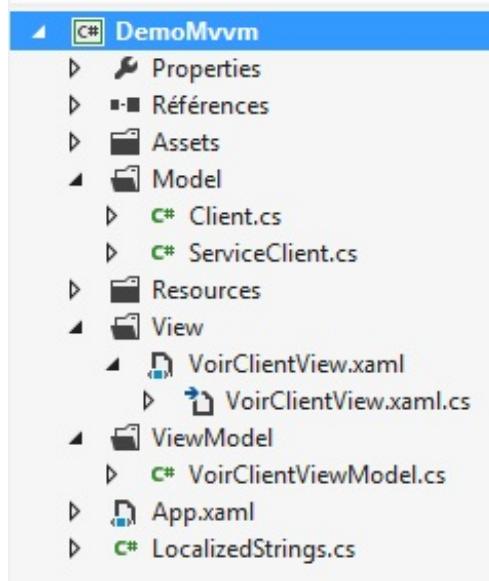
vue.

Commençons par créer un nouveau projet pour mettre en place une version simplifiée de MVVM. Comme l'idée est de séparer les responsabilités, nous allons en profiter pour créer des répertoires pour notre modèle, nos vues et nos view-models.

Créons donc les répertoires et les fichiers suivants :

- Model
 - Client.cs
 - ServiceClient.cs
- View
 - VoirClientView.xaml
- ViewModel
 - VoirClientViewModel.cs

Et profitons-en pour supprimer le fichier MainPage.xaml, de manière à avoir la même architecture que sur la figure suivante.



Architecture de la solution MVVM

Par convention, vous aurez compris que le modèle se place dans le répertoire `Model`, que les vues se placent dans le répertoire `View` et que les view-models se placent dans le répertoire `ViewModel`. De même, on suffixera les vues par « `View` » et les view-models par « `ViewModel` ».

En l'état, notre application ne pourra pas démarrer ainsi, car notre application va essayer de démarrer en naviguant sur le fichier `MainPage.xaml`, que nous avons supprimé. Nous devons donc lui indiquer un nouveau point d'entrée. Cela se fait depuis le fichier `WMAppManifest.xml` qui est sous le répertoire `Properties`. Ouvrez-le et modifiez la Page de navigation, comme nous l'avons déjà fait, pour y mettre :`View/VoirClientView.xaml`.

Cela nous permet de dire que l'application doit démarrer en affichant la page `VoirClientView.xaml`.

Commençons par créer un modèle ultra simple, qui consiste en une classe client qui contient un prénom, un âge et un booléen indiquant s'il est un bon client :

Code : C#

```
public class Client
{
    public string Prenom { get; set; }
    public int Age { get; set; }
    public bool EstBonClient { get; set; }
}
```

Ainsi qu'un service qui va nous simuler le chargement d'un client :

Code : C#

```
public class ServiceClient
```

```

    {
        public Client Charger()
        {
            return new Client { Prenom = "Nico", Age = 30, EstBonClient
= true };
        }
    }
}

```

Maintenant, réalisons la vue. Nous allons simplement afficher le prénom et l'âge du client. Ceux-ci seront sur un fond vert si le client est un bon client et en rouge si c'est un mauvais client. Quelque chose comme ça :

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
        <TextBlock x:Name="PageTitle" Text="Fiche client"
Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"
Background="{Binding BonClient}">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto" />
            <RowDefinition Height="auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <TextBlock Text="Prénom : " />
        <TextBlock Grid.Column="1" Text="{Binding Prenom}" />
        <TextBlock Grid.Row="1" Text="Age : " />
        <TextBlock Grid.Column="1" Grid.Row="1" Text="{Binding Age}" />
    </>
    </Grid>
</Grid>

```

On utilise les expressions de balisage pour indiquer les valeurs grâce au binding. Sauf qu'il est difficile de se rendre compte ainsi si la vue est bien construite, car il nous manque les valeurs de design. Qu'à cela ne tienne, nous savons désormais comment créer un contexte de design. Créons un nouveau répertoire sous le répertoire `ViewModel` que nous appelons `Design` et une nouvelle classe s'appelant `DesignVoirClientViewModel.cs` qui contiendra les valeurs de design suivantes :

Code : C#

```

public class DesignVoirClientViewModel
{
    public string Prenom
    {
        get { return "Nico"; }
    }

    public int Age
    {
        get { return 30; }
    }

    public SolidColorBrush BonClient

```

```
        {
            get { return new SolidColorBrush(Color.FromArgb(100, 0, 255,
0)); }
        }
    }
```

Pour rappel, `SolidColorBrush` se trouve dans l'espace de nom `System.Windows.Media`.

Il faut ensuite lier le contexte de design au view-model de design :

Code : XML

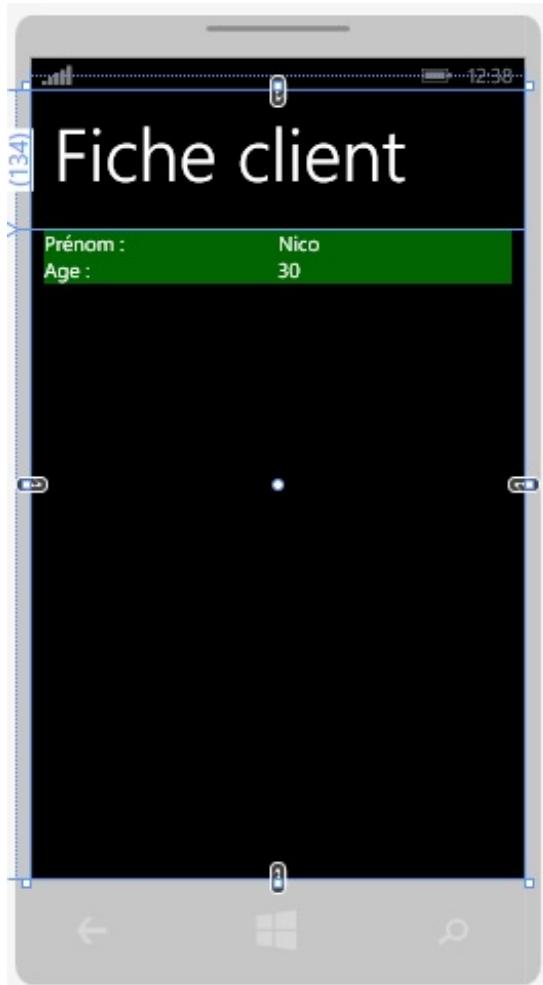
```
<d:DesignProperties.DataContext>
    <design:DesignVoirClientViewModel />
</d:DesignProperties.DataContext>
```

sans oublier d'importer l'espace de nom correspondant :

Code : XML

```
xmlns:design="clr-namespace:DemoMvvm.ViewModel.Design"
```

Ainsi, nous pourrons avoir en mode design le résultat affiché à la figure suivante.



Affichage des données en mode design grâce à MVVM

Ce qui est le résultat attendu. Chouette ! 😊

Passons enfin au view-model. Nous avons vu qu'il devait implémenter l'interface `INotifyPropertyChanged`:

Code : C#

```
public class VoirClientViewModel : INotifyPropertyChanged
{
    private string prenom;
    public string Prenom
    {
        get { return prenom; }
        set { NotifyPropertyChanged(ref prenom, value); }
    }

    private int age;
    public int Age
    {
        get { return age; }
        set { NotifyPropertyChanged(ref age, value); }
    }

    private SolidColorBrush bonClient;
    public SolidColorBrush BonClient
    {
        get { return bonClient; }
        set { NotifyPropertyChanged(ref bonClient, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }
}
```

Rien de sorcier, nous définissons également les propriétés `Prenom`, `Age` et `BonClient`. Reste à charger notre modèle depuis notre view-model et à affecter les propriétés du view-model à partir des valeurs du modèle :

Code : C#

```
public void ChargeClient()
{
    ServiceClient service = new ServiceClient();
    Client client = service.Chaiger();

    Prenom = client.Prenom;
    Age = client.Age;
    if (client.EstBonClient)
        BonClient = new SolidColorBrush(Color.FromArgb(100, 0, 255,
0));
}
```

```
        else
            BonClient = new SolidColorBrush(Color.FromArgb(100, 255, 0,
0));
    }
}
```

Il nous manque une dernière chose, hautement indispensable, qui est de lier la vue au view-model. Pour l'instant, nous avons vu que nous pouvions le faire depuis le code-behind de la vue, avec :

Code : C#

```
public partial class VoirClientView : PhoneApplicationPage
{
    public VoirClientView()
    {
        InitializeComponent();
        DataContext = new VoirClientViewModel();
    }
}
```

Il y a une autre solution qui évite de passer par le code, en utilisant le XAML :

Code : XML

```
<phone:PhoneApplicationPage.DataContext>
    <viewmodel:VoirClientViewModel />
</phone:PhoneApplicationPage.DataContext>
```

Ce qui revient au même, vu qu'on positionne la propriété `DataContext` de la page à une instance du view-model. C'est cette solution que nous allons privilégier ici.

Vous n'aurez bien sûr pas oublié d'inclure l'espace de nom qui va bien :

Code : XML

```
xmlns:viewmodel="clr-namespace:DemoMvvm.ViewModel"
```

Revenons à présent un peu sur ce que nous avons fait. Nous avons créé une vue, la page XAML, liée à l'exécution au view-model `VoirClientViewModel` et liée en design au pseudo view-model `DesignVoirClientViewModel`. Le pseudo view-model de design expose des données en dur, pour nous permettre d'avoir des données dans le designer alors que le view-model utilise et transforme le modèle pour exposer les mêmes données.

Une bonne pratique ici serait de définir une interface avec les données à exposer et que nos deux view-models l'implémentent. Créons donc un répertoire Interface dans le répertoire `ViewModel` et créons l'interface `IVoirClientViewModel` :

Code : C#

```
public interface IVoirClientViewModel
{
    string Prenom { get; set; }
    int Age { get; set; }
    SolidColorBrush BonClient { get; set; }
}
```

Nos deux view-models doivent implémenter cette interface :

Code : C#

```
public class VoirClientViewModel : INotifyPropertyChanged,  
IVoirClientViewModel  
{  
    ...  
}
```

Et :

Code : C#

```
public class DesignVoirClientViewModel : IVoirClientViewModel  
{  
    public string Prenom  
    {  
        get { return "Nico"; }  
        set { }  
    }  
  
    public int Age  
    {  
        get { return 30; }  
        set { }  
    }  
  
    public SolidColorBrush BonClient  
    {  
        get { return new SolidColorBrush(Color.FromArgb(100, 0, 255,  
0)); }  
        set { }  
    }  
}
```

Vous aurez remarqué que nous avons rajouté le mutateur set dans le view-model de design, et qu'elle n'a besoin de rien faire.

Nous pouvons encore faire une petite amélioration. Ici, elle est mineure car nous n'avons qu'un seul view-model, mais elle sera intéressante dès que nous en aurons plusieurs. En effet, chaque view-model doit implémenter l'interface `INotifyPropertyChanged` et avoir le code suivant :

Code : C#

```
public event PropertyChangedEventHandler PropertyChanged;  
  
public void NotifyPropertyChanged(string nomPropriete)  
{  
    if (PropertyChanged != null)  
        PropertyChanged(this, new  
PropertyChangedEventArgs(nomPropriete));  
}  
  
private bool NotifyPropertyChanged<T>(ref T variable, T valeur,  
[CallerMemberName] string nomPropriete = null)  
{  
    if (object.Equals(variable, valeur)) return false;  
  
    variable = valeur;  
    NotifyPropertyChanged(nomPropriete);  
    return true;  
}
```

Nous pouvons factoriser ce code dans une classe de base dont vont dériver tous les view-models. Créons pour cela un répertoire FrameworkMvvm et dedans, mettons-y la classe ViewModelBase :

Code : C#

```
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    public bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }
}
```

N'oubliez pas de passer la méthode NotifyPropertyChanged en public.

Il ne reste plus qu'à supprimer ce code du view-model et de le faire hériter de cette classe de base :

Code : C#

```
public class VoirClientViewModel : ViewModelBase,
IVoirClientViewModel
{
    ...
}
```

Les commandes

Bon, c'est très bien tout ça, mais nous n'avons fait qu'une partie du chemin... MVVM ce n'est pas que du binding avec une séparation entre la vue et les données. Il faut être capable de faire des actions. Imaginons que nous souhaitions charger les données du client suite à un appui sur un bouton. Avant MVVM, nous aurions utilisé un événement sur le clic du bouton, puis nous aurions chargé les données dans le code-behind et nous les aurions affichées sur notre page.

Avec notre découpage, le view-model n'est pas au courant d'une action sur l'interface, car c'est un fichier à part. Il n'est donc pas directement possible de réaliser une action dans le view-model lors d'un clic sur le bouton.

On peut résoudre ce problème d'une première façon très simple mais pas tout à fait parfaite. Créons tout d'abord un bouton dans notre XAML :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"
Background="{Binding BonClient}">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
```

```
<TextBlock Text="Prénom : " />
<TextBlock Grid.Column="1" Text="{Binding Prenom}" />
<TextBlock Grid.Row="1" Text="Age : " />
<TextBlock Grid.Column="1" Grid.Row="1" Text="{Binding Age}" />
<Button Grid.Row="2" Grid.ColumnSpan="2" Content="Charger
client" Tap="Button_Tap" />
</Grid>
```

Voyons à présent comment résoudre simplement ce problème. Il suffit d'utiliser l'événement Tap et de faire quelque chose comme ceci dans le code-behind de la page, dans la méthode associée à l'événement de clic :

Code : C#

```
public partial class VoirClientView : PhoneApplicationPage
{
    public VoirClientView()
    {
        InitializeComponent();
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        ((IVoirClientViewModel)DataContext).ChargeClient();
    }
}
```

On utilise la propriété DataContext pour récupérer l'instance du view-model. En effet, n'oubliez pas que nous avons lié la vue au view-model via la propriété DataContext. Cela implique de rajouter une méthode ChargeClient() dans l'interface et de la définir dans notre view-model ainsi que dans celui de design :

Code : C#

```
public interface IVoirClientViewModel
{
    string Prenom { get; set; }
    int Age { get; set; }
    SolidColorBrush BonClient { get; set; }
    void ChargeClient();
}

public class DesignVoirClientViewModel : IVoirClientViewModel
{
    public string Prenom
    {
        get { return "Nico"; }
        set { }
    }

    public int Age
    {
        get { return 30; }
        set { }
    }

    public SolidColorBrush BonClient
    {
        get { return new SolidColorBrush(Color.FromArgb(100, 0, 255,
0)); }
        set { }
    }

    public void ChargeClient()
```

```
        {  
    }  
}  
  
public class VoirClientViewModel : ViewModelBase,  
IVoirClientViewModel  
{  
    [...code supprimé pour plus de clarté...]  
  
    public VoirClientViewModel()  
    {  
    }  
  
    public void ChargeClient()  
    {  
        ServiceClient service = new ServiceClient();  
        Client client = service.Cha...  
  
        Prenom = client.Prenom;  
        Age = client.Age;  
        if (client.EstBonClient)  
            BonClient = new SolidColorBrush(Color.FromArgb(100, 0,  
255, 0));  
        else  
            BonClient = new SolidColorBrush(Color.FromArgb(100, 255,  
0, 0));  
    }  
}
```

Vous pouvez tester cette solution, cela fonctionne. Cependant, elle est imparfaite car cela crée un couplage entre la vue et le view-model, c'est-à-dire que la vue connaît l'instance du view-model car c'est effectivement elle qui l'instancie avec la déclaration que nous avons vue dans le XAML. Mais bien qu'imparfaite, n'écartez pas non plus complètement cette solution de votre esprit, elle reste bien pratique et ne rajoute pas tant de code que ça dans la vue (de plus, ce code ne fait pas de traitement, il fonctionne juste comme un relai du traitement).

Pour résoudre plus proprement ce problème, il y a une autre solution : les commandes.

Les commandes correspondent à des actions faites sur la vue. Le XAML dispose d'un mécanisme léger de gestion de commandes via l'interface **ICommand**. Par exemple, le contrôle bouton possède (par héritage) une propriété **Command** du type **ICommand** permettant d'invoquer une commande lorsque le bouton est appuyé.

Ainsi, il sera possible de remplacer :

Code : XML

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Content="Charger client"  
Tap="Button_Tap" />
```

par :

Code : XML

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Content="Charger client"  
Command="{Binding ChargerClientCommand}" />
```

Ici, nous avons enlevé l'événement Tap et la méthode associée (vous pouvez donc supprimer cette méthode dans le code behind) pour la remplacer par un binding d'une commande. Cette commande devra être définie dans le view-model :

Code : C#

```
public ICommand ChargerClientCommand { get; private set; }
```

N'oubliez pas d'inclure :

Code : C#

```
using System.Windows.Input;
```

Elle sera du type `ICommand` et évidemment en lecture seule afin que seul le view-model puisse instancier la commande. Cette commande doit ensuite être reliée à une méthode et pour faire cela, nous allons utiliser un délégué et plus particulièrement un délégué de type `Action`. Créons donc une classe qui implémente `ICommand` et qui associe la commande à une action. Nous pouvons placer cette classe dans le répertoire `FrameworkMvvm` et la nommer `RelayCommand` (ce nom n'est pas choisi au hasard, vous verrez plus loin pourquoi). L'interface `ICommand` impose de définir la méthode `CanExecute` qui permet d'indiquer si la commande peut être exécutée ou non. Nous allons renvoyer vrai dans tous les cas pour cet exemple. Elle oblige également à définir un événement `CanExecuteChanged` que nous n'allons pas utiliser et une méthode `Execute` qui appellera la méthode associée à la commande.

La classe `RelayCommand` pourra donc être :

Code : C#

```
public class RelayCommand : ICommand
{
    private readonly Action actionAExecuter;

    public RelayCommand(Action action)
    {
        actionAExecuter = action;
    }

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public event EventHandler CanExecuteChanged;

    public void Execute(object parameter)
    {
        actionAExecuter();
    }
}
```

Ce qui fait que nous allons pouvoir instancier un objet du type `RelayCommand` que nous stockerons dans la propriété `ChargerClientCommand` dans notre view-model, par exemple depuis le constructeur :

Code : C#

```
public VoirClientViewModel()
{
    ChargerClientCommand = new RelayCommand(ChargeClient);
}
```

Et voilà, la méthode `ChargeClilent()` reste la même que précédemment. Vous pouvez donc la supprimer de l'interface, du view-model de design et même si vous le souhaitez, vous pouvez la passer en privée dans le view-model. En ce qui concerne la propriété `ChargerClientCommand`, vous pouvez la déclarer dans l'interface mais ce n'est pas forcément utile car le view-model de design n'a pas besoin de connaître la commande et ne saura d'ailleurs pas quoi mettre comme méthode dedans.

Super ces commandes sauf que la propriété Command que nous avons vu est présente sur le contrôle ButtonBase, dont hérite le contrôle Button ainsi que le contrôle de case à cocher, CheckBox. Elle n'est pas présente par contre sur tous les autres contrôles. Nous allons être bloqués pour associer une commande à un autre événement, par exemple la sélection d'un élément d'une ListBox. Nous allons voir plus loin comment résoudre ce problème.

Les frameworks à la rescousse : MVVM-Light

Entre la classe ViewModelBase et la classe RelayCommand, nous sommes en train d'écrire un vrai mini framework pour nous aider à implémenter MVVM.

Alors, je vous le dis tout de suite, nous allons nous arrêter là dans l'implémentation de ce framework... car d'autres l'ont déjà fait ; et en mieux ! 😊 Il existe beaucoup de framework pour utiliser MVVM et certains sont utilisables avec Windows Phone, comme par exemple le populaire MVVM Light Toolkit, que vous pouvez [trouver ici](#).

Malgré sa dénomination, il n'est en fait pas si light que ça car il est utilisable avec WPF, Silverlight, Windows Phone et Windows 8.

Je ne vais pas faire un cours entier sur ce toolkit, car cela demanderait beaucoup trop de pages. Nous allons cependant regarder quelques points intéressants qui pourraient vous servir dans une application Windows Phone. Après, à vous de voir si vous avez besoin d'embarquer la totalité du framework ou si vous pouvez simplement vous inspirer de quelques idées...

Toujours est-il qu'il y a beaucoup de bonnes choses dans ce toolkit. Vous vous rappelez de nos deux classes ViewModelBase et RelayCommand ? Et bien, MVVM Light possède également ces deux classes... et en plus fournies ! Par exemple la classe RelayCommand. Nous en avons écrit une version ultra simplifiée. Celle de ce toolkit est complète et permet même d'y associer un paramètre. Cela peut-être utile lorsque la même commande est associée à plusieurs boutons et que chaque bouton possède un paramètre différent.

De même, ce toolkit propose une solution au problème que j'évoquais plus haut, à savoir de pouvoir relier n'importe quel événement à une commande.

Il propose également une solution pour résoudre le problème de couplage entre la vue et le view-model que nous avons rencontré précédemment. Il utilise en effet un patron de conception prévu pour ce genre de cas, le service locator, qui est un patron de conception d'inversion de dépendance.

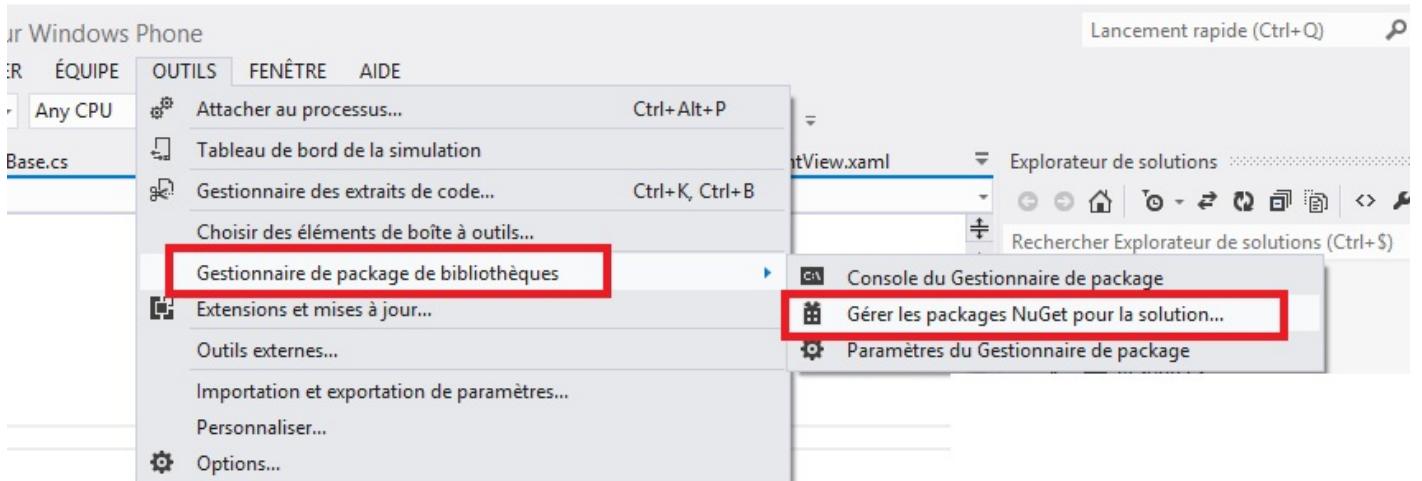
Je ne parlerai pas en détail de ce patron de conception ici, vous pouvez trouver [plus d'informations en anglais sur ce site](#). Sachez simplement que le principe général est d'avoir un gros objet contenant une liste de services et qui sait en retrouver un à partir d'une clé. Cette clé peut être de plusieurs sortes, comme une chaîne de caractère, un type (en général une interface), etc.

Pour éviter le couplage entre la vue et le view-model, ce patron de conception permettra de faire en sorte que la vue connaisse simplement une clé afin d'obtenir son view-model, sans forcément connaître l'instance et le type du view-model.

Remarquez que pour moi, ce couplage fort entre la vue et le view-model n'est pas un vrai problème, au risque de faire hurler les puristes. Le but premier théorique est de pouvoir éventuellement changer facilement de view-model pour une même vue, sans avoir à tout modifier. En pratique, il s'avère qu'il existe toujours un et un seul view-model associé à une vue, et que c'est toujours le même. C'est pourquoi je trouve que la liaison du view-model depuis le XAML proposée dans le chapitre précédent est bien souvent suffisante.

Installation du toolkit

Voyons à présent comment créer une application MVVM Light. Nous allons installer la dernière version stable du framework à l'heure où j'écris ces lignes, à savoir la version 4.1.25. Le plus simple pour télécharger les bibliothèques est de passer par NuGet, qui est un gestionnaire de package .NET open source permettant de télécharger des bibliothèques externes très facilement. Pour cela, allez dans le menu outils, puis gestionnaire de package de bibliothèques, puis Gérer les packages NuGet pour la solution, comme indiqué à la figure suivante.



Démarrer NuGet

Vous arrivez dans NuGet où vous allez pouvoir cliquer sur En ligne à gauche et commencer à rechercher le MVVM Light Toolkit (voir la figure suivante).

The screenshot shows the 'Gérer les packages NuGet' (Manage NuGet packages) window for the 'DemoMvvmLight.sln' solution. The search bar at the top right contains the text 'Mvvm light'. The left sidebar has 'En ligne' selected. The search results list several packages:

- MVVM Light**: Description: 'The MVVM Light Toolkit is a set of components helping people to get started in the Model-View-ViewModel pattern i...'. The 'Installer' (Install) button is highlighted with a blue box.
- MVVM Light libraries only**: Description: 'The MVVM Light Toolkit is a set of components helping people to get started in the Model-View-ViewModel pattern i...'. The 'Installer' (Install) button is highlighted with a blue box.
- MVVM Light [Preview]**: Description: 'The MVVM Light Toolkit is a set of components helping people to get started in the Model-View-ViewModel pattern i...'
- MVVM Light libraries only [Preview]**: Description: 'The MVVM Light Toolkit is a set of components helping people to get started in the Model-View-ViewModel pattern i...'
- knockoutjs**: Description: 'A JavaScript MVVM library to help you create rich, dynamic user interfaces with clean maintainable code'
- MVVMT4**: Description: 'T4 Templates for generating view models, and views for WPF, Silverlight, Windows Phone using T4 Toolbox and MVVM Lig...
- MadProps.MvvmLight**: Description: 'A contrib project for MVVM Light, adding support for coroutines and a screen conductor.'

The right pane displays detailed information for the selected 'Mvvm light' package, including developer details, version, last update, download count, and a full description. A note at the bottom left states: 'Chaque package vous est concédé sous licence par son propriétaire. Microsoft n'est pas responsable et n'accorde pas de licence pour les packages de sociétés tierces.'

Installation du Mvvm Light Toolkit via NuGet



Vous devez bien sûr être connectés à internet pour utiliser NuGet.

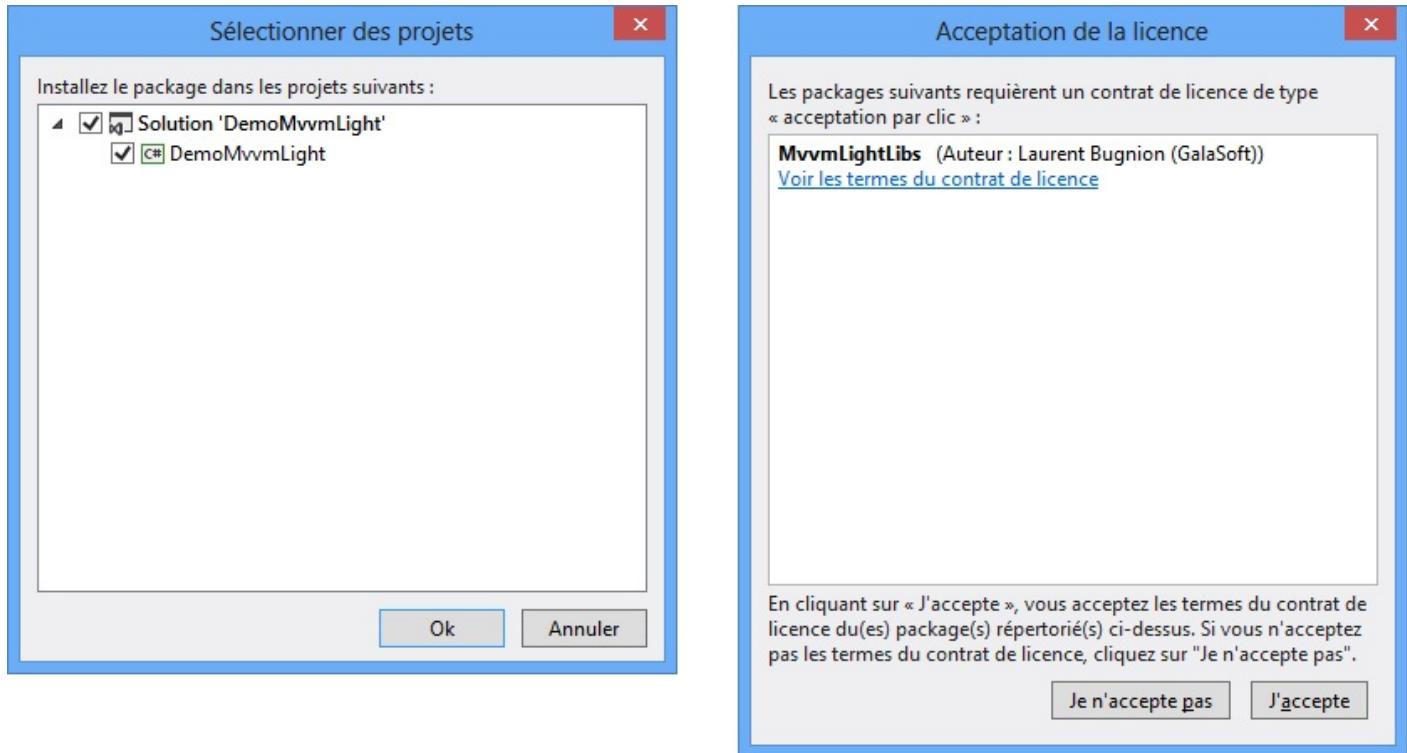
Vous pouvez choisir d'installer la version complète ou uniquement la bibliothèque. La version complète contient les binaires du toolkit, les templates et les snippets.

Les binaires sont indispensables pour utiliser le toolkit. Il s'agit des assemblies contenant le code du toolkit.

Les templates sont des modèles de projet permettant de créer facilement une application utilisant MVVM Light. Le projet ainsi créé contient déjà les bonnes références, un premier exemple de vue, de locator, etc.
Les snippets sont des extraits de code qui sont utilisés pour pouvoir créer plus rapidement des applications MVVM.

Comme je ne souhaite pas vous présenter tout MVVM Light et que je veux me concentrer sur la compréhension des éléments du patron de conception, je vais simplement installer les binaires en choisissant MVVM Light Libraries only. Cliquez sur Installer.

Après téléchargement (rapide) il me propose de l'installer dans ma solution en cours, comme vous pouvez le voir sur la figure suivante.



Installation de Mvvm Light Toolkit

Faites Ok et acceptez également la licence. Et voilà, c'est installé !



Vous pouvez aussi tout classiquement télécharger l'installeur sur le site.

Vous pouvez voir que dans votre projet, les assemblies suivantes ont automatiquement été référencées :

- GalaSoft.MvvmLight.Extras.WP8.dll
- GalaSoft.MvvmLight.WP8.dll
- Microsoft.Practices.ServiceLocation
- System.Windows.Interactivity.dll

sachant que la dernière est une assembly de Blend et que l'avant dernière permet d'utiliser le service locator.

Recréez alors un répertoire View pour y mettre votre page `VoirClientView.xaml`, un répertoire Model avec la classe Client et un répertoire ViewModel avec une classe `VoirClientViewModel`. La classe Client est la même que précédemment, le service aussi. Nous allons simplement lui rajouter une interface en plus :

Code : C#

```
public class Client
{
    public string Prenom { get; set; }
```

```
        public int Age { get; set; }
        public bool EstBonClient { get; set; }
    }

    public interface IServiceClient
    {
        Client Charger();
    }

    public class ServiceClient : IServiceClient
    {
        public Client Charger()
        {
            return new Client { Prenom = "Nico", Age = 30, EstBonClient
= true };
        }
    }
```

Puis nous allons rajouter une classe de design pour notre service. Au contraire de ce que nous avons fait dans le chapitre précédent, ce n'est pas ici le view-model qui sera en mode design, mais le model :

Code : C#

```
public class DesignServiceClient : IServiceClient
{
    public Client Charger()
    {
        return new Client { Prenom = "Nico", Age = 30, EstBonClient
= true };
    }
}
```



Ici, la classe ServiceClient et la classe DesignServiceClient sont identiques, mais dans une « vraie » application, la classe ServiceClient irait sûrement chercher les données en base ou autre...

Puis notre view-model va hériter de ViewModelBase qui se situe dans l'espace de nom GalaSoft.MvvmLight :

Code : C#

```
using GalaSoft.MvvmLight;

public class VoirClientViewModel : ViewModelBase
{}
```

Le service locator

Il est temps de créer le service locator afin de bénéficier d'un couplage faible entre notre vue et le view-model, mais également entre le view-model et le service, ce qui n'est pas indispensable mais ne fait pas de mal. Pour cela, créons une nouvelle classe ViewModelLocator, je la place à la racine du projet :

Code : C#

```
public class ViewModelLocator
{
```

```
static ViewModelLocator()
{
    ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

    if (ViewModelBase.IsInDesignModeStatic)
        SimpleIoc.Default.Register<IServiceClient,
DesignServiceClient>();
    else
        SimpleIoc.Default.Register<IServiceClient,
ServiceClient>();

    SimpleIoc.Default.Register<VoirClientViewModel>();
}

public VoirClientViewModel VoirClientVM
{
    get { return
ServiceLocator.Current.GetInstance<VoirClientViewModel>(); }
}
```



N'oubliez pas d'importer les espaces de nom `Microsoft.Practices.ServiceLocation`, `GalaSoft.MvvmLight.Ioc` et `GalaSoft.MvvmLight`.

Elle contient un constructeur statique qui s'occupe de l'inversion de dépendance en elle-même, à savoir associer le service client de design avec l'interface lorsqu'on est en mode design, et le vrai service sinon. De même, il enregistre une instance du view-model. Il possède également une propriété permettant d'accéder à l'instance du ViewModel : `VoirClientVM`.



J'ai choisi d'appeler la propriété `VoirClientVM` alors que je l'aurai naturellement appelée `VoirClientViewModel`, afin de bien voir la différence entre la propriété et la classe du view-model.

Ce locator devra être déclaré dans les ressources de l'application, ainsi il sera accessible depuis n'importe qu'elle vue. Modifiez donc le fichier `App.xaml` pour avoir :

Code : XML

```
<Application.Resources>
<... />

<vm:ViewModelLocator x:Key="Locator" d:IsDataSource="true" />
</Application.Resources>
```

Sans oublier de déclarer l'espace de nom :

Code : XML

```
xmlns:vm="clr-namespace:DemoMvvmLight"
```

et d'inclure les déclarations suivantes, qu'il n'est pas nécessaire de comprendre :

Code : XML

```
xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
mc:Ignorable="d"
```

Ceci correspond à l'implémentation du locator fourni dans les exemples du toolkit. Il existe plusieurs implémentations possibles de ce locator, plus ou moins parfaites.

Lier la vue au view-model

Maintenant, pour lier le modèle à la vue il suffira de modifier la propriété `Datacontext` de votre page pour avoir :

Code : XML

```
<phone:PhoneApplicationPage  
    x:Class="DemoMvvmLight.View.VoirClientView"  
    ...  
    DataContext="{Binding VoirClientVM, Source={StaticResource  
    Locator}}">
```

La liaison s'effectue donc sur la propriété publique `VoirClientVM` du locator, trouvé en ressource. Modifions notre vue pour afficher notre client :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">  
    <Grid.RowDefinitions>  
        <RowDefinition Height="Auto"/>  
        <RowDefinition Height="auto"/>  
    </Grid.RowDefinitions>  
  
    <StackPanel x:Name="TitlePanel" Grid.Row="0"  
    Margin="12,17,0,28">  
        <TextBlock x:Name="PageTitle" Text="Fiche client"  
    Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>  
    </StackPanel>  
  
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"  
    Background="{Binding BonClient}">  
        <Grid.RowDefinitions>  
            <RowDefinition Height="auto" />  
            <RowDefinition Height="auto" />  
        </Grid.RowDefinitions>  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition Width="*" />  
            <ColumnDefinition Width="*" />  
        </Grid.ColumnDefinitions>  
        <TextBlock Text="Prénom : " />  
        <TextBlock Grid.Column="1" Text="{Binding Prenom}" />  
        <TextBlock Grid.Row="1" Text="Age : " />  
        <TextBlock Grid.Column="1" Grid.Row="1" Text="{Binding Age}" />  
    </Grid>  
    </Grid>  
</Grid>
```

Maintenant, passons au view-model. Il ressemble beaucoup à notre précédent view-model :

Code : C#

```
public class VoirClientViewModel : ViewModelBase  
{
```

```
    private readonly IServiceProvider serviceClient;

    private string prenom;
    public string Prenom
    {
        get { return prenom; }
        set { NotifyPropertyChanged(ref prenom, value); }
    }

    private int age;
    public int Age
    {
        get { return age; }
        set { NotifyPropertyChanged(ref age, value); }
    }

    private SolidColorBrush bonClient;
    public SolidColorBrush BonClient
    {
        get { return bonClient; }
        set { NotifyPropertyChanged(ref bonClient, value); }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        RaisePropertyChanged(nomPropriete);
        return true;
    }

    public VoirClientViewModel(IServiceProvider service)
    {
        serviceClient = service;

        Client client = serviceClient.GetClient();
        Prenom = client.Prenom;
        Age = client.Age;
        if (client.EstBonClient)
            BonClient = new SolidColorBrush(Color.FromArgb(100, 0,
255, 0));
        else
            BonClient = new SolidColorBrush(Color.FromArgb(100, 255,
0, 0));
    }
}
```

La différence vient bien sûr en premier lieu de l'héritage à `ViewModelBase` mais surtout de l'application de l'inversion de dépendance dans le constructeur du view-model. Ainsi, notre view-model récupère automatiquement une instance de notre service.

Notons aussi la différence dans la méthode `NotifyPropertyChanged` qui doit appeler maintenant la méthode `RaisePropertyChanged` de la classe de base, `ViewModelBase`. Cette méthode correspond à notre méthode `NotifyPropertyChanged` mais on se demande pourquoi le créateur du toolkit n'a pas écrit de surcharge de cette méthode utilisant les nouveautés du framework 4.5 et notamment l'attribut `CallerMemberName`, du coup nous sommes toujours obligés d'utiliser une variation de la méthode `NotifyPropertyChanged`.

N'oubliez pas de faire en sorte que ce soit la bonne vue qui s'affiche lors du démarrage de l'application et vous pourrez observer votre application qui fonctionne grâce au toolkit MVVM Light. Bravo !



Vous aurez également constaté qu'en mode design, nous pouvons aussi voir des données.

Jusque-là, nous ne sommes pas trop dépayrés à part dans l'utilisation du service locator.

Les commandes

Passons maintenant aux commandes sur le bouton. Je vous ai indiqué qu'il est possible de passer un paramètre à un bouton. Pour illustrer ce fonctionnement, nous allons créer une nouvelle page qui affichera une liste de clients. Modifions notre service pour avoir la méthode suivante :

Code : C#

```
public interface IServiceClient
{
    Client Charger();
    List<Client> ChargerTout();
}

public class ServiceClient : IServiceClient
{
    public Client Charger()
    {
        return new Client { Prenom = "Nico", Age = 30, EstBonClient =
= true };
    }

    public List<Client> ChargerTout()
    {
        return new List<Client>
        {
            new Client { Age = 30, EstBonClient = true, Prenom =
"Nico" },
            new Client { Age = 20, EstBonClient = false, Prenom =
"Jérémie" },
            new Client { Age = 30, EstBonClient = true, Prenom =
"Delphine" }
        };
    }
}
```

Et pareil dans la classe DesignServiceClient.

Rajoutons une nouvelle vue dans le répertoire View que nous allons appeler ListeClientsView.xaml. Et enfin, créons un nouveau view-model que nous appellerons ListeClientsViewModel et qui héritera de ViewModelBase. N'oubliez pas de déclarer le nouveau view-model dans le locator :

Code : C#

```
public class ViewModelLocator
{
    static ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

        if (ViewModelBase.IsInDesignModeStatic)
            SimpleIoc.Default.Register<IServiceClient,
DesignServiceClient>();
        else
            SimpleIoc.Default.Register<IServiceClient,
ServiceClient>();

        SimpleIoc.Default.Register<VoirClientViewModel>();
        SimpleIoc.Default.Register<ListeClientsViewModel>();
    }

    public VoirClientViewModel VoirClientVM
```

```

    {
        get { return
    ServiceLocator.Current.GetInstance<VoirClientViewModel>(); }
    }

    public ListeClientsViewModel ListeClientsVM
    {
        get { return
    ServiceLocator.Current.GetInstance<ListeClientsViewModel>(); }
    }
}

```

Votre vue va posséder une ListBox dont le template contiendra un bouton :

Code : XML

```

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <ListBox ItemsSource="{Binding ListeClients}">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Button Content="Qui suis-je ?" Command="{Binding
    QuiSuisJeCommand}" CommandParameter="{Binding}" />
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Grid>

```



Le code XAML précédent contient volontairement une erreur que nous corrigerais un peu plus loin.

Nous voyons que notre ListBox est liée à une propriété ListeClients qui devra être présente dans notre view-model. Le bouton présent dans le template possède une commande liée à la commande QuiSuisJeCommand. Remarquons la propriété CommandParameter qui permet de positionner l'élément en cours comme paramètre de la commande.
Liez ensuite le view-model à la vue en modifiant la propriété Datacontext avec notre nouvelle propriété dans le locator :

Code : XML

```

DataContext="{Binding ListeClientsVM, Source={StaticResource
Locator}}">

```

Passons à notre view-model désormais :

Code : C#

```

public class ListeClientsViewModel : ViewModelBase
{
    private List<Client> listeClients;
    public List<Client> ListeClients
    {
        get { return listeClients; }
        set { NotifyPropertyChanged(ref listeClients, value); }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;
    }
}

```

```
        variable = valeur;
        RaisePropertyChanged(nomPropriete);
        return true;
    }

    public ICommand QuiSuisJeCommand { get; set; }

    public ListeClientsViewModel(IServiceClient service)
    {
        ListeClients = service.Cha
```

Remarque, la classe RelayCommand nécessite l'import suivant :

Code : C#

```
using GalaSoft.MvvmLight.Command;
```

Nous avons créé la propriété ListeClients que nous avons alimentée dans le constructeur grâce au modèle. Puis nous voyons comment définir une commande qui accepte un paramètre. En l'occurrence, ici le paramètre sera du type Client car c'est le type que nous passons dans :

Code : XML

```
CommandParameter="{Binding}"
```

L'extension de balisage {Binding} prend ici l'élément courant de la propriété énumérable ListeClients qui est bien de type Client.

Une fois le bouton cliqué, nous pourrons alors afficher le client sélectionné grâce à la boîte de message MessageBox. Vérifions cela en démarrant l'application. N'oubliez pas de changer le point d'entrée de l'application dans le fichier WMAppManifest.xml afin qu'il démarre sur la vue ListeClientsView.xaml.

Sauf que... cela ne marche pas. 😊 Le clic sur le bouton ne fait rien !

Si vous observez bien la fenêtre de sortie de Visual Studio, vous pouvez constater l'erreur suivante, erreur que vous aurez l'occasion de souvent voir :

Code : Autre

```
System.Windows.Data Error: BindingExpression path error: 'QuiSuisJeCommand' prope
(HashCode=23288300). BindingExpression: Path='QuiSuisJeCommand' DataItem='DemoMv
(Name='')'; target property is 'Command' (type 'System.Windows.Input.ICommand')..
```

Une erreur de binding... Argh !

On nous indique que la propriété QuiSuisJeCommand est introuvable sur l'objet Client. Ce qui est vrai en soit, notre classe Client ne possède pas cette commande ! Mais nous, ce que nous voulions c'est que la commande QuiSuisJeCommand soit celle du view-model. Comme ce que nous avions fait précédemment.

C'est rageant, on fait pareil, mais cela ne fonctionne pas ! Essayons de comprendre pourquoi.

Je vous donne un indice : Datacontext.

En effet, dans notre exemple précédent, notre commande est liée à un bouton dont le contexte est celui hérité de celui de la page, car il n'a pas été changé. Nous avions lié la propriété Datacontext de la page au view-model et tous les Datacontext des objets de la page ont hérité de ce contexte.

Ici, nous avons changé le contexte de la ListBox pour lui fournir une liste de clients. Dans chaque template des éléments de la ListBox, nous sommes sur un objet Client et plus sur le view-model. C'est pour cela qu'il ne trouve pas la propriété QuiSuisJeCommand et qu'il ne peut pas faire correctement le binding.

C'est une erreur très classique que l'on peut résoudre de plusieurs façons. La première est de faire en sorte que le contexte courant puisse connaître le view-model. En l'occurrence, on pourrait ajouter une propriété pointant sur le view-model à l'objet Client ou même mieux créer un nouvel objet dédié au binding qui contient cette propriété. Créons donc une classe ClientBinding :

Code : C#

```
public class ClientBinding
{
    public string Prenom { get; set; }
    public int Age { get; set; }
    public bool EstBonClient { get; set; }
    public ListeClientsViewModel ViewModel { get; set; }
}
```

Puis changeons le type de la propriété ListeClient pour avoir une List<ClientBinding>. Et modifions le constructeur pour avoir :

Code : C#

```
public ListeClientsViewModel(IServiceClient service)
{
    ListeClients = service.ChaireTout().Select(c => new
ClientBinding { Age = c.Age, EstBonClient = c.EstBonClient, Prenom =
c.Prenom, ViewModel = this }).ToList();

    QuiSuisJeCommand = new RelayCommand<ClientBinding>(QuiSuisJe);
}
```

Remarquez que j'ai aussi changé le type générique de RelayCommand pour avoir un ClientBinding vu que désormais, c'est un objet de ce type qui est lié au paramètre de la commande. Ce qui implique également de changer le type du paramètre de la méthode QuiSuisJe () :

Code : C#

```
private void QuiSuisJe(ClientBinding client)
{
    MessageBox.Show("Je suis " + client.Prenom);
}
```

(n'oubliez pas de rajouter le using de l'espace de nom System.Linq;).

Chaque client possède donc une propriété ViewModel contenant le view-model en cours grâce à this. Il ne reste plus qu'à modifier l'extension de balisage pour avoir :

Code : XML

```
<Button Content="Qui suis-je ?" Command="{Binding  
ViewModel.QuiSuisJeCommand}" CommandParameter="{Binding}" />
```

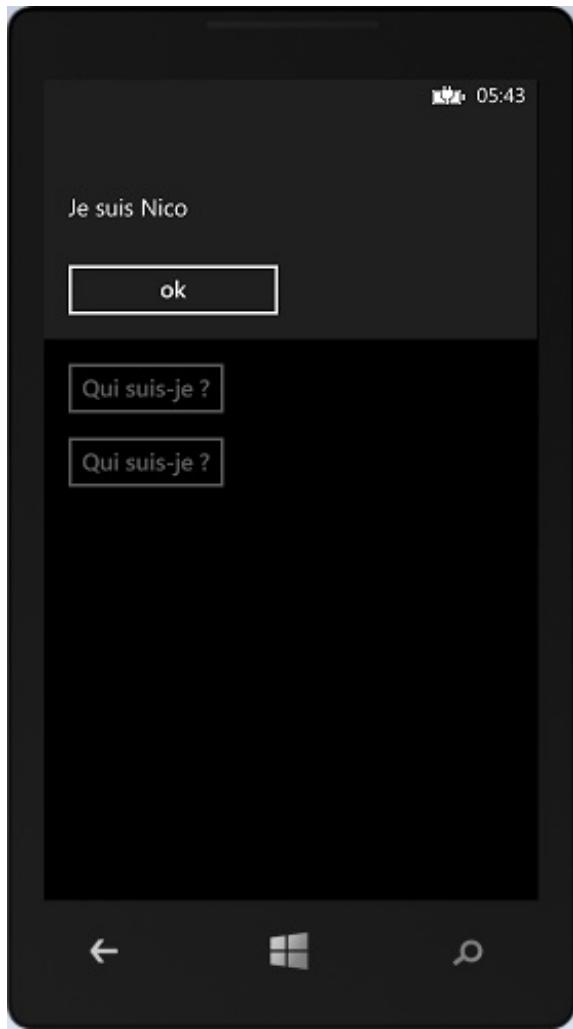
Et voilà, cela fonctionne. Mais cela implique pas mal de changement...

L'autre solution est de lier directement la propriété `Command` à la propriété `QuiSuisJeCommand` de la propriété du locator pour accéder au view-model. Plus besoin de classe intermédiaire qui contient une référence vers le view-model. Il suffit d'écrire :

Code : XML

```
<Button Content="Qui suis-je ?" Command="{Binding  
Source={StaticResource Locator},  
Path=ListeClientsVM.QuiSuisJeCommand}" CommandParameter="{Binding}" />
```

Et le résultat est le même, ainsi que vous pouvez le constater sur la figure suivante.



Affichage d'un message suite à l'activation de la commande

Pouvoir passer un paramètre à une commande est très pratique dans ce genre de situation. La classe `RelayCommand` du toolkit nous aide bien pour récupérer ce paramètre, que l'on retrouve en paramètre de la méthode `QuiSuisJe()`.

Elle sait faire encore une chose intéressante, à savoir de permettre de savoir si la commande est utilisable ou pas. Rappelez-vous, c'est ce que nous avions vu plus haut. Il s'agissait de la méthode `CanExecute` de l'interface `ICommand`. J'avais décidé arbitrairement que cette méthode renverrait toujours vrai. La classe `RelayCommand` de MVVM Light permet d'associer une condition à la possibilité d'exécuter une commande. Par exemple, on pourrait imaginer qu'on ne puisse cliquer sur le bouton que des clients qui sont des bons clients. C'est de la ségrégation, mais c'est comme ça. Les mauvais clients resteront inconnus ! Pour ce faire, on utilisera le deuxième paramètre du constructeur de la classe `RelayCommand` qui permet de définir une méthode qui servira de prédictat permettant d'indiquer si la commande peut être exécutée ou non. Ici, nous aurons simplement

besoin de renvoyer la valeur du booléen `EstBonClient`, mais cela pourrait être une méthode plus complexe. Notre instantiation de commande devient donc :

Code : C#

```
QuiSuisJeCommand = new RelayCommand<Client>(QuiSuisJe,  
    CanExecuteQuiSuisJe);
```

avec :

Code : C#

```
private bool CanExecuteQuiSuisJe(Client client)  
{  
    if (client == null)  
        return false;  
    return client.EstBonClient;  
}
```

Ainsi, non seulement il ne sera pas possible d'exécuter la commande en cliquant sur le bouton, mais le bouton est également grisé, signe de son état désactivé (voir la figure suivante).



Le bouton est grisé quand la commande n'est pas utilisable

Plutôt pratique quand un bouton doit être désactivé.

Juste avant de terminer ce point, remarquons que le prédictat associé à la possibilité d'exécution d'une commande est exécuté une unique fois. Si jamais notre client venait à devenir un bon client, notre bouton resterait dans un état désactivé car il n'aura

pas été mis au courant de ce changement. À ce moment-là, MVVM Light fournit une méthode qui permet à ce prédictat de se réévaluer et ainsi modifier éventuellement l'état du bouton. Il s'agit de la méthode `RaiseCanExecuteChanged`. On pourra l'utiliser ainsi :

Code : C#

```
((RelayCommand)QuiSuisJeCommand).RaiseCanExecuteChanged();
```

Continuons cet aperçu de MVVM Light et de ses commandes en vous indiquant comment relier n'importe quel événement à une commande. Par exemple, l'événement de sélection d'un élément dans une `ListBox`.

Première chose à faire, modifier ma vue pour ne plus avoir ce bouton, mais simplement pour avoir le prénom du client afin qu'il soit facilement sélectionnable :

Code : XML

```
<ListBox ItemsSource="{Binding ListeClients}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Prenom}" Margin="10 30 0 0" />
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Avant, pour savoir quand un élément d'un `ListBox` est sélectionné, on se serait abonné à l'événement `SelectionChanged`. Grâce à MVVM Light, on peut utiliser l'action `EventToCommand` :

Code : XML

```
<ListBox ItemsSource="{Binding ListeClients}">
    <Interactivity:Interaction.Triggers>
        <Interactivity:EventTrigger EventName="SelectionChanged" >
            <Command:EventToCommand Command="{Binding SelectionElementCommand}" PassEventArgsToCommand="True"/>
        </Interactivity:EventTrigger>
    </Interactivity:Interaction.Triggers>
    <ListBox.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Prenom}" Margin="10 30 0 0" />
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Sachant qu'il faudra importer les espaces de noms suivants :

Code : XML

```
xmlns:Command="clr-namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.WP8"
xmlns:Interactivity="clr-namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
```

Le XAML est un peu plus verbeux, je vous le concède. Le principe est d'utiliser les triggers de Blend qui correspondent au déclenchement d'un événement, de préciser l'événement choisi dans la propriété `EventName` et on pourra alors se brancher sur n'importe quel événement, ici l'événement `SelectionChanged`.

Reste à définir la commande dans le view-model :

Code : C#

```
public ICommand SelectionElementCommand { get; set; }
```

et à l'instancier, par exemple dans le constructeur :

Code : C#

```
SelectionElementCommand = new  
RelayCommand<SelectionChangedEventArgs>(OnSelectionElement);
```

Avec la méthode associée :

Code : C#

```
private void OnSelectionElement(SelectionChangedEventArgs args)  
{  
}
```

Remarquez que nous avons positionné la propriété PassEventArgsToCommand de EventToCommand à true et que nous pouvons ainsi obtenir l'argument de l'événement en paramètre de la commande. En l'occurrence, pour l'événement SelectionChanged, nous obtenons un paramètre du type SelectionChangedEventArgs.

La messagerie

Ça commence à ressembler à quelque chose, mais qu'est-ce qu'on pourrait bien faire une fois un client sélectionné ? Vous me voyez venir... on pourrait naviguer sur la vue VoirClientView.xaml et afficher le détail du client. Rien de plus simple, nous savons naviguer dans notre application, on l'a vu dans la partie précédente... sauf que nous nous heurtons à un problème de taille. Vous devinez ?

Le service de navigation est une propriété de la PhoneApplicationPage. À cause de notre séparation des responsabilités, la méthode associée à la commande se trouve dans le view-model qui n'a aucune connaissance de la vue.



Aie. Comment retrouver notre service de navigation ?

Ceci fait partie d'un problème plus général, à savoir : comment faire pour que le view-model puisse agir sur la présentation à part en utilisant les mécanismes du binding ? La navigation se retrouve exactement dans ce cas-là. C'est le code-behind qui aurait normalement pris en charge cette navigation, sauf que là, c'est impossible. On retrouve un cas similaire lorsque l'on cherche à afficher une boîte de dialogue, autre que la MessageBox.

MVVM Light propose une solution pour résoudre cet épique problème à travers son système de messagerie. Ce système offre la possibilité de pouvoir communiquer de manière découpée entre un view-model et sa vue ou entre view-models. Le principe est que l'émetteur envoie un message au système de messagerie, qui le diffuse à ceux qui s'y sont abonné. Dans notre cas, il faut donc que le code-behind s'abonne au message :

Code : C#

```
public partial class ListeClientsView : PhoneApplicationPage  
{  
    public ListeClientsView()
```

```
{  
    InitializeComponent();  
    Messenger.Default.Register<Client>(this, AfficheClient);  
}  
  
private void AfficheClient(Client client)  
{  
    PhoneApplicationService.Current.State["Client"] = client;  
    NavigationService.Navigate(new  
Uri("/View/VoirClientView.xaml", UriKind.Relative));  
}
```

Cela se fait grâce à la méthode Register du Messenger, qui se trouve dans l'espace de nom GalaSoft.MvvmLight.Messaging. On indique que l'on s'abonne aux messages qui prendront un client en paramètre et que dans ce cas, la méthode AfficheClient est appelée. La méthode AfficheClient fait une navigation toute simple, comme on l'a déjà vu.

Il faut maintenant que le view-model émette le message, mais avant ça, nous allons ajouter une liaison de données pour récupérer l'élément sélectionné de la ListBox.

Remarque, on pourrait le faire avec la valeur de l'argument, mais c'est plus propre de faire comme ça. Donc changeons notre ListBox pour avoir la liaison sur l'élément sélectionné :

Code : XML

```
<ListBox ItemsSource="{Binding ListeClients}" SelectedItem="{Binding  
Selection, Mode=TwoWay}">
```

Étant donné que notre propriété sera mise à jour à partir de l'interface, le binding doit être dans les deux sens, d'où le mode TwoWay.

Ajoutons la propriété Selection dans le view-model :

Code : C#

```
private Client selection = null;  
public Client Selection  
{  
    get { return selection; }  
    set { NotifyPropertyChanged(ref selection, value); }  
}
```

Il n'y a plus qu'à envoyer le message depuis la commande de sélection. On utilise pour cela la méthode Send du Messenger :

Code : C#

```
private void OnSelectionElement(SelectionChangedEventArgs args)  
{  
    Messenger.Default.Send(Selection);  
}
```

Résumons.

- L'utilisateur sélectionne un élément de la ListBox
- La commande associée à l'événement est déclenchée sur le view-model
- Le view-model émet un message avec le client sélectionné

- Le code-behind de la vue, qui s'est abonné à ce type de message, reçoit le message émis par le view-model et déclenche la navigation

Pour terminer proprement la petite application, il faudrait que la vue qui affiche un client utilise les données positionnées dans le dictionnaire d'état. Alors, comment feriez-vous ?

Il y a plusieurs solutions, je vous propose la plus simple.

Nous allons profiter qu'un message est diffusé à chaque sélection d'un élément pour mettre à jour les propriétés du view-model :

Code : C#

```
public VoirClientViewModel()
{
    Messenger.Default.Register<Client>(this, MetAJourClient);
    Client client =
    (Client)PhoneApplicationService.Current.State["Client"];
    MetAJourClient(client);
}

private void MetAJourClient(Client client)
{
    Prenom = client.Prenom;
    Age = client.Age;
    if (client.EstBonClient)
        BonClient = new SolidColorBrush(Color.FromArgb(100, 0, 255,
0));
    else
        BonClient = new SolidColorBrush(Color.FromArgb(100, 255, 0,
0));
}
```

Il suffit de s'abonner également à la réception de ce message afin de mettre à jour les propriétés avec le nouveau client courant. Ce message est bien celui émis par le view-model de la liste des clients et c'est le view-model qui permet de voir un client qui le reçoit après s'y être abonné.

Il faudra faire attention à l'initialisation où nous utiliserons le dictionnaire d'état pour récupérer la première sélection.

Notons enfin que vous devez réinitialiser la propriété Selection afin que la ListBox ne conserve pas la sélection lors du retour arrière sur la page :

Code : C#

```
private void OnSelectionElement(SelectionChangedEventArgs args)
{
    if (Selection == null)
        return;
    Messenger.Default.Send(Selection);
    Selection = null;
}
```

Voilà pour ce petit tour de MVVM Light. Nous avons vu l'essentiel de ce toolkit qui propose des solutions pour aider à la mise en place du patron de conception MVVM. N'oubliez pas que malgré sa dénomination de light, c'est un framework complet qui prend sa place (110 ko). Il est en fait light par rapport à d'autres framework, comme PRISM qui est utilisé avec WPF. A utiliser en connaissance de cause.

D'autres frameworks MVVM

MVVM Light n'est pas le seul framework aidant à la mise en place de MVVM. C'est assurément l'un des plus connus, mais d'autres existent respectant plus ou moins bien le patron de conception et apportant des outils différents. Citons par exemple :

- Calcium <http://calcium.codeplex.com>

- Caliburn Micro <http://caliburnmicro.codeplex.com/>
- Catel <http://catel.codeplex.com/>
- nRoute <http://nroute.codeplex.com/>
- Simple MVVM Toolkit <http://simplemvvmtoolkit.codeplex.com/>
- UltraLight.mvvm <http://ultralightmvvm.codeplex.com/>

Je ne peux pas bien sur tous les présenter et d'ailleurs je ne les ai pas tous testés .

J'aime bien l'UltraLight.mvvm qui, via son locator, offre une liaison avec la page ce qui permet facilement de démarrer une navigation sans passer par l'utilisation d'une messagerie.

N'hésitez pas à les tester pour vous faire votre propre opinion et pour vous permettre de voir ce que vous souhaitez garder de MVVM et ce dont vous pouvez vous passer.

Faut-il utiliser systématiquement MVVM ?

MVVM est complexe à appréhender. Pour bien le comprendre, il faut pratiquer. Ce n'est que petit à petit que vous verrez vraiment de quoi vous avez besoin et à quel moment.

J'imagine que pour l'instant, vous avez l'impression que MVVM pose plus de problèmes qu'il n'en résout et qu'on se complique la vie pour pas grand-chose. Franchement, le coup de la navigation et de la messagerie, c'est censé nous simplifier la vie ?

Il est vrai que lorsque l'on réalise des petites applications, respecter parfaitement le patron de conception MVVM est sans doute un peu démesuré. Cela implique toute une mécanique qui est plutôt longue à mettre en place et parfois ennuyeuse, pour un gain pas forcément évident.

N'oubliez cependant pas que le but premier de MVVM est de séparer les responsabilités, notamment en séparant les données de la vue. Cela facilite les opérations de maintenance en limitant l'éternel problème du plat de spaghetti où la moindre correction a des impacts sur un autre bout de code. Mon avis sur MVVM est que peu importe si vous ne respectez pas parfaitement MVVM, le principe de ce pattern est de vous aider dans la réalisation de votre application et surtout dans sa maintenabilité.

L'intérêt également est qu'il devient possible de faire des tests unitaires sur le view-model, sans avoir besoin de charger l'application et de cliquer partout. Cela permet de tester chaque fonctionnalité, dans un processus automatisé, ce qui dans une grosse application est un atout considérable pour éviter les régressions.

En tous cas, n'ayez crainte. Vous n'êtes pas obligés de pratiquer MVVM tout de suite. En tant que débutant, vous aurez plutôt intérêt à commencer à créer des applications et ensuite à chercher à appliquer des bonnes pratiques. Dans ce cas, n'hésitez pas à revenir lire ce chapitre.



Remarque : par souci de simplicité et de concision, je ne respecterai pas MVVM dans la suite de ce cours.

- MVVM est un patron de conception qui aide à la réalisation d'applications d'envergure utilisant le XAML.
- Des frameworks gratuits nous aident à la mise en place de ce patron de conception en fournissant des bibliothèques remplies de classes éprouvées.
- Le respect et l'utilité de MVVM se découvrent en pratiquant. Ne soyez pas forcément trop pressés de respecter parfaitement MVVM.

Gestion des états visuels

Maintenant que nous connaissons les templates, nous allons revenir sur un point qui vous sera sûrement utile dans le développement de vos applications. Il s'agit de la gestion des états visuels. Mais qu'est-ce donc ?

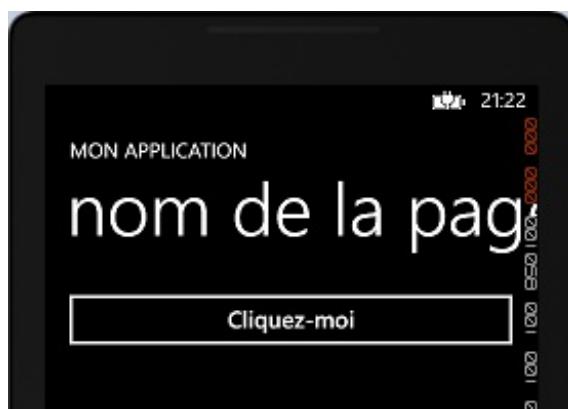
Prenons un bouton par exemple, il possède plusieurs états visuels. Il y a l'état quand il est cliqué, l'état quand il est désactivé et l'état lorsqu'il est au repos.

C'est la même chose pour une `ListBox`, nous avons par exemple vu un état où un élément est sélectionné, qui est d'ailleurs mis en avant grâce à la couleur d'accentuation du téléphone.

Tous les contrôles ont potentiellement plusieurs représentations visuelles en fonction de leurs états. Voyons voir comment cela fonctionne.

Les états d'un contrôle

Observons les états de notre bouton plus en détail. À la figure suivante, vous pouvez le voir dans son état normal, au repos.



Etat du bouton au repos

À la figure suivante, vous voyez son état quand il est cliqué.



Etat du bouton cliqué

Et sur la figure suivante, lorsqu'il est désactivé et donc non cliquable.



Etat du bouton désactivé

Ils correspondent respectivement aux états :

- Normal
- Pressed
- Disabled

Chaque contrôle dispose de différents états. Les états changent automatiquement en fonction d'une action utilisateur ou d'une propriété. Par exemple, c'est en cliquant sur le bouton que celui-ci passe de l'état `Normal` à `Pressed` et en relâchant le clic que celui-ci passe de `Pressed` à `Normal`, changeant au passage l'apparence du contrôle. Pour l'état désactivé, ce changement se fait quand on passe la propriété `IsEnabled` à `false`.

Le bouton possède d'autres états qui ne sont pas utilisés pour Windows Phone, comme l'état `Focused` et l'état `Unfocused` qui correspondent au fait que le bouton ait le focus ou pas, ce qui ne sert pas vraiment dans une application pour Windows Phone, ainsi que l'état `MouseOver` qui correspond au passage de la souris sur le bouton.



On dit qu'un contrôle possède le focus quand il est la cible des actions de saisie de l'utilisateur, comme le clavier. C'est très valable pour une application PC, mais beaucoup moins pour une application de téléphone.

Ces trois états du bouton sont un héritage de Silverlight pour PC, ils ne servent pas avec le XAML pour Windows Phone. Un contrôle peut être dans plusieurs états à la fois, par exemple si c'était valable on pourrait envisager qu'il soit dans un état `Pressed` et un état où il ait le focus. Par contre, d'autres états sont exclusifs, il n'est bien sûr pas possible que le bouton soit dans l'état `Pressed` et dans l'état `Normal`... Pour représenter ceci, les états appartiennent à des groupes. Le bouton ne peut avoir qu'un seul état par groupe. En l'occurrence, notre bouton possède deux groupes avec les états suivants :

Groupe `FocusStates` :

- `Focused`
- `Unfocused`

Groupe `CommonStates` :

- `Pressed`
- `Disabled`
- `Normal`
- `MouseOver`

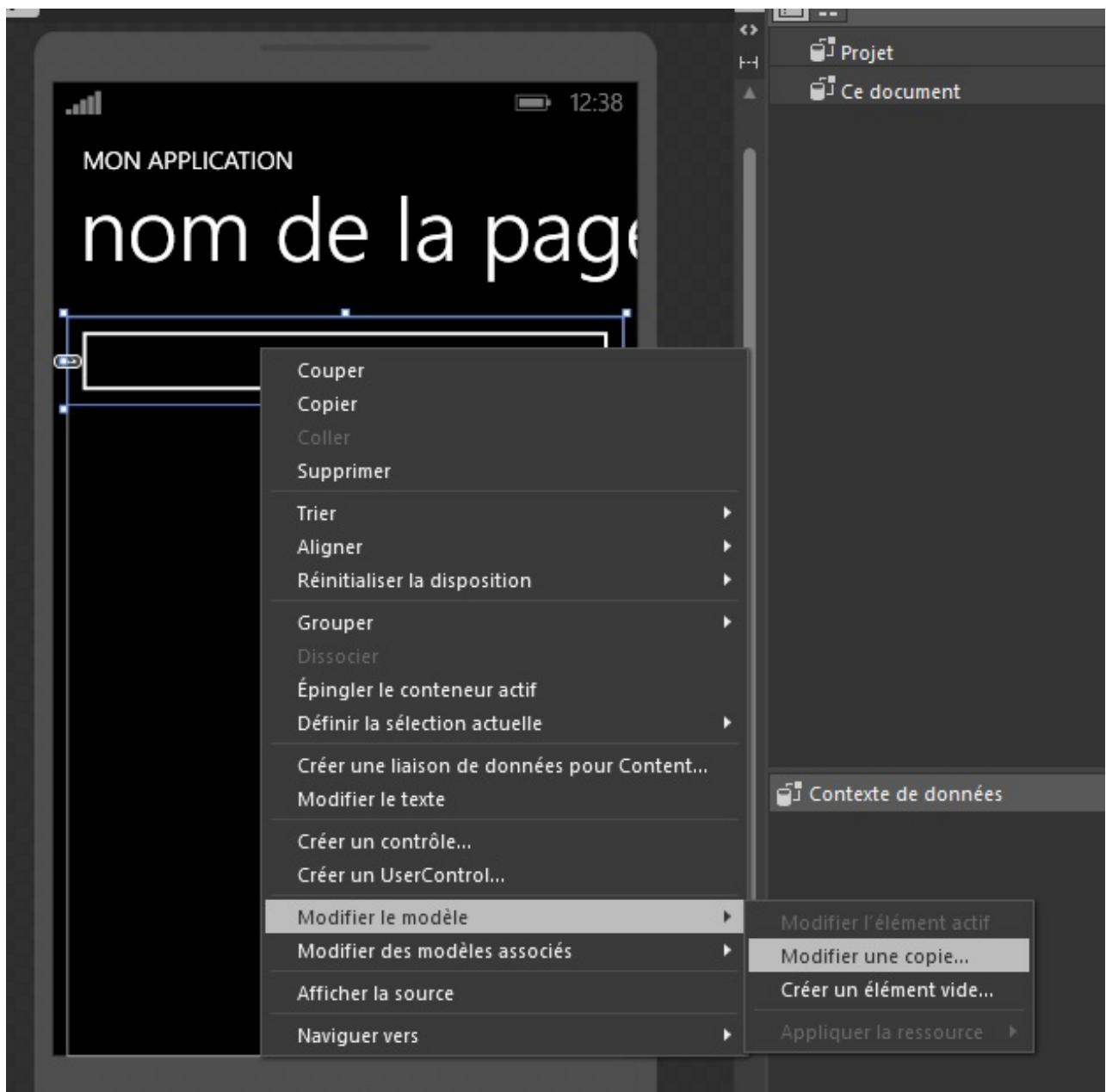
À chaque état donc son apparence... ce qui implique que nous pouvons modifier les apparences de chaque état via des templates que nous allons définir dans un style.

Pour comprendre, le plus simple est d'utiliser Blend. Ajoutons un bouton dans notre XAML :

Code : XML

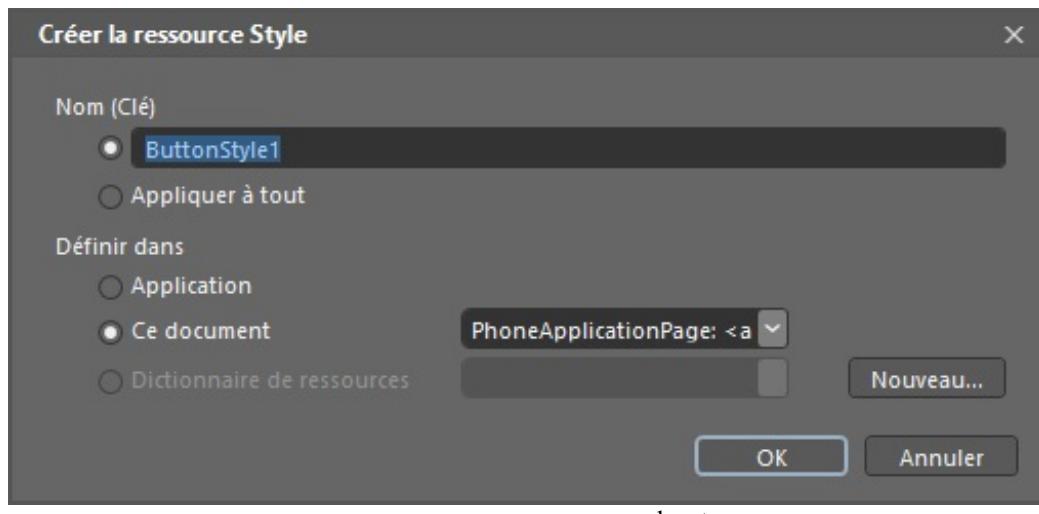
```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <Button x:Name="But" Content="Cliquez-moi !" />
    </StackPanel>
</Grid>
```

Puis démarrons Blend en faisant un clic droit sur le projet, puis ouvrir dans expression blend. Une fois ouvert, cliquez sur le bouton pour le sélectionner et faites un clic droit pour modifier une copie du modèle, comme indiqué à la figure suivante.

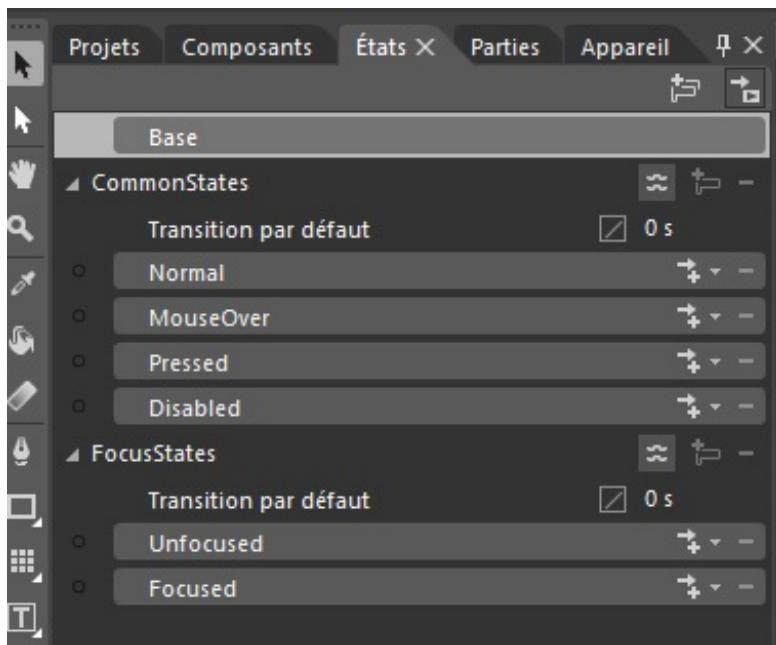


Modification du modèle du bouton dans Blend

Cela permet de créer un style automatiquement (voir la figure suivante).



Maintenant, nous pouvons voir dans l'onglet Etats, les différents états du bouton, comme vous pouvez le constater sur la figure suivante.



Les différents états du bouton

Remarquons que le XAML est désormais :

Code : XML

```
<phone:PhoneApplicationPage
    x:Class="DemoEtatVisuel.MainPage"
    ...>
<phone:PhoneApplicationPage.Resources>
    <Style x:Key="ButtonStyle1" TargetType="Button">
        <Setter Property="Background" Value="Transparent"/>
        <Setter Property="BorderBrush" Value="{StaticResource
PhoneForegroundBrush}"/>
        <Setter Property="Foreground" Value="{StaticResource
PhoneForegroundBrush}"/>
        <Setter Property="BorderThickness" Value="{StaticResource
PhoneBorderThickness}"/>
        <Setter Property="FontFamily" Value="{StaticResource
PhoneFontFamilySemiBold}"/>
        <Setter Property="FontSize" Value="{StaticResource
PhoneFontSizeMedium}"/>
        <Setter Property="Padding" Value="10,5,10,6"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Grid Background="Transparent">
                        <VisualStateManager.VisualStateGroups>
                            <VisualStateGroup x:Name="CommonStates">
                                <VisualState x:Name="Normal"/>
                                <VisualState x:Name="MouseOver"/>
                                <VisualState x:Name="Pressed">
                                    <Storyboard>
                                        <ObjectAnimationUsingKeyFrames
Storyboard.TargetProperty="Foreground"
Storyboard.TargetName="ContentContainer">
                                            <DiscreteObjectKeyFrame KeyTime="0"
Value="{StaticResource PhoneButtonBasePressedForegroundBrush}"/>
                                        </ObjectAnimationUsingKeyFrames>
                                        <ObjectAnimationUsingKeyFrames
Storyboard.TargetProperty="Background"
Storyboard.TargetName="ButtonBackground">
                                            <DiscreteObjectKeyFrame KeyTime="0"
Value="{StaticResource PhoneAccentBrush}"/>

```

```
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ObjectAnimationUsingKeyFrames
            Storyboard.TargetProperty="Foreground"
            Storyboard.TargetName="ContentContainer">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{StaticResource PhoneDisabledBrush}" />
        </ObjectAnimationUsingKeyFrames>
        <ObjectAnimationUsingKeyFrames
            Storyboard.TargetProperty="BorderBrush"
            Storyboard.TargetName="ButtonBackground">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="{StaticResource PhoneDisabledBrush}" />
        </ObjectAnimationUsingKeyFrames>
        <ObjectAnimationUsingKeyFrames
            Storyboard.TargetProperty="Background"
            Storyboard.TargetName="ButtonBackground">
            <DiscreteObjectKeyFrame KeyTime="0"
                Value="Transparent" />
        </ObjectAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateManager.VisualStateGroups>
<Border x:Name="ButtonBackground"
    BorderBrush="{TemplateBinding BorderBrush}"
    BorderThickness="{TemplateBinding BorderThickness}"
    Background="{TemplateBinding Background}" CornerRadius="0"
    Margin="{StaticResource PhoneTouchTargetOverhang}">
    <ContentControl x:Name="ContentContainer"
        ContentTemplate="{TemplateBinding ContentTemplate}"
        Content="{TemplateBinding Content}" Foreground="{TemplateBinding
        Foreground}" HorizontalContentAlignment="{TemplateBinding
        HorizontalContentAlignment}" Padding="{TemplateBinding Padding}"
        VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"
        VerticalContentAlignment}"/>
    </Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</phone:PhoneApplicationPage.Resources>

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
        Margin="12,17,0,28">
        <TextBlock Text="MON APPLICATION" Style="{StaticResource
        PhoneTextNormalStyle}" Margin="12,0" />
        <TextBlock Text="nom de la page" Margin="9,-7,0,0"
        Style="{StaticResource PhoneTextTitle1Style}" />
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <Button x:Name="But" Content="Cliquez-moi !" Style="{StaticResource
            ButtonStyle1}" />
        </StackPanel>
    </Grid>
</Grid>
</phone:PhoneApplicationPage>
```

Beaucoup de choses, mais ce qu'il faut regarder précisément c'est que Blend a déclaré un nouveau template du bouton, celui-ci est simplement :

Code : XML

```
<Border x:Name="ButtonBackground" BorderBrush="{TemplateBinding BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}" Background="{TemplateBinding Background}" CornerRadius="0" Margin="{StaticResource PhoneTouchTargetOverhang}">
    <ContentControl x:Name="ContentContainer"
        ContentTemplate="{TemplateBinding ContentTemplate}"
        Content="{TemplateBinding Content}" Foreground="{TemplateBinding Foreground}" HorizontalContentAlignment="{TemplateBinding HorizontalContentAlignment}" Padding="{TemplateBinding Padding}" VerticalContentAlignment="{TemplateBinding VerticalContentAlignment}"/>
</Border>
```

Bref, un contrôle Border contenant un contrôle ContentControl, ce qui est précisément le look du bouton tel que nous le connaissons.



L'extension de balisage TemplateBinding permet de lier la valeur d'une propriété dans un modèle afin de la définir comme valeur d'une autre propriété exposée dans le contrôle. Typiquement ici, le fait de définir par exemple la propriété Foreground sur le contrôle Button va en fait appliquer cette valeur au contrôle ContentControl.

Ensuite, il faut regarder l'intérieur de la balise `<VisualStateManager.VisualStateGroups>`. À l'intérieur sont définis tous les groupes d'états du contrôle :

Code : XML

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal"/>
        <VisualState x:Name="MouseOver"/>
        <VisualState x:Name="Pressed">
            <Storyboard>
                ...
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Disabled">
            <Storyboard>
                ...
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
    <VisualStateGroup x:Name="FocusStates"/>
</VisualStateManager.VisualStateGroups>
```

Ce qu'on peut voir c'est que les états définissent une animation qui permet de changer la valeur de certaines propriétés lorsque le contrôle change d'état. Par exemple, en passant à l'état Pressed, nous pouvons constater que la propriété Foreground passe à la valeur de PhoneButtonBasePressedForegroundBrush.

Voilà comment sont définis les états, dans des modèles.

Modifier un état

Ceci nous permet de faire ce que nous voulons avec les états des contrôles afin d'améliorer le look de nos boutons. Rappelez-vous dans la première partie, nous avons modifié l'apparence d'un bouton en modifiant sa propriété Content :

Code : XML

```
<Button x:Name="But">
    <Button.Content>
        <StackPanel>
            <Image Source="rouge.png" Width="100" Height="100" />
            <TextBlock Text="Cliquez-moi !" />
        </StackPanel>
    </Button.Content>
</Button>
```

Comme on peut s'en rendre compte maintenant que nous avons vu le modèle original du contrôle, nous n'avons modifié que ce qui correspondait au contenu du ContentControl. Le cadre est donc conservé, mais plus encore, la même animation sur le fond du cadre existe toujours.

Ceci ne correspond peut-être pas à ce que nous souhaitons avoir lorsque le bouton est cliqué. En l'occurrence, moi ce que je voudrais, c'est que le rond rouge devienne vert et que le texte passe à « cliqué ». Pour cela, il suffit de modifier le template de l'état Pressed de notre contrôle... Reprenons donc notre bouton :

Code : XML

```
<Button x:Name="But" Content="Cliquez-moi !" Style="{StaticResource
    ButtonStyle1}" />
```

et modifions ses templates à l'intérieur de son style :

Code : XML

```
<ControlTemplate TargetType="Button">
    <Grid Background="Transparent">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal"/>
                <VisualState x:Name="MouseOver"/>
                <VisualState x:Name="Pressed">
                    <Storyboard>
                        <ObjectAnimationUsingKeyFrames
                            Storyboard.TargetProperty="Source" Storyboard.TargetName="MonRond">
                            <DiscreteObjectKeyFrame KeyTime="0"
                                Value="vert.png"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames
                            Storyboard.TargetProperty="Text" Storyboard.TargetName="MonTexte">
                            <DiscreteObjectKeyFrame KeyTime="0"
                                Value="Cliqué :)" />
                        </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <ObjectAnimationUsingKeyFrames
                            Storyboard.TargetProperty="Visibility"
                            Storyboard.TargetName="MonRond">
                            <DiscreteObjectKeyFrame KeyTime="0"
                                Value="Collapsed"/>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames
                            Storyboard.TargetProperty="Text" Storyboard.TargetName="MonTexte">
                            <DiscreteObjectKeyFrame KeyTime="0"
                                Value="Pas touche ..." />
                        </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager>
    </Grid>
</ControlTemplate>
```

```
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="ButtonBackground"
BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}"
Background="{TemplateBinding Background}" CornerRadius="0"
Margin="{StaticResource PhoneTouchTargetOverhang}">
    <StackPanel>
        <Image x:Name="MonRond" Source="rouge.png"
Width="100" Height="100" />
        <TextBlock x:Name="MonTexte" Text="Cliquez-moi !" />
    </StackPanel>
</Border>
</Grid>
</ControlTemplate>
```

Vous pouvez voir que j'ai modifié l'apparence du contrôle pour qu'il contienne notre image et notre texte :

Code : XML

```
<Border x:Name="ButtonBackground" BorderBrush="{TemplateBinding
BorderBrush}" BorderThickness="{TemplateBinding BorderThickness}"
Background="{TemplateBinding Background}" CornerRadius="0"
Margin="{StaticResource PhoneTouchTargetOverhang}">
    <StackPanel>
        <Image x:Name="MonRond" Source="rouge.png" Width="100"
Height="100" />
        <TextBlock x:Name="MonTexte" Text="Cliquez-moi !" />
    </StackPanel>
</Border>
```

Puis, dans l'état `Pressed`, j'ai animé les propriétés `Source` de l'image et `Text` du `TextBlock` pour charger une nouvelle image et changer le texte :

Code : XML

```
<VisualState x:Name="Pressed">
    <Storyboard>
        <ObjectAnimationUsingKeyFrames
Storyboard.TargetProperty="Source" Storyboard.TargetName="MonRond">
            <DiscreteObjectKeyFrame KeyTime="0" Value="vert.png"/>
        </ObjectAnimationUsingKeyFrames>
        <ObjectAnimationUsingKeyFrames
Storyboard.TargetProperty="Text" Storyboard.TargetName="MonTexte">
            <DiscreteObjectKeyFrame KeyTime="0" Value="Cliqué :)" />
        </ObjectAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
```

Bien sûr, il faudra rajouter les images `rouge.png` et `vert.png` à la solution.

De la même façon, dans l'état `Disabled`, j'ai changé la visibilité de l'image pour la faire disparaître, puis j'ai animé le texte pour le changer :

Code : XML

```
<VisualState x:Name="Disabled">
    <Storyboard>
        <ObjectAnimationUsingKeyFrames
```

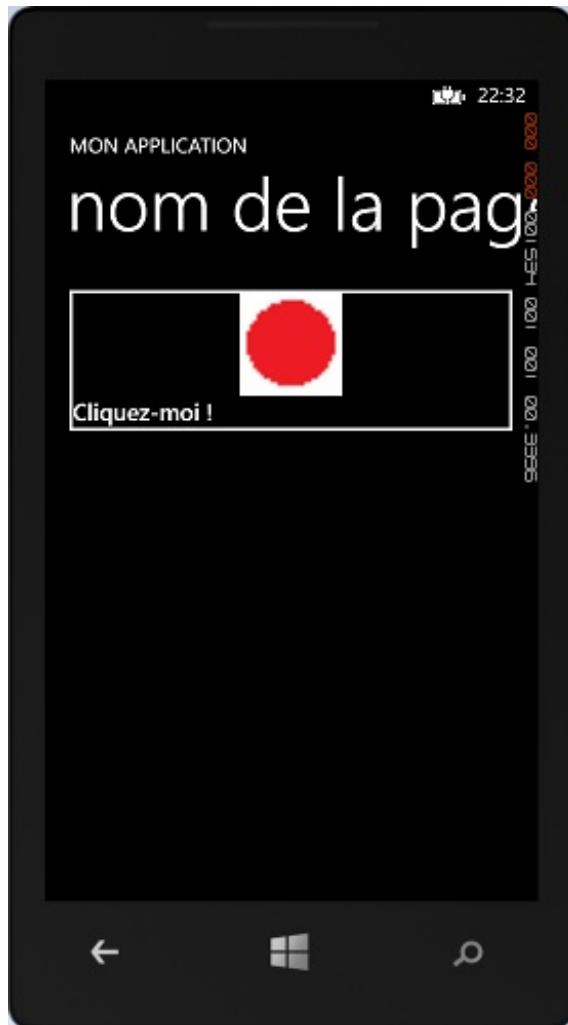
```
Storyboard.TargetProperty="Visibility"
Storyboard.TargetName="MonRond">
    <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames
Storyboard.TargetProperty="Text" Storyboard.TargetName="MonTexte">
    <DiscreteObjectKeyFrame KeyTime="0" Value="Pas touche
..."/>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
```

Du coup, dans mon XAML, je me retrouve avec mon bouton :

Code : XML

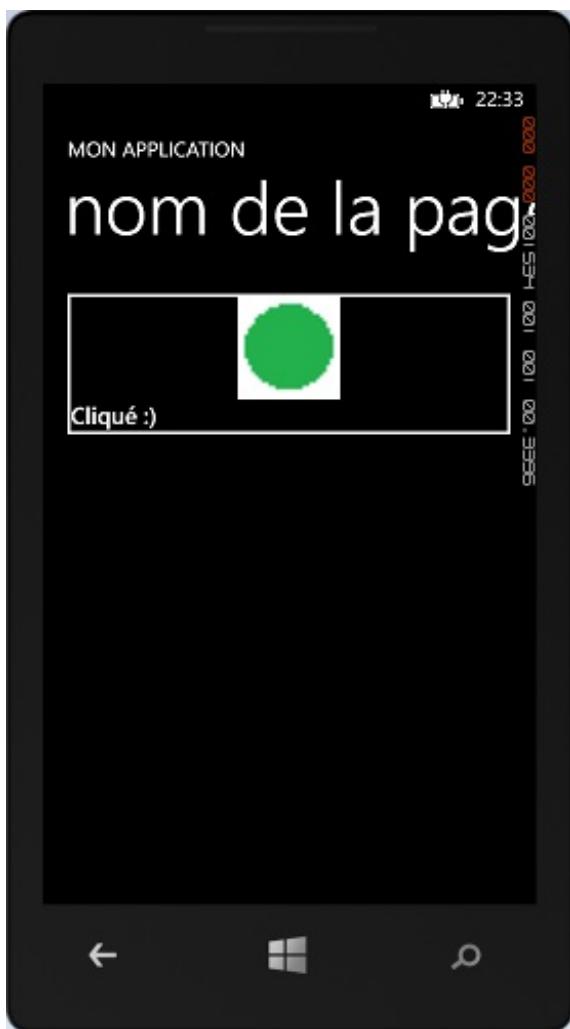
```
<Button x:Name="But" Content="Cliquez-moi !" Style="{StaticResource
ButtonStyle1}" />
```

Qui, lorsqu'il est au repos, ressemble à ce que vous pouvez voir sur la figure suivante.



Etat du bouton modifié au repos

Lorsqu'il est cliqué, il est plutôt ainsi (voir la figure suivante).



Etat du bouton modifié lorsqu'il est cliqué

Et lorsqu'il est désactivé, il est comme vous pouvez le voir à la figure suivante.



Etat du bouton modifié lorsqu'il est désactivé



Ici, nous avons fait les modifications des animations des différents états directement en XAML, mais il aurait été tout à fait possible d'utiliser Blend pour le faire. C'est notamment plus pratique lorsqu'il y a des animations complexes.

Changer d'état

Bon... j'avoue ! Dans le chapitre précédent j'ai triché ! Mais ne le dites à personne 😊.

Pour faire mes copies d'écrans, je n'ai pas cherché à appuyer sur la touche de copie d'écran tout en maintenant le clic sur le bouton... trop compliqué, je fais attention à l'état... de mes doigts. 🤪 J'ai donc pour l'occasion changé l'état du bouton par code.

Et oui, il n'y a pas que les actions de l'utilisateur qui peuvent changer l'état d'un contrôle. Il est très simple de changer l'état d'un contrôle avec une ligne de code. On utilise pour cela [le VisualStateManager](#). Par exemple, pour passer sur l'état `Pressed`, je peux utiliser :

Code : C#

```
VisualStateManager.GoToState(But, "Pressed", true);
```

Il suffit de connaître le nom de l'état à atteindre et le tour est joué.

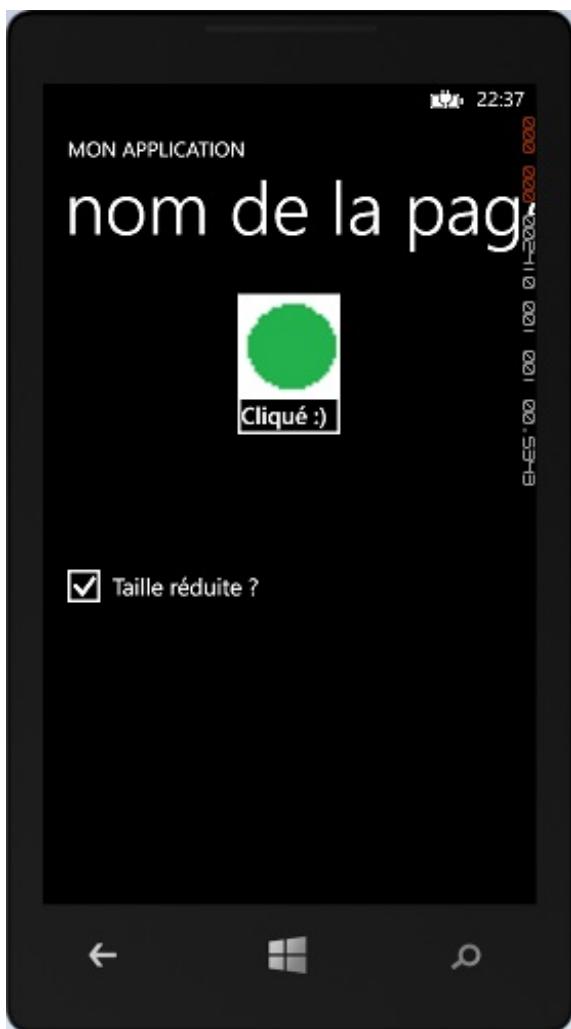
Créer un nouvel état

Et vous savez quoi ? Il est même possible de rajouter des états à un contrôle. Imaginons par exemple que je souhaite que mon bouton puisse être dans un état « TailleReduite » et un autre « TailleNormale » où vous l'aurez compris notre bouton pourra avoir deux apparences différentes en fonction de son état. Ce nouvel état sera bien sûr cumulatif aux autres états, comme `Pressed` ou `Disabled`. Comme nous l'avons vu, il va falloir commencer par créer un nouveau groupe d'état. Pour le rajouter, il suffit de se placer à l'intérieur de la propriété `<VisualStateManager.VisualStateGroups>`. Et nous aurons par exemple :

Code : XML

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="TailleStates">
        <VisualState x:Name="TailleNormale"/>
        <VisualState x:Name="TailleReduite">
            <Storyboard>
                <ObjectAnimationUsingKeyFrames
                    Storyboard.TargetProperty="Width"
                    Storyboard.TargetName="ButtonBackground">
                    <DiscreteObjectKeyFrame KeyTime="0"
                        Value="100"/>
                </ObjectAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal"/>
        <VisualState x:Name="MouseOver"/>
        <VisualState x:Name="Pressed">
            ...
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Ici, j'ai simplement choisi de réduire la taille de la propriété Width du contrôle Border afin de réduire la taille du bouton. Le voici à la figure suivante donc dans un état Pressed et TailleReduite.



Le bouton est dans l'état Pressed et TailleReduite

Pour obtenir cela, dans le XAML, j'aurai toujours mon bouton, mais j'ai également rajouté une case à cocher pour pouvoir positionner l'état réduit :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <Button x:Name="But" Content="Cliquez-moi !" Style="{StaticResource ButtonStyle1}" />
        <CheckBox Margin="0 100 0 0" Content="Taille réduite ?" Checked="CheckBox_Checked" Unchecked="CheckBox_Unchecked" />
    </StackPanel>
</Grid>
```

Avec dans le code-behind :

Code : C#

```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    VisualStateManager.GoToState(But, "TailleReduite", true);
}

private void CheckBox_Unchecked(object sender, RoutedEventArgs e)
{
    VisualStateManager.GoToState(But, "TailleNormale", true);
}
```



On peut également facilement ajouter un nouvel état grâce à Blend, nous le verrons dans la partie suivante lors de la gestion de l'orientation.

- Un contrôle peut posséder plusieurs états, comme un bouton qui peut être cliqué ou non cliqué.
- À chaque état est associée une représentation visuelle différente, qu'il est possible de modifier grâce aux templates.
- Il est possible de créer un nouvel état ou un nouveau groupe d'état pour un contrôle.
- On change un état par code grâce à la classe VisualStateManager.
- Blend peut se révéler très utile dans la création ou la modification d'états.

Le traitement des données

Généralement, dans nos applications, nous allons avoir besoin de traiter des données. Des clients, des commandes ou tout autre chose qui a toute sa place dans une ListBox ou dans d'autres contrôles. Nous venons en plus de parler de modèle dans le chapitre sur MVVM en simulant un peu maladroitement un chargement de données, avec des données en dur dans le code. Dans une application de gestion, les données évoluent au fur et à mesure. La liste de clients s'agrandit, le nombre de produits du catalogue augmente, les prix changent, etc.

Bref, nous allons avoir besoin de gérer des données, aussi nous allons nous attarder dans ce chapitre à considérer différentes solutions pour récupérer des données et les manipuler.

HttpRequest & WebClient

Dans une application pour mobile, il y a souvent beaucoup d'occasions pour récupérer des informations disponibles sur internet, notamment avec la présence de plus en plus importante du cloud.

Récupérer des données sur internet consiste généralement en trois choses :

- Demander la récupération de données.
- Attendre la fin du téléchargement.
- Interpréter ces données.

Dans nos applications Windows Phone, la récupération de données est obligatoirement asynchrone pour éviter de bloquer l'application et garder une interface fonctionnelle. Cela veut dire qu'on va lancer le téléchargement et être notifié de la fin par un événement. C'est à ce moment-là que l'on pourra interpréter les données. Il y a plusieurs façons de faire des appels, la première est d'utiliser les classes `WebClient` et `HttpWebRequest`.

Si vous avez simplement besoin de récupérer une chaîne (ou du XML brut) le plus simple est d'utiliser la classe `WebClient`. Par exemple, la page internet http://www.siteduzero.com/uploads/fr/ftp/windows_phone/script_nico.php renvoie la chaîne « Bonjour tout le monde ». Pour y accéder, nous pouvons écrire le code suivant :

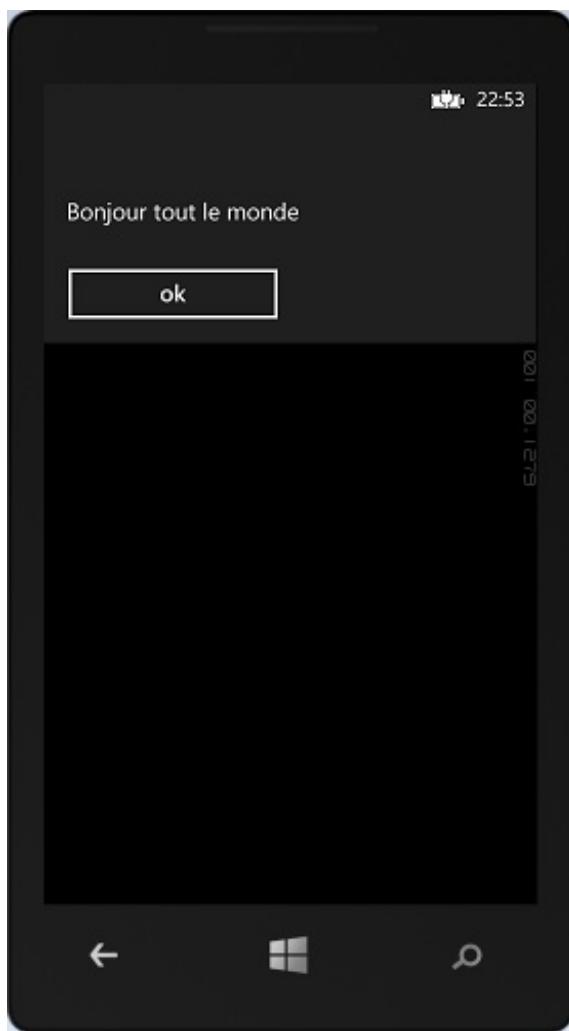
Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        WebClient client = new WebClient();
        client.DownloadStringCompleted += client_DownloadStringCompleted;
        client.DownloadStringAsync(new
Uri("http://www.siteduzero.com/uploads/fr/ftp/windows_phone/script_nico.php"));
    }

    private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            string texte = e.Result;
            MessageBox.Show(texte);
        }
        else
        {
            MessageBox.Show("Impossible de récupérer les données sur internet :
" + e.Error);
        }
    }
}
```

Ce qui donne le résultat de la figure suivante.



Affichage d'une donnée récupérée sur internet

Le principe est de s'abonner à l'événement de fin de téléchargement, de déclencher le téléchargement asynchrone et de récupérer le résultat. Ici, il n'y a pas de traitement à faire pour interpréter les données, vu que nous souhaitons afficher directement la chaîne récupérée.

La méthode `DownloadStringAsync` fonctionne très bien pour tout ce qui est texte brut. Si vous voulez télécharger des données binaires, vous pourrez utiliser la méthode `OpenReadAsync`.

La classe `WebClient` est parfaite pour faire des téléchargements simples mais elle devient vite limitée lorsque nous devons faire des opérations plus pointues sur les requêtes, comme modifier les entêtes HTTP ou envoyer des données en POST, etc. C'est là que nous allons utiliser la classe `HttpWebRequest`. Elle offre un contrôle plus fin sur la requête web. Nous allons illustrer ceci en faisant une requête sur le formulaire PHP du cours PHP du site OpenClassrooms, disponible à cet emplacement : http://fr.openclassrooms.com/uploads/fr/ftp/mateo21/php/form_text/formulaire.php

Le principe est d'envoyer des données en POST, notamment la donnée : "prenom=Nicolas".

Le code est le suivant :

Code : XML

```
<TextBlock x:Name="Resultat" TextWrapping="Wrap"/>
```

Avec le code behind qui suit :

Code : C#

```
public MainPage()
{
    InitializeComponent();
```

```
        HttpWebRequest requete =
    (HttpWebRequest)HttpWebRequest.Create("http://fr.openclassrooms.com/uploads/fr/i
    requete.Method = "POST";
    requete.ContentType = "application/x-www-form-urlencoded";

    requete.BeginGetRequestStream(DebutReponse, requete);
}

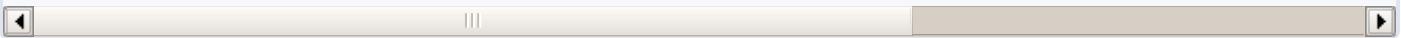
private void DebutReponse(IAsyncResult resultatAsynchrone)
{
    HttpWebRequest requete = (HttpWebRequest)resultatAsynchrone.AsyncState;
    Stream postStream = requete.EndGetRequestStream(resultatAsynchrone);
    string donneesAEnvoyer = "prenom=Nicolas";

    byte[] tableau = Encoding.UTF8.GetBytes(donneesAEnvoyer);
    postStream.Write(tableau, 0, donneesAEnvoyer.Length);
    postStream.Close();
    requete.BeginGetResponse(FinReponse, requete);
}

private void FinReponse(IAsyncResult resultatAsynchrone)
{
    HttpWebRequest requete = (HttpWebRequest)resultatAsynchrone.AsyncState;
    WebResponse webResponse = requete.EndGetResponse(resultatAsynchrone);
    Stream stream = webResponse.GetResponseStream();

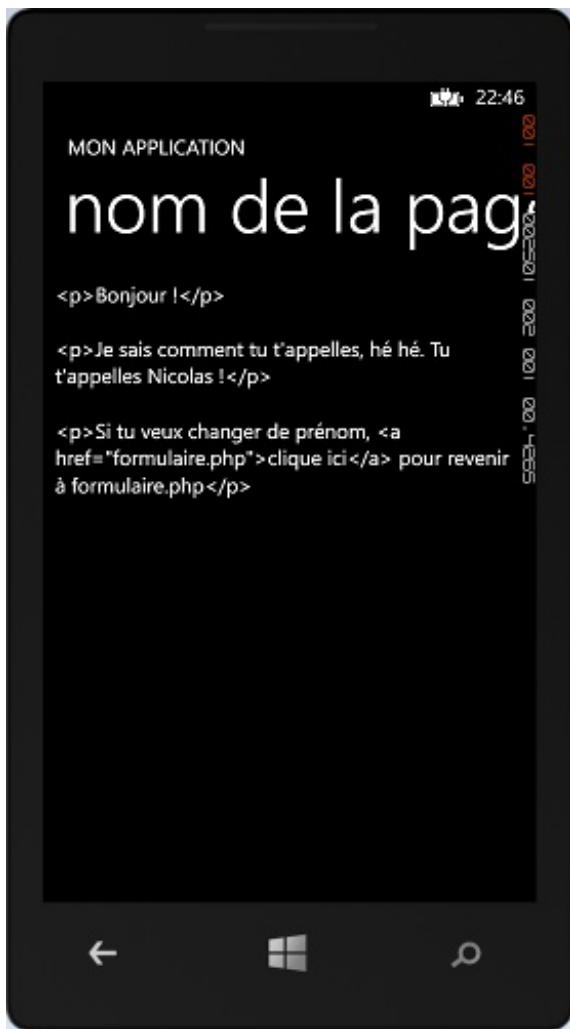
    StreamReader streamReader = new StreamReader(stream);
    string reponse = streamReader.ReadToEnd();
    stream.Close();
    streamReader.Close();
    webResponse.Close();

    Dispatcher.BeginInvoke(() => Resultat.Text = reponse);
}
```



Stream et StreamReader sont dans l'espace de nom System.IO.Encoding est dans l'espace de nom System.Text.

Ce code peut paraître un peu compliqué, mais c'est toujours le même pour faire ce genre de requête. On commence par créer la requête en lui indiquant la méthode POST. Ensuite dans la première méthode on écrit les données à envoyer dans le flux et on invoque la requête asynchrone. Dans la deuxième méthode, on récupère le retour (voir la figure suivante).



Envoi d'un formulaire POST à une page PHP et affichage du retour

Ici, le retour est du HTML, ce qui est normal vu que ce formulaire a été prévu pour une page web. Il aurait pu être judicieux d'interpréter le résultat, en retirant les balises HTML par exemple...



Attention, un téléchargement fait avec `HttpWebRequest` s'exécute sur un thread en arrière-plan, cela veut dire que si nous voulons mettre à jour l'interface, nous aurons besoin d'utiliser le dispatcher pour re-basculer sur le thread de l'interface.

C'est ce que nous avons fait ici avec :

Code : C#

```
Dispatcher.BeginInvoke(() => resultat.Text = reponse);
```

Ce qui permet de mettre à jour la propriété `Text` du `TextBlock`.



Remarquons que si nous utilisons la classe `WebClient` dans un thread différent de celui permettant de mettre à jour l'interface, le résultat sera aussi dans un thread différent, il faudra donc utiliser un dispatcher dans ce cas aussi. Nous reviendrons sur ces notions de Thread et de Dispatcher dans une prochaine partie.

Nous pouvons également consommer facilement des services web avec une application Windows Phone. Un service web est une espèce d'application web qui répond à des requêtes permettant d'appeler une méthode avec des paramètres et de recevoir en réponse le retour de la méthode.

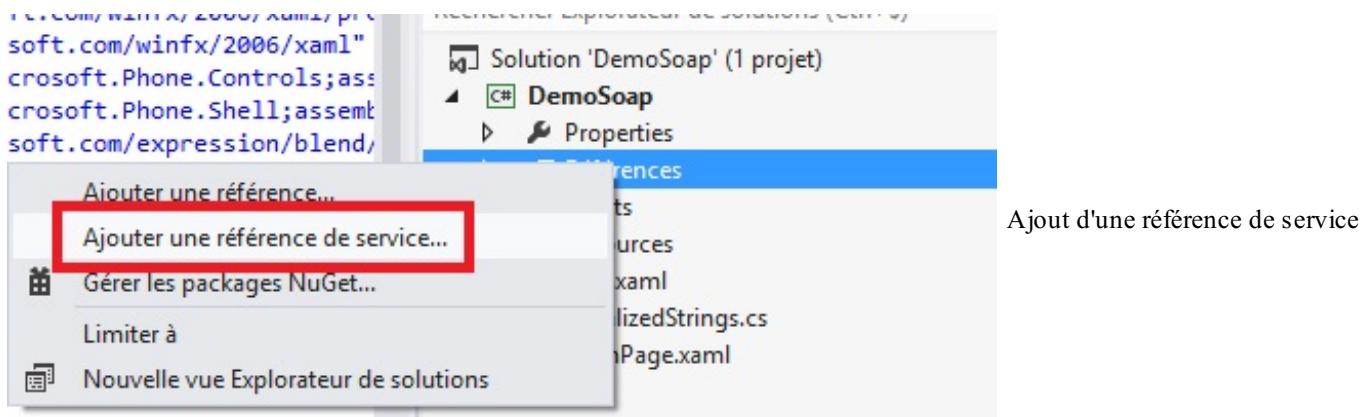


Mais comment appeler un service web ? Quelle adresse ? Comment connaît-on le nom des méthodes à appeler ?
Comment indiquer les paramètres et les valeurs que l'on souhaite passer ?

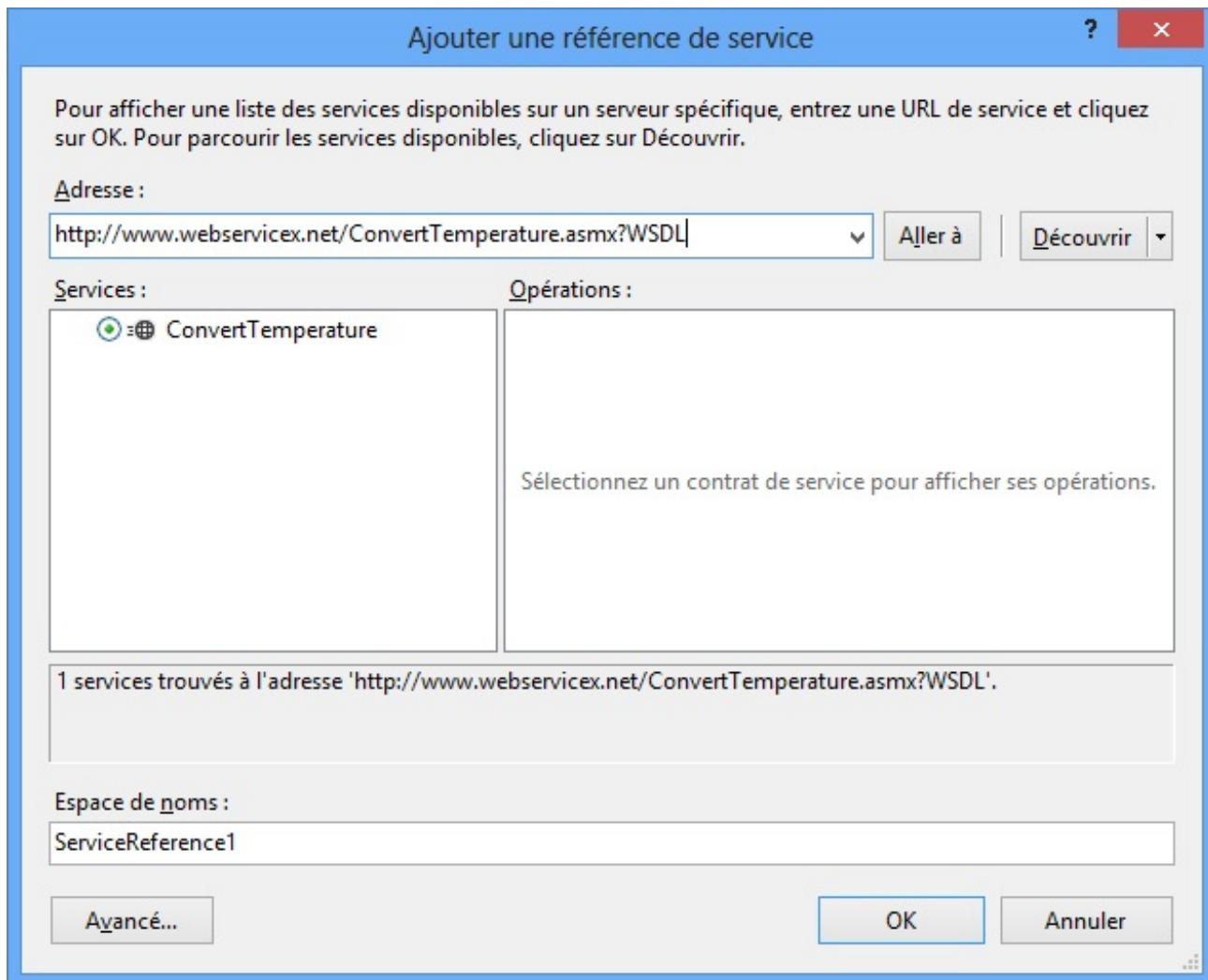
Eh oui, il faut une syntaxe définie, sinon on peut faire ce que l'on veut. C'est là qu'interviennent les organismes de standardisation. Ils ont défini plusieurs normes permettant de décrire le format des échanges. C'est le cas par exemple du protocole **SOAP** qui est basé sur du XML. Il est associé au **WSDL** qui permet de décrire le service web. Nous avons à notre disposition également les services web de type **REST** qui exposent les fonctionnalités comme des URL.

Pour illustrer ce fonctionnement, nous allons utiliser un service web gratuit qui permet de transformer des températures, par exemple de degrés Celsius en degré Fahrenheit. Ce service web est disponible à l'adresse suivante : <http://www.webservicex.net/ConvertTemperature.asmx>, et plus précisément, sa description est accessible sur <http://www.webservicex.net/ConvertTemperature.asmx?WSDL>.

Nous devons ensuite ajouter une référence web, pour cela faites un clic droit sur les références et ajoutez une référence de service, comme indiqué sur la figure suivante.



Ensuite, il faut saisir l'adresse du WSDL <http://www.webservicex.net/ConvertTemperature.asmx?WSDL> et cliquer sur *Aller à* (voir la figure suivante).



Remarquez que je laisse l'espace de noms à la valeur ServiceReference1, mais n'hésitez pas à le changer si besoin. Vous aurez alors besoin d'inclure cet espace de nom afin de pouvoir appeler le service web. Une fois validé, Visual Studio nous génère un proxy en se basant sur le WSDL du service web. Ce proxy va s'occuper d'encapsuler tout la logique d'appel du web service pour nous simplifier la tâche.

Remarque : un proxy est une ou plusieurs classes qui se placent entre deux autres pour faciliter ou surveiller leurs échanges. Le proxy est en fait un patron de conception que l'on peut décrire assez facilement avec un exemple du monde réel, dans le cas où deux personnes qui ne parlent pas la même langue vont recourir à un interprète pour les faire communiquer entre elles. Cet interprète, c'est le proxy.

Cela veut dire concrètement que Visual Studio travaille pour nous. À partir de l'URL de nos services web, il va analyser le type des données qui doivent être passées en paramètres ainsi que les données que l'on obtient en retour et générer des classes qui leurs correspondent. De même, il génère tout une classe qui encapsule les différents appels aux différentes méthodes du service web. C'est cette classe que l'on appelle un proxy car elle sert de point d'entrée pour tous les appels des méthodes du service web. Toutes ces classes ont été créées dans l'espace de nom que nous avons indiqué.

Nous pourrons alors simplement l'utiliser comme ceci :

Code : C#

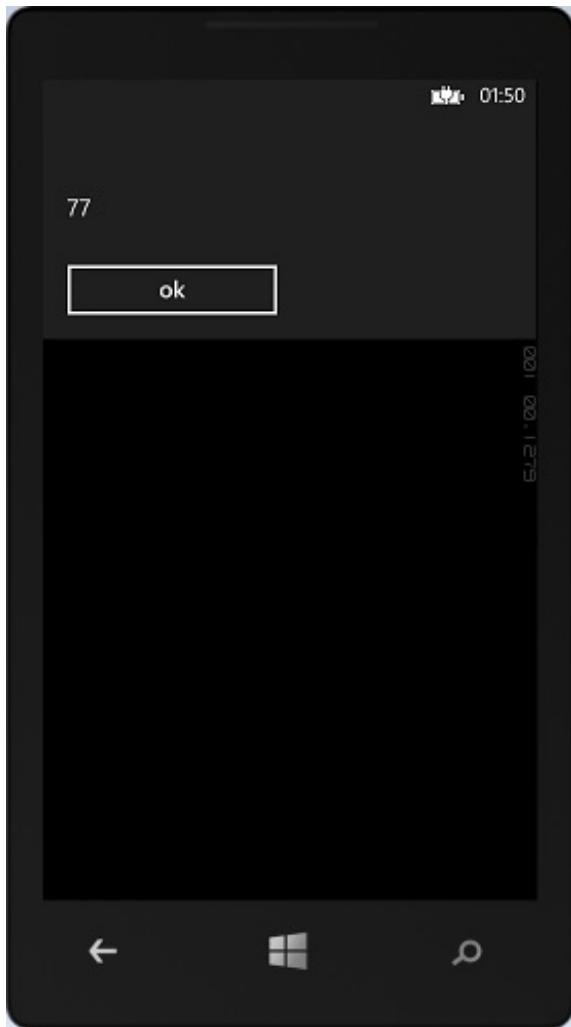
```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        TemperatureService.ConvertTemperatureSoapClient client = new
```

```
TemperatureService.ConvertTemperatureSoapClient();
    client.ConvertTempCompleted += client_ConvertTempCompleted;
    client.ConvertTempAsync(25,
TemperatureService.TemperatureUnit.degreeCelsius,
TemperatureService.TemperatureUnit.degreeFahrenheit);
}

private void client_ConvertTempCompleted(object sender,
TemperatureService.ConvertTempCompletedEventArgs e)
{
    if (e.Error == null)
    {
        MessageBox.Show(e.Result.ToString());
    }
}
```

Ce qui donnera la figure suivante.



Résultat de la conversion degré Celsius en degré Fahrenheit

Et voilà, 25 degré Celsius font 77 degré Fahrenheit !

Linq-To-Json

Parlons à présent un peu des services REST. Je ne présenterai pas comment faire un appel REST parce que vous savez déjà le faire, dans la mesure où il s'agit d'une simple requête HTTP. Par contre, ce qu'il est intéressant d'étudier ce sont les solutions pour interpréter le résultat d'un appel REST. De plus en plus, les services REST renvoient du JSON car c'est un format de description de données beaucoup moins verbeux que le XML.

.NET sait interpréter le JSON mais malheureusement l'assembly `System.Json.dll` n'est pas portée pour Windows Phone. Nous pouvons quand même l'utiliser mais c'est à nos risques et périls. Elle a été installée avec le SDK de Silverlight, chez moi à cet emplacement :

C:\Program Files\Microsoft SDKs\Silverlight\v4.0\Libraries\Client\System.Json.dll.

Cette assembly nous permet de faire du Linq To Json et d'avoir accès aux objets `JsonObject` et `JsonArray`. Lorsque vous allez référencer cette assembly, vous aurez le message d'avertissement suivant :



Fenêtre d'avertissement lors de la

référence à System.Json.dll

Vous pouvez cliquer sur oui pour continuer.

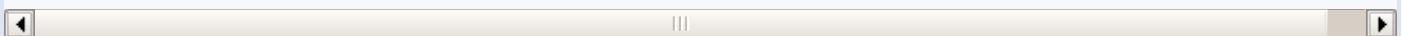
Nous allons illustrer le fonctionnement de ces objets à travers un appel REST qui consistera à réaliser une recherche avec le moteur de recherche Google. Par exemple, si je veux chercher la chaîne « openclassrooms » sur Google, je pourrai utiliser le service web REST suivant : <http://ajax.googleapis.com/ajax/services/search/web?q=openclassrooms>

Qui me renverra quelque chose comme :

Code : Autre

```
{  
    "responseData": {  
        "results": [  
            {  
                "GsearchResultClass": "GwebSearch",  
                "unescapeUrl": "http://fr.openclassrooms.com/",  
                "url": "http://fr.openclassrooms.com/",  
                "visibleUrl": "fr.openclassrooms.com",  
                "cacheUrl": "http://www.google.com/search?  
q\u003dcache:Wy4OFwwsrHMJ:fr.openclassrooms.com",  
                "title": "\u003cb\u003eOpenClassrooms\u003c/b\u003e, Le Site du Zéro  
Les cours les plus ouverts du Web",  
                [...]  
            },  
            {  
                "GsearchResultClass": "GwebSearch",  
                "unescapeUrl": "http://fr.openclassrooms.com/informatique/cours",  
                "url": "http://fr.openclassrooms.com/informatique/cours",  
                [...]  
            },  
            [...]  
        ],  
        "cursor": {  
            "resultCount": "119 000",  
            "pages": [  
                {  
                    "start": "0",  
                    "label": 1  
                },  
                [...]  
            ],  
            "estimatedResultCount": "119000",  
            "currentPageIndex": 0,  
            [...]  
            "searchResultTime": "0,21"  
        }  
    },  
}
```

```
        "responseDetails":null,  
        "responseStatus":200  
    }
```



Le format JSON est relativement compréhensible à l'œil nu, mais sa lecture fait un peu mal aux yeux. Nous pouvons quand même décrypter que le résultat contient une série de valeurs correspondant à la recherche.

Construisons une mini application qui effectuera le téléchargement des données, vous savez faire, on utilise la classe WebClient :

Code : C#

```
public partial class MainPage : PhoneApplicationPage  
{  
    public MainPage()  
    {  
        InitializeComponent();  
        WebClient client = new WebClient();  
        client.DownloadStringCompleted +=  
client_DownloadStringCompleted;  
        client.DownloadStringAsync(new  
Uri("http://ajax.googleapis.com/ajax/services/search/web?  
v=1.0&q=openclassrooms"));  
    }  
  
    private void client_DownloadStringCompleted(object sender,  
DownloadStringCompletedEventArgs e)  
    {  
        if (e.Error == null)  
        {  
            JsonObject json =  
(JsonObject)JsonObject.Parse(e.Result);  
        }  
    }  
}
```

Nous allons pouvoir obtenir un objet JsonObject grâce à la méthode statique Parse. Le JsonObject est en fait une espèce de dictionnaire où nous pouvons accéder à des valeurs à partir de leur clé. Par exemple, vous pouvez voir que le json récupéré possède la propriété responseData qui contient une sous propriété cursor, contenant elle-même la propriété resultCount fournissant le nombre de résultats de la requête. Nous pouvons y accéder de cette façon :

Code : C#

```
JsonObject json = (JsonObject)JsonObject.Parse(e.Result);  
string nombreResultat =  
json["responseData"]["cursor"]["resultCount"];
```

Dans ce cas, chaque JsonObject renvoie un nouvel JsonObject qui est lui-même toujours cette espèce de dictionnaire. En effet, responseData, cursor et resultCount sont des propriétés simples. Ce n'est pas le cas par contre de la propriété results qui est un tableau. On utilisera alors un JSONArray pour pouvoir le parcourir. Et c'est là que *Linq To Json* rentre en action, nous allons pouvoir requêter sur ce tableau. Par exemple :

Code : C#

```
List<string> resultats = new List<string>();  
JsonObject json = (JsonObject)JsonObject.Parse(e.Result);  
string nombreResultat =  
json["responseData"]["cursor"]["resultCount"];  
var listeResultat =
```

```
        from resultat in
    (JsonArray)(json["responseData"]["results"])
        where ((string)resultat["content"]).IndexOf("cours",
StringComparison.CurrentCultureIgnoreCase) >= 0
            select resultat;

foreach (var resultat in listeResultat)
{
    resultats.Add(resultat["titleNoFormatting"]);
}
```

Ici, je récupère les titres de chaque résultats dont le contenu contient le mot cours, sans faire attention à la casse. Ainsi, avec une ListBox :

Code : XML

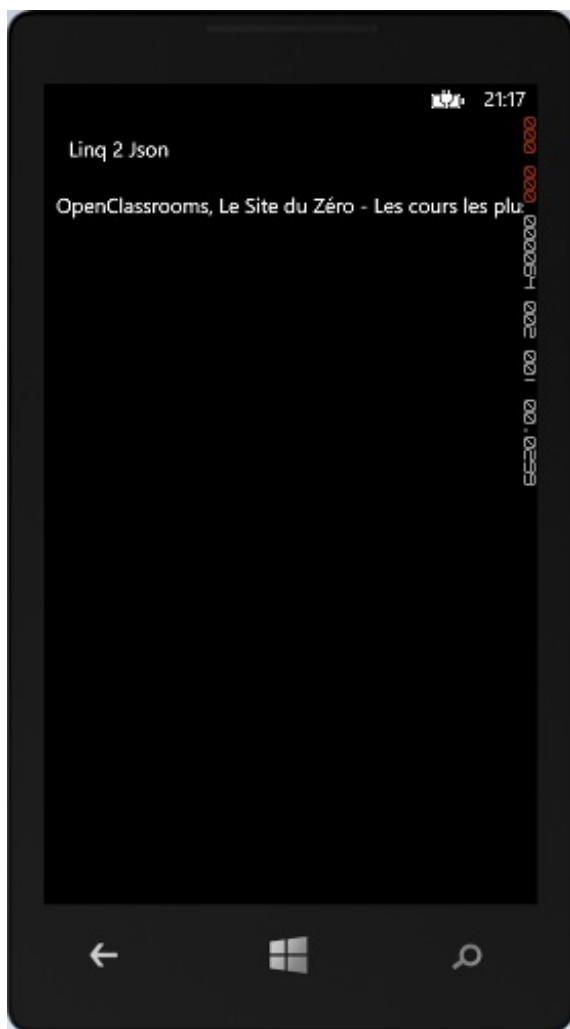
```
<ListBox x:Name="MaListBox" />
```

Je pourrais les afficher :

Code : C#

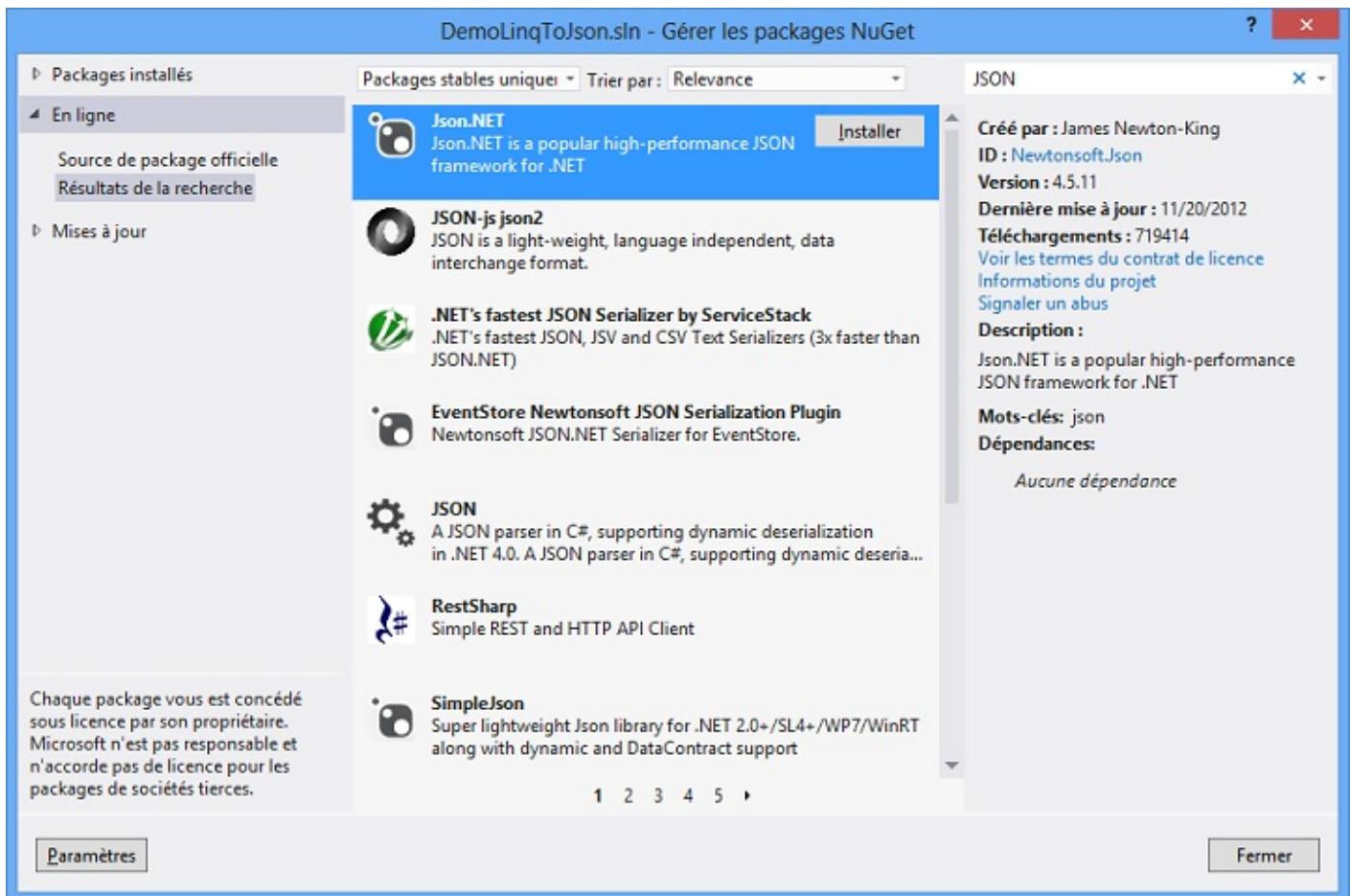
```
MaListBox.ItemsSource = resultats;
```

Et obtenir :



Résultat de la recherche google dans la ListBox

Vous aurez remarqué que traiter du JSON de cette façon n'est pas formidable. C'est plutôt lourd à mettre en place. Heureusement, il y a une autre solution plus intéressante et qui n'a pas besoin d'aller chercher l'assembly `System.Json.dll`. Il s'agit de transformer le JSON obtenu en objet. Cela peut se faire avec le [DataContractJsonSerializer](#) du framework .NET, qui se trouve dans l'assembly `System.ServiceModel.Web`. Celui-ci souffre cependant de quelques limitations. On pourra le remplacer avec la bibliothèque open-source `JSON.NET` que l'on peut télécharger sur [codeplex](#). Téléchargez la dernière version et référez l'assembly dans votre projet, la version Windows Phone bien sûr ou alors utilisez NuGet :



Installation de Json.NET via NuGet

La première chose à faire est de regarder la réponse renvoyée car nous allons avoir besoin de construire un ou plusieurs objets mappant ce résultat. Nous pouvons les construire à la main ou bien profiter du fait que certaines personnes ont réalisé des outils pour nous simplifier la vie. Allons par exemple sur le site <http://json2csharp.com/> où nous pouvons copier le résultat de la requête. Ce site nous génère les classes suivantes :

Code : C#

```
public class Page
{
    public int label { get; set; }
    public string start { get; set; }
}

public class Cursor
{
    public int currentPageIndex { get; set; }
    public string estimatedResultCount { get; set; }
    public string moreResultsUrl { get; set; }
    public List<Page> pages { get; set; }
    public string resultCount { get; set; }
    public string searchResultTime { get; set; }
}

public class Result
{
    public string GsearchResultClass { get; set; }
    public string cacheUrl { get; set; }
    public string content { get; set; }
    public string title { get; set; }
    public string titleNoFormatting { get; set; }
    public string unescapedUrl { get; set; }
    public string url { get; set; }
    public string visibleUrl { get; set; }
}
```

```
}

public class ResponseData
{
    public Cursor cursor { get; set; }
    public List<Result> results { get; set; }
}

public class RootObject
{
    public ResponseData responseData { get; set; }
    public object responseDetails { get; set; }
    public int responseStatus { get; set; }
}
```

que nous pouvons inclure dans notre projet.

Ces classes représentent exactement le résultat de la requête sous la forme de plusieurs objets.
Il ne reste plus qu'à faire un appel web comme on l'a vu :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    WebClient client = new WebClient();
    client.DownloadStringCompleted +=
    client_DownloadStringCompleted;
    client.DownloadStringAsync(new
    Uri("http://ajax.googleapis.com/ajax/services/search/web?
    v=1.0&q=openclassrooms"));
}

private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        RootObject resultat =
JsonConvert.DeserializeObject<RootObject>(e.Result);
        MaListBox.ItemsSource =
resultat.responseData.results.Select(r => r.titleNoFormatting);
    }
}
```

et à utiliser la classe `JsonConvert` pour déserialiser le JSON récupéré et le mettre dans les classes qui ont été générées.



« Srialisé » signifie que l'instance d'un objet subit une transformation afin de pouvoir être stockée au format texte, ou binaire. Inversement, la désrialisation permet de reconstruire une instance d'un objet à partir de ce texte ou de binaire.

Ensuite, j'extraie uniquement le titre pour l'afficher dans ma `ListBox`. Et le tour est joué !

Grâce à la bibliothèque `JSON.NET`, nous pouvons facilement interpréter des données JSON venant d'internet afin d'être efficace dans nos applications.

Remarquez que maintenant que nous avons des objets, nous pouvons utiliser les extensions `Linq` pour requêter sur les informations issues du JSON.

La bibliothèque de Syndication

Maintenant que nous savons récupérer des données depuis internet, pourquoi ne pas essayer d'en faire quelque chose d'un peu intéressant ? Comme un lecteur de flux RSS par exemple...

Vous connaissez sans doute tous [le RSS](#), c'est ce format qui nous permet de nous abonner à nos blogs favoris afin d'être avertis des nouveaux articles publiés.

Le flux RSS est produit sous forme de XML standardisé, contenant des informations sur le nom du site, les billets qui le

composent, le titre du billet, la description, etc.

Le site OpenClassrooms possède bien évidemment des flux RSS, comme le flux d'actualité de son blog disponible à cet emplacement : <http://www.simple-it.fr/blog/feed/>.

Si vous naviguez sur ce lien, vous pouvez facilement voir le titre du site, la description, ainsi que les différentes actualités ; et tout ça au format XML.

Prenons un autre site pour l'exemple, le blog de l'équipe Windows Phone. Le flux RSS est accessible via cette page : [http://blogs.windows.com/windows_phone \[...\] hone/rss.aspx](http://blogs.windows.com/windows_phone [...] hone/rss.aspx).

Vous savez déjà récupérer du XML grâce à la classe WebClient. Je vais en profiter pour vous communiquer une petite astuce dont je n'ai pas parlé dans les chapitres précédents. Il est possible de fournir un objet de contexte à la requête de téléchargement et ainsi pouvoir utiliser la même méthode pour l'événement de fin de téléchargement et identifier ainsi différentes requêtes. Il suffit de lui passer en paramètre de l'appel à la méthode DownloadStringAsync. Cet objet de contexte sera récupéré dans l'événement de fin de téléchargement, dans la propriété UserState de l'objet résultat :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private WebClient client;

    public MainPage()
    {
        InitializeComponent();

        client = new WebClient();
        client.DownloadStringCompleted += client_DownloadStringCompleted;
        client.DownloadStringAsync(new Uri("http://www.simple-
it.fr/blog/feed/"), "OC");
    }

    private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
    {
        if (e.Error == null)
        {
            if ((string)e.UserState == "OC")
            {
                // ce sont les données venant du flux du blog
                // OpenClassrooms

                // lancer le téléchargement suivant
                client.DownloadStringAsync(new
                    Uri("http://windowsteamblog.com/windows_phone/b/windowsphone/rss.aspx"),
                    "WP");
            }
            if ((string)e.UserState == "WP")
            {
                // ce sont les données venant du flux blog de l'équipe
                // windows phone
            }
        }
    }
}
```

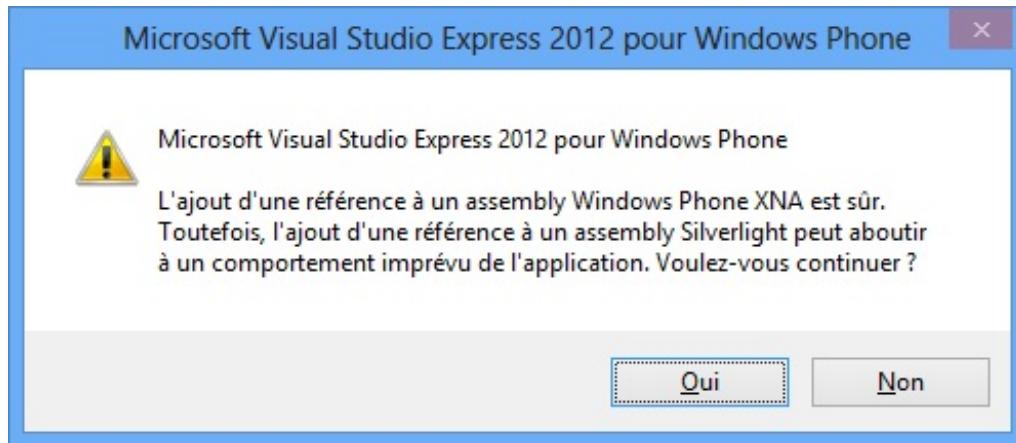
Ceci nous permettra d'utiliser la même méthode pour l'événement de fin de téléchargement.

Nous avons maintenant besoin d'interpréter les données du flux XML retourné. Étant donné que les flux RSS sont standards, il existe une bibliothèque Silverlight qui permet de travailler avec ce genre de flux. C'est la bibliothèque System.ServiceModel.Syndication.dll. Encore une fois, cette assembly n'a pas été écrite pour Windows Phone, mais elle est quand même utilisable avec nos applications Windows Phone.

Pour l'utiliser, nous devons ajouter une référence à celle-ci. Elle se trouve dans le répertoire suivant : C:\Program Files\Microsoft SDKs\Silverlight\v4.0\Libraries\Client.

Comme pour System.Json.dll, vous aurez une boîte de dialogue d'avertissement où vous pouvez cliquer sur Oui (voir la

figure suivante).



Fenêtre d'avertissement lors de la

référence à System.ServiceModel.Syndication.dll

Nous allons donc pouvoir charger l'objet de Syndication à partir du résultat, cet objet sera du type [SyndicationFeed](#). Pour commencer, je me rajoute une variable privée :

Code : C#

```
private List<SyndicationFeed> listeFlux;
```

que j'initialise dans le constructeur :

Code : C#

```
listeFlux = new List<SyndicationFeed>();
```

N'oubliez pas de rajouter le using qui va bien :

Code : C#

```
using System.ServiceModel.Syndication;
```

puis je consolide ma liste à partir du retour de l'appel web :

Code : C#

```
private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        if ((string)e.UserState == "OC")
        {
            // ce sont les données venant du flux du blog
OpenClassrooms
            AjouteFlux(e.Result);

            // lancer le téléchargement suivant
            client.DownloadStringAsync(new
Uri("http://windowsteamblog.com/windows_phone/b/windowsphone/rss.aspx"),
"WP");
    }
}
```

```

        }
        if ((string)e.UserState == "WP")
        {
            // ce sont les données venant du flux blog de l'équipe
            windows phone
            AjouteFlux(e.Result);
        }
    }

private void AjouteFlux(string flux)
{
    StringReader stringReader = new StringReader(flux);
    XmlReader xmlReader = XmlReader.Create(stringReader);
    SyndicationFeed feed = SyndicationFeed.Load(xmlReader);
    listeFlux.Add(feed);
}

```

Pour charger un flux de syndication, il suffit d'utiliser la méthode `Load` de la classe `SyndicationFeed` en lui passant un `XmlReader`, présent dans l'espace de nom `System.Xml`.



L'objet `StringReader` permet de lire un flux et de l'exploiter en tant que conteur de caractères. De même, l'objet `XmlReader` permet de traiter ce flux sous la forme de données XML.

Et voilà, nous avons créé une liste d'objet `SyndicationFeed` qui possède les éléments du flux RSS du blog OpenClassrooms ainsi que ceux du blog de l'équipe Windows Phone. Chaque objet `SyndicationFeed` contient une liste de billets, sous la forme d'objets `SyndicationItem`. Par exemple, la date de publication est accessible via la propriété `PublishDate` d'un `SyndicationItem`. Le titre du billet est accessible via la propriété `Title.Text`...

Nous pourrons par exemple afficher le titre de chaque post, ainsi que sa date dans une `ListBox` :

Code : XML

```

<ListBox x:Name="LaListBox">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <TextBlock Text="{Binding PublishDate}" />
                <TextBlock Text="{Binding Title.Text}" />
            </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>

```

Il nous faudra lier la propriété `ItemsSource` de la `ListBox` à, par exemple une `ObservableCollection` que nous construisons une fois le dernier flux reçu, et qui sera triée par ordre de date de publication de la plus récente à la plus ancienne :

Code : C#

```

if ((string)e.UserState == "WP")
{
    // ce sont les données venant du flux blog de l'équipe windows
    phone
    AjouteFlux(e.Result);
    ObservableCollection<SyndicationItem> listeBillets = new
    ObservableCollection<SyndicationItem>();
    foreach (SyndicationFeed flux in listeFlux)
    {
        foreach (SyndicationItem billet in flux.Items)
        {

```

```
        listeBillets.Add(billet);
    }
    LaListBox.ItemsSource = listeBillets.OrderByDescending(billet => billet.PublishDate);
}
```

Ce qui donnera le résultat présenté dans la figure suivante.



Affichage du flux RSS dans la ListBox

La présentation laisse à désirer, mais c'est fait exprès. Nous en restons là pour l'instant, mais ne vous inquiétez pas, vous allez y revenir bientôt.

Asynchronisme avancé

Bon, c'est très bien les méthodes asynchrones, mais c'est une gymnastique un peu compliquée. On s'abonne à un événement de fin de téléchargement puis on démarre le téléchargement et quand le téléchargement est terminé, on exploite le résultat dans une autre méthode perdant au passage le contexte de l'appel. Alors oui... il y a des astuces, comme celle que nous venons de voir... mais je vous dis pas les noeuds au cerveau lorsqu'il y a des appels dans tous les sens !

Bonne nouvelle, avec Windows Phone 8 il est possible de se simplifier grandement l'asynchronisme. En fait, c'est surtout le framework 4.5 qu'il faut remercier, mais peu importe, si vous créez une application pour Windows Phone 8, vous pourrez bénéficier de 2 formidables nouveaux petits mot-clés : `async` et `await`.

Certaines API de Windows Phone 8 ont été réécrites pour tirer parti de ces nouveaux mots clés, c'est le cas par exemple de certaines opérations sur les fichiers qui peuvent prendre du temps et que nous allons voir juste après. Ça aurait également pu être le cas pour les classes d'accès à Internet comme `WebClient` et `HttpWebRequest`, mais malheureusement celles-ci n'ont pas été réécrites.

Heureusement, nous pouvons écrire un wrapper pour bénéficier de ces éléments avec la classe `WebClient`. Ceci va nous permettre de nous simplifier grandement l'asynchronisme.



Je ne vais pas rentrer dans le détail des Task car il ne s'agit pas d'un cours sur le C# et au fond ce n'est pas vraiment ce qui nous intéresse ici.

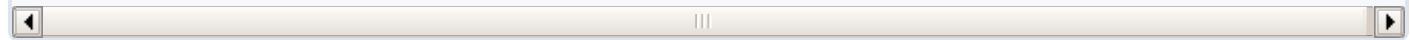
Ce qui va nous intéresser c'est la construction suivante, que nous avons vue au tout début de ce chapitre :

Code : C#

```
public MainPage()
{
    InitializeComponent();

    WebClient client = new WebClient();
    client.DownloadStringCompleted += client_DownloadStringCompleted;
    client.DownloadStringAsync(new
Uri("http://www.siteduzero.com/uploads/fr/ftp/windows_phone/script_nico.php"));
}

private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string texte = e.Result;
        MessageBox.Show(texte);
    }
    else
    {
        MessageBox.Show("Impossible de récupérer les données sur internet : " +
e.Error);
    }
}
```



Commençons par utiliser la classe `TaskCompletionSource` dans une méthode d'extension :

Code : C#

```
public static class Extensions
{
    public static Task<string> DownloadStringTaskAsync(this
WebClient webClient, Uri uri)
    {
        TaskCompletionSource<string> taskCompletionSource = new
TaskCompletionSource<string>();
        DownloadStringCompletedEventHandler downloadCompletedHandler =
null;
        downloadCompletedHandler = (s, e) =>
        {
            webClient.DownloadStringCompleted -=
downloadCompletedHandler;
            if (e.Error != null)
                taskCompletionSource.TrySetException(e.Error);
            else
                taskCompletionSource.TrySetResult(e.Result);
        };

        webClient.DownloadStringCompleted +=
downloadCompletedHandler;
        webClient.DownloadStringAsync(uri);

        return taskCompletionSource.Task;
    }
}
```

Le principe est d'encapsuler l'appel à `DownloadStringAsync` et de renvoyer un objet de type `Task<string>`, `string` étant le type de ce que l'on récupère en résultat de l'appel.

Ainsi, nous allons pouvoir remplacer l'appel du début par :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    LanceTelechargementAsync();
}

private async void LanceTelechargementAsync()
{
    WebClient client = new WebClient();
    string texte = await client.DownloadStringTaskAsync(new
Uri("http://www.siteduzero.com/uploads/fr/ftp/windows_phone/script_nico.php"));
    MessageBox.Show(texte);
}
```

La méthode `LanceTelechargementAsync` doit posséder le mot-clé `async` avant son type de retour pour indiquer qu'elle est asynchrone et doit également par convention avoir son nom qui se termine par `Async`. Nous appelons la méthode d'extension `DownloadStringTaskAsync` et nous attendons son résultat de manière asynchrone grâce au mot-clé `await`. Pas de méthode appelée une fois le téléchargement terminé... Juste un mot-clé qui nous permet d'attendre le résultat de manière asynchrone.

Plutôt simplifié comme construction, non ?

Allez, pour le plaisir, on se simplifie le téléchargement des flux RSS que l'on a fait juste au dessus ? Gardons toujours la même classe d'extension et écrivons désormais :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    LanceLeTelechargementAsync();
}

private async void LanceLeTelechargementAsync()
{
    listeFlux = new List<SyndicationFeed>();

    client = new WebClient();
    string rss = await client.DownloadStringTaskAsync(new
Uri("http://www.simple-it.fr/blog/feed/"));
    AjouteFlux(rss);
    rss = await client.DownloadStringTaskAsync(new
Uri("http://windowsteamblog.com/windows_phone/b/windowsphone/rss.aspx"));
    AjouteFlux(rss);

    ObservableCollection<SyndicationItem> listeBillets = new
ObservableCollection<SyndicationItem>();
    foreach (SyndicationFeed flux in listeFlux)
    {
        foreach (SyndicationItem billet in flux.Items)
        {
            listeBillets.Add(billet);
        }
    }
}
```

```
        }
        LaListBox.ItemsSource = listeBillets.OrderByDescending(billet =>
billet.PublishDate);
    }
```

C'est quand même bien plus clair !



Il n'y a plus de traitement d'erreur dans la construction précédente. Vous pourrez alors encadrer l'appel dans un `try/catch`.

Le répertoire local

Il n'est pas toujours possible d'accéder à internet pour récupérer des infos ou les mettre à jour. Nous avons encore à notre disposition un emplacement pour faire persister de l'information : à l'intérieur du téléphone.

On appelle cet emplacement le répertoire local (*local folder* en anglais ou anciennement *isolated storage*) dans la mesure où nous n'avons pas accès directement au système de fichiers, mais plutôt à un emplacement mémoire isolé dont nous pouvons ignorer le fonctionnement. Tout ce qu'il faut savoir c'est qu'il est possible d'y stocker des informations, comme du texte mais aussi des objets serialisés.

Il y a deux grandes façons d'utiliser le répertoire local. La plus simple est d'utiliser le dictionnaire `ApplicationSettings`. On peut y ranger des objets qui seront associés à une chaîne de caractères. Par exemple, en imaginant que l'on crée une application où l'on demande le nom de notre utilisateur, il pourra être judicieux d'éviter de le redemander à chaque ouverture de l'application... Pour cela, nous pouvons le faire persister dans le répertoire local. On utilisera alors :

Code : C#

```
IsolatedStorageSettings.ApplicationSettings["prenom"] = "Nicolas";
```

Nb : pour utiliser la classe `IsolatedStorageSettings`, nous devons inclure :

Code : C#

```
using System.IO.IsolatedStorage;
```

J'associe ici la chaîne de caractères "prenom" à la chaîne de caractères "Nicolas".

Au prochain démarrage de l'application, on pourra vérifier si le prénom existe déjà en tentant d'accéder à sa clé "prenom". S'il y a quelque chose d'associé à cette clé, on pourra le récupérer pour éviter de demander à re-saisir le prénom de l'utilisateur :

Code : C#

```
if (IsolatedStorageSettings.ApplicationSettings.Contains("prenom"))
    prenom =
(string)IsolatedStorageSettings.ApplicationSettings["prenom"];
```

Nous pouvons mettre des objets complexes dans le répertoire local, pas seulement des chaînes de caractères :

Code : C#

```
public class Utilisateur
{
    public int Age { get; set; }
    public string Prenom { get; set; }
}
```

```
Utilisateur nicolas = new Utilisateur { Age = 30, Prenom = "Nicolas"
};
IsolatedStorageSettings.ApplicationSettings["utilisateur"] =
nicolas;
```

Et de la même façon, on pourra récupérer cette valeur très facilement :

Code : C#

```
if (IsolatedStorageSettings.ApplicationSettings.Contains("utilisateur"))
    nicolas =
(Utilisateur)IsolatedStorageSettings.ApplicationSettings["utilisateur"];
```

Attention, le répertoire local fonctionne très bien avec l'émulateur à la seule condition de ne pas fermer brutalement votre application, en arrêtant le débogueur par exemple, en fermant l'émulateur ou en ayant une exception. Pour que le répertoire local persiste d'une utilisation à l'autre, il faut que vous conserviez l'émulateur ouvert et que vous terminiez l'application en cliquant sur le bouton retour, jusqu'à ce qu'il n'y ait plus rien dans la pile des pages et que l'application se termine. Sinon, vous devrez appeler la méthode de sauvegarde explicite suivante :

Code : C#

```
IsolatedStorageSettings.ApplicationSettings.Save();
```

Le stockage d'objets dans le répertoire local est quand même très pratique. Vous vous servirez très souvent de ce fonctionnement simple et efficace. Parfois vous aurez peut-être besoin d'un peu plus de contrôle sur ce que vous voulez stocker. À ce moment-là, on peut se servir du répertoire local comme d'un flux classique, comme lorsque l'on souhaite enregistrer des données dans un fichier.

Regardons l'enregistrement :

Code : C#

```
using (IsolatedStorageFileStream stream = new
IsolatedStorageFileStream("sauvegarde.txt", FileMode.Create,
IsolatedStorageFile.GetUserStoreForApplication()))
{
    using (StreamWriter writer = new StreamWriter(stream))
    {
        writer.WriteLine(30);
        writer.WriteLine("Nicolas");
    }
}
```

Puis la lecture :

Code : C#

```
using (IsolatedStorageFileStream stream = new
IsolatedStorageFileStream("sauvegarde.txt", FileMode.Open,
IsolatedStorageFile.GetUserStoreForApplication()))
{
    using (StreamReader reader = new StreamReader(stream))
    {
        int age = Convert.ToInt32(reader.ReadLine());
    }
}
```

```
        string prenom = reader.ReadLine();  
    }  
}
```

Cela ressemble beaucoup à des opérations d'écriture et de lecture dans un fichier classique... Enfin, si besoin, on pourra supprimer le fichier :

Code : C#

```
IsolatedStorageFile racine =  
IsolatedStorageFile.GetUserStoreForApplication();  
if (racine.FileExists("sauvegarde.txt"))  
    racine.DeleteFile("sauvegarde.txt");
```

L'objet `IsolatedStorageFile` que l'on récupère avec la méthode `GetUserStoreForApplication` permet de créer un fichier, un répertoire, de le supprimer, etc. Bref, quasiment tout ce que permet un système de fichier classique.

À noter que cette technique est tout à fait appropriée pour stocker des images par exemple, pour éviter d'avoir à les redécharger à chaque fois.



Remarquez qu'au contraire d'une application Silverlight s'exécutant dans un navigateur, le répertoire local dans une application Windows Phone n'a pas de limite de taille. Tant que le téléphone dispose encore de mémoire, celle-ci est toute à vous.

Enfin, avec Windows Phone 8 nous pouvons tirer parti des nouvelles API de stockages asynchrones. Celles-ci nous permettent de ne pas bloquer le thread courant lorsque nous avons besoin de lire et écrire potentiellement de grosses données dans le répertoire local. Voici par exemple comment écrire des données dans le répertoire local de manière asynchrone :

Code : C#

```
private async void SauvegardeAsync()  
{  
    IStorageFolder applicationFolder =  
    ApplicationData.Current.LocalFolder;  
  
    IStorageFile storageFile = await  
    applicationFolder.CreateFileAsync("sauvegarde.txt",  
    CreationCollisionOption.ReplaceExisting);  
    using (Stream stream = await  
    storageFile.OpenStreamForWriteAsync())  
    {  
        byte[] bytes = Encoding.UTF8.GetBytes("30;Nicolas");  
        await stream.WriteAsync(bytes, 0, bytes.Length);  
    }  
}
```

Et voici comment lire les données précédemment sauvegardées :

Code : C#

```
private async void LectureAsync()  
{  
    IStorageFolder applicationFolder =  
    ApplicationData.Current.LocalFolder;  
  
    IStorageFile storageFile = await  
    applicationFolder.GetFileAsync("sauvegarde.txt");
```

```
IRandomAccessStream accessStream = await  
storageFile.OpenReadAsync();  
  
using (Stream stream =  
accessStream.AsStreamForRead((int)accessStream.Size))  
{  
    byte[] bytes = new byte[stream.Length];  
    await stream.ReadAsync(bytes, 0, bytes.Length);  
    string chaine = Encoding.UTF8.GetString(bytes, 0,  
bytes.Length);  
    string[] tableau = chaine.Split(';');  
    int age = int.Parse(tableau[0]);  
    string prenom = tableau[1];  
}  
}
```

Ici, j'ai choisi de stocker mes données sous la forme de texte séparés par des points virgules, mais libre à vous de spécifier le format de fichier de votre choix. Nous remarquons l'utilisation des mots-clés `async` et `await`, dignes témoins de l'asynchronisme de ces méthodes.

- Le XAML/C# pour Windows Phone dispose de toute une gamme de solutions pour utiliser des données depuis internet ou en local sur le téléphone.
- Les classes `HttpRequest` et `WebClient` permettent de faire des requêtes sur le protocole HTTP.
- On utilise la bibliothèque open-source `JSON.NET` pour interpréter les données au format JSON.
- Il est très facile d'exploiter des flux RSS grâce à la bibliothèque de syndication.
- L'asynchronisme est grandement facilité dans Windows Phone 8 grâce aux mots-clés `async` et `await`.
- Le répertoire local correspond à un emplacement sur le téléphone, dédié à notre application, où l'on peut faire persister de l'information entre les divers lancements d'une application.

Ça y est, vous savez traiter les données dans vos applications Windows Phone.

Nous avons commencé par étudier le système de navigation ainsi que la `ListBox`. C'est un contrôle très puissant qui va vous servir énormément dans vos applications, dès que vous aurez besoin d'afficher des listes de quoi que ce soit. Son mécanisme de modèle est particulièrement efficace pour personnaliser l'affichage des éléments.

Nous avons ensuite vu la liaison de données. Il s'agit d'un chapitre très important que vous devez absolument comprendre pour tirer parti du meilleur des applications utilisant le XAML. N'hésitez pas à le relire et à vous entraîner. Il peut paraître obscur et difficile à maîtriser. Je vous rassure, vous ferez plein d'erreurs de liaison de données, c'est tout à fait normal. N'oubliez pas de vérifier de temps en temps dans la fenêtre de sortie s'il n'y a pas un message d'erreur, témoin qu'une liaison de données n'a pas fonctionné. Cela vous sera très utile.

MVVM a ensuite pointé le bout de son nez. N'hésitez pas à revenir plus tard sur ce chapitre si vous n'avez pas encore bien compris son intérêt. Ce n'est qu'après un peu de pratique que vous pourrez saisir toute la plus-value d'un tel patron de conception.

Enfin, nous avons vu différentes solutions pour traiter les données, qu'elles viennent d'internet ou bien simplement de l'intérieur du téléphone.

Vous verrez que cette partie est une partie clé dont vous aurez régulièrement besoin. C'est le socle de toute application de gestion réalisée avec XAML/C# pour Windows Phone. N'hésitez pas à vous entraîner et à y revenir si besoin.

Partie 3 : Une bibliothèque de contrôles

Le XAML pour Windows Phone, c'est également tout une série de contrôles qui nous facilitent la vie et qui sont particulièrement adaptés à un téléphone.

Dans cette partie nous allons découvrir différents contrôles dont vous allez forcément avoir besoin à un moment où un autre pour afficher des données d'une façon particulière ou bien offrir une expérience riche à vos utilisateurs.

Inutile de réinventer la roue et découvrons à présent ces contrôles, qui en plus ont été élaborés avec un look Modern UI.

Panorama et Pivot

Nous avons vu dans la partie précédente que nous pouvions naviguer entre les pages, c'est bien ! Mais sachez que nous pouvons également naviguer entre les données. C'est encore mieux 😊

C'est là qu'interviennent deux contrôles très utiles qui permettent de naviguer naturellement entre des données : le contrôle Panorama et le contrôle Pivot.

Le contrôle Panorama sert en général à voir un petit bout d'un plus gros écran, qui ne rentre pas dans l'écran du téléphone. Le principe est qu'on peut mettre beaucoup d'informations sur une grosse page et la mécanique du contrôle Panorama incite l'utilisateur à se déplacer avec le doigt sur le reste du plus gros écran.

Le contrôle Pivot quant à lui permet plutôt de voir la même donnée sur plusieurs pages. La navigation entre les pages se fait en faisant glisser le doigt, comme si l'on tournait une page. Par exemple pour une application météo, la première page permet d'afficher la météo du jour, la page suivante permet d'afficher la météo de demain, etc.

Découvrons à présent ces deux contrôles.

Panorama

Le panorama est donc un contrôle qui sert à voir un petit bout d'un plus gros écran dont la taille dépasse celle de l'écran du téléphone. On l'illustre souvent avec une image de ce genre (voir la figure suivante).

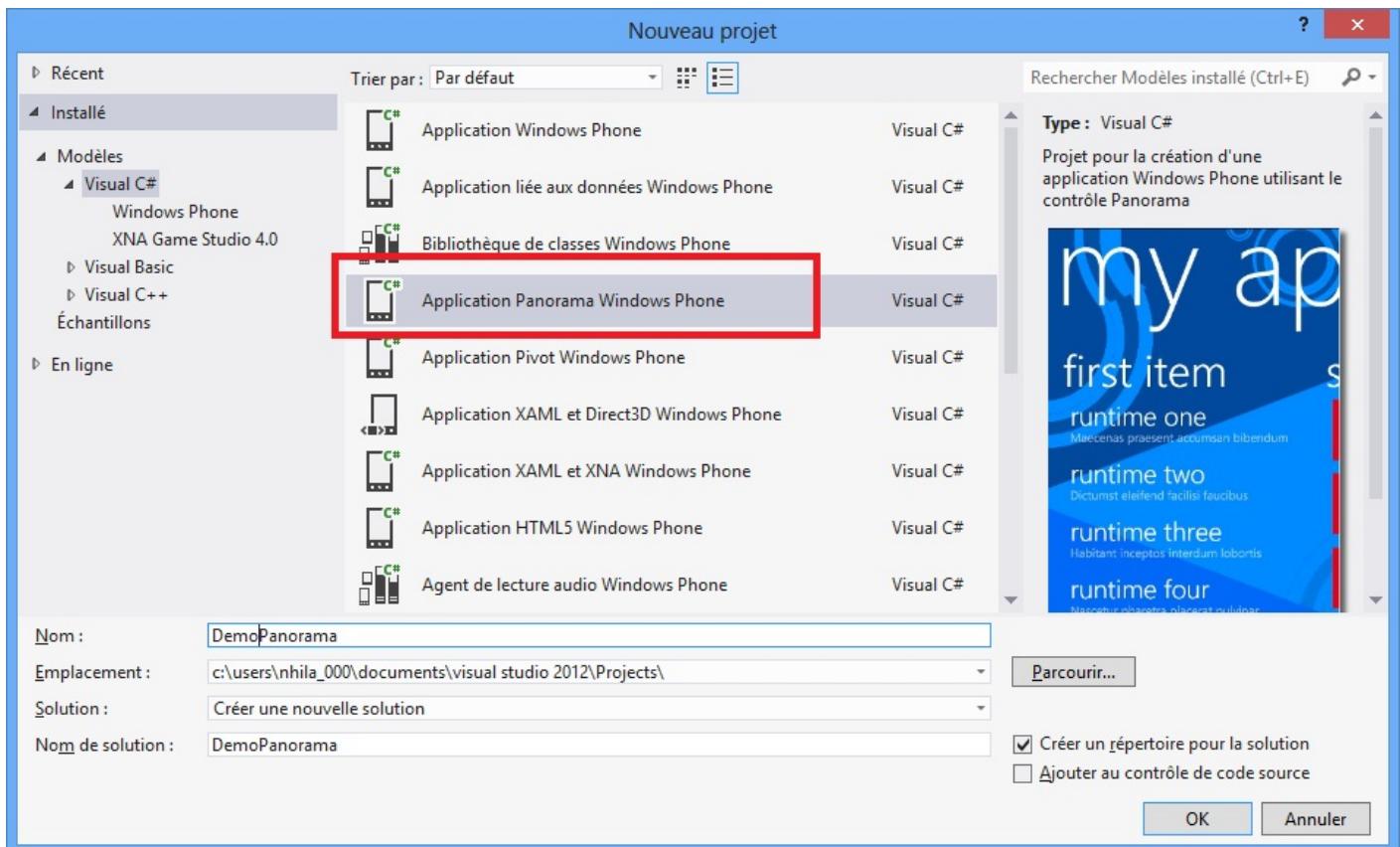


Représentation du contrôle Panorama

Vous vous rappelez l'introduction du cours et le passage sur les hubs ? Ce contrôle est exactement le même. Nous pouvons l'intégrer dans nos applications et tirer parti de son élégance et de ses fonctionnalités.

Pour découvrir le panorama, le plus simple est de créer un nouveau projet. Vous avez sûrement constaté que Visual Studio nous proposait de créer différents modèles de projet, dont un projet s'appelant « Application Panorama Windows Phone » et un autre

s'appelant « Application Pivot Windows Phone ». Choisissons le projet « Application Panorama Windows Phone », ainsi qu'indiqué à la figure suivante.



Création d'un projet Panorama

Si nous démarrons immédiatement l'application, nous pouvons voir qu'elle contient un panorama existant. Wahou... bon ok, passons sur la relative traduction des divers éléments. 😊

Ce qu'il faut remarquer ici, c'est qu'il est possible de faire glisser l'écran en cours de gauche à droite, affichant trois éléments en tout, et en boucle, sachant que le troisième élément occupe plus d'espace qu'un écran (voir la figure suivante).



La panorama du projet exemple, composé de 3 écrans

Il faut également remarquer que l'affichage de chaque écran incite l'utilisateur à aller voir ce qu'il y a à droite. En effet, on peut voir que le titre n'est pas complet. Pareil pour le carré jaune, on se doute qu'il doit y avoir quelque chose à côté... Bref, tout est

fait pour donner envie d'aller voir ce qu'il y a plus loin. C'est le principe du panorama.

Voyons à présent le XAML qui a été généré pour obtenir cet écran :

Code : XML

```
<phone:Panorama Title="mon application">
    <phone:Panorama.Background>
        <ImageBrush
            ImageSource="/DemoPanorama;component/Assets/PanoramaBackground.png"/>
    </phone:Panorama.Background>

    <!--Élément un de panorama-->
    <phone:PanoramaItem Header="first item">
        <!--Liste simple trait avec habillage du texte-->
        <phone:LongListSelector Margin="0,0,-22,0"
            ItemsSource="{Binding Items}">
            [...]
        </phone:LongListSelector>
    </phone:PanoramaItem>

    <!--Élément deux de panorama-->
    <phone:PanoramaItem>
        <!--Liste double trait avec espace réservé pour une image et
            habillage du texte utilisant un en-tête flottant qui défile avec le
            contenu-->
        <phone:LongListSelector Margin="0,-38,-22,2"
            ItemsSource="{Binding Items}">
            [...]
        </phone:LongListSelector>
    </phone:PanoramaItem>

    <!--Élément trois de panorama-->
    <phone:PanoramaItem Header="third item"
        Orientation="Horizontal">
        <!--Double largeur de panorama avec espaces réservés pour
            grandes images-->
        <Grid>
            [...]
        </Grid>
    </phone:PanoramaItem>
</phone:Panorama>
```

Ce qu'on peut constater déjà c'est qu'il est composé de trois parties qui sont toutes les trois des PanoramaItem. Un PanoramaItem correspond donc à une « vue » de la totalité du Panorama. La navigation se passe entre ces trois éléments. Nous pouvons d'ailleurs voir dans le designer le rendu du premier PanoramaItem. Vous pouvez également voir le second en allant vous positionner dans le XAML au niveau du second PanoramaItem, et de même pour le troisième. Plutôt pas mal, le rendu est assez fidèle.

Nous pouvons également constater que le contrôle Panorama est défini dans un espace de noms différent de ceux que nous avons déjà utilisés, on voit notamment qu'il est préfixé par phone qui correspond à :

Code : XML

```
xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
```

Ce contrôle se situe donc dans l'espace de noms Microsoft.Phone.Controls et dans l'assembly Microsoft.Phone.



Pour les utilisateurs du SDK pour Windows Phone 7, le contrôle panorama se situe dans l'assembly Microsoft.Phone.Controls et doit être explicitement référencée.

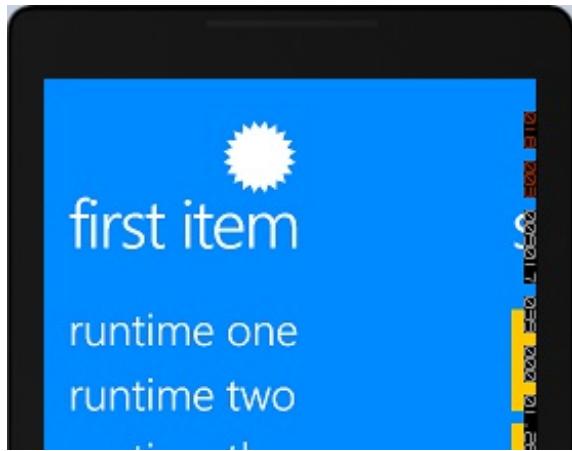
Le panorama possède un titre qui est affiché tout en haut du contrôle, ici, en haut de la page. On le remplit via la propriété `Title`. Vous pouvez d'ailleurs constater que ce titre n'est pas affiché en entier et que cela nous incite encore à aller voir plus à droite s'il n'y a pas autre chose. Ce titre est une propriété de contenu que nous pouvons remplacer par n'importe quoi, comme avec le bouton. À titre d'exemple, remplacez la propriété `Title` par :

Code : XML

```
<phone:Panorama>
    <phone:Panorama.Title>
        <Image Source="/Assets/ApplicationIcon.png" Margin="150 60 0
-30" />
    </phone:Panorama.Title>
    ...

```

Vous pouvez voir le résultat à la figure suivante.



Le titre est un contrôle de contenu

De la même façon, vous pouvez mettre un fond d'écran au panorama via la propriété `Background`. Notez que l'image doit absolument avoir son action de génération à `Resource`, sinon elle risque de ne pas apparaître immédiatement et d'être chargée de manière asynchrone.

Chaque élément du panorama a un titre, représenté par la propriété `Header`. Dans le deuxième, on peut remarquer que le titre commence par un « s » et qu'il dépasse du premier élément. Tout ceci est fait automatiquement sans que l'on ait à faire quoi que ce soit de supplémentaire.

Nous pouvons créer autant de `PanoramaItem` que nous le voulons et y mettre ce que nous voulons. Ici, il a été mis des listes de type `LongListSelector`, et dans le troisième élément une grille mais cela pourrait être n'importe quoi d'autre vu que le `Panorama` fait office de conteneur.

Soyez vigilant quant à l'utilisation du contrôle `Panorama`. Il doit être utilisé à des emplacements judicieux afin de ne pas perturber l'utilisateur. Bien souvent, il est utilisé comme page d'accueil d'une application.

Vous vous doutez bien qu'on peut faire beaucoup de choses avec ce panorama. Il est par exemple possible de s'abonner à l'événement de changement de `PanoramaItem`, ou se positionner directement sur un élément précis du panorama. Illustrons ceci avec le XAML suivant :

Code : XML

```
<phone:Panorama x:Name="MonPanorama" Title="Mes tâches"
    Loaded="Panorama_Loaded"
    SelectionChanged="Panorama_SelectionChanged">
    <phone:PanoramaItem Header="Accueil">
        <StackPanel>
            <TextBlock Text="Blablabla" HorizontalAlignment="Center"
/>
        <Button Content="Allez à aujourd'hui" Tap="Button_Tap"
Margin="0 50 0 0" />
    
```

```

        </StackPanel>
    </phone:PanoramaItem>
    <phone:PanoramaItem Header="Aujourd'hui">
        <ListBox>
            <ListBoxItem>Tondre la pelouse</ListBoxItem>
            <ListBoxItem>Arroser les plantes</ListBoxItem>
        </ListBox>
    </phone:PanoramaItem>
    <phone:PanoramaItem Header="Demain">
        <StackPanel>
            <TextBlock Text="Passer l'aspirateur" Margin="30 50 0
60" />
            <TextBlock Text="Laver la voiture" Margin="30 50 0 60"
/>
        </StackPanel>
    </phone:PanoramaItem>
</phone:Panorama>

```

Mon panorama contient trois éléments, un accueil, des tâches pour aujourd’hui et des tâches pour demain. Je me suis abonné à l’événement de chargement du panorama ainsi qu’à l’événement de changement de sélection. Ici, cela fonctionne un peu comme une `ListBox`. Notons également que l’écran d’accueil possède un bouton avec un événement de clic.

Passons au code-behind à présent :

Code : C#

```

public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void Panorama_Loaded(object sender, RoutedEventArgs e)
    {
        if (IsolatedStorageSettings.ApplicationSettings.Contains("PageCourante"))
        {
            MonPanorama.DefaultItem =
MonPanorama.Items[(int)IsolatedStorageSettings.ApplicationSettings["PageCourante"]]
        }
    }

    private void Button_Tap(object sender, System.Windows.Input.GestureEventArgs e)
    {
        MonPanorama.DefaultItem = MonPanorama.Items[1];
    }

    private void Panorama_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
        IsolatedStorageSettings.ApplicationSettings["PageCourante"] =
MonPanorama.SelectedIndex;
    }
}

```

La méthode `Panorama_SelectionChanged` est appelée à chaque changement de sélection. Dans cette méthode, je stocke dans le répertoire local l’index de la page en cours, obtenu comme pour la `ListBox` avec la propriété `SelectedIndex`. Ce qui me permet, au chargement du panorama, de me repositionner sur la dernière page visitée s’il y en a une. Cela se fait grâce à la propriété `DefaultItem` que je renseigne avec le `PanoramaItem` trouvé à l’indice de la propriété `Items`, indice qui est celui stocké dans le répertoire local. De la même façon, je peux me positionner sur un `PanoramaItem` choisi lorsque je clique sur le bouton.

Même si c’est un peu plus rare, il est possible d’utiliser le binding avec le contrôle `Panorama`. Reproduisons plus ou moins notre exemple précédent en utilisant un contexte de données (je retire la page accueil et le bouton, ce sera plus simple). Tout

d'abord le XAML :

Code : XML

```
<phone:Panorama x:Name="MonPanorama" Title="Mes tâches"
    Loaded="Panorama_Loaded"
    SelectionChanged="Panorama_SelectionChanged" ItemsSource="{Binding
    ListeEcrans}">
    <phone:Panorama.HeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Titre}" />
        </DataTemplate>
    </phone:Panorama.HeaderTemplate>
    <phone:Panorama.ItemTemplate>
        <DataTemplate>
            <ListBox ItemsSource="{Binding ListeDesTaches}">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding}" Margin="0 20 0
0" />
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>
        </DataTemplate>
    </controls:Panorama.ItemTemplate>
</controls:Panorama>
```

Ici, c'est comme pour la `ListBox`. Le contrôle `Panorama` possède aussi des modèles, que nous pouvons utiliser. Il y a le modèle `HeaderTemplate` qui nous permet de définir un titre et le modèle `ItemTemplate` qui nous permet de gérer le contenu. Le contrôle `Panorama` a sa propriété `ItemsSource` qui est liée à la propriété `ListeEcrans` que nous retrouvons dans le code-behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage, INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

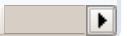
    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private List<Ecran> _listeEcrans;
    public List<Ecran> ListeEcrans
    {
        get { return _listeEcrans; }
        set { NotifyPropertyChanged(ref _listeEcrans, value); }
    }

    public MainPage()
```

```
{  
    InitializeComponent();  
  
    ListeEcrans = new List<Ecran>  
    {  
        new Ecran  
        {  
            Titre = "Aujourd'hui",  
            ListeDesTaches = new List<string> { "Tondre la pelouse",  
"Arroser les plantes"}  
        },  
        new Ecran  
        {  
            Titre = "Demain",  
            ListeDesTaches = new List<string> { "Passer l'aspirateur",  
"Laver la voiture"}  
        }  
    };  
  
    DataContext = this;  
}  
  
private void Panorama_Loaded(object sender, RoutedEventArgs e)  
{  
    if (IsolatedStorageSettings.ApplicationSettings.Contains("PageCourante"))  
    {  
        MonPanorama.DefaultItem =  
MonPanorama.Items[(int)IsolatedStorageSettings.ApplicationSettings["PageCourante"]]  
    }  
}  
  
private void Panorama_SelectionChanged(object sender, SelectionChangedEventArgs e)  
{  
    IsolatedStorageSettings.ApplicationSettings["PageCourante"] =  
MonPanorama.SelectedIndex;  
}
```



Avec la classe Ecran suivante :

Code : C#

```
public class Ecran  
{  
    public string Titre { get; set; }  
    public List<string> ListeDesTaches { get; set; }  
}
```

Et le tour est joué. Vous n'avez plus qu'à démarrer l'application pour obtenir ce résultat (voir la figure suivante).

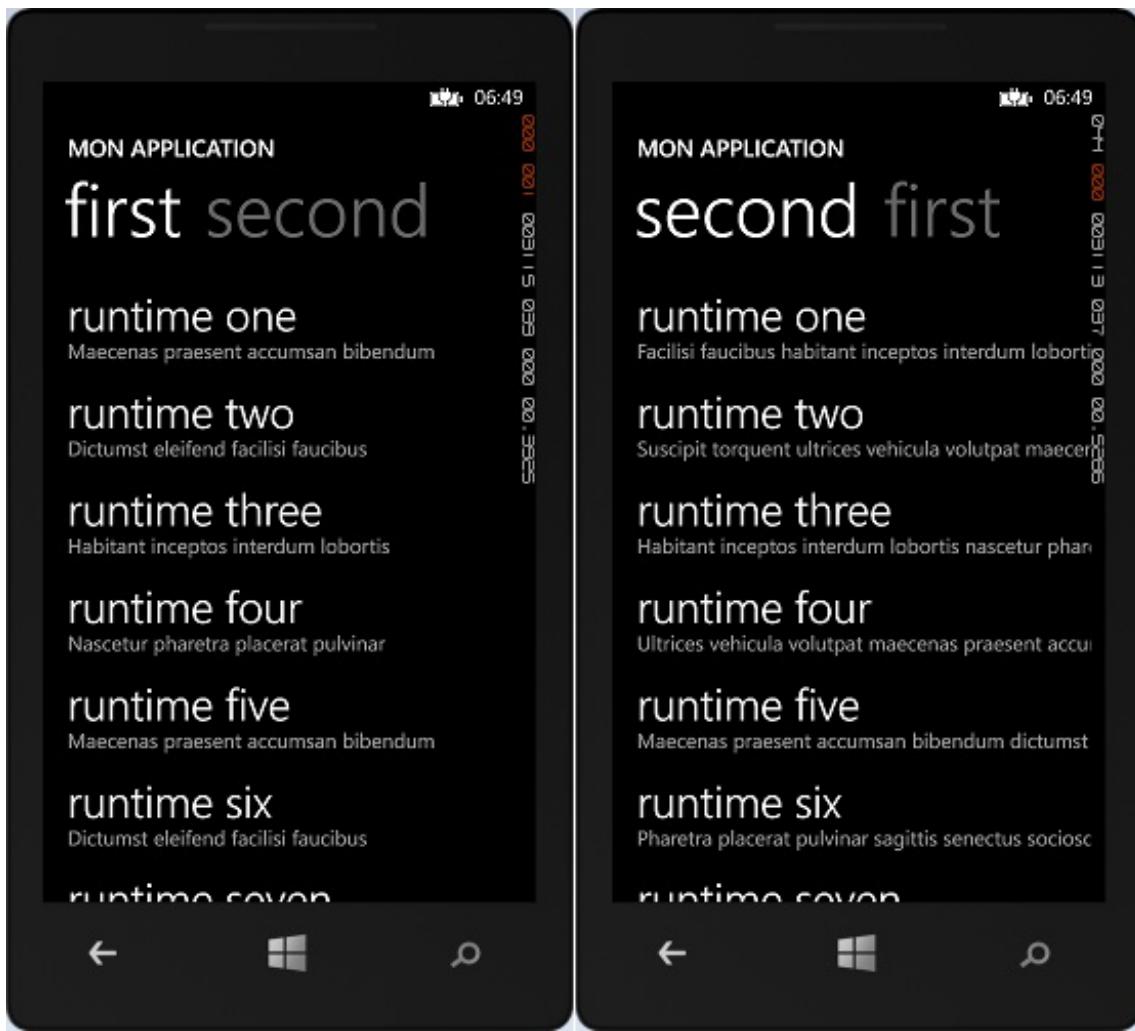


Binding

Pivot

Passons maintenant à l'autre contrôle très pratique, le **Pivot**, qui est un peu le petit frère du panorama. Il permet plutôt de voir la même donnée sur plusieurs pages. La navigation entre les pages se fait en faisant glisser le doigt, comme si l'on tournait une page. On pourrait le comparer à un contrôle de gestion d'onglets. On passe à l'onglet suivant en faisant glisser son doigt...

Voyons à présent comme fonctionne **le contrôle Pivot**. Pour cela, créez le deuxième type de projet que nous avons vu, à savoir « Application Pivot Windows Phone » et démarrons l'application exemple (voir la figure suivante). On constate qu'on peut également naviguer en faisant glisser la page sur la droite ou sur la gauche avec le doigt (ou la souris 😊).



Rendu du projet Pivot

exemple

Ici, visuellement, il y a seulement le titre des pages qui nous renseigne sur la présence d'un autre élément. Voyons à présent le code XAML :

Code : XML

```
<phone:Pivot Title="MON APPLICATION">
    <!--Élément un de tableau croisé dynamique-->
    <phone:PivotItem Header="first">
        <!--Liste double trait avec habillage du texte-->
        <phone:LongListSelector Margin="0,0,-12,0"
            ItemsSource="{Binding Items}">
            [... code supprimé pour plus de clarté...]
        </phone:LongListSelector>
    </phone:PivotItem>

    <!--Élément deux de tableau croisé dynamique-->
    <phone:PivotItem Header="second">
        <!--Liste double trait, aucun habillage du texte-->
        <phone:LongListSelector Margin="0,0,-12,0"
            ItemsSource="{Binding Items}">
            [... code supprimé pour plus de clarté...]
        </phone:LongListSelector>
    </phone:PivotItem>
</phone:Pivot>
```

Ici, le principe est le même que pour le Panorama. Le contrôle Pivot est composé de deux `PivotItem`, chacun faisant office de container. Dedans il y a un `LongListSelector` mais tout autre contrôle y trouve sa place. Encore une fois, c'est la propriété

Header qui va permettre de donner un titre à la page.

Vous pouvez également voir dans le designer les différents rendus des PivotItem en vous positionnant dans le XAML à leurs niveaux.

Tout comme pour le panorama, vous pouvez allégrement modifier les différentes propriétés, Title, Background... afin de personnaliser ce contrôle. De même, il possède des événements bien pratiques pour être notifié d'un changement de vue et également de quoi se positionner sur celle que l'on veut.

Il est également possible d'utiliser le binding avec ce contrôle, et c'est d'ailleurs ce que vous aurez tendance à souvent faire. Reprenons l'exemple de la liste des tâches qui est particulièrement adapté au contrôle Pivot et améliorons cet exemple. Voici dans un premier temps le XAML :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <phone:Pivot Title="Mes tâches" SelectedIndex="{Binding Index,
Mode=TwoWay}" Loaded="Pivot_Loaded" ItemsSource="{Binding
ListeEcrans}">
        <phone:Pivot.HeaderTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding Titre}" />
            </DataTemplate>
        </phone:Pivot.HeaderTemplate>
        <phone:Pivot.ItemTemplate>
            <DataTemplate>
                <ListBox ItemsSource="{Binding ListeDesTaches}">
                    <ListBox.ItemTemplate>
                        <DataTemplate>
                            <TextBlock Text="{Binding}" Margin="0 20
0 0" />
                        </DataTemplate>
                    </ListBox.ItemTemplate>
                </ListBox>
            </DataTemplate>
        </phone:Pivot.ItemTemplate>
    </phone:Pivot>
</Grid>
```

Cela ressemble beaucoup à ce que nous avons fait pour le panorama. Une des premières différences vient de la propriété SelectedIndex du Pivot, qui fonctionne comme pour la ListBox. Je l'ai liée à une propriété Index en mode TwoWay afin que la mise à jour de la propriété depuis le code-behind affecte le contrôle mais qu'en inversement, un changement de valeur depuis le contrôle mette à jour la propriété.

Du coup, je n'ai plus besoin de l'événement de changement de sélection qui existe également sur le contrôle Pivot. Le reste du XAML est semblable au Panorama.

Passons au code-behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;
```

```
        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private List<Ecran> _listeEcrans;
    public List<Ecran> ListeEcrans
    {
        get { return _listeEcrans; }
        set { NotifyPropertyChanged(ref _listeEcrans, value); }
    }

    private int _index;
    public int Index
    {
        get { return _index; }
        set
        {

IsolatedStorageSettings.ApplicationSettings["PageCourante"] = value;
            NotifyPropertyChanged(ref _index, value);
        }
    }

    public MainPage()
    {
        InitializeComponent();

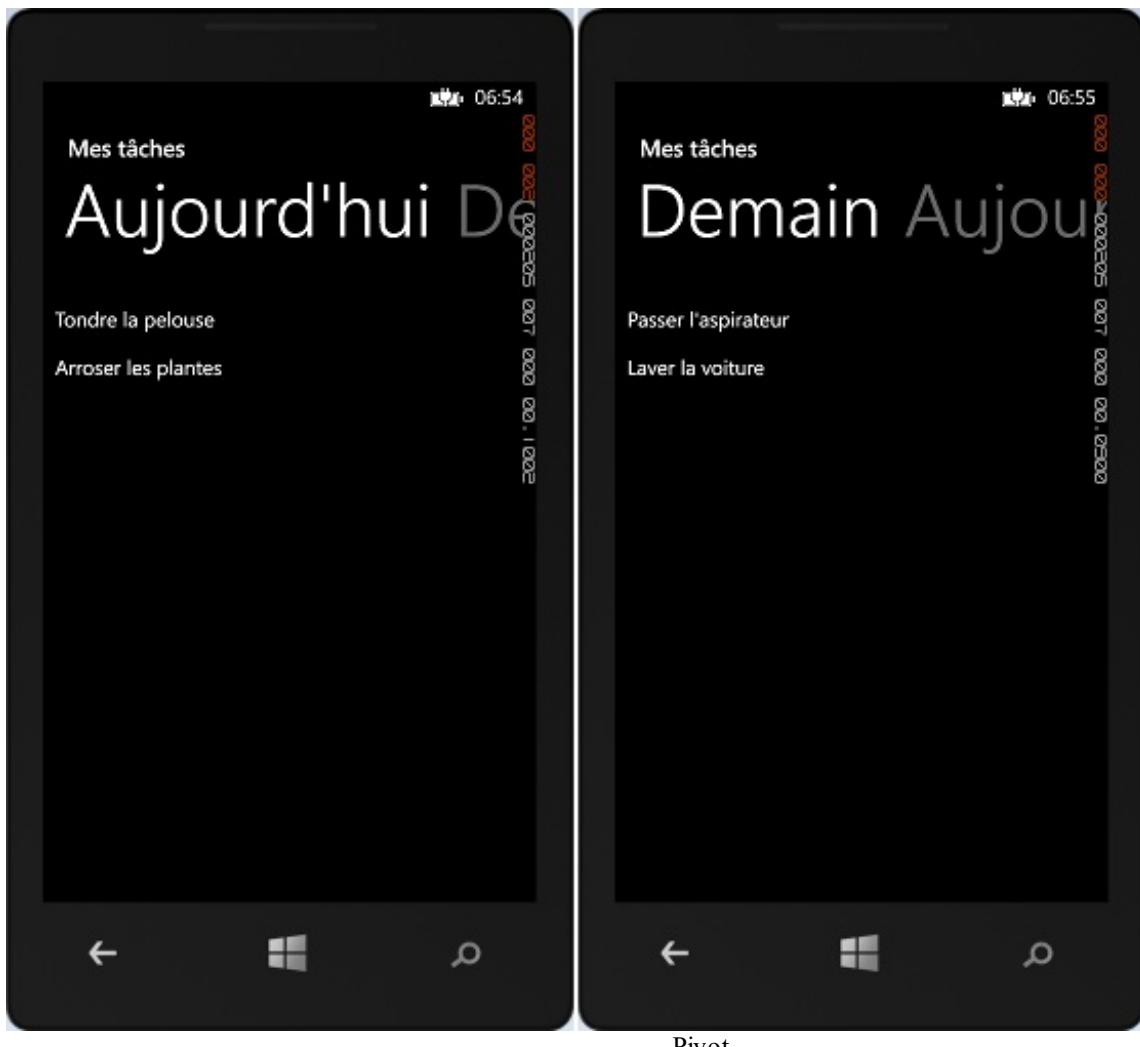
        ListeEcrans = new List<Ecran>
        {
            new Ecran
            {
                Titre = "Aujourd'hui",
                ListeDesTaches = new List<string> { "Tondre la
pelouse", "Arroser les plantes" }
            },
            new Ecran
            {
                Titre = "Demain",
                ListeDesTaches = new List<string> { "Passer
l'aspirateur", "Laver la voiture" }
            }
        };

        DataContext = this;
    }

    private void Pivot_Loaded(object sender, RoutedEventArgs e)
    {
        if
(IsolatedStorageSettings.ApplicationSettings.Contains("PageCourante"))
        {
            Index =
(int)IsolatedStorageSettings.ApplicationSettings["PageCourante"];
        }
    }
}
```

Nous voyons que j'ai rajouté une propriété `Index`, et qu'à l'intérieur de son modificateur, j'enregistre dans le répertoire local la valeur de la sélection. Ensuite, au chargement du pivot et lorsque la valeur existe, je positionne la propriété `Index` à la valeur enregistrée lors d'une visite précédente.

Et voilà, vous pouvez admirer le rendu à la figure suivante.



Le binding du contrôle

Pivot

Vous pourriez trouver que les deux contrôles se ressemblent, et ce n'est pas complètement faux. Je vous rappelle juste que le contrôle Panorama permet d'afficher plusieurs données sur une seule grosse page alors que le contrôle Pivot est utilisé pour présenter la même donnée sur plusieurs pages. Vous apprêhenderez au fur et à mesure la subtile différence entre ces deux contrôles. 😊

- Le Panorama et le Pivot permettent de « naviguer » à l'intérieur des données.
- Le Panorama sert à voir un petit bout d'un plus gros écran dont la taille dépasse celle de l'écran du téléphone.
- Le Pivot permet plutôt de voir la même donnée sur plusieurs pages.
- Chaque changement de vue se fait grâce à un glissement de doigt.
- Le contrôle Pivot est particulièrement bien adapté au binding.

Navigateur web

Malgré tous les superbes contrôles dont dispose Windows Phone, vous allez parfois avoir besoin d'afficher du HTML ou bien directement une page web.

C'est ainsi que le SDK de Windows Phone dispose d'un contrôle bien pratique : [le WebBrowser](#). Vous pouvez le voir comme un mini Internet Explorer que l'on peut mettre où on veut dans une page et qui n'a pas toute la gestion des barres d'adresses, des favoris, ...

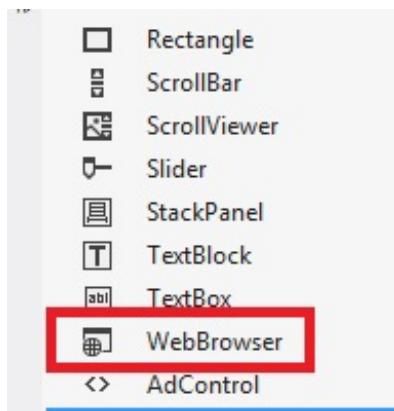
Si nos téléphones possèdent déjà un navigateur web, en l'occurrence Internet Explorer, à quoi pourrait bien servir un tel contrôle ?

Les scénarios sont divers, cela peut aller de l'affichage d'un billet issu d'un flux RSS à une authentification via un formulaire HTML, sur un réseau social par exemple. Ou pourquoi pas un jeu en HTML5 ?

De plus, il est également possible de communiquer entre le Javascript d'une page web et notre page XAML. Regardons tout cela de plus près.

Naviguer sur une page web

Tout d'abord, il nous faut ce fameux contrôle. Vous pouvez le mettre dans votre XAML via la boîte à outils, comme indiqué à la figure suivante.



Le contrôle WebBrowser dans la boîte à outils

Ou comme nous en avons désormais l'habitude, directement dans le XAML :

Code : XML

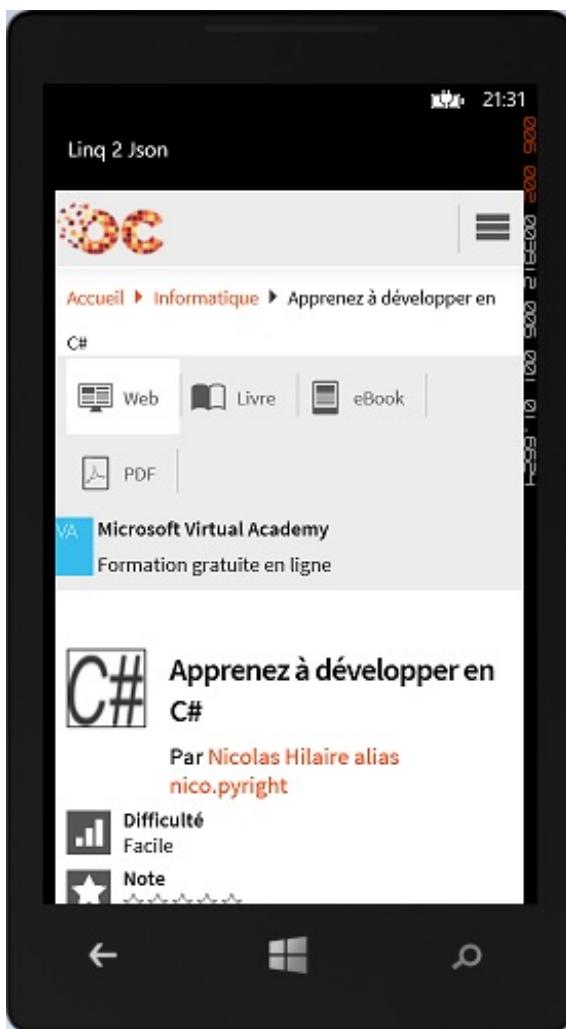
```
<phone:WebBrowser x:Name="MonWebBrowser" />
```

Il est ensuite possible de naviguer sur une page web grâce à la méthode `Navigate()` qui prend une URI en paramètre :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    MonWebBrowser.Navigate(new Uri("http://fr.openclassrooms.com/informatique/cours/apprenez-a-
    developper-en-c", UriKind.Absolute));
}
```

Ici, j'effectue la navigation dans le constructeur de la page. La page internet s'affiche donc comme on peut le voir sur la figure suivante.



La page web s'affiche dans le contrôle

(si bien sûr vous êtes connectés à internet !)

Événements de navigation

Le contrôle `WebBrowser` possède également des événements qui permettent de savoir par exemple quand une page est chargée, il s'agit de l'événement `Navigated`. Ce qui est pratique si l'on souhaite afficher un message d'attente, ou si on veut n'afficher vraiment le `WebControl` qu'une fois la page complètement chargée.

Il y a également un autre événement intéressant qui permet de savoir si la navigation a échoué, par exemple si l'utilisateur ne capte plus internet. Il s'agit de l'événement `NavigationFailed`. Cet événement nous fournit notamment une exception qui peut nous donner plus d'informations sur l'erreur.

Il est toujours intéressant d'indiquer à l'utilisateur si la navigation a échoué. Il est également approprié de lui proposer un bouton lui permettant de retenter sa navigation, si jamais il se trouve à nouveau dans une zone de couverture.

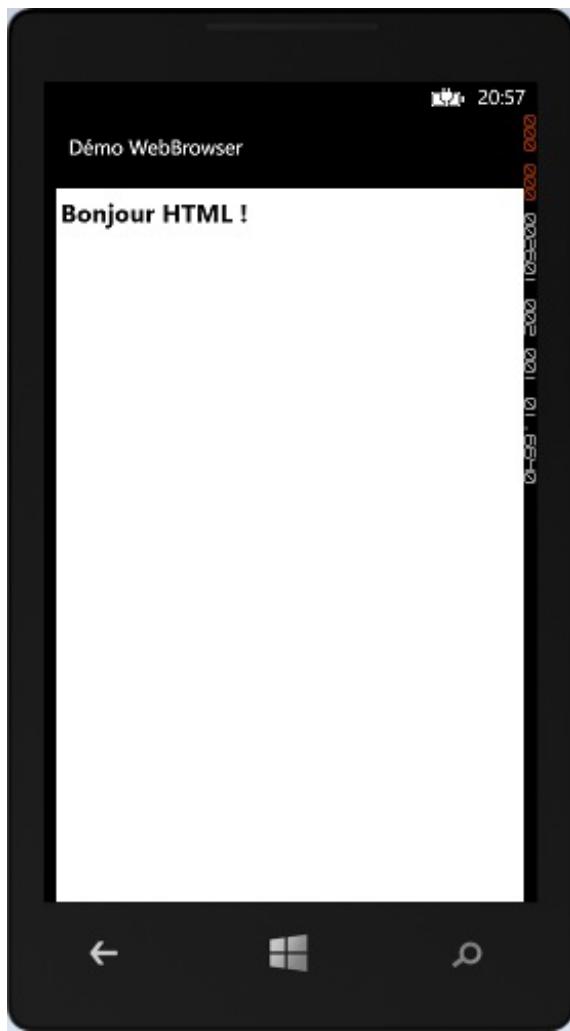
Navigation interne

Mais nous pouvons également créer notre propre HTML et l'afficher grâce à la méthode `NavigateToString` :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    MonWebBrowser.NavigateToString(
        @"<html>
<body>
<h3>Bonjour HTML !</h3>
</body>
</html>" );
}
```

Qui donnera la figure suivante.



Affichage direct de HTML



Le caractère @ est utilisé ici pour pouvoir avoir une chaîne de caractères définie sur plusieurs lignes. Vous pouvez tout à faire écrire tout en ligne ou faire des concaténations avec le caractère \n.

Vous avouerez que ce n'est pas très pratique de devoir saisir le HTML dans le code. Cela serait plus pratique de pouvoir intégrer un fichier HTML à notre application et de naviguer dessus. Pour ce faire, il va falloir dans un premier temps intégrer le fichier dans le répertoire local. Commencez déjà par ajouter un nouveau fichier de type texte au projet que vous nommez `hello.html` et qui possède le code suivant :

Code : HTML

```
<html>
  <body>
    <h3>Bonjour HTML local!</h3>
  </body>
</html>
```

Ensuite, dans les propriétés du fichier, vérifiez que l'action de génération est à « contenu ». Puis, on peut utiliser le code suivant pour ajouter un fichier HTML dans le répertoire local :

Code : C#

```
public MainPage()
{
    InitializeComponent();
```

```

        IntegreHtmlDansRepertoireLocalSiNonPresent("hello.html");
        MonWebBrowser.Navigate(new Uri("hello.html", UriKind.Relative));
    }

    private void IntegreHtmlDansRepertoireLocalSiNonPresent(string
nomFichier)
{
    IsolatedStorageFile systemeDeFichier =
IsolatedStorageFile.GetUserStoreForApplication();

    if (!systemeDeFichier.FileExists(nomFichier))
    {
        // lecture du fichier depuis les ressources
        StreamResourceInfo sr = Application.GetResourceStream(new
Uri(nomFichier, UriKind.Relative));

        using (StreamReader reader = new StreamReader(sr.Stream))
        {
            string html = reader.ReadToEnd();
            using (IsolatedStorageFileStream stream = new
IsolatedStorageFileStream(nomFichier, FileMode.Create,
IsolatedStorageFile.GetUserStoreForApplication()))
            {
                using (StreamWriter writer = new
StreamWriter(stream))
                {
                    writer.Write(html);
                    writer.Close();
                }
            }
        }
    }
}

```

Le principe est de vérifier la présence du fichier. S'il n'existe pas alors on le lit depuis les ressources comme on l'a déjà vu, puis on écrit le contenu dans le répertoire local.

Il ne restera plus qu'à naviguer sur ce fichier créé grâce à la méthode Navigate comme on peut le voir plus haut.

Communiquer entre XAML et HTML

Le WebBrowser a aussi la capacité de faire communiquer la page web et la page XAML. Plus particulièrement, il est possible d'invoquer une méthode Javascript depuis notre page XAML et inversement, la page web peut déclencher un événement dans notre application.

Pour que ceci soit possible, vous devez positionner la propriété IsScriptEnabled à true dans votre WebBrowser et vous abonner à l'événement ScriptNotify :

Code : XML

```
<phone:WebBrowser x:Name="MonWebBrowser" IsScriptEnabled="True"
ScriptNotify="MonWebBrowser_ScriptNotify" />
```

Pour que cela soit plus simple, je vais illustrer le fonctionnement avec du HTML et du Javascript que j'embarquerais dans mon application avec la méthode montrée précédemment. Mais il est bien sûr possible de faire la même chose avec une page sur internet.

L'événement ScriptNotify est levé lorsque la page web invoque la méthode Javascript window.external.notify(). Cette méthode accepte un paramètre sous la forme d'une chaîne de caractères qui pourra être récupérée dans l'événement ScriptNotify.

Pour l'illustrer,修改ons notre page hello.html pour avoir :

Code : HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Demo javascript</title>
    <script type="text/javascript">
      function EnvoyerMessage()
      {
        window.external.notify("Bonjour, je suis " + prenom.value);
        resultat.innerHTML = "Message bien envoyé";
      }
    </script>
  </head>
  <body>
    <h3>Communication entre page web et XAML</h3>
    <p>Saisissez votre prénom : </p>
    <input type="text" id="prenom" />
    <input type="button" value="Dire bonjour"
    onclick="EnvoyerMessage(); " />
    <div id="resultat" />
  </body>
</html>

```

Cette page possède une zone de texte pour saisir un prénom et un bouton qui invoque la méthode Javascript `EnvoyerMessage()`. Cette méthode récupère la valeur du champ saisi, la concatène à la chaîne « Bonjour, je suis » et l'envoie à notre application Windows Phone via la méthode `window.external.notify`. Enfin, elle affiche un message sur la page web pour indiquer que le message a bien été envoyé.

Côté code-behind, nous avons juste à naviguer sur notre page et à récupérer la valeur envoyée :

Code : C#

```

public partial class MainPage : PhoneApplicationPage
{
  public MainPage()
  {
    InitializeComponent();
    IntegreHtmlDansRepertoireLocalSiNonPresent("hello.html",
true);
    MonWebBrowser.Navigate(new Uri("hello.html",
UriKind.Relative));
  }

  private void IntegreHtmlDansRepertoireLocalSiNonPresent(string
nomFichier, bool force)
  {
    IsolatedStorageFile systemeDeFichier =
IsolatedStorageFile.GetUserStoreForApplication();

    if (!systemeDeFichier.FileExists(nomFichier) || force)
    {
      // lecture du fichier depuis les ressources
      StreamResourceInfo sr =
Application.GetResourceStream(new Uri(nomFichier,
UriKind.Relative));

      using (StreamReader reader = new
StreamReader(sr.Stream))
      {
        string html = reader.ReadToEnd();
        using (IsolatedStorageFileStream stream = new
IsolatedStorageFileStream(nomFichier, FileMode.Create,
IsolatedStorageFile.GetUserStoreForApplication()))
        {
          using (StreamWriter writer = new
StreamWriter(stream))
          {
            writer.Write(html);
            writer.Close();
          }
        }
      }
    }
  }
}

```

```
        }
    }
}

private void MonWebBrowser_ScriptNotify(object sender,
NotifyEventArgs e)
{
    MessageBox.Show(e.Value);
}
}
```

Notez que j'ai rajouté un paramètre à la méthode permettant de stocker la page HTML dans le répertoire local, qui force l'enregistrement afin d'être sûr d'avoir toujours la dernière version.

Rappelez-vous, nous recevons la valeur envoyée par la page web grâce à l'événement `ScriptNotify`. Lorsque cette chaîne est reçue, on l'affiche simplement avec une boîte de message. Démarrons l'application, la page web s'affiche, comme vous pouvez le voir à la figure suivante.



La page web avant envoi du message

Notez que nous sommes un peu obligés de zoomer pour pouvoir voir le contenu de la page web et cliquer sur le bouton. Pour rappel, le zoom s'effectue avec deux doigts, en posant les doigts sur l'écran et en les étirant vers l'extérieur. Ce mouvement s'appelle le *Pinch-to-zoom*. Il est également possible de double-cliquer dans l'émulateur afin de produire le même effet. Nous verrons comment corriger ce problème plus loin.

Saisissez une valeur et cliquez sur le bouton. Le message est bien envoyé par la page HTML et est bien reçu par notre application, comme en témoigne la figure suivante.



L'application Windows Phone reçoit un message de la page web

Mais alors, cette page HTML illisible, on ne peut pas un peu l'améliorer ?

Il y a plusieurs solutions pour ce faire. La première est d'utiliser un tag meta que le navigateur va interpréter. Cette balise se met à l'intérieur de la balise <head> :

Code : HTML

```
<meta name="viewport" content="width=device-width, user-scalable=no" />
```

Elle permet de définir la taille de la fenêtre. On peut y mettre une valeur numérique allant de 320 à 10000 ou l'ajuster directement à la taille du téléphone avec la valeur device-width. Remarquez, que la propriété user-scalable empêchera l'utilisateur de pouvoir zoomer dans la page. Voici le rendu à la figure suivante.



Le zoom est adapté à la largeur de la page

Ce qui est beaucoup plus lisible ! 😊

L'autre solution est d'utiliser une propriété CSS pour ajuster la taille du texte :

Code : HTML

```
<h3 style="-ms-text-size-adjust:300%">Communication entre page web  
et XAML</h3>
```

Ici, j'augmente la taille de cette balise de 300%.

Remarquez que cette balise est incompatible avec la balise `viewport`.

Il est possible de faire communiquer nos deux éléments également dans l'autre sens. Ainsi, nous pouvons invoquer une méthode Javascript présente sur la page web depuis le code C#. Modifions notre page pour qu'elle possède une méthode Javascript qui accepte deux paramètres :

Code : HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <meta name="viewport" content="width=device-width, user-  
scalable=no" />  
    <title>Demo javascript</title>  
    <script type="text/javascript">  
      function ReceptionMessage(texte, heure)  
      {
```

```
        resultat.innerHTML = texte + ". Il est " + heure;
    return "OK";
}
</script>
</head>
<body>
    <h3>En attente d'un envoi ...</h3>
    <div id="resultat" />
</body>
</html>
```

Rajoutons ensuite dans notre XAML un bouton qui va permettre d'envoyer des informations à la page web :

Code : HTML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="auto" />
    </Grid.RowDefinitions>
    <phone:WebBrowser x:Name="MonWebBrowser" IsScriptEnabled="True"
        ScriptNotify="MonWebBrowser_ScriptNotify" />
        <Button Content="Envoyer l'heure" Tap="Button_Tap" Grid.Row="1"
    />
</Grid>
```

Et dans le code-behind nous aurons :

Code : C#

```
private void Button_Tap(object sender,
    System.Windows.Input.GestureEventArgs e)
{
    string retour =
        (string)MonWebBrowser.InvokeScript("ReceptionMessage", "Bonjour
depuis Windows Phone", DateTime.Now.ToString());
```

On utilise la méthode `InvokeScript` de l'objet `WebBrowser` en lui passant en paramètre le nom de la méthode Javascript à appeler. Puis nous pouvons passer ensuite autant de paramètres que nous le souhaitons, sous la forme d'une chaîne de caractères. Il faut bien sûr qu'il y ait autant de paramètres que peut accepter la méthode Javascript. Ce qui donne la figure suivante.



Réception par la page HTML du message envoyé par l'application

Windows Phone

Notez que le Javascript peut renvoyer une valeur au code C#. Ici je renvoie la chaîne OK, que je stocke dans la variable retour pour une éventuelle interprétation.

Remarquez qu'il est également possible de passer des objets complexes grâce au JSON. Le principe consiste à envoyer une version sérialisée d'un objet à la page web et la page web désérialise cet objet pour pouvoir l'utiliser. Utilisons donc la bibliothèque open-source JSON.NET que nous avions précédemment utilisée afin de sérialiser un objet. Référez-vous à l'assembly Newtonsoft.Json.dll et créez une classe avec des propriétés :

Code : C#

```
public class Informations
{
    public string Message { get; set; }
    public DateTime Date { get; set; }
}
```

Puis, sérialisez un objet pour l'envoyer à notre méthode Javascript :

Code : C#

```
private void Button_Tap(object sender, System.Windows.Input.GestureEventArgs e)
{
    Informations informations = new Informations { Message = "Bonjour depuis Wind
Date = DateTime.Now };
    string objetSerialise = JsonConvert.SerializeObject(informations);
    Retour retour =
JsonConvert.DeserializeObject<Retour>((string)MonWebBrowser.InvokeScript("Recepti
```

```
    objetSerialise));  
}
```

De la même façon, on pourra désérialiser le retour de la méthode, dans l'objet Retour suivant :

Code : C#

```
public class Retour  
{  
    public string Resultat { get; set; }  
    public bool Succes { get; set; }  
}
```

Il ne reste plus qu'à modifier le Javascript de la page :

Code : HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
    <head>  
        <meta name="viewport" content="width=device-width, user-  
scalable=no" />  
        <title>Demo javascript</title>  
        <script type="text/javascript">  
            function ReceptionMessage(objet)  
            {  
                var infos = JSON.parse(objet);  
                var d = new Date(infos.Date);  
                resultat.innerHTML = infos.Message + ". Il est " +  
d.getHours() + 'h' + d.getMinutes();  
                var retour = {  
                    Resultat : "OK",  
                    Succes : true  
                }  
                return JSON.stringify(retour);  
            }  
        </script>  
    </head>  
    <body>  
        <h3>En attente d'un envoi ...</h3>  
        <div id="resultat" />  
    </body>  
</html>
```

Pour désérialiser du JSON côté javascript, on utilise la méthode `JSON.parse` et pour sérialiser, on utilisera `JSON.stringify`.

Et voilà. Plutôt puissant n'est-ce pas ?

- Le contrôle `WebBrowser` nous permet facilement d'afficher du HTML dans nos applications Windows Phone.
- Il est possible d'interagir entre l'application et la page web grâce à du Javascript.
- Grâce au JSON, nous pourrons passer des paramètres complexes entre l'application et la page web.

TP : Création d'un lecteur de flux RSS simple

Oula, mais ça fait longtemps qu'on a pas un peu testé nos connaissances dans un contexte bien concret. Il est temps d'y remédier avec ce TP où nous allons mettre en pratique les derniers éléments que nous avons étudiés.

Avec à la clé, un petit début d'application sympathique pour lire nos flux RSS 😊. Allez, c'est parti, à vous de travailler.

Instructions pour réaliser le TP

Vous l'avez compris, nous allons réaliser une petite application qui va nous permettre de lire nos flux RSS préférés. Voici ce que devra faire l'application :

- L'application possédera une page de menu qui permettra soit d'aller voir les flux déjà enregistrés dans l'application, soit d'aller en ajouter de nouveaux.
- La page de saisie de nouveau flux offrira un moyen de saisir et de faire persister des URL de flux RSS.
- La page de consultation possèdera une `ListBox` présentant la liste de tous les titres, image et description des flux.
- Lors du clic sur un élément, elle renverra sur une nouvelle page contenant un contrôle `Pivot`, branché encore une fois sur la liste de tous les flux et automatiquement positionné sur le flux sélectionné dans la `ListBox`. Ce qui permettra de naviguer de flux en flux grâce à un glissement de doigt. Ce pivot présentera le titre du blog ainsi qu'une `ListBox` contenant la liste des titres des billets classés par ordre décroissant de dernière mise à jour.
- La sélection d'un billet de cette `ListBox` renverra sur une page contenant un `WebBrowser` et affichant ce fameux billet

Voilà pour l'énoncé de ce TP. Il va nous permettre de vérifier que nous avons bien compris comment accéder à des données sur internet, comment utiliser la bibliothèque de syndication, comment utiliser la `ListBox` et le `Pivot` avec le binding. Vous devrez également utiliser le répertoire local - bien sûr, nous n'allons pas re-saisir nos flux RSS à chaque lancement d'application -, la navigation et le contrôle `WebBrowser`. Bref, que des bonnes choses ! 😊

Vous vous sentez prêt pour relever ce défi ? Alors, n'hésitez pas à vous lancer. Petit bonus ? Utiliser une solution pour marquer différemment les billets qui ont été lus des billets qui ne l'ont pas encore été...

Vous pouvez bien sûr si vous le souhaitez respecter le patron de conception MVVM, mais ceci n'est pas une obligation.

Si cela vous effraie un peu (et ça peut !), essayons de décortiquer un peu les différents éléments.

- La page de menu ne devrait pas poser de problèmes, il suffit d'utiliser le service de navigation.
- La page de saisie d'URL de flux RSS ne devrait pas non plus poser de problèmes. Pensez juste à faire persister toute modification dans le répertoire local.
- Passons à la page qui affiche la liste des flux RSS dans une `ListBox`. Le point critique consiste à charger les différents flux RSS, mais ça, vous devriez savoir le faire car nous l'avons vu dans la partie précédente. N'oubliez pas que la classe `WebClient` ne peut télécharger qu'un élément à la fois, il va donc falloir attendre que le téléchargement précédent soit terminé. Ensuite, étant donné qu'on doit afficher le titre, l'image et la description, il est tout à fait opportun de créer une classe dédiée qui ne contiendra que les éléments que nous souhaitons afficher. Le titre est disponible dans la propriété `Title.Text` d'un `SyndicationFeed`, l'image via la propriété `ImageUrl` et la description via la propriété `Description.Text`. Chaque flux possédera une liste de billets qui contiendront la date de dernière publication (propriété `LastUpdatedTime.DateTime`), le titre (propriété `Title.Text`) et l'URL du billet (premier élément de la liste `Links`, propriété `Uri`). Ensuite tout est une histoire de binding, ce qui n'est pas forcément le plus facile mais c'est un point très important où vous devez vous entraîner. 😊
- Enfin, la page avec le `WebBrowser` ne devrait pas être trop compliquée à faire, le chapitre sur le `WebBrowser` étant tout fraîchement lu.

N'oubliez pas que vous pouvez utiliser le dictionnaire d'état pour communiquer entre les pages. Allez, j'en ai beaucoup dit. Pour le reste, c'est à vous de chercher un peu. 😊 Bon courage.

Correction

Ahhhhh, une première vraie application. Enfin ! Pas si facile finalement ?

On se rend compte qu'il y a plein de petites choses, plein de légers bugs qui apparaissent au fur et à mesure de nos tests... Il faut avancer petit à petit à partir du moment où ça se complique un peu et lorsque l'application commence à grandir.

Il n'y a pas une unique solution pour ce TP. Toute solution fonctionnelle est bonne. Je vais vous présenter la mienne. J'ai porté une attention quasi nulle à la présentation histoire que cela soit assez simple. J'espère que de votre côté, vous en avez profité pour faire quelque chose de joli. 😊

J'ai donc commencé par créer une nouvelle application, nommée `TpFluxRSS`. Étant donné que nous avons besoin de

manipuler des flux RSS, il faut référencer l'assembly System.ServiceModel.Syndication.dll.
Voici la page de menu :

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--TitlePanel contient le nom de l'application et le titre de
la page-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP Lecteur de
flux RSS" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Menu" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <!--ContentPanel - placez tout contenu supplémentaire ici-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <Button Content="Voir les flux RSS" Tap="VoirFluxTap" />
            <Button Content="Ajouter un flux RSS"
Tap="AjouterFluxTap" />
        </StackPanel>
    </Grid>
</Grid>

```

Avec le code-behind suivant :

Code : C#

```

public partial class MainPage : PhoneApplicationPage
{
    // Constructeur
    public MainPage()
    {
        InitializeComponent();

        if
(IsolatedStorageSettings.ApplicationSettings.Contains("ListeUrl"))
        {
            PhoneApplicationService.Current.State["ListeUrl"] =
IsolatedStorageSettings.ApplicationSettings["ListeUrl"];
        }
    }

    private void VoirFluxTap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        NavigationService.Navigate(new Uri("/VoirFlux.xaml",
UriKind.Relative));
    }

    private void AjouterFluxTap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        NavigationService.Navigate(new Uri("/AjouterFlux.xaml",
UriKind.Relative));
    }
}

```

On observe dans le constructeur qu'on commence par charger un objet que l'on met directement dans le dictionnaire d'état. Le nom de la clé nous fait pressentir qu'il s'agit de la liste des URL de flux RSS que notre utilisateur a saisi dans notre application. Ce qui nous permet de les faire persister d'un lancement à l'autre. Pour le reste, il s'agit d'afficher deux boutons qui nous renvoient sur d'autres pages, les pages VoirFlux.xaml et AjouterFlux.xaml (voir la figure suivante).



La page de menu du lecteur de flux RSS

Commençons par la page AjouterFlux.xaml qui, sans surprises, affiche une page permettant de saisir des URL de flux RSS. Le XAML de la page est encore très simple :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
    Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP Lecteur de
        flux RSS" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Ajouter un flux"
        Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="Saisir une url" />
            <TextBox x:Name="Url" InputScope="Url" />
        </StackPanel>
    </Grid>
</Grid>
```

```
<Button Content="Ajouter" Tap="AjouterTap" />
</StackPanel>
</Grid>
</Grid>
```

Il permet d'afficher une zone de texte, où le clavier sera adapté pour la saisie d'URL (InputScope="Url") et possédant un bouton pour faire l'ajout de l'URL, comme vous pouvez le voir sur la figure suivante.



La page d'ajout des flux du lecteur de flux RSS

C'est dans le code-behind que tout se passe :

Code : C#

```
public partial class AjouterFlux : PhoneApplicationPage
{
    private List<Uri> listeUrl;

    public AjouterFlux()
    {
        InitializeComponent();
    }

    protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        Url.Text = "http://";
        if
(PhoneApplicationService.Current.State.ContainsKey("ListeUrl"))
            listeUrl =
```

```
(List<Uri>) PhoneApplicationService.Current.State["ListeUrl"];  
    else  
        listeUrl = new List<Uri>();  
    base.OnNavigatedTo(e);  
}  
  
private void AjouterTap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    Uri uri;  
    if (!Uri.TryCreate(Url.Text, UriKind.Absolute, out uri))  
        MessageBox.Show("Le format de l'url est incorrect");  
    else  
    {  
        listeUrl.Add(uri);  
        IsolatedStorageSettings.ApplicationSettings["ListeUrl"]  
= listeUrl;  
        PhoneApplicationService.Current.State["ListeUrl"] =  
listeUrl;  
        MessageBox.Show("L'url a été correctement ajoutée");  
    }  
}
```

On commence par obtenir la liste des URL potentiellement déjà chargée, ensuite nous l'ajoutons à la liste si elle n'existe pas déjà, puis nous mettons à jour cette liste dans le répertoire local ainsi que dans le dictionnaire d'état.



Remarque : il pourrait être judicieux de vérifier ici si l'URL correspond vraiment à un flux RSS.

Maintenant, il s'agit de réaliser la page `VoirFlux.xaml`, qui contient la liste des titres des différents flux. Pour l'exemple, je m'en suis ajouté quelques uns, comme vous pouvez le voir sur la figure suivante.



La liste de tous les flux du lecteur de flux RSS

Ici le XAML intéressant se situe dans la deuxième grille :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP Lecteur de flux RSS" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Voir les flux" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock x:Name="Changement" HorizontalAlignment="Center" Foreground="Red" FontSize="20" FontWeight="Bold" />
        <ListBox x:Name="ListBoxFlux" ItemsSource="{Binding ListeFlux}" Grid.Row="1" SelectionChanged="ListBox_SelectionChanged">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <Grid>
```

```

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto" />
            <RowDefinition Height="auto" />
            <RowDefinition Height="auto" />
        </Grid.RowDefinitions>
        <Image Source="{Binding UrlImage}"
    Height="50" Width="50" />
        <TextBlock Grid.Column="1" Text="{Binding
Titre}" TextWrapping="Wrap" Style="{StaticResource
PhoneTextTitle3Style}" />
        <TextBlock Grid.ColumnSpan="2" Grid.Row="1"
Text="{Binding Description}" TextWrapping="Wrap" Margin="0 0 0 20"
FontStyle="Italic" />
        <Line X1="0" X2="480" Y1="0" Y2="0"
StrokeThickness="5" Stroke="Blue" Grid.Row="2" Grid.ColumnSpan="2"
/>
    </Grid>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</Grid>

```

On y remarque un `TextBlock` qui servira d'indicateur de chargement des flux, puis une `ListBox` qui est liée à la propriété `ListeFlux`. Le modèle des éléments de la `ListBox` affiche l'image du flux, via la liaison à la propriété `UrlImage`, le titre via la liaison à la propriété `Titre` ainsi que la description... Notons un unique effet d'esthétique permettant de séparer deux flux, via une magnifique ligne bleue. 

Passons au code behind :

Code : C#

```

public partial class VoirFlux : PhoneApplicationPage,
INotifyPropertyChanged
{
    private WebClient client;

    private ObservableCollection<Flux> listeFlux;
    public ObservableCollection<Flux> ListeFlux
    {
        get { return listeFlux; }
        set { NotifyPropertyChanged(ref listeFlux, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }
}

```

```
public VoirFlux()
{
    InitializeComponent();
    DataContext = this;
}

protected override async void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    List<Uri> listeUrl;
    if (PhoneApplicationService.Current.State.ContainsKey("ListeUrl"))
        listeUrl =
(List<Uri>)PhoneApplicationService.Current.State["ListeUrl"];
    else
        listeUrl = new List<Uri>();

    if (listeUrl.Count == 0)
    {
        MessageBox.Show("Vous devez d'abord ajouter des flux");
        if (NavigationService.CanGoBack)
            NavigationService.GoBack();
    }
    else
    {
        Chargement.Text = "Chargement en cours ...";
        Chargement.Visibility = Visibility.Visible;
        ListeFlux = new ObservableCollection<Flux>();
        client = new WebClient();

        queue = new Queue<Uri>();
        foreach (Uri uri in listeUrl)
        {
            try
            {
                string rss = await
client.DownloadStringTaskAsync(uri);
                AjouteFlux(rss);
            }
            catch (Exception)
            {
                MessageBox.Show("Impossible de lire le flux à
l'adresse : " + uri + "\nVérifiez votre connexion internet");
            }
        }
        Chargement.Text = string.Empty;
        Chargement.Visibility = Visibility.Collapsed;
    }
    base.OnNavigatedTo(e);
}

private void AjouteFlux(string flux)
{
    StringReader stringReader = new StringReader(flux);
    XmlReader xmlReader = XmlReader.Create(stringReader);
    SyndicationFeed feed = SyndicationFeed.Load(xmlReader);
    listeFlux.Add(ConstruitFlux(feed));
    PhoneApplicationService.Current.State["ListeFlux"] =
ListeFlux.ToList();
}

private Flux ConstruitFlux(SyndicationFeed feed)
{
    Flux flux = new Flux { Titre = feed.Title.Text, UrlImage =
feed.ImageUrl, Description = feed.Description == null ? string.Empty :
feed.Description.Text, ListeBillets = new List<Billet>() };
    foreach (SyndicationItem item in feed.Items.OrderByDescending(e =>
e.LastUpdatedTime.DateTime))
    {
        Uri url = item.Links.Select(e => e.Uri).FirstOrDefault();
        if (url != null)
```

```

        {
            Billet billet = new Billet { Id = url.AbsolutePath,
DatePublication = item.LastUpdatedTime.DateTime, Titre = item.Title.Text,
EstDejaLu = EstDejaLu(url.AbsolutePath), Url = url };
            flux.ListeBillets.Add(billet);
        }
    }
    return flux;
}

private bool EstDejaLu(string id)
{
    if
(IsolatedStorageSettings.ApplicationSettings.Contains("ListeDejaLus"))
    {
        List<string> dejaLus =
(List<string>)IsolatedStorageSettings.ApplicationSettings["ListeDejaLus"];
        bool any = dejaLus.Any(e => e == id);
        return any;
    }
    return false;
}

private void ListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (ListBoxFlux.SelectedItem != null)
    {
        Flux flux = (Flux)ListBoxFlux.SelectedItem;
        PhoneApplicationService.Current.State["FluxCourant"] = flux;
        NavigationService.Navigate(new Uri("/VoirFluxPivot.xaml",
UriKind.Relative));
        ListBoxFlux.SelectedItem = null;
    }
}
}

```

C'est le code-behind le plus long de l'application. Nous voyons l'implémentation classique de `INotifyPropertyChanged` ainsi qu'une propriété `ListeFlux` qui est une `ObservableCollection` de `Flux`. L'objet `Flux` contiendra toutes les propriétés d'un `Flux` que nous souhaitons afficher sur la prochaine page :

Code : C#

```
public class Flux
{
    public string Titre { get; set; }
    public Uri UrlImage { get; set; }
    public string Description { get; set; }
    public List<Billet> ListeBillets { get; set; }
}
```

Un titre, l'URL de l'image, la description ainsi qu'une liste de billets, sachant que la classe `Billet` sera :

Code : C#

```
public class Billet
{
    public string Id { get; set; }
    public DateTime DatePublication { get; set; }
    public string Titre { get; set; }
    public Uri Url { get; set; }
    public bool EstDejaLu { get; set; }
}
```

Ensuite, il y a le chargement des flux. Voyez comme c'est facile grâce à `await`, une simple boucle et on n'en parle plus. 😊

La méthode `AjouteFlux` crée un objet `SyndicationFeed` comme on l'a déjà vu puis assemble un objet `Flux` à partir du `SyndicationFeed`. La liste d'objets `Flux` est ensuite mise à jour dans le dictionnaire d'état.

Notons que lors de l'assemblage, je dois déterminer si le billet a déjà été lu ou non. Pour cela, je tente de lire dans le répertoire local la liste de tous les billets déjà lus, identifiés par leur id. Si je le trouve, c'est que le billet est déjà lu. Cette liste est alimentée lorsqu'on visionne réellement le billet.



Il serait judicieux de ne lire qu'une unique fois la liste des billets déjà lus depuis le répertoire local, par exemple dans la méthode `OnNavigatedTo`, afin d'améliorer les performances. Je ne l'ai pas fait ici pour que cela soit plus clair, mais n'hésitez pas à le faire.

Enfin, l'événement de sélection d'un flux s'occupe de récupérer le flux sélectionné, de le mettre dans le dictionnaire d'état et de naviguer sur la page `VoirFluxPivot.xaml`.

Finalement, ce n'est pas très compliqué. Il faut juste enchaîner tranquillement les actions sans rien oublier.

Passons à présent à la page `VoirFluxPivot.xaml` où le but est d'afficher dans un pivot la liste des flux, en se positionnant sur celui choisi. Ce qui nous permettra de naviguer de flux en flux en faisant un glissement de doigt. Chaque vue du pivot contiendra une `ListBox` avec tous les billets du flux en cours. Notons au passage que j'ai choisi de marquer les billets déjà lus en italique (voir la figure suivante).



Ici c'est le XAML qui est sans doute le plus compliqué. Nous avons besoin du contrôle `Pivot`. Notre contrôle `Pivot` est lié à la propriété `ListeFlux` :

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <phone:Pivot x:Name="PivotFlux" ItemsSource="{Binding
ListeFlux}" Loaded="PivotFlux_Loaded">
        <phone:Pivot.HeaderTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding Titre}" />
            </DataTemplate>
        </controls:Pivot.HeaderTemplate>
        <phone:Pivot.ItemTemplate>
            <DataTemplate>
                <StackPanel>
                    <TextBlock Text="{Binding Titre}"
FontWeight="Bold" FontSize="20" />
                    <ListBox ItemsSource="{Binding ListeBillets}"
SelectionChanged="ListBoxBillets_SelectionChanged">
                        <ListBox.ItemTemplate>
                            <DataTemplate>
                                <TextBlock Text="{Binding Titre}"
Margin="0 0 0 40" FontStyle="{Binding EstDejaLu,
Converter={StaticResource FontFamilyConverter}}" />
                            </DataTemplate>
                        </ListBox.ItemTemplate>
                    </ListBox>
                </StackPanel>
            </DataTemplate>
        </controls:Pivot.ItemTemplate>
    </controls:Pivot>
</Grid>

```

Nous nous abonnons également à l'événement de chargement de Pivot. L'entête du Pivot est liée au titre du flux, via le modèle d'entête. Quant au corps du Pivot, il possède un modèle où il y a notamment une ListBox, liée à la propriété ListeBillets. Notons que nous nous sommes abonnés à l'événement de changement de sélection de la ListBox. Le corps de la ListBox consiste à lier le titre du billet avec une subtilité où j'utilise un converter pour positionner le style de la police en fonction de si le billet a déjà été lu ou pas. Ce converter devra donc être déclaré dans les ressources :

Code : XML

```

<phone:PhoneApplicationPage.Resources>
    <converter:FontFamilyConverter x:Key="FontFamilyConverter" />
</phone:PhoneApplicationPage.Resources>

```

Avec l'espace de nom qui va bien :

Code : XML

```
xmlns:converter="clr-namespace:TpFluxRss"
```

La classe de conversion devra donc convertir un booléen en FontStyle :

Code : C#

```

public class FontFamilyConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object
parameter, CultureInfo culture)
    {
        return (bool)value ? FontStyles.Italic : FontStyles.Normal;
    }
}

```

```
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Ici, nul besoin d'implémenter la méthode de conversion de FontStyle vers booléen, elle ne sera pas utilisée. La conversion consiste à avoir le style italique si le booléen est vrai, normal sinon.

Reste le code-behind :

Code : C#

```
public partial class VoirFluxPivot : PhoneApplicationPage, INotifyPropertyChanged
{
    private ObservableCollection<Flux> listeFlux;
    public ObservableCollection<Flux> ListeFlux
    {
        get { return listeFlux; }
        set { NotifyPropertyChanged(ref listeFlux, value); }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur, [CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    public VoirFluxPivot()
    {
        InitializeComponent();
        DataContext = this;
    }

    protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        ListeFlux = new
        ObservableCollection<Flux>((List<Flux>)PhoneApplicationService.Current.State["ListeFlux"]);
        base.OnNavigatedTo(e);
    }

    private void PivotFlux_Loaded(object sender, RoutedEventArgs e)
    {
        PivotFlux.SelectedItem =
(Flux)PhoneApplicationService.Current.State["FluxCourant"];
        PivotFlux.SelectionChanged += PivotFlux_SelectionChanged;
    }

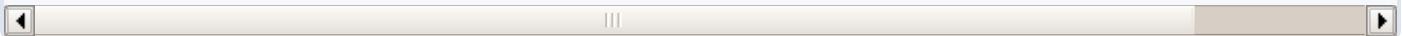
    private void ListBoxBillets_SelectionChanged(object sender, SelectionChange
```

```

        if (listBox.SelectedItem != null)
    {
        Billet billet = (Billet)listBox.SelectedItem;
        PhoneApplicationService.Current.State["BilletCourrant"] = billet;
        NavigationService.Navigate(new Uri("/VoirBillet.xaml", UriKind.Relative));
        listBox.SelectedItem = null;
    }
}

private void PivotFlux_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    PhoneApplicationService.Current.State["FluxCourant"] = PivotFlux.SelectedItem;
}
}

```



On retrouve notre propriété `ListeFlux`, ainsi que son implémentation classique. On peut voir que lors de l'arrivée sur la page, je récupère dans le dictionnaire d'état la liste des flux et que je construis une `ObservableCollection` avec, pour la mettre dans la propriété `ListeFlux`.

Dans l'événement de chargement du `Pivot`, j'en profite pour récupérer le flux courant et ainsi positionner le pivot sur le flux précédemment sélectionné dans la `ListBox`.

Enfin, lors de la sélection d'un billet dans la `ListBox`, il ne me reste plus qu'à positionner le billet dans le dictionnaire d'état et de naviguer sur la page `VoirBillet.xaml`.

Sachant que la `ListBox` est dans un contrôle qui possède plusieurs vues, il n'est pas possible d'obtenir la `ListBox` en cours par son nom. En effet, en fait il y a autant de `ListBox` que de flux. On utilisera donc le paramètre `sender` de l'événement de sélection qui nous donne la `ListBox` concernée.

Remarquons que nous devons modifier le flux courant lorsque nous changeons d'élément dans le `Pivot`. On utilise l'événement de changement de sélection auquel on s'abonne à la fin du chargement du `Pivot`. On fait ceci une fois que l'élément en cours est positionné, afin que l'événement ne soit pas levé.

Il ne reste plus que la page `VoirBillet.xaml`, qui est plutôt simple. En tous cas, le XAML ne contient que le contrôle `WebBrowser`:

Code : XML

```

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP Lecteur de flux RSS" Style="{StaticResource PhoneTextNormalStyle}" />
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <phone:WebBrowser x:Name="PageBillet" NavigationFailed="PageBillet_NavigationFailed" />
    </Grid>
</Grid>

```

Nous nous sommes quand même abonnés à l'événement d'erreur de navigation pour afficher un petit message, au cas où... Dans le code-behind, nous aurons juste besoin de démarrer la navigation et de marquer le billet comme lu :

Code : C#

```

public partial class VoirBillet : PhoneApplicationPage
{
    private Billet billet;
}

```

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

protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    billet =
(Billet)PhoneApplicationService.Current.State["BilletCourrant"];
    billet.EstDejaLu = true;

    List<string> dejaLus;
    if
(IsolatedStorageSettings.ApplicationSettings.Contains("ListeDejaLus"))
        dejaLus =
(List<string>) IsolatedStorageSettings.ApplicationSettings["ListeDejaLus"];
    else
        dejaLus = new List<string>();
    if (!dejaLus.Any(elt => elt == billet.Id))
        dejaLus.Add(billet.Id);

    IsolatedStorageSettings.ApplicationSettings["ListeDejaLus"] =
dejaLus;
    PageBillet.Navigate(billet.Url);
    base.OnNavigatedTo(e);
}

private void PageBillet_NavigationFailed(object sender,
System.Windows.Navigation.NavigationFailedEventArgs e)
{
    MessageBox.Show("Impossible de charger la page, vérifiez votre
connexion internet");
}
```

N'oublions pas de faire persister la liste des billets lus dans le répertoire local...
Et voici le résultat à la figure suivante.



Affichage d'un billet dans le lecteur de flux RSS

Et voilà, notre application commence à être fonctionnelle. J'espère que vous aurez réussi à maîtriser notamment les histoires de binding et de sélection d'éléments. C'est un travail très classique dans la création d'application Windows Phone avec le XAML. Ce sont des éléments que vous devez maîtriser. N'hésitez pas à refaire ce TP si vous ne l'avez pas complètement réussi. Vous pouvez aussi compléter cette application qui est loin d'être parfaite, vous entraîner à faire des jolies interfaces dans le style Modern UI, rajouter des transitions, des animations, etc.

Aller plus loin

Vous avez remarqué que dans la correction, je ne stocke dans le dictionnaire d'état que les objets que j'ai moi-même créés, à savoir les objets Flux et Billet. Vous me direz, pourquoi ne pas mettre directement les objets SyndicationFeed et SyndicationItem ?

Eh bien pour une bonne raison, ceci introduit un bug pas forcément visible du premier coup. Ce bug survient lorsque notre application est désactivée, par exemple si l'on reçoit un coup de fil ou si on affiche le menu. Cela vient du fait que les objets SyndicationFeed et SyndicationItem ne sont pas sérialisables, ce qui fait que lorsqu'on passe en désactivé, la sérialisation échoue et fait planter l'application. En effet, le dictionnaire d'état est un emplacement qui est disponible tant que notre application est désactivée. Il est plus rapide d'accès que le répertoire local, mais ceci implique que les objets qu'on y stocke soient sérialisables.

Pour résoudre ce point, il suffit de ne pas mettre ces objets dans le dictionnaire d'état et par exemple de mettre plutôt directement nos objets Flux et Billet, comme ce que j'ai proposé en correction. Ceci est d'ailleurs plus logique.

L'autre solution est de vider le dictionnaire d'état dans l'événement de désactivation de l'application, avec par exemple :

Code : C#

```
private void Application_Deactivated(object sender,
DeactivatedEventArgs e)
{
    PhoneApplicationService.Current.State["ListeFlux"] = null;
}
```

Évidemment, il faut re-remplir le dictionnaire d'état dans la méthode d'activation de l'application :

Code : C#

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    // code à faire, utilisation du WebClient pour recharger tous
    les flux
}
```

Ou alors renvoyer l'utilisateur sur la page où s'effectue ce chargement, en utilisant le service de navigation.

Vous serez aussi d'accord avec moi, l'ajout de l'URL d'un flux n'est pas des plus commodes. Si vous voulez approfondir l'application, vous pouvez également essayer d'importer des flux à partir d'une autre source... depuis un autre syndicateur de flux, à partir [d'un fichier OPML](#).

Gérer l'orientation

Jusqu'à présent, nous avons toujours utilisé notre téléphone (enfin, surtout l'émulateur) dans sa position **portrait**, où les boutons sont orientés vers le bas, le téléphone ayant son côté le plus long dans le sens vertical, droit devant nous. Il peut se tenir aussi horizontalement, en mode **paysage**. Suivant les cas, les applications sont figées et ne peuvent se tenir que dans un sens. Dans d'autres cas, elles offrent la possibilité de tenir son téléphone dans plusieurs modes, révélant au passage une interface légèrement différente.

Si on tient le téléphone en mode paysage, il y a plus de place en largeur, il peut être pertinent d'afficher plus d'informations...

Les différentes orientations

L'orientation portrait est particulièrement adaptée à la visualisation des listes déroulantes pouvant contenir beaucoup d'éléments. Il devient également naturel de faire défiler les éléments vers le bas.

Par contre, en mode paysage, il y a plus de place en largeur. Si on garde le même type d'affichage, il risque d'y avoir un gros trou sur la droite, et la liste d'éléments deviendra sans doute un peu moins visible. De plus, dans ce mode-là, il semble plus naturel de faire défiler les éléments de gauche à droite. Et quel mode portrait adopter quand on tourne son téléphone vers la droite, pour avoir les boutons à gauche ou quand on tourne le téléphone vers la gauche et ainsi avoir les boutons à droite ?

Un téléphone Windows Phone sait détecter quand il change d'orientation. Il est capable de lever un événement nous permettant de réagir à ce changement d'orientation... mais n'anticipons pas et revenons à la base, la page. Si nous regardons en haut dans les propriétés de la page, nous pouvons voir que nous avons régulièrement créé des pages qui possèdent ces propriétés :

Code : XML

```
SupportedOrientations="Portrait" Orientation="Portrait"
```

Cela indique que l'application supporte le mode portrait et que la page démarre en mode portrait. Il est possible de changer les valeurs de ces propriétés. On peut par exemple affecter les valeurs suivantes à la propriété `SupportedOrientation` :

- Portrait
- Landscape
- PortraitOrLandscape

qui signifient respectivement portrait, paysage et portrait-ou-paysage. Ainsi, si l'on positionne cette dernière valeur, l'application va automatiquement réagir au changement d'orientation. Modifions donc cette propriété pour utiliser le mode portrait et paysage :

Code : XML

```
SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
```

et utilisons par exemple le code XAML suivant :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
    Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="MON APPLICATION"
        Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Hello World" Margin="9,-
        7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>
```

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock Text="Bonjour à tous" />
        <Button Content="Cliquez-moi" />
    </StackPanel>
</Grid>
</Grid>
```

Si nous démarrons l'application nous obtenons la figure suivante.



Emulateur en mode portrait avec les boutons de l'émulateur pour le

faire pivoter

Il est possible de simuler un changement d'orientation avec l'émulateur, il suffit de cliquer sur les boutons disponibles dans la barre en haut à droite de l'émulateur, comme indiqué sur l'image du dessus. L'écran est retourné et l'interface est automatiquement adaptée à la nouvelle orientation, comme vous pouvez le voir à la figure suivante.



L'émulateur en mode paysage

Ceci est possible grâce à la propriété que nous avons ajoutée. Les contrôles se redessinent automatiquement lors du changement d'orientation.

Remarquez que lorsqu'on retourne le téléphone dans l'autre mode paysage, le principe reste le même.

Ici, le changement d'orientation reste globalement élégant. Mais si nous avions utilisé un contrôle en positionnement absolu grâce à un Canvas par exemple, il est fort possible qu'il se retrouve à un emplacement non voulu lors du changement d'orientation. C'est la même chose avec un StackPanel, peut être que tous les éléments tiennent dans la hauteur en mode portrait, mais il faudra sûrement rajouter un ScrollViewer en mode paysage...

Par exemple, le XAML suivant en mode portrait semble avoir ses composants centrés :

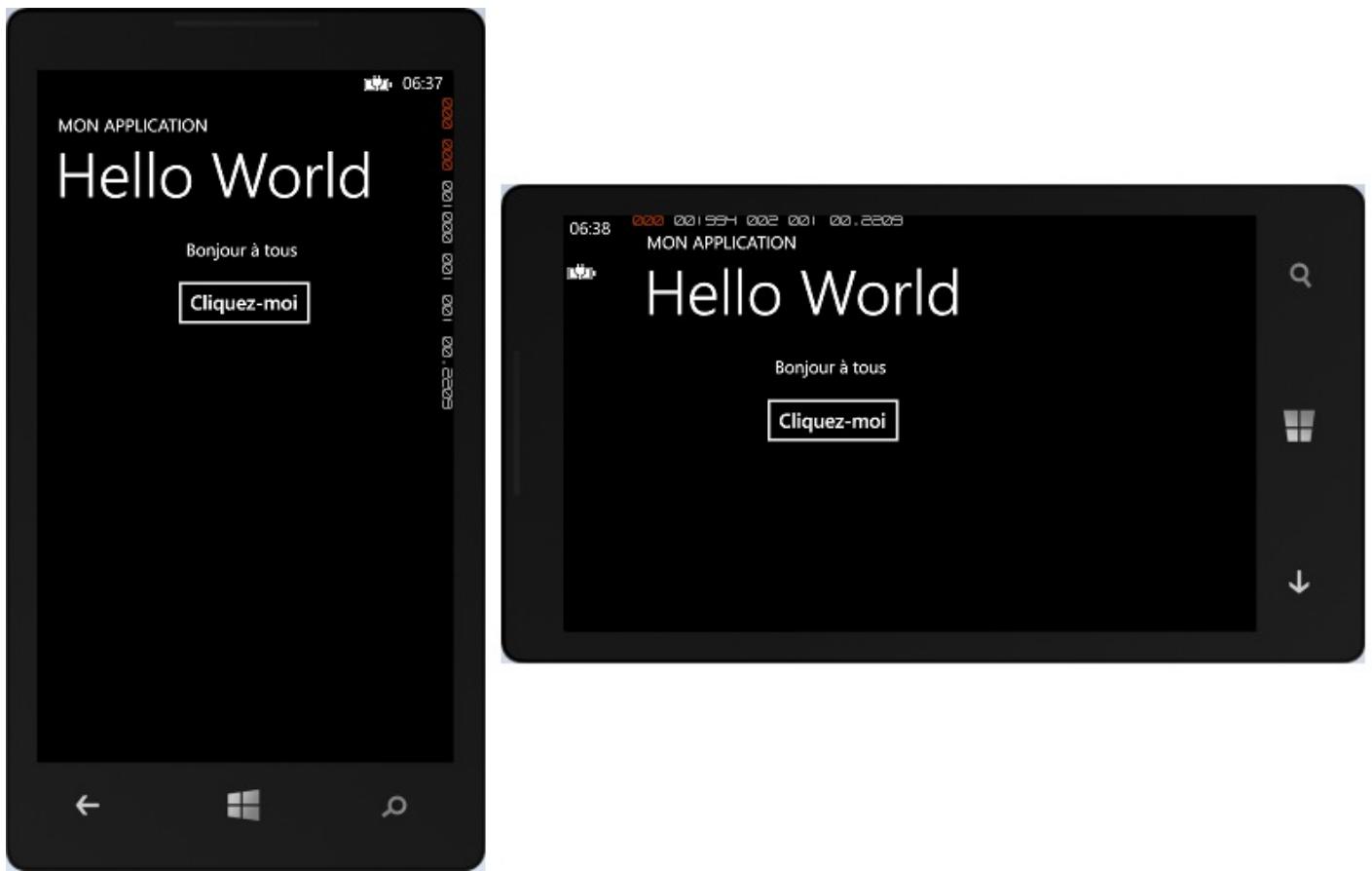
Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
    Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="MON APPLICATION"
        Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Hello World" Margin="9,-
        7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Canvas>
            <TextBlock x:Name="MonTextBlock" Text="Bonjour à tous"
            Canvas.Left="160" />
            <Button x:Name="MonBouton" Content="Cliquez-moi"
            Canvas.Left="140" Canvas.Top="40" />
        </Canvas>
    </Grid>
</Grid>
```

Alors que si on le retourne, on peut voir un beau trou à droite (voir la figure suivante).



Positionnement décalé suivant l'orientation

Déetecter les changements d'orientation

Il est possible de détecter le type d'orientation d'un téléphone en consultant la propriété Orientation d'une page.

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();

        switch (Orientation)
        {
            case PageOrientation.Landscape:
            case PageOrientation.LandscapeLeft:
            case PageOrientation.LandscapeRight:
                MessageBox.Show("Mode paysage");
                break;
            case PageOrientation.Portrait:
            case PageOrientation.PortraitDown:
            case PageOrientation.PortraitUp:
                MessageBox.Show("Mode portrait");
                break;
        }
    }
}
```

Que l'on peut également simplifier de cette façon :

Code : C#

```
if ((Orientation & PageOrientation.Landscape) ==
PageOrientation.Landscape)
```

```

    {
        MessageBox.Show("Mode paysage");
    }
else
{
    MessageBox.Show("Mode portrait");
}

```

Et, bien souvent plus utile, il est possible d'être notifié des changements d'orientation. Cela pourra permettre par exemple de réaliser des ajustements. Pour être prévenu, il suffit de substituer la méthode `OnOrientationChanged` :

Code : C#

```

protected override void
OnOrientationChanged(OrientationChangedEventArgs e)
{
    switch (e.Orientation)
    {
        case PageOrientation.Landscape:
        case PageOrientation.LandscapeLeft:
        case PageOrientation.LandscapeRight:
            // faire des choses pour le mode paysage
            break;
        case PageOrientation.Portrait:
        case PageOrientation.PortraitDown:
        case PageOrientation.PortraitUp:
            // faire des choses pour le mode portrait
            break;
    }
    base.OnOrientationChanged(e);
}

```

que l'on peut à nouveau simplifier en :

Code : C#

```

protected override void
OnOrientationChanged(OrientationChangedEventArgs e)
{
    if ((e.Orientation & PageOrientation.Landscape) ==
PageOrientation.Landscape)
    {
        // faire des choses pour le mode paysage
    }
    else
    {
        //faire des choses pour le mode portrait
    }
    base.OnOrientationChanged(e);
}

```

Prenons notre précédent exemple. Nous pouvons modifier les propriétés de dépendance `Canvas.Left` pour avoir :

Code : C#

```

protected override void
OnOrientationChanged(OrientationChangedEventArgs e)
{
    if ((e.Orientation & PageOrientation.Landscape) ==

```

```
        PageOrientation.Landscape)
    {
        MonTextBlock.SetValue(Canvas.LeftProperty, 320.0);
        MonBouton.SetValue(Canvas.LeftProperty, 300.0);
    }
    else
    {
        MonTextBlock.SetValue(Canvas.LeftProperty, 160.0);
        MonBouton.SetValue(Canvas.LeftProperty, 140.0);
    }
    base.OnOrientationChanged(e);
}
```

Ainsi, nous « recentrons » les contrôles à chaque changement d'orientation.

Stratégies de gestion d'orientation

Bien sûr, l'exemple précédent est un mauvais exemple, vous l'aurez compris. La bonne solution est d'utiliser correctement le système de placement du XAML pour centrer nos composants, avec uniquement du XAML :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <TextBlock x:Name="MonTextBlock" Text="Bonjour à tous"
            HorizontalAlignment="Center" />
        <Button x:Name="MonBouton" Content="Cliquez-moi"
            HorizontalAlignment="Center" />
    </StackPanel>
</Grid>
```

Mais modifier des propriétés de contrôles permet quand même de pouvoir faire des choses intéressantes. Prenez l'exemple suivant où nous positionnons des contrôles dans une grille :

Code : XML

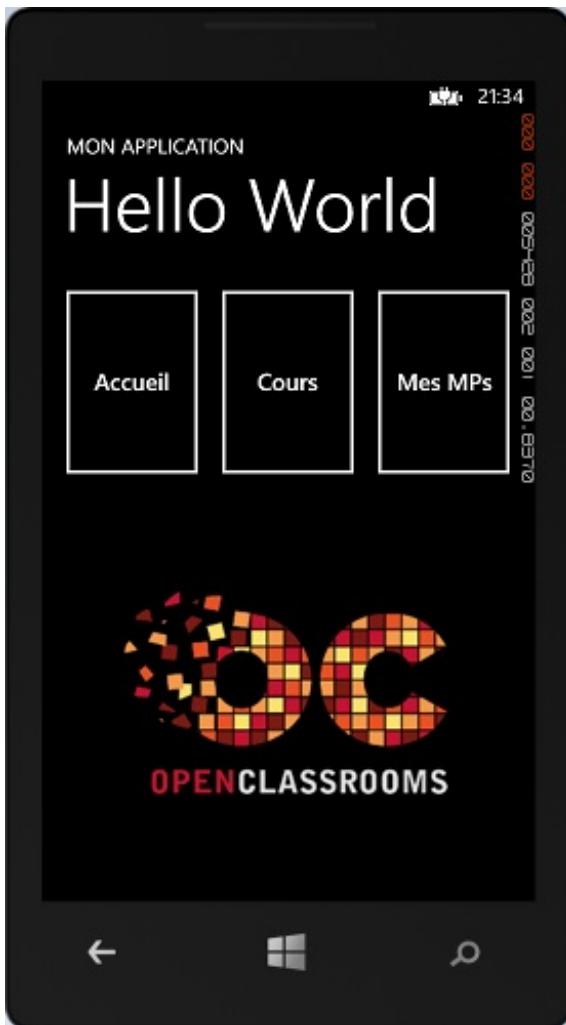
```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
        Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="MON APPLICATION"
            Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Hello World" Margin="9,-
            7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Button x:Name="Accueil" Content="Accueil" />
        <Button Grid.Column="1" x:Name="Cours" Content="Cours" />
        <Button Grid.Column="2" x:Name="MesMPs" Content="Mes MPs" />
    </Grid>
</Grid>
```

```
<Image x:Name="ImageSdz" Grid.Row="1" Grid.RowSpan="2"
Grid.ColumnSpan="3" Source="http://open-e-education-
2013.openclassrooms.com/img/logos/logo-openclassrooms.png" />
</Grid>
</Grid>
```

Observez la figure suivante pour le rendu.



La grille en mode portrait

Nous pouvons utiliser la technique précédente pour changer la disposition des contrôles dans la grille afin de produire un autre rendu en mode paysage. Pour cela, voyez le code-behind suivant :

Code : C#

```
protected override void
OnOrientationChanged(OrientationChangedEventArgs e)
{
    if ((e.Orientation & PageOrientation.Landscape) ==
PageOrientation.Landscape)
    {
        Grid.SetRow(ImageSdz, 0);
        Grid.SetRow(Accueil, 0);
        Grid.SetRow(Cours, 1);
        Grid.SetRow(MesMPs, 2);

        Grid.SetColumn(ImageSdz, 0);
        Grid.SetColumn(Accueil, 2);
        Grid.SetColumn(Cours, 2);
        Grid.SetColumn(MesMPs, 2);
    }
}
```

```
        Grid.SetRowSpan(ImageSdz, 3);
        Grid.SetColumnSpan(ImageSdz, 2);
    }
    else
    {
        Grid.SetRow(ImageSdz, 1);
        Grid.SetRow(Accueil, 0);
        Grid.SetRow(Cours, 0);
        Grid.SetRow(MesMPs, 0);

        Grid.SetColumn(ImageSdz, 0);
        Grid.SetColumn(Accueil, 0);
        Grid.SetColumn(Cours, 1);
        Grid.SetColumn(MesMPs, 2);

        Grid.SetRowSpan(ImageSdz, 2);
        Grid.SetColumnSpan(ImageSdz, 3);
    }
    base.OnOrientationChanged(e);
}
```

Ainsi, on déplace les contrôles d'une colonne à l'autre, on arrange la fusion de certaines lignes, etc. Regardez la figure suivante pour le rendu.



La grille réordonnée en mode

paysage

Et voilà un positionnement complètement différent, plutôt sympathique. N'oubliez bien sûr pas de repositionner les contrôles en mode portrait également, sinon ils garderont la disposition précédente.

Cependant, il est parfois judicieux de ne pas gérer le changement d'orientation et de forcer l'application à une unique orientation. Beaucoup de jeux sont bloqués en mode paysage et beaucoup d'applications, notamment dès qu'il y a un panorama ou un pivot forcent l'orientation en portrait. C'est un choix qui peut se justifier et il vaut parfois mieux proposer une expérience utilisateur optimale dans un mode, qu'une expérience bancale dans les deux modes. Nous avons vu qu'il suffisait de positionner la propriété `SupportedOrientations` à `Portrait` ou `Landscape`. Sachant que ceci peut également se faire via le code-behind, si par exemple toute l'application gère la double orientation, mais qu'un écran en particulier peut uniquement être utilisé en mode portrait.

Si vous le pouvez, c'est également très pratique que vos applications gèrent différentes orientations. La solution la plus pratique est d'utiliser la mise en page automatique du XAML, comme ce que nous avons fait en centrant le bouton et la zone de texte grâce à la propriété `HorizontalAlignment`. La première chose à faire est de toujours utiliser des conteneurs qui peuvent se redimensionner automatiquement, d'éviter de fixer des largeurs ou des hauteurs et de penser à des contrôles pouvant faire défiler leur contenu (`ListBox`, `ScrollView`, etc.). Cela peut par contre parfois donner des rendus pas toujours esthétiques, avec un bouton qui prend toute la longueur de l'écran par exemple. Pensez-bien au design de vos écrans et n'oubliez pas de tester dans toutes les orientations possibles.

Comme on l'a déjà vu, vous pouvez également faire vos ajustements lors de la détection du changement d'orientation. Vous pouvez modifier la hauteur/largeur d'un contrôle, changer son positionnement, en ajouter un nouveau ou au contraire, supprimer un contrôle qui ne rentre plus. L'inconvénient de cette technique est qu'elle requiert plus de code et qu'elle est plus compliquée à mettre en place avec une approche MVVM.

Vous pouvez également rediriger l'utilisateur sur une page adaptée à un mode précis lorsqu'il y a un changement d'orientation. Par exemple sur la `PagePortrait.xaml` :

Code : C#

```
protected override void  
OnOrientationChanged(OrientationChangedEventArgs e)  
{  
    if ((e.Orientation & PageOrientation.Landscape) ==  
        PageOrientation.Landscape)  
    {  
        NavigationService.Navigate(new Uri("/PagePaysage.xaml",  
        UriKind.Relative));  
    }  
    base.OnOrientationChanged(e);  
}
```

et sur la `PagePaysage.xaml` :

Code : C#

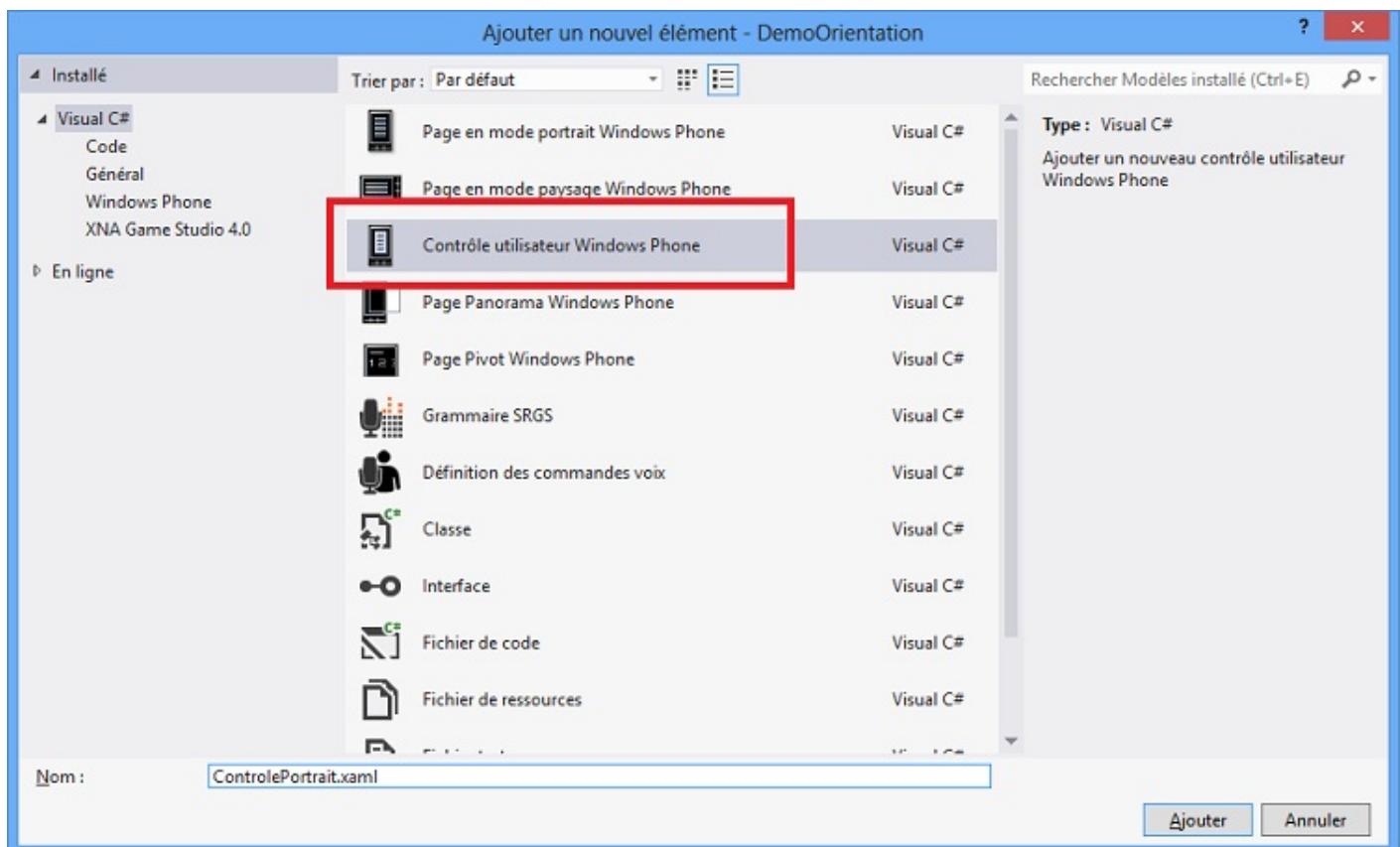
```
protected override void  
OnOrientationChanged(OrientationChangedEventArgs e)  
{  
    if (!((e.Orientation & PageOrientation.Landscape) ==  
          PageOrientation.Landscape))  
    {  
        NavigationService.Navigate(new Uri("/PagePortrait.xaml",  
        UriKind.Relative));  
    }  
    base.OnOrientationChanged(e);  
}
```

L'inconvénient est que cela fait empiler beaucoup de pages dans la navigation. Il faut alors retirer la page précédente de la pile de navigation. Il suffit de redéfinir la méthode `OnNavigatedTo` et d'utiliser la méthode `RemoveBackEntry` :

Code : C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)  
{  
    NavigationService.RemoveBackEntry();  
    base.OnNavigatedTo(e);  
}
```

Dans le même style, nous pouvons créer des contrôles utilisateurs dédiés à une orientation, que nous affichons et masquons en fonction de l'orientation. Un contrôle utilisateur est un bout de page que nous pouvons réutiliser dans n'importe quelle page. Pour créer un nouveau contrôle utilisateur, on utilise le modèle « Contrôle utilisateur Windows Phone » lors d'un clic droit sur le projet, ajouter, nouvel élément (voir la figure suivante).



Ajout d'un contrôle utilisateur

Le contrôle utilisateur portrait pourra contenir le XAML suivant :

Code : XML

```
<Grid x:Name="LayoutRoot">
    <StackPanel>
        <TextBlock Text="Je suis en mode portrait"
HorizontalAlignment="Center" />
    </StackPanel>
</Grid>
```

Alors que le contrôle utilisateur paysage contiendra :

Code : XML

```
<Grid x:Name="LayoutRoot">
    <StackPanel>
        <TextBlock Text="Je suis en mode paysage"
HorizontalAlignment="Center" />
    </StackPanel>
</Grid>
```

Nous pourrons alors avoir le code-behind suivant dans la page principale :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private ControlePortrait controlePortrait;
```

```
private ControlePaysage controlePaysage;

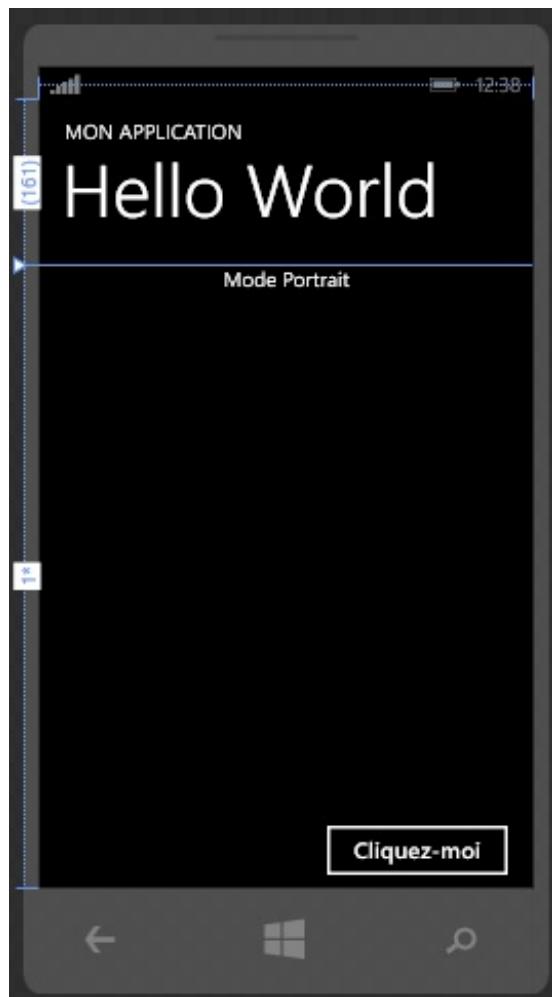
public MainPage()
{
    InitializeComponent();
    controlePortrait = new ControlePortrait();
    controlePaysage = new ControlePaysage();

    ContentPanel.Children.Add(controlePortrait);
}

protected override void
OnOrientationChanged(OrientationChangedEventArgs e)
{
    if ((e.Orientation & PageOrientation.Landscape) ==
        PageOrientation.Landscape)
    {
        ContentPanel.Children.Remove(controlePortrait);
        ContentPanel.Children.Add(controlePaysage);
    }
    else
    {
        ContentPanel.Children.Remove(controlePaysage);
        ContentPanel.Children.Add(controlePortrait);
    }
    base.OnOrientationChanged(e);
}
```

Le principe est d'instancier les deux contrôles et de les ajouter ou de les retirer à la grille en fonction de l'orientation. L'inconvénient, comme pour la précédente stratégie, est qu'il faut potentiellement dupliquer pas mal de code dans chacun des contrôles.

Enfin, nous pouvons utiliser le gestionnaire d'état. De la même façon que nous l'avons vu dans la partie précédente pour les contrôles, une page peut avoir plusieurs états. Il suffit de considérer que l'état normal est celui en mode portrait et que nous allons créer un état pour le mode paysage. Pour cela, le plus simple est d'utiliser Blend. Démarrons-le et commençons à créer notre page (voir la figure suivante).



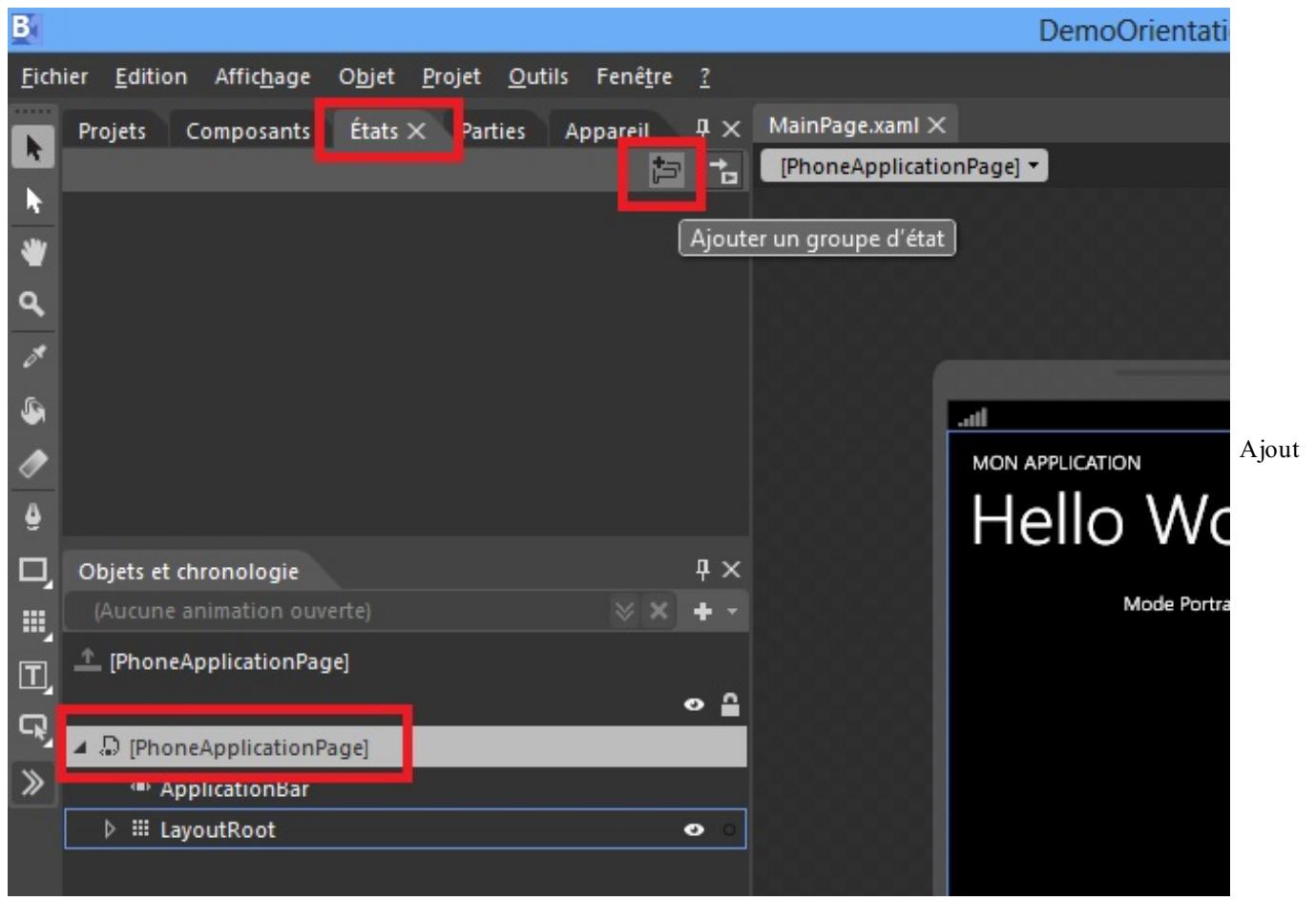
Page portrait dans Blend

qui correspond au XAML suivant :

Code : XML

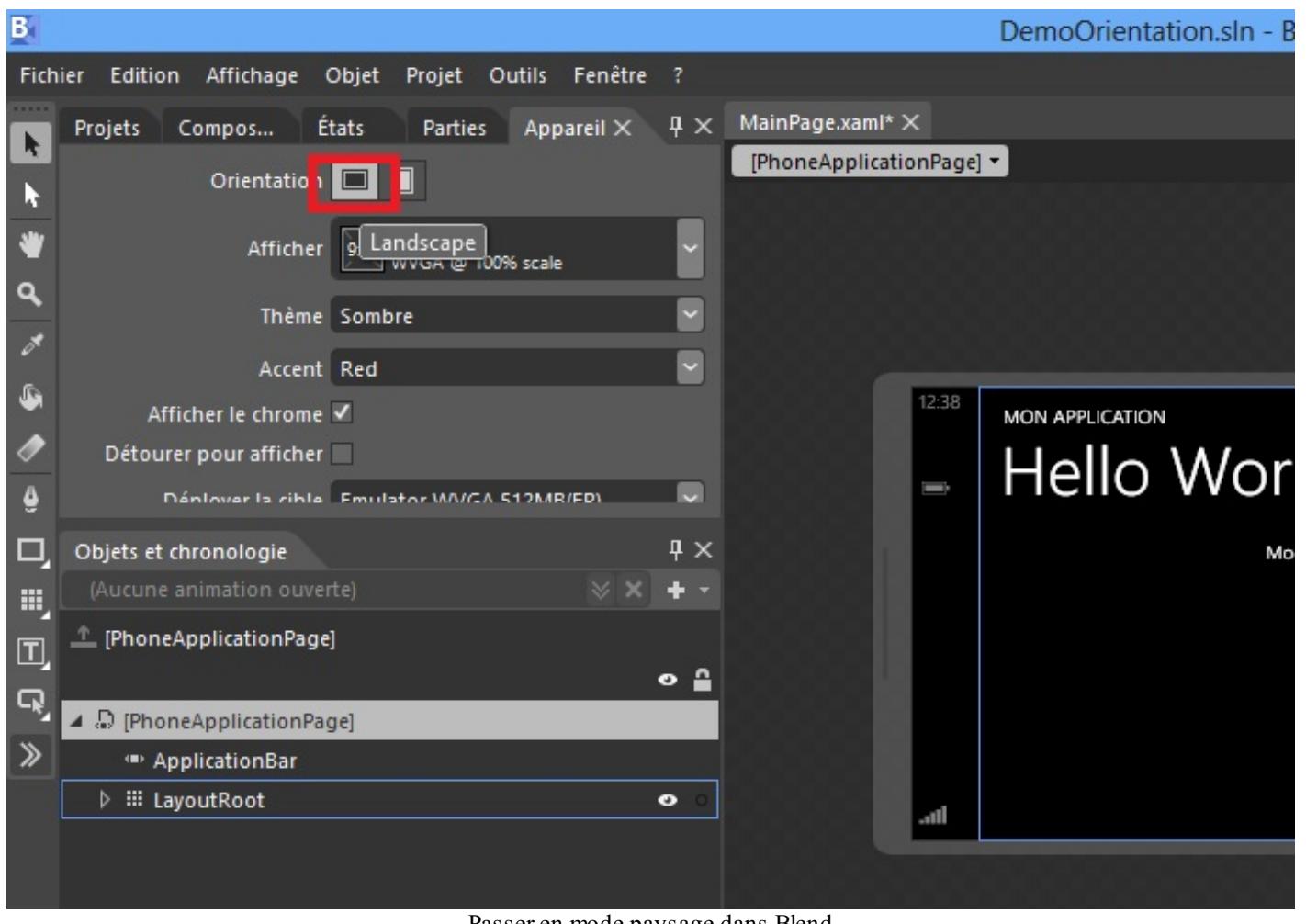
```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <TextBlock Text="Mode Portrait" HorizontalAlignment="Center" />
    <Button Content="Cliquez-moi" Width="200"
        VerticalAlignment="Bottom" HorizontalAlignment="Right" />
</Grid>
```

Maintenant, nous allons créer un nouvel état. Sélectionnez la page dans l'arbre visuel (en bas à gauche) puis dans l'onglet état, cliquez sur le petit bouton à droite permettant d'ajouter un groupe d'état, comme indiqué sur la figure suivante.



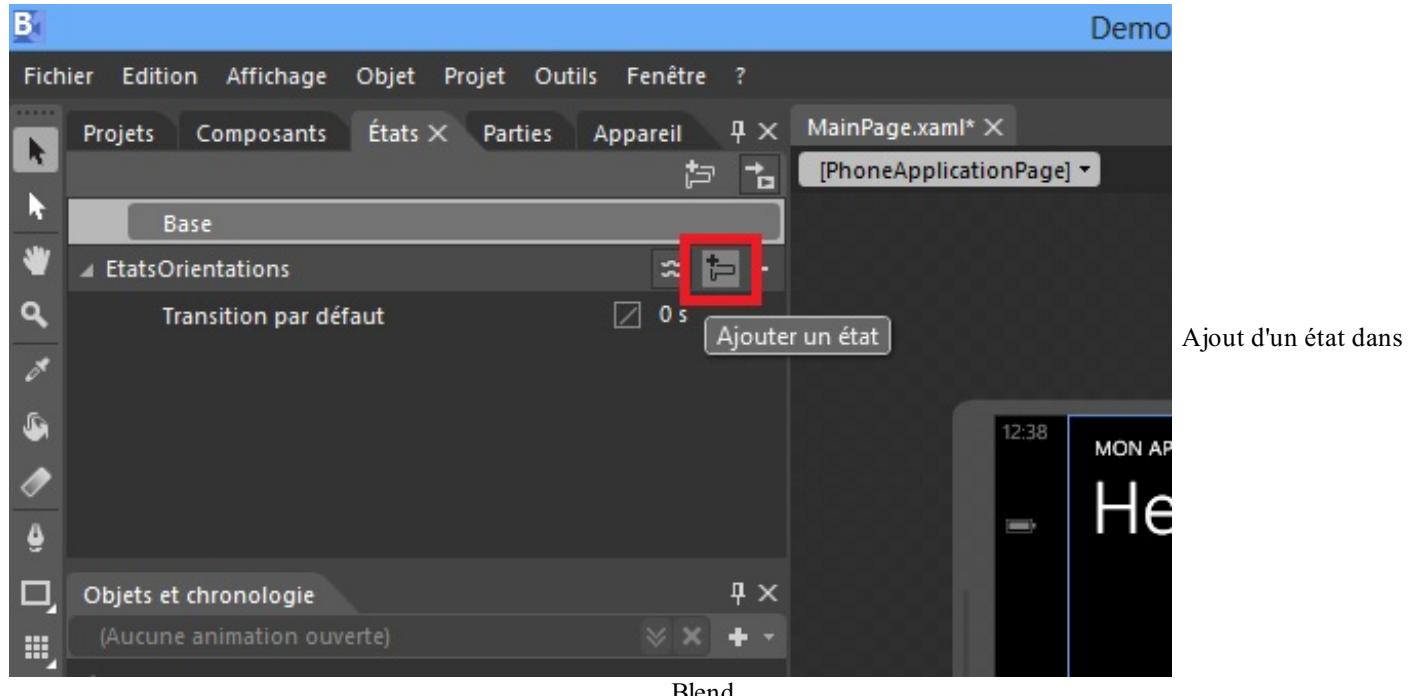
Nous pouvons le nommer par exemple `EtatsOrientations`.

Avant de pouvoir créer un état propre au mode paysage, nous allons positionner le designer en mode paysage. Pour cela il faut aller dans l'onglet `Appareil` qui est un peu plus à droite que l'onglet états et basculer en `landscape` (voir la figure suivante).



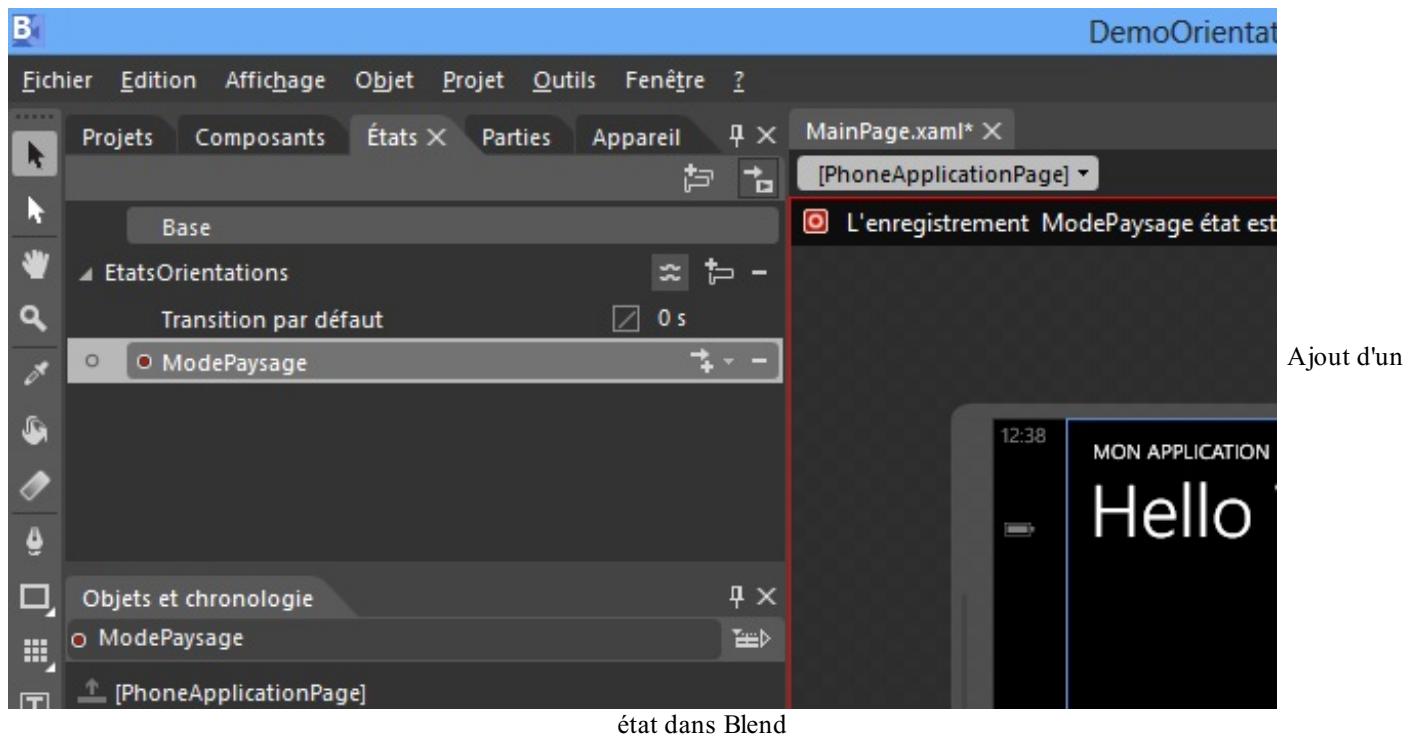
Passer en mode paysage dans Blend

Revenez dans l'onglet état, et cliquez à présent à droite sur « ajouter un état », comme indiqué à la figure suivante.



Blend

Nous pouvons appeler ce nouvel état ModePaysage (voir la figure suivante).



Nous pouvons voir que l'état est en mode enregistrement, c'est-à-dire que nous allons pouvoir maintenant modifier la position des contrôles pour créer notre écran en mode paysage. Changez la disposition et changez la propriété Text du TextBlock pour afficher Mode Paysage, comme indiqué à la figure suivante.



Vous pouvez maintenant sauvegarder vos modifications, fermer Blend et revenir dans Visual Studio. Remarquons que Blend nous a modifié notre XAML, notamment pour rajouter de quoi faire une animation des contrôles :

Code : XML

```
<TextBlock x:Name="textBlock" Text="Mode Portrait"
HorizontalAlignment="Center" RenderTransformOrigin="0.5,0.5" >
<TextBlock.RenderTransform>
```

```
<CompositeTransform/>
</TextBlock.RenderTransform>
</TextBlock>
<Button x:Name="button" Content="Cliquez-moi" Width="200"
VerticalAlignment="Bottom" HorizontalAlignment="Right"
RenderTransformOrigin="0.5,0.5" >
    <Button.RenderTransform>
        <CompositeTransform/>
    </Button.RenderTransform>
</Button>
```

De même, il a rajouté de quoi gérer les états :

Code : XML

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="EtatsOrientations">
        <VisualState x:Name="ModePaysage">
            <Storyboard>
                <DoubleAnimation Duration="0" To="-254"
Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.Trans
Storyboard.TargetName="textBlock" d:IsOptimized="True"/>
                <DoubleAnimation Duration="0" To="182"
Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.Trans
Storyboard.TargetName="textBlock" d:IsOptimized="True"/>
                <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(TextBlock.Text
Storyboard.TargetName="textBlock">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="Mode Paysage"/>
                </ObjectAnimationUsingKeyFrames>
                <DoubleAnimation Duration="0" To="-300"
Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.Trans
Storyboard.TargetName="button" d:IsOptimized="True"/>
                <DoubleAnimation Duration="0" To="-208"
Storyboard.TargetProperty="(UIElement.RenderTransform).(CompositeTransform.Trans
Storyboard.TargetName="button" d:IsOptimized="True"/>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Nous pouvons constater qu'il s'opère des changements de coordonnées via translations, ainsi que la modification du texte du TextBlock. À cet état, ajoutons un état ModePortrait vide pour signifier l'état de départ, le plus simple ici est de le faire dans le XAML, avec :

Code : XML

```
<VisualStateGroup x:Name="EtatsOrientations">
    <VisualState x:Name="ModePortrait" />
    <VisualState x:Name="ModePaysage">
        <Storyboard>
            ...
        </Storyboard>
    </VisualState>
</VisualStateGroup>
```

Enfin, vous allez avoir besoin d'indiquer que vous gérez le multi-orientation et que vous démarrez en mode portrait avec :

Code : XML

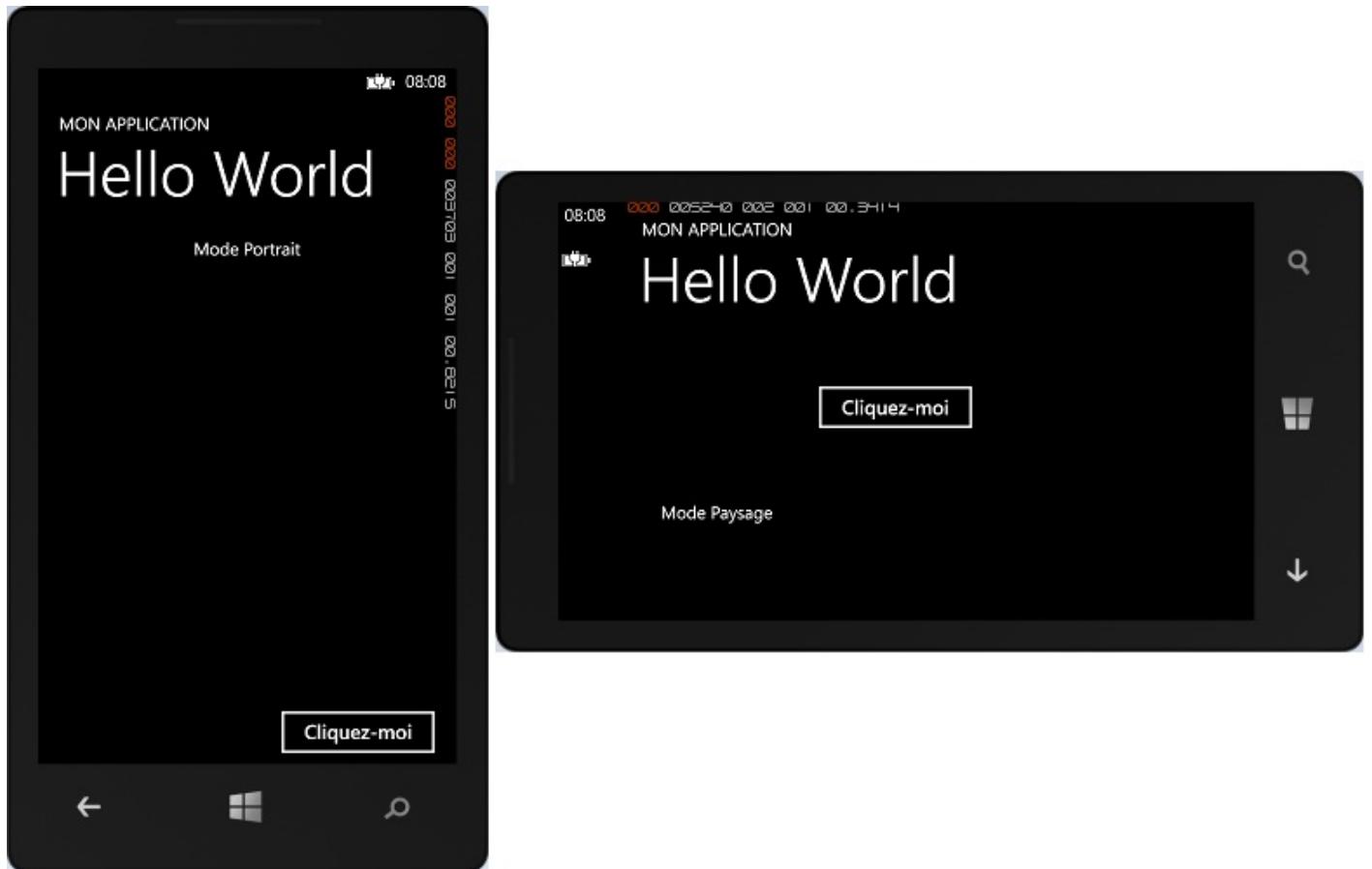
```
SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
```

Puis maintenant, il faudra réagir à un changement d'orientation et modifier l'état de la page grâce au VisualStateManager :

Code : C#

```
protected override void  
OnOrientationChanged(OrientationChangedEventArgs e)  
{  
    if ((e.Orientation & PageOrientation.Landscape) ==  
        PageOrientation.Landscape)  
    {  
        VisualStateManager.GoToState(this, "ModePaysage", true);  
    }  
    else  
    {  
        VisualStateManager.GoToState(this, "ModePortrait", true);  
    }  
  
    base.OnOrientationChanged(e);  
}
```

Maintenant, vous pouvez changer l'orientation de l'émulateur ; vous pouvez voir le rendu à la figure suivante.



Changement d'état de la page lors du changement d'orientation

Et voilà ! 😊

Si vous décidez de gérer les deux orientations dans votre application, c'est une bonne idée de prévoir une configuration permettant de figer l'orientation dans un unique sens. Par exemple, il est très désagréable de voir son



écran se tourner quand on utilise son téléphone en position allongé. Pouvoir figer l'orientation à la demande de l'utilisateur s'avère être un plus pour les pauvres utilisateurs fatigués ou alités. 😊

- Un téléphone sait détecter les changements d'orientation, ainsi une application peut s'exécuter en mode portrait ou en mode paysage.
- Pour être averti d'un changement d'orientation, il suffit de redéfinir la méthode `OnOrientationChanged`.
- Il faut toujours tester à fond notre application avec différentes orientations si l'on souhaite gérer le changement d'orientation.

Gérer les multiples résolutions

Alors que Windows Phone 7.X ne supportait que des téléphones ayant la résolution 480x800, Windows Phone 8 permet maintenant de supporter 3 résolutions. Ceci est un avantage pour les utilisateurs qui peuvent ainsi avoir des téléphones avec des résolutions différentes, mais cela devient un inconvénient pour le développeur qui doit maintenant faire en sorte que son application fonctionne pour toutes les résolutions des appareils.

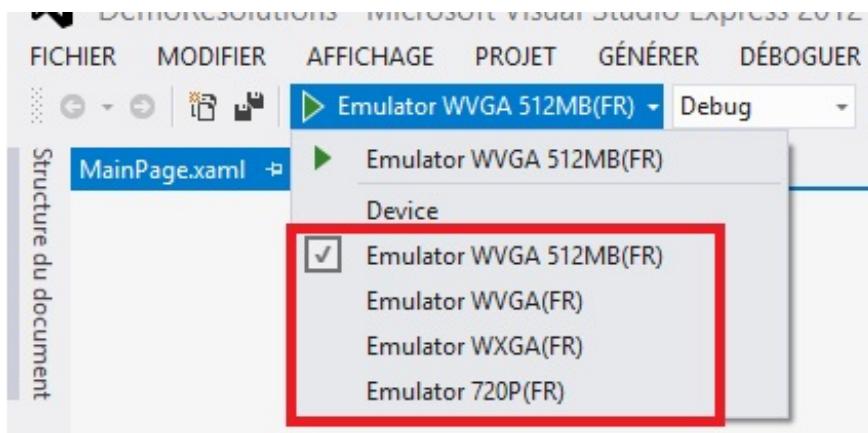
Voyons maintenant ce qu'il faut savoir.

Les différentes résolutions

Windows Phone 8 supporte trois résolutions :

| Résolution | Taille | Format |
|------------|----------|--------|
| WVGA | 480x800 | 15/9 |
| WXGA | 768x1280 | 15/9 |
| 720p | 720x1280 | 16/9 |

C'est aussi le cas de l'émulateur, que nous pouvons démarrer suivant plusieurs résolutions, en la changeant dans la liste déroulante, comme indiqué sur la figure suivante.



Les différentes résolutions de l'émulateur

Voici à présent le même écran présenté dans les 3 différentes résolutions, avec de gauche à droite WVGA, WXGA et 720P (voir la figure suivante).



Le même écran dans les trois résolutions



Pour réaliser cet exemple, je me suis servi d'un contrôle du Windows Phone Toolkit que nous allons découvrir dans quelques chapitres : le `WrapPanel`. J'ai positionné une série de boutons qui font tous 90 pixels de longueur et de largeur.

La première chose à remarquer est que le rendu est le même entre la résolution WVGA et WXGA, nonobstant la différence de résolution. En effet, le format étant le même, à savoir du 15/9 ième, Windows Phone prend automatiquement en charge la mise à l'échelle de la page et ramène finalement la résolution 768x1280 à une taille de 480x800.

Ce n'est pas le cas pour la résolution 720p, qui est du 16/9 ième, et où la mise à l'échelle automatique donne du 480x853. On peut voir en effet sur la copie d'écran que nous avons un peu plus de choses qui sont affichées en bas du 720p.

Gérer plusieurs résolutions

Alors, comment gérer toutes ces belles résolutions ?

Vous aurez sûrement deviné que c'est grossièrement le même principe que pour les différentes orientations. Il y a plusieurs techniques pour gérer les résolutions :

- Utiliser le redimensionnement automatique des contrôles, n'hésitez pas à utiliser l'étoile (*) dans les grilles ou la valeur auto, les alignements centrés, etc.
- Faire un ajustement avec le code-behind.
- Rediriger vers les bonnes pages ou utiliser les bons contrôles utilisateurs en fonction de la résolution.
- Modifier l'état de la page en fonction de la résolution.
- ...

La seule différence est qu'il ne faut pas le faire en réception à un événement de changement de taille, car en effet il est très rare qu'un téléphone puisse changer de taille d'écran en cours d'utilisation. 😊 Par contre, il est possible de détecter la résolution au lancement de l'application afin de faire les ajustements adéquats.

Il suffit d'utiliser les propriétés présentes dans `Application.Current.Host.Content`, comme :

- ActualHeight qui donne la hauteur de l'écran
- ActualWidth qui donne la largeur de l'écran
- ScaleFactor qui donne le facteur d'échelle

Et nous aurons pour chaque résolution, les valeurs suivantes :

| Résolution | Hauteur | Largeur | Facteur d'échelle |
|------------|---------|---------|-------------------|
| WVGA | 800 | 480 | 100 |
| WXGA | 800 | 480 | 160 |
| 720p | 853 | 480 | 150 |

Ces valeurs vont donc nous permettre de détecter la résolution du téléphone.

Les images

Étant donné que nous avons différentes résolutions, il se pose la question des images. Mon image va-t-elle être jolie dans toutes les résolutions ?

La première idée tentante serait d'inclure 3 images différentes, chacune optimisée pour une résolution d'écran. Nous pourrions alors écrire un petit helper qui nous détecterait la résolution et nous permettrait de mettre la bonne image au bon moment :

Code : C#

```
public static class ResolutionHelper
{
    public static bool EstWvga
    {
        get { return Application.Current.Host.Content.ActualHeight
== 800 && Application.Current.Host.Content.ScaleFactor == 100; }
    }

    public static bool EstWxga
    {
        get { return Application.Current.Host.Content.ScaleFactor ==
160; }
    }

    public static bool Est720p
    {
        get { return Application.Current.Host.Content.ScaleFactor ==
150; }
    }
}
```

Il serait ainsi facile de charger telle ou telle image en fonction de la résolution :

Code : C#

```
if (ResolutionHelper.EstWvga)
    MonImage.Source = new BitmapImage(new
Uri("/Assets/Images/Wvga/fond.png", UriKind.Relative));
if (ResolutionHelper.EstWxga)
    MonImage.Source = new BitmapImage(new
Uri("/Assets/Images/Wxga/fond.png", UriKind.Relative));
if (ResolutionHelper.Est720p)
    MonImage.Source = new BitmapImage(new
Uri("/Assets/Images/720p/fond.png", UriKind.Relative));
```

C'est une bonne solution sauf que cela augmentera considérablement la taille de notre .xap et la consommation mémoire de notre application. Étant donné que le facteur est le même entre WVGA et WXGA, il est possible de n'inclure que les images optimisées pour la résolution WXGA et de laisser le système redimensionner automatiquement les images pour la résolution WVGA. De même, si on peut faire en sorte de ne pas inclure d'images en 720p et que ce soit la mise en page qui soit légèrement différente pour cette résolution, c'est toujours une image en moins de gagnée dans le .xap final et en mémoire dans l'application. Cependant, il peut parfois être justifié d'inclure les images en différentes résolutions et d'utiliser notre petit helper.

À noter que nous pouvons utiliser notre helper pour choisir la bonne image dans le XAML également. Créons une nouvelle classe :

Code : C#

```
public class ChoisisseurImage
{
    public Uri MonImageDeFond
    {
        get
        {
            if (ResolutionHelper.EstWxga)
                return new Uri("/Assets/Images/Wxga/fond.png",
UriKind.Relative);
            if (ResolutionHelper.Est720p)
                return new Uri("/Assets/Images/720p/fond.png",
UriKind.Relative);
            return new Uri("/Assets/Images/Wvga/fond.png",
UriKind.Relative);
        }
    }

    public Uri ImageRond
    {
        get
        {
            if (ResolutionHelper.EstWxga)
                return new Uri("/Assets/Images/Wxga/rond.png",
UriKind.Relative);
            if (ResolutionHelper.Est720p)
                return new Uri("/Assets/Images/720p/rond.png",
UriKind.Relative);
            return new Uri("/Assets/Images/Wvga/rond.png",
UriKind.Relative);
        }
    }
}
```

Que nous allons déclarer en ressources de notre application :

Code : XML

```
<Application
    x:Class="DemoResolution.App"
    ...
    xmlns:local="clr-namespace:DemoResolution">

    <Application.Resources>
        <local:ChoisisseurImage x:Key="ChoisisseurImageResource"/>
    </Application.Resources>

    [...]
</Application>
```

Ainsi, nous pourrons utiliser cette ressource dans le XAML :

Code : XML

```
<Image Source="{Binding MonImageDeFond, Source={StaticResource ChoisisseurImageResource}}"/>
<Image Source="{Binding ImageRond, Source={StaticResource ChoisisseurImageResource}}"/>
```

L'image de l'écran d'accueil

Vous vous rappelez que nous avons vu comment ajouter une image pour notre écran d'accueil, le fameux splash screen ? Vous pouvez également fournir des images de splash screen dans différentes résolutions, il suffit d'utiliser trois images dans les bonnes résolutions portant ces noms :

- SplashScreenImage.Screen-WVGA.jpg
- SplashScreenImage.Screen-WXGA.jpg
- SplashScreenImage.Screen-720p.jpg

Vous devez quand même garder l'image SplashScreenImage.jpg qui est l'image par défaut.

- Windows Phone 8 apporte 3 résolutions différentes que nous devons gérer en tant que développeur pour offrir la meilleure qualité d'application possible.
- Les différentes stratégies pour gérer les résolutions multiples sont les mêmes que pour gérer les différentes orientations.
- Il est possible d'utiliser des images de différentes résolutions dans son application mais gardez à l'esprit qu'elles occuperont de la mémoire et augmenteront la taille du .xap.

L'application Bar

Nous ne l'avons pas encore vue, mais vous la connaissez sûrement si vous possédez un téléphone équipé de Windows Phone. La barre d'application, si elle est présente, est située en bas de l'écran et possède des boutons que nous pouvons cliquer. Elle fait office plus ou moins de menu, accessible tout le temps, un peu comme le menu en haut des fenêtres qui existaient sur nos vieilles applications Windows.

Elle peut s'avérer très pratique et est plutôt simple d'accès, deux bonnes raisons pour pousser un peu plus loin sa découverte.

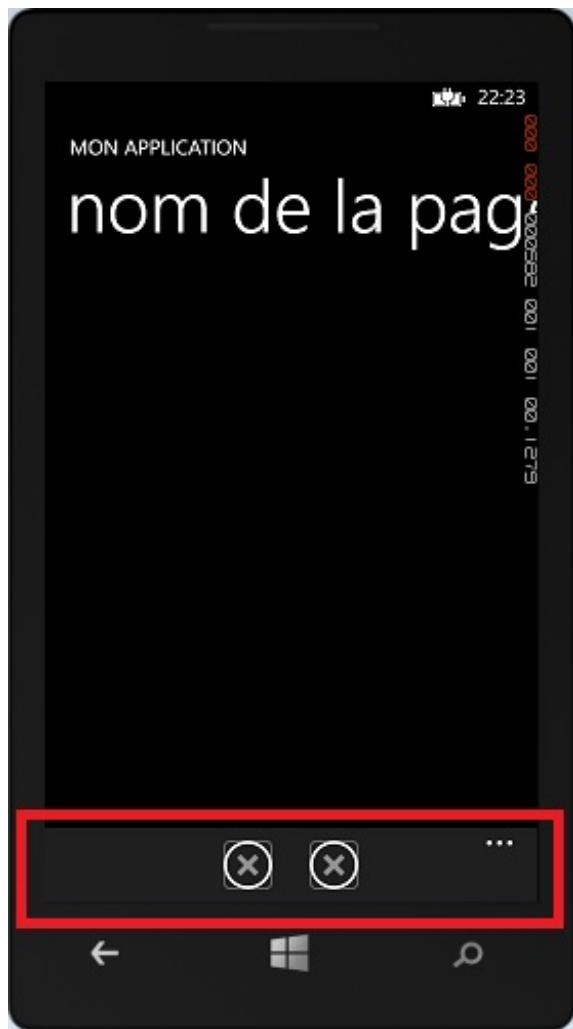
Présentation et utilisation

Si vous utilisez le SDK pour Windows Phone 7, vous l'avez sûrement déjà vue dans le code XAML sans vraiment y faire attention. Il y a un exemple d'utilisation commenté dans chaque nouvelle page créée :

Code : XML

```
<!--Exemple de code illustrant l'utilisation d'ApplicationBar-->
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button1.png" Text="Bouton 1"/>
        <shell:ApplicationBarIconButton
IconUri="/Images/appbar_button2.png" Text="Bouton 2"/>
        <shell:ApplicationBar.MenuItems>
            <shell:ApplicationBarMenuItem Text="ÉlémentMenu 1"/>
            <shell:ApplicationBarMenuItem Text="ÉlémentMenu 2"/>
        </shell:ApplicationBar.MenuItems>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Si vous dé-commentez ces lignes ou que vous les recopiez dans votre page Windows Phone 8, que vous démarrez l'émulateur, vous aurez la figure suivante.



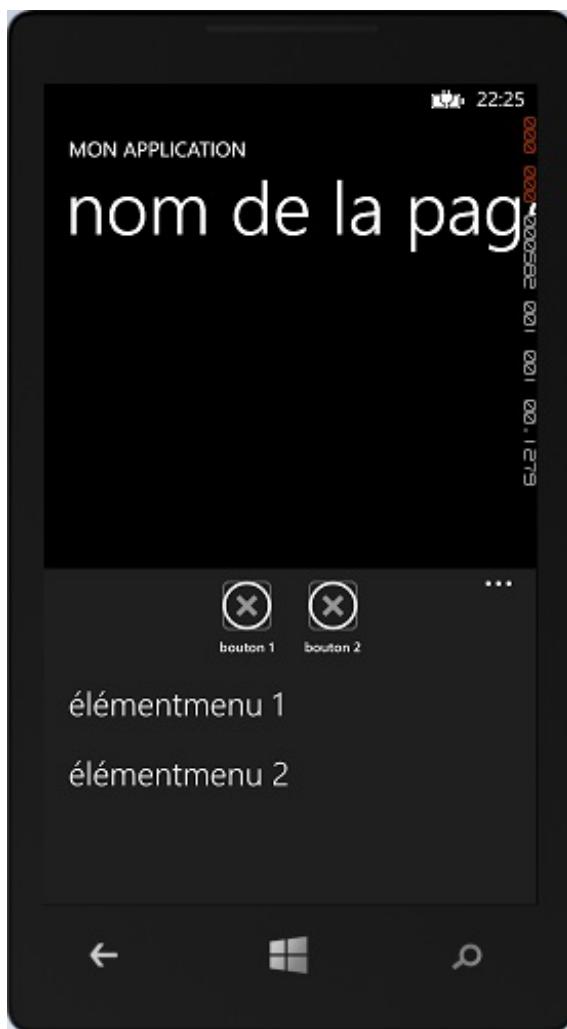
Affichage de la barre d'application

La barre d'application est bien sûr ce que j'ai encadré en rouge.
Ce XAML doit être au même niveau que la page, c'est-à-dire hors du conteneur racine :

Code : XML

```
<phone:PhoneApplicationPage  
    x:Class="DemoBarreApplication.MainPage"  
    ...>  
  
    <Grid x:Name="LayoutRoot" Background="Transparent">  
        ...  
    </Grid>  
  
    <phone:PhoneApplicationPage.ApplicationBar>  
        ...  
    </phone:PhoneApplicationPage.ApplicationBar>  
</phone:PhoneApplicationPage>
```

Le XAML est plutôt facile à comprendre. On constate que la barre est visible, que le menu est activé et qu'elle possède deux boutons, « bouton 1 » et « bouton 2 », ainsi que deux éléments de menu, « ÉlémentMenu 1 » et « ÉlémentMenu 2 ». Pour afficher les éléments de menu, il faut cliquer sur les trois petits points en bas à droite. Cela permet également de voir le texte associé à un bouton, comme vous pouvez le voir sur la figure suivante.



Les éléments de menu de la barre

Par contre, nos images ne sont pas très jolies. En effet, le code XAML fait référence à des images qui n'existent pas dans notre solution. La croix affichée représente donc l'absence d'image.

Ces images sont des icônes. Elles sont toujours visibles sur la barre d'application alors que ce n'est pas forcément le cas des éléments de menu. Il est donc primordial que ces icônes soient les plus représentatives possibles, car pour voir la légende de l'icône, il faut cliquer sur les trois petits points permettant d'afficher la suite de la barre. Les icônes doivent avoir la taille de 48x48, sachant que le cercle est ajouté automatiquement en surimpression par Windows Phone. Cela implique que la zone visible de votre icône doit être de 26x26 au centre de l'icône. Enfin, l'icône doit être de couleur blanche, sur fond transparent.

C'est un look très Modern UI que nous impose Microsoft afin de privilégier la sémantique du dessin plutôt que sa beauté. Pour nous aider, Microsoft propose un pack d'icône « Modern UI » que nous pouvons librement utiliser dans nos applications. Ce pack a été installé avec le SDK de Windows Phone, vous pouvez le retrouver à cet emplacement :

C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.0\Icons.



Vous pouvez voir qu'il y a plusieurs répertoires. Vous n'avez qu'à vous soucier du répertoire `dark` qui correspond au thème foncé. Windows Phone utilise automatiquement la bonne image en fonction du thème sélectionné.

Maintenant, ajoutons deux images à notre application, disons `delete.png` et `edit.png`. Pour ce faire, créez par exemple un répertoire `Icons` sous le répertoire `Assets` et ajoutez les icônes dedans en ajoutant un élément existant. Ensuite, dans la fenêtre de propriétés, modifiez l'action de génération en « Contenu » ainsi que la copie dans le répertoire de sortie en « copier si plus récent ».

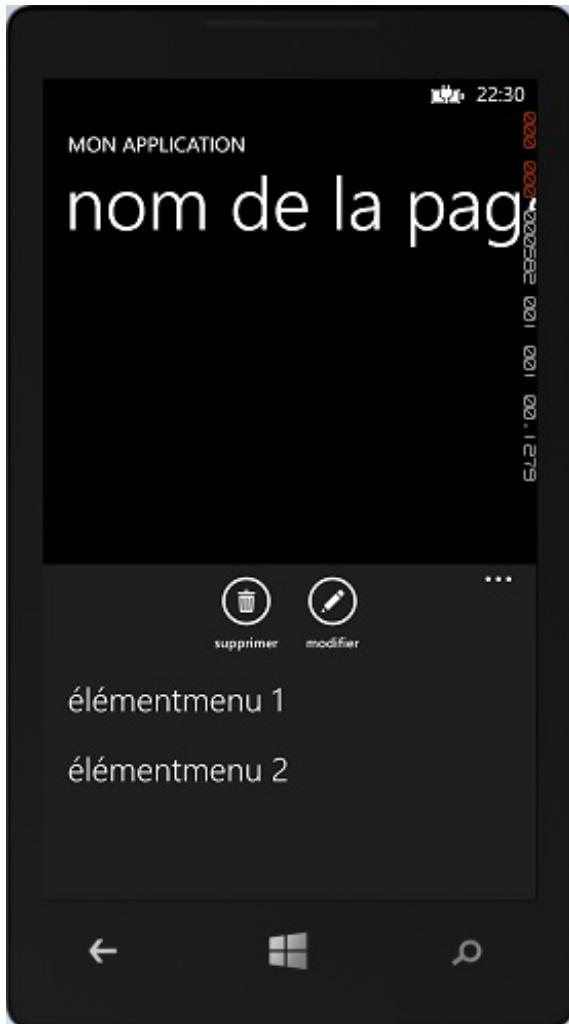
Modifiez ensuite le XAML pour utiliser nos nouvelles icônes :

Code : XML

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:AppBar IsVisible="True" IsMenuEnabled="True">
        <shell:AppBarIconButton
IconUri="/Assets/Icons/delete.png" Text="Supprimer"/>
        <shell:AppBarIconButton
```

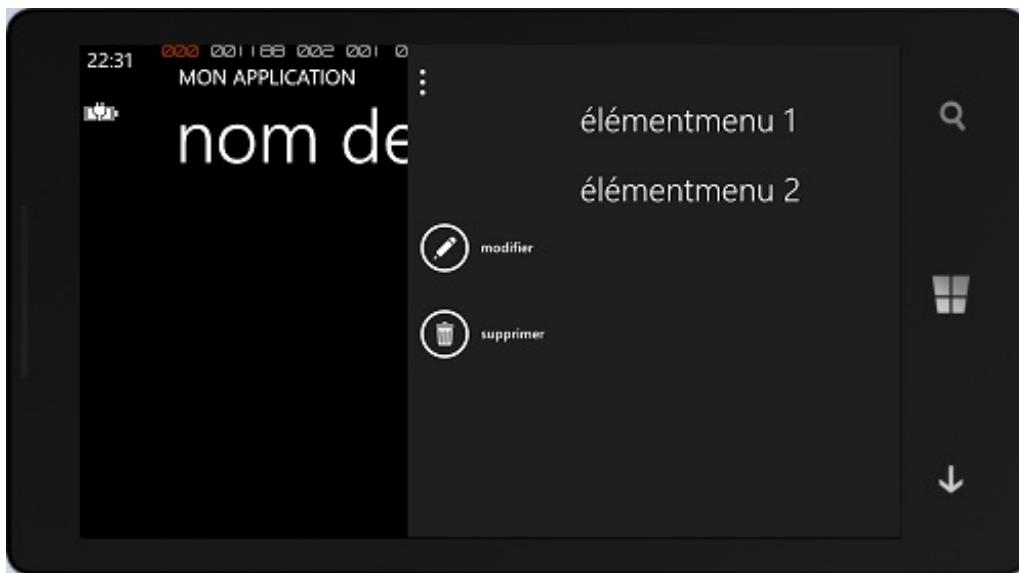
```
IconUri="/Assets/Icones/edit.png" Text="Modifier"/>
<shell:ApplicationBar.MenuItems>
    <shell:ApplicationBarItem Text="ÉlémentMenu 1"/>
    <shell:ApplicationBarItem Text="ÉlémentMenu 2"/>
</shell:ApplicationBar.MenuItems>
</shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Et voilà à la figure suivante le beau résultat !



Utilisation des icônes de la barre d'application

La barre d'application est positionnée en bas de l'écran, juste au-dessus des trois boutons physiques. On ne peut pas la changer de place, mais elle sait cependant s'orienter différemment suivant l'orientation du téléphone. Par exemple, en mode paysage, nous pouvons voir la rotation des images et du texte :



rotation lors du changement d'orientation du téléphone

La barre d'application subit une

Il faudra bien sûr au préalable faire en sorte que la page supporte les deux orientations comme nous l'avons vu dans un chapitre précédent.



Remarquez qu'on ne peut avoir que 4 boutons dans la barre d'application. Si vous tentez d'en mettre plus, vous aurez une exception.

Bon, une barre avec des boutons c'est bien, mais si on pouvait cliquer dessus pour faire des choses, ça serait pas plus mal non ? Rien de plus simple, vous pouvez ajouter l'événement de clic sur un bouton ou sur un élément de menu :

Code : XML

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton
            IconUri="/Assets/Icones/delete.png" Text="Supprimer"
            Click="ApplicationBarIconButton_Click_1"/>
        <shell:ApplicationBarIconButton
            IconUri="/Assets/Icones/edit.png" Text="Modifier"/>
        <shell:ApplicationBar.MenuItems>
            <shell:ApplicationBarMenuItem Text="ÉlémentMenu 1"
                Click="ApplicationBarMenuItem_Click_1"/>
            <shell:ApplicationBarMenuItem Text="ÉlémentMenu 2"/>
        </shell:ApplicationBar.MenuItems>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```



Ici, nous utilisons l'événement Click, il n'existe pas d'événement Tap pour la barre d'application.

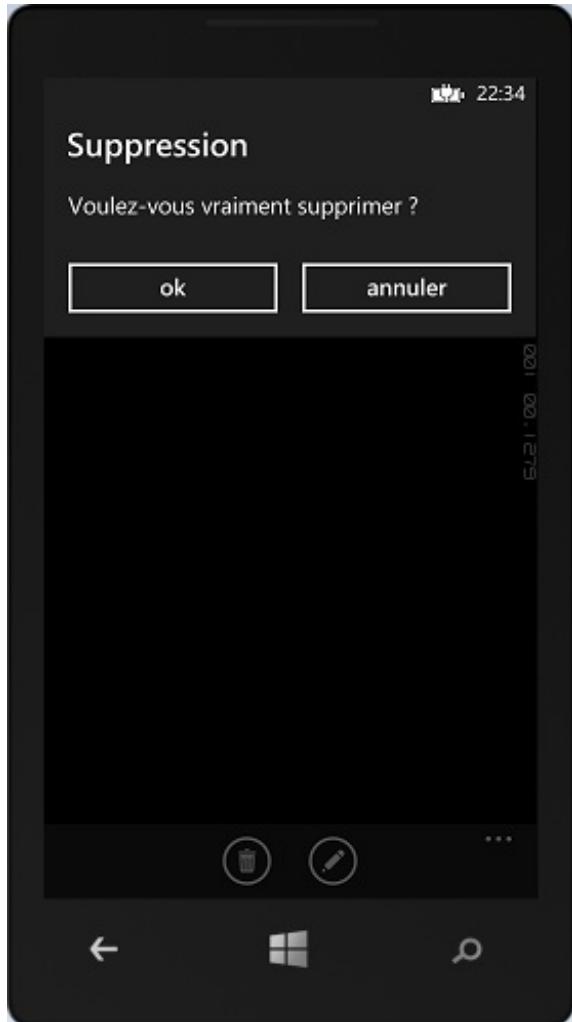
Avec :

Code : C#

```
private void ApplicationBarIconButton_Click_1(object sender,
EventArgs e)
{
    MessageBox.Show("Voulez-vous vraiment supprimer ?",
    "Suppression", MessageBoxButton.OKCancel);
}
```

```
private void ApplicationBarItem_Click_1(object sender, EventArgs e)
{
    MessageBox.Show("Menu !");
}
```

Observez la figure suivante pour le rendu.



Le clic sur un bouton de menu déclenche l'affichage du message

On peut jouer sur les propriétés de la barre d'application. Citons par exemple sa propriété `IsVisible` qui permet de la rendre visible ou invisible. Citons encore la propriété `Opacity` qui permet d'afficher des contrôles sous la barre d'application, par exemple avec le XAML suivant :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>

<StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
    <TextBlock Text="MON APPLICATION" Style="{StaticResource
PhoneTextNormalStyle}" Margin="12,0"/>
    <TextBlock Text="nom de la page" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Canvas>
        <TextBlock Text="Je suis caché" Foreground="Red"
FontSize="50" Canvas.Top="500" />
    </Canvas>
</Grid>
</Grid>

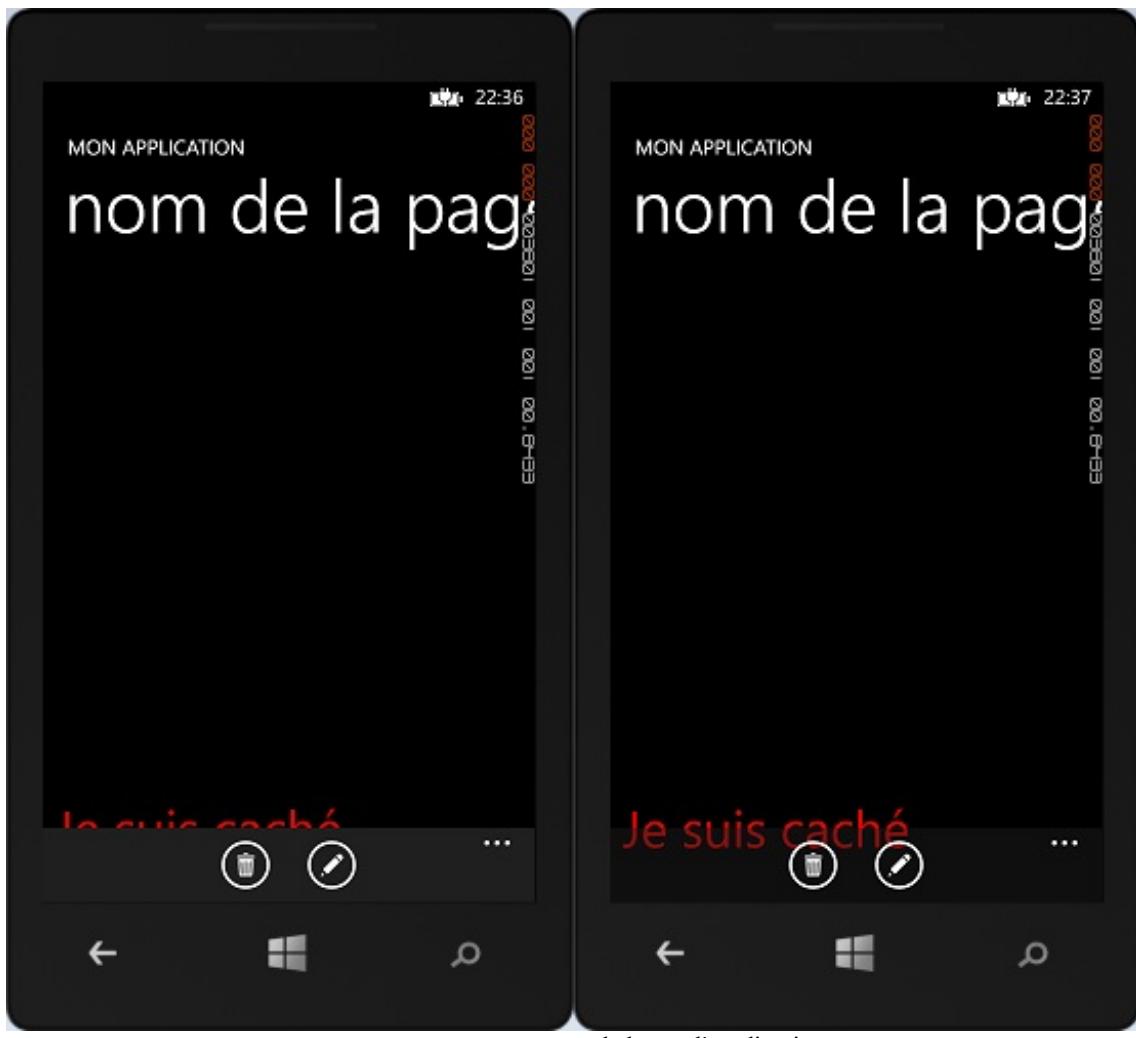
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:AppBar IsVisible="True" IsMenuEnabled="True">
        <shell:AppBarIconButton
IconUri="/Assets/Icones/delete.png" Text="Supprimer"
Click="AppBarIconButton_Click_1"/>
        <shell:AppBarIconButton
IconUri="/Assets/Icones/edit.png" Text="Modifier"/>
        <shell:AppBar.MenuItems>
            <shell:AppBarMenuItem Text="ÉlémentMenu 1"
Click="AppBarMenuItem_Click_1"/>
            <shell:AppBarMenuItem Text="ÉlémentMenu 2"/>
        </shell:AppBar.MenuItems>
    </shell:AppBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Si vous démarrez l'application, le texte rouge est invisible car il est derrière la barre d'application. Si vous changez l'opacité en mettant par exemple 0.5 :

Code : XML

```
<shell:AppBar IsVisible="True" IsMenuEnabled="True"
Opacity="0.5">
```

le texte apparaîtra sous la barre d'application (voir la figure suivante).



Utilisation de l'opacité

sur la barre d'application

Pour les boutons, on peut remarquer en plus la propriété `IsEnabled` qui permet de désactiver ou d'activer un bouton. Il est bien sûr possible de manipuler la barre d'application par le code-behind, via la propriété `ApplicationBar`. Par exemple, je peux rendre invisible la barre d'application de cette façon :

Code : C#

```
ApplicationBar.Visible = false;
```

Ou modifier le texte du premier bouton :

Code : C#

```
((ApplicationBarIconButton)ApplicationBar.Buttons[0]).Text =  
"Changement dynamique";
```

Pour les éléments de menu, le principe est le même que pour les boutons.

Notons juste qu'il est possible de s'abonner à un événement qui permet de savoir si l'état de la barre change, c'est-à-dire quand les éléments de menu sont affichés ou non. Cela permet par exemple de faire en sorte que certaines infos soient affichées différemment si jamais le menu les cache. Il s'agit de l'événement `StateChanged` :

Code : C#

```
private void ApplicationBar_StateChanged(object sender,
ApplicationBarStateChangedEventArgs e)
{
    if (e.IsMenuVisible)
    {
        // ...
    }
}
```

Enfin, remarquons une dernière petite chose qui peut-être vous a intrigué. Il s'agit de l'endroit dans le XAML où est définie la barre d'application :

Code : XML

```
<phone:PhoneApplicationPage
...
<Grid x:Name="LayoutRoot" Background="Transparent">
...
</Grid>

<phone:PhoneApplicationPage.ApplicationBar>
...
</phone:PhoneApplicationPage.ApplicationBar>
</phone:PhoneApplicationPage>
```

J'en ai parlé un peu en haut mais vous avez vu ? Elle est au même niveau que la grille ! Alors que nous avons dit qu'un seul conteneur racine était autorisé. C'est parce que l'objet `AppBar` ne dérive pas de `FrameworkElement`, pour notre plus grand malheur d'ailleurs. La propriété `AppBar` de la `PhoneApplicationPage` est une propriété de dépendance : [http://msdn.microsoft.com/fr-fr/library \[...\] entry\(v=vs.92\)](http://msdn.microsoft.com/fr-fr/library [...] entry(v=vs.92)). C'est pour cela que nous pouvons (et d'ailleurs devons !) la mettre au niveau de la page.



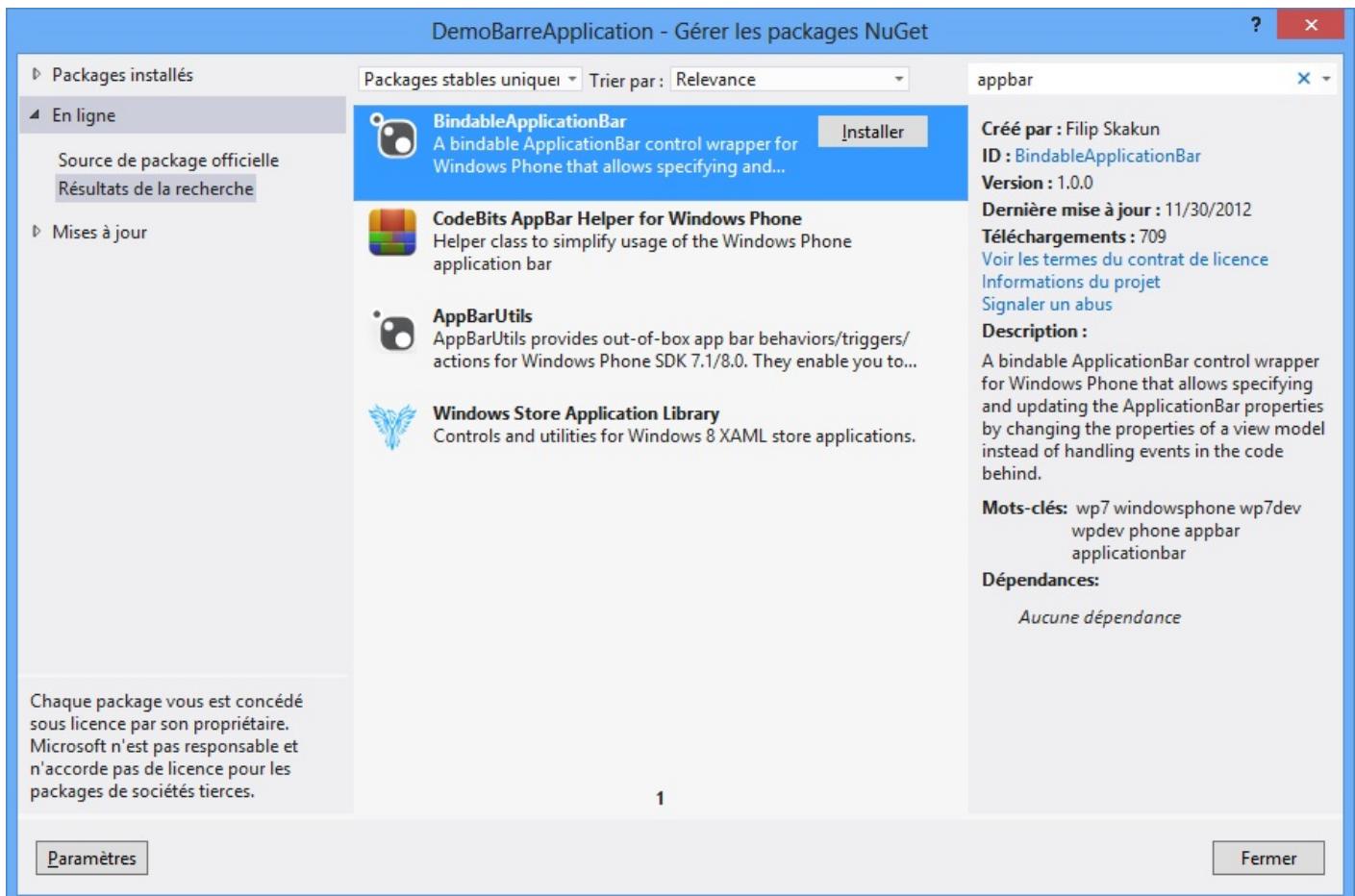
Attention : il est recommandé de ne pas avoir de trop long textes dans les éléments de menu, entre 14 et 20 caractères.

Appliquer le Databinding

Étant donné que notre barre d'application n'est pas un `FrameworkElement`, elle ne sait pas gérer la liaison de données. Il n'est donc pas possible d'utiliser les extensions de balisage `{Binding}` ni les commandes sur cet élément. Ce qui pose un problème lorsque l'on fait du MVVM ou même quand il s'agit de traduire notre application.

Il y a cependant une petite astuce pour rendre notre barre application fonctionnelle avec la liaison de données ainsi qu'avec le système de commandes. Il suffit de développer une nouvelle classe qui encapsule les fonctionnalités de la classe `AppBar`. On appelle communément ce genre de classe un wrapper.

Nous n'allons pas faire cet exercice ici vu que d'autres personnes l'ont déjà fait pour nous. C'est le cas par exemple de la bibliothèque `BindableAppBar` que nous pouvons trouver à cet emplacement : <http://bindableapplicationb.codeplex.com/>. Vous pouvez télécharger cette bibliothèque ou tout simplement utiliser NuGet (voir la figure suivante).



Utilisation de NuGet pour télécharger la barre d'application bindable

Il ne reste plus qu'à importer l'espace de nom suivant :

Code : XML

```
xmlns:barre="clr-
namespace:BindableApplicationBar;assembly=BindableApplicationBar"
```

et à modifier notre barre comme suit :

Code : XML

```
<barre:Bindable.ApplicationBar>
    <barre:BindableApplicationBar IsVisible="{Binding
EstBarreVisible}">
        <barre:BindableApplicationBarButton Command="{Binding
SupprimerCommand}" Text="{Binding SupprimerText}"
IconUri="/Assets/Icones/delete.png" />
        <barre:BindableApplicationBar.MenuItems>
            <barre:BindableApplicationBarMenuItem Text="{Binding
ElementText}" Command="{Binding ElementCommand}" />
        </barre:BindableApplicationBar.MenuItems>
    </barre:BindableApplicationBar>
</barre:Bindable.ApplicationBar>
```

Notons la liaison de données sur la propriété `IsVisible` de la barre d'application, de la propriété `Text` du bouton ainsi que l'utilisation d'une commande lors du clic sur le bouton. Nous aurons donc un contexte qui pourra être :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    public void NotifyPropertyChanged(string nomPropriete)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
PropertyChangedEventArgs(nomPropriete));
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private bool estBarreVisible;
    public bool EstBarreVisible
    {
        get { return estBarreVisible; }
        set { NotifyPropertyChanged(ref estBarreVisible, value); }
    }

    public string SupprimerText
    {
        get { return "Supprimer"; }
    }

    public string ElementText
    {
        get { return "Elément"; }
    }

    public ICommand SupprimerCommand { get; set; }
    public ICommand ElementCommand { get; set; }

    public MainPage()
    {
        InitializeComponent();
        SupprimerCommand = new RelayCommand(OnSupprimer);
        ElementCommand = new RelayCommand(OnElement);
        EstBarreVisible = true;

        DataContext = this;
    }

    private void OnSupprimer()
    {
        MessageBox.Show("Voulez-vous vraiment supprimer ?",
"Suppression", MessageBoxButton.OKCancel);
    }

    private void OnElement()
    {
        EstBarreVisible = false;
    }
}
```

Notez que vous aurez besoin d'une classe `RelayCommand`, venant de votre framework MVVM préféré ou bien l'exemple simple que nous avons créé précédemment.

Grâce à cette classe, nous pouvons désormais respecter correctement le pattern MVVM, chose qui était plus difficile sans.



Le framework Ultralight MVVM que nous avons cité dans un chapitre précédent propose une solution pour gérer la liaison de données avec la barre d'application.

Mode plein écran

Vous avez dû remarquer qu'il y a tout en haut de l'écran, un petit bout qui est réservé à l'affichage de la batterie, de la qualité de la connexion, etc. Il s'agit de la barre système. Ce n'est pas vraiment la barre d'application, mais je trouve qu'il est pertinent d'en parler ici.

Il n'est pas possible de modifier le contenu de la barre système, mais il est par contre possible de la masquer afin d'avoir un mode « plein écran », grâce à une propriété.

Cette propriété est positionnée à vrai par défaut lorsque l'on crée un nouvelle page XAML, elle rend la barre système visible :

Code : XML

```
<phone:PhoneApplicationPage  
...  
    shell:SystemTray.Visibile="True"
```

Nous pouvons modifier ici sa valeur en mettant `False`. Ou bien par code, en utilisant la classe statique `SystemTray` :

Code : C#

```
SystemTray.Visibile = false;
```

Cela permet de récupérer 16 pixels en hauteur.

Vous aurez besoin d'inclure l'espace de nom suivant :

Code : C#

```
using Microsoft.Phone.Shell;
```



Attention cependant, ce n'est pas toujours pertinent de masquer cette barre système car elle fournit des informations intéressantes. Personnellement, lorsque je dois être connecté à internet, j'aime bien vérifier la force du signal histoire de savoir s'il vaut mieux relancer la requête, ou tout simplement pour avoir l'heure... À utiliser judicieusement.

- La barre d'application est un contrôle utile lorsqu'il y a besoin de faire des actions récurrentes dans une application.
- Elle fonctionne un peu comme un menu, accessible de partout.
- Ne supportant pas la liaison de données par défaut, il est possible d'utiliser des wrappers pour la rendre fonctionnelle avec le binding.
- Il est recommandé de ne pas masquer la barre système, mais c'est cependant possible pour passer en mode plein écran.

Le toolkit Windows Phone

Bien que déjà bien fournis, les contrôles Windows Phone ne font pas tout. Nous avons déjà vu des bibliothèques tierces, comme le MVVM Light toolkit qui permet de faire en sorte que son application respecte plus facilement le patron de conception MVVM. Nous avons également utilisé une bibliothèque qui permet d'utiliser le binding avec la barre d'application.

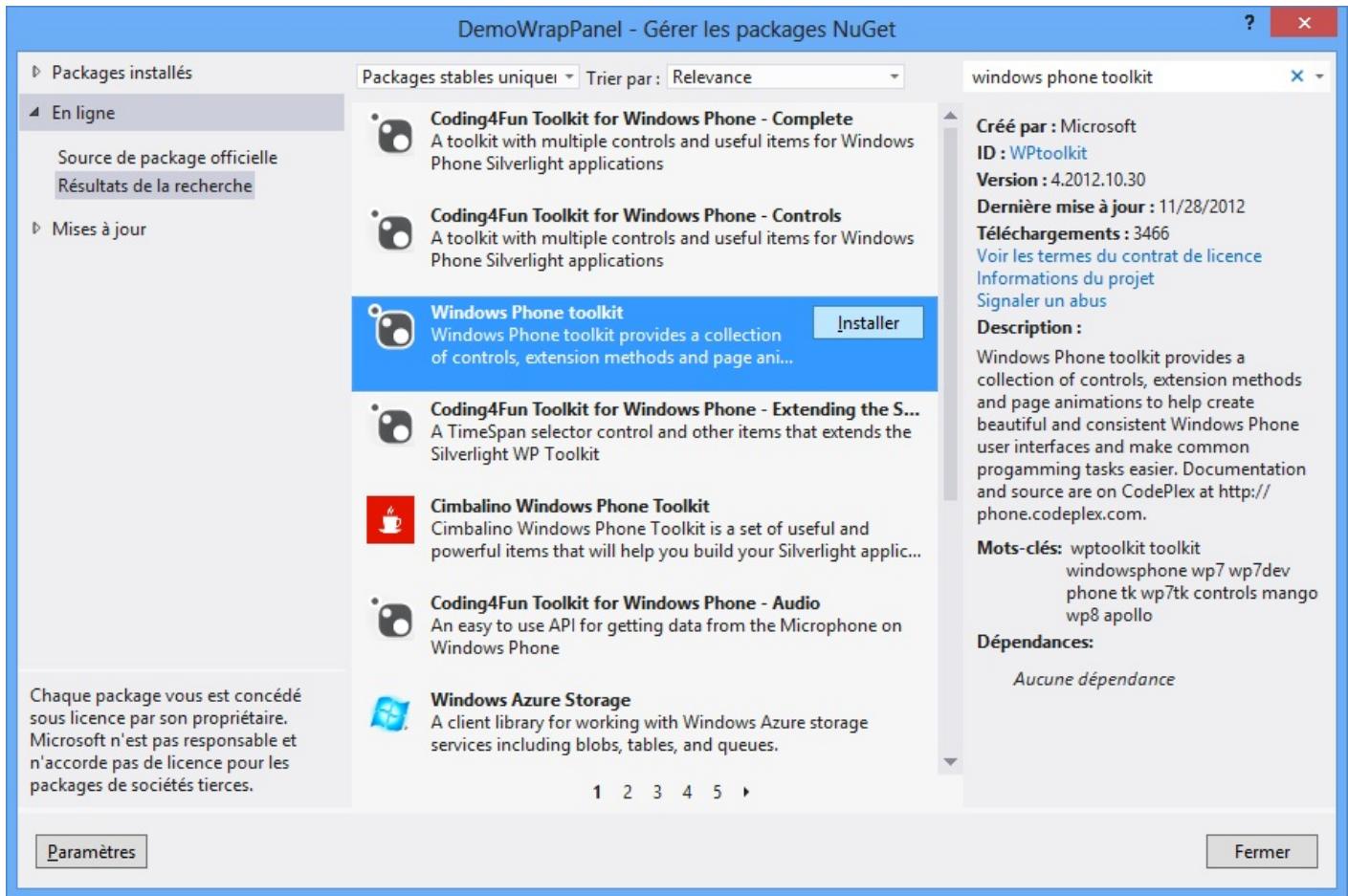
Il existe d'autres bibliothèques bien pratiques contenant des contrôles évolués qui vont nous permettre d'enrichir nos applications et d'offrir avec le moindre effort une expérience utilisateur des plus sympathiques.

En plus, il y en a qui sont gratuites, alors pourquoi s'en priver ?

Présentation et installation du toolkit Windows Phone

La bibliothèque la plus connue est sans nul doute le Toolkit pour Windows Phone. Il s'agit d'un projet très suivi qui contient beaucoup de contrôles pour Windows Phone. Historiquement, ce toolkit proposait déjà toute une gamme de contrôles supplémentaires pour Silverlight. Depuis la sortie de Silverlight pour Windows Phone 7.X, c'est tout naturellement qu'une bibliothèque parallèle, dédiée à Windows Phone, a vu le jour. Et aujourd'hui, avec l'arrivée de Windows Phone 8, il en existe une version pour lui.

Comme beaucoup de bibliothèques, elle est open-source, utilisable dans beaucoup de situations mais pas toujours exempte de bug. Il y a tout une communauté qui travaille sur ces projets et qui continue régulièrement à l'améliorer. Cette bibliothèque est téléchargeable sur <http://phone.codeplex.com/>. À l'heure où j'écris ces lignes, nous pouvons télécharger la version de novembre 2012. Vous pouvez télécharger les binaires sur le site ou encore une fois utiliser NuGet qui facilite la récupération de bibliothèques tierces (voir la figure suivante).



DemoWrapPanel - Gérer les packages NuGet

Packages installés

En ligne

Source de package officielle

Résultats de la recherche

Mises à jour

Packages stables uniquement Trier par : Relevance

windows phone toolkit

Créé par : Microsoft
ID : WPtoolkit
Version : 4.2012.10.30
Dernière mise à jour : 11/28/2012
Téléchargements : 3466
[Voir les termes du contrat de licence](#)
[Informations du projet](#)
[Signaler un abus](#)

Description :

Windows Phone toolkit provides a collection of controls, extension methods and page ani...
Installer

Windows Phone toolkit
Windows Phone toolkit provides a collection of controls, extension methods and page ani...

Coding4Fun Toolkit for Windows Phone - Complete
A toolkit with multiple controls and useful items for Windows Phone Silverlight applications

Coding4Fun Toolkit for Windows Phone - Controls
A toolkit with multiple controls and useful items for Windows Phone Silverlight applications

Windows Phone toolkit
Windows Phone toolkit provides a collection of controls, extension methods and page ani...

Coding4Fun Toolkit for Windows Phone - Extending the S...
A TimeSpan selector control and other items that extends the Silverlight WP Toolkit

Cimalino Windows Phone Toolkit
Cimalino Windows Phone Toolkit is a set of useful and powerful items that will help you build your Silverlight applic...

Coding4Fun Toolkit for Windows Phone - Audio
An easy to use API for getting data from the Microphone on Windows Phone

Windows Azure Storage
A client library for working with Windows Azure storage services including blobs, tables, and queues.

1 2 3 4 5

Paramètres Fermer

Chaque package vous est concédé sous licence par son propriétaire. Microsoft n'est pas responsable et n'accorde pas de licence pour les packages de sociétés tierces.

Utilisation de NuGet pour télécharger le Windows Phone Toolkit

NuGet nous a donc ajouté la référence à l'assembly `Microsoft.Phone.Controls.Toolkit.dll`.

Je ne vais pas vous présenter tous les contrôles tellement il y en a. Je vous encourage à télécharger les sources ainsi que l'application exemple et de la tester pour voir ce qu'elle renferme. Mais voyons-en quand même quelques uns 😊.

PerformanceProgressBar

Je souhaitais vous parler de la barre de progression `PerformanceProgressBar` car c'est un très bon exemple de la puissance du toolkit. La barre de progression qui était originellement présente avec le SDK pour Windows Phone 7.X posait des problèmes de performances. Celle du toolkit profite de toutes les optimisations possibles et gère également beaucoup mieux les

threads. Nous parlerons des Threads dans un prochain chapitre.

Toujours est-il que pour Windows Phone 7, c'était cette barre de progression qui était recommandée un peu partout sur le net et par Microsoft (pour la petite histoire, c'est un développeur de Microsoft qui a créé cette nouvelle barre de progression). Elle a depuis été intégrée dans le SDK de Windows Phone 8 afin de tirer parti de ces améliorations. Je vais vous montrer ici comment s'en servir dans une application pour Windows Phone 7 et après je rebasculerai sur Windows Phone 8. Créez donc une application qui cible le SDK 7.1. Ensuite, pour utiliser la barre, rien de plus simple, il vous suffit d'importer l'espace de nom du toolkit :

Code : XML

```
xmlns:toolkit="clr-  
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"
```

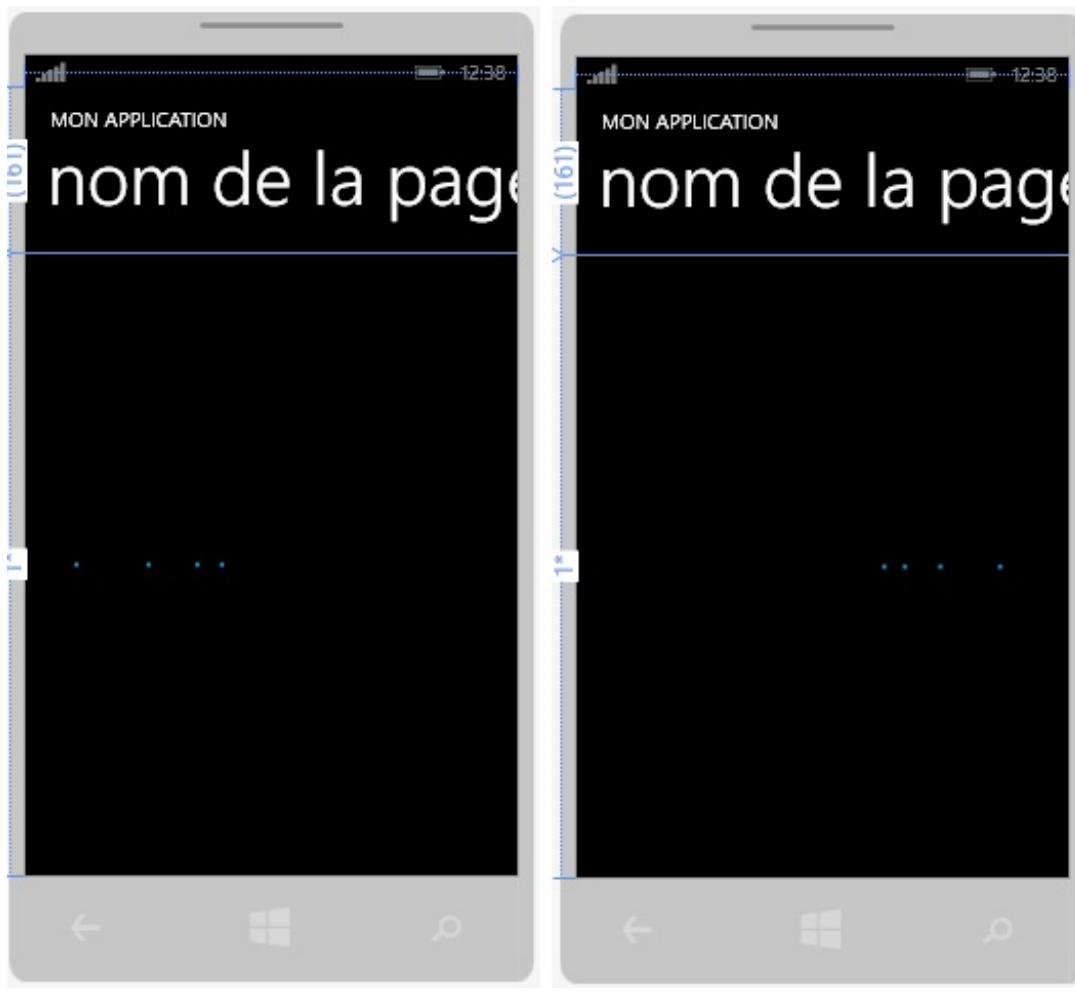
et de la déclarer où vous souhaitez qu'elle s'affiche :

Code : XML

```
<toolkit:PerformanceProgressBar IsIndeterminate="True" />
```

La barre de progression n'en est en fait pas vraiment une, du moins pas dans le sens où on les connaît : en l'occurrence, elle n'affiche pas un pourcentage de progression sur une tâche dont on connaît la durée totale. Il s'agit plutôt d'une petite animation qui montre qu'il se passe un chargement, sans que l'on sache vraiment où nous en sommes.

Vous pouvez voir son fonctionnement dans le designer de Visual Studio, la barre s'anime et affiche des points, comme indiqué sur la figure suivante.



Animation de la barre dans

le designer

La barre est visible dans un état où la progression est indéterminée. C'est pour cela que la propriété s'appelle `IsIndeterminate` et permet d'indiquer si elle est visible ou non. Passez la à `False` pour la voir se masquer. En général, ce que vous ferez, c'est dans un premier temps mettre la propriété à vrai, démarrer quelque chose de long de manière asynchrone, par exemple un téléchargement, et une fois terminé vous passerez la propriété à faux. Nous pouvons simuler ce fonctionnement avec un `DispatcherTimer`, il s'agit d'une classe qui va nous permettre d'appeler une méthode à une fréquence déterminée. Nous en reparlerons plus tard, mais par exemple ici, mon Timer va me permettre de masquer ma barre de progression au bout de 7 secondes :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    private DispatcherTimer timer;

    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new
PropertyChangedEventArgs(propertyName));
        }
    }

    private bool chargementEnCours;
    public bool ChargementEnCours
    {
        get { return chargementEnCours; }
        set
        {
            chargementEnCours = value;
            NotifyPropertyChanged("ChangementEnCours");
        }
    }

    public MainPage()
    {
        InitializeComponent();
        DataContext = this;

        ChargementEnCours = true;
        timer = new DispatcherTimer { Interval =
TimeSpan.FromSeconds(7) };
        timer.Tick += timer_Tick;
        timer.Start();
    }

    private void timer_Tick(object sender, EventArgs e)
    {
        ChargementEnCours = false;
        timer.Stop();
    }
}
```

N'oubliez pas d'inclure l'espace de noms suivant pour pouvoir utiliser le `DispatcherTimer` :

Code : C#

```
using System.Windows.Threading;
```

Et dans le XAML, nous aurons le binding suivant :

Code : XML

```
<toolkit:PerformanceProgressBar IsIndeterminate="{Binding ChargementEnCours}" />
```

Remarquez qu'il est possible de changer facilement la couleur des points grâce à la propriété Foreground.

À noter que son utilisation est globalement la même avec le SDK pour Windows Phone 8, on utilisera cependant le contrôle ProgressBar directement. Sachant que pour reproduire exactement le même fonctionnement, il faudra masquer la barre de progression une fois son statut indéterminé activé :

Code : XML

```
<ProgressBar IsIndeterminate="{Binding ChargementEnCours}" Visibility="{Binding ChargementEnCours, Converter={StaticResource VisibilityConverter}}" />
```

Usez et abusez de cette barre dès qu'il faut avertir l'utilisateur que quelque chose de potentiellement long se passe.

ListPicker

Le contrôle ListPicker est une espèce de ListBox simplifiée. Il est l'équivalent d'un contrôle connu mais qui manquait dans les contrôles Windows Phone, la ComboBox.

Ce ListPicker permet de choisir un élément parmi une liste d'éléments relativement restreinte. Puissant grâce à son système de modèle, il permet de présenter l'information de manière évoluée. Voyons à présent comment il fonctionne.

La manière la plus simple de l'utiliser est de le lier à une liste de chaînes de caractères :

Code : XML

```
<toolkit:ListPicker ItemsSource="{Binding Langues}" />
```

Avec dans le code-behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage, INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
    }
}
```

```
        return true;
    }

    private List<string> langues;
    public List<string> Langues
    {
        get { return langues; }
        set { NotifyPropertyChanged(ref langues, value); }
    }

    public MainPage()
    {
        InitializeComponent();
        DataContext = this;

        Langues = new List<string> { "Français", "English",
"Deutsch", "Español" };
    }
}
```

Nous obtenons la figure suivante.



Le ListPicker plié

Et lorsque nous cliquons dessus, il se déplie pour nous permettre de sélectionner un élément, comme vous pouvez le voir à la figure suivante.



Le ListPicker déplié nous permet de sélectionner un élément

Pour la sélection, ce contrôle fonctionne comme la ListBox. Nous pouvons présélectionner un élément et être informés de quel élément est sélectionné. Donnons un nom à notre ListPicker et abonnons-nous à l'événement SelectionChanged :

Code : XML

```
<toolkit:ListPicker ItemsSource="{Binding Langues}" x:Name="Liste"
SelectionChanged="Liste_SelectionChanged" />
```

Et dans le code-behind :

Code : C#

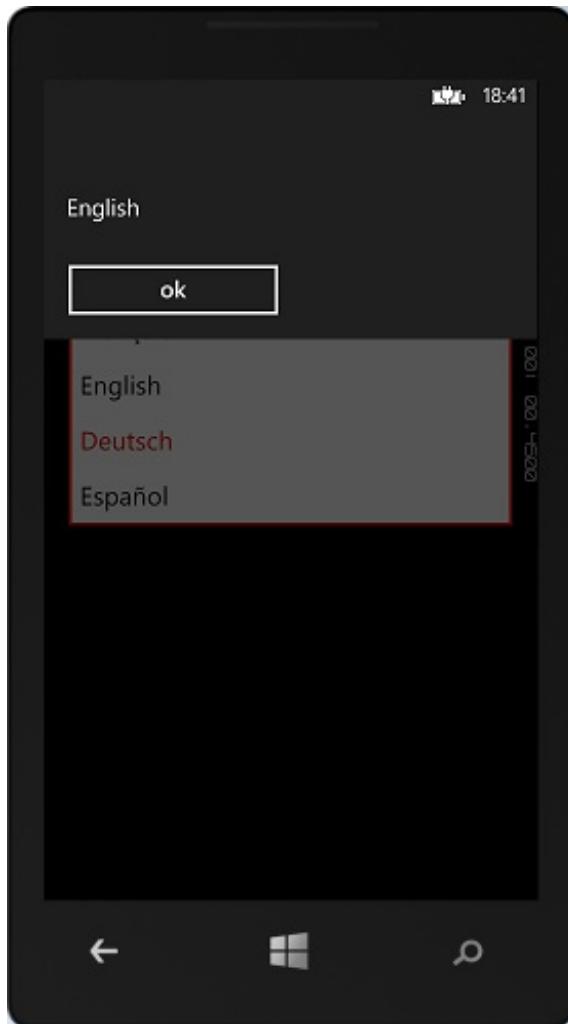
```
public MainPage()
{
    InitializeComponent();
    DataContext = this;

    Langues = new List<string> { "Français", "English", "Deutsch",
"Español" };
    Liste.SelectedIndex = 2;
}

private void Liste_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (Liste.SelectedItem != null)
    {
        MessageBox.Show(Liste.SelectedItem.ToString());
    }
}
```

```
    }  
}
```

Nous obtenons la figure suivante.



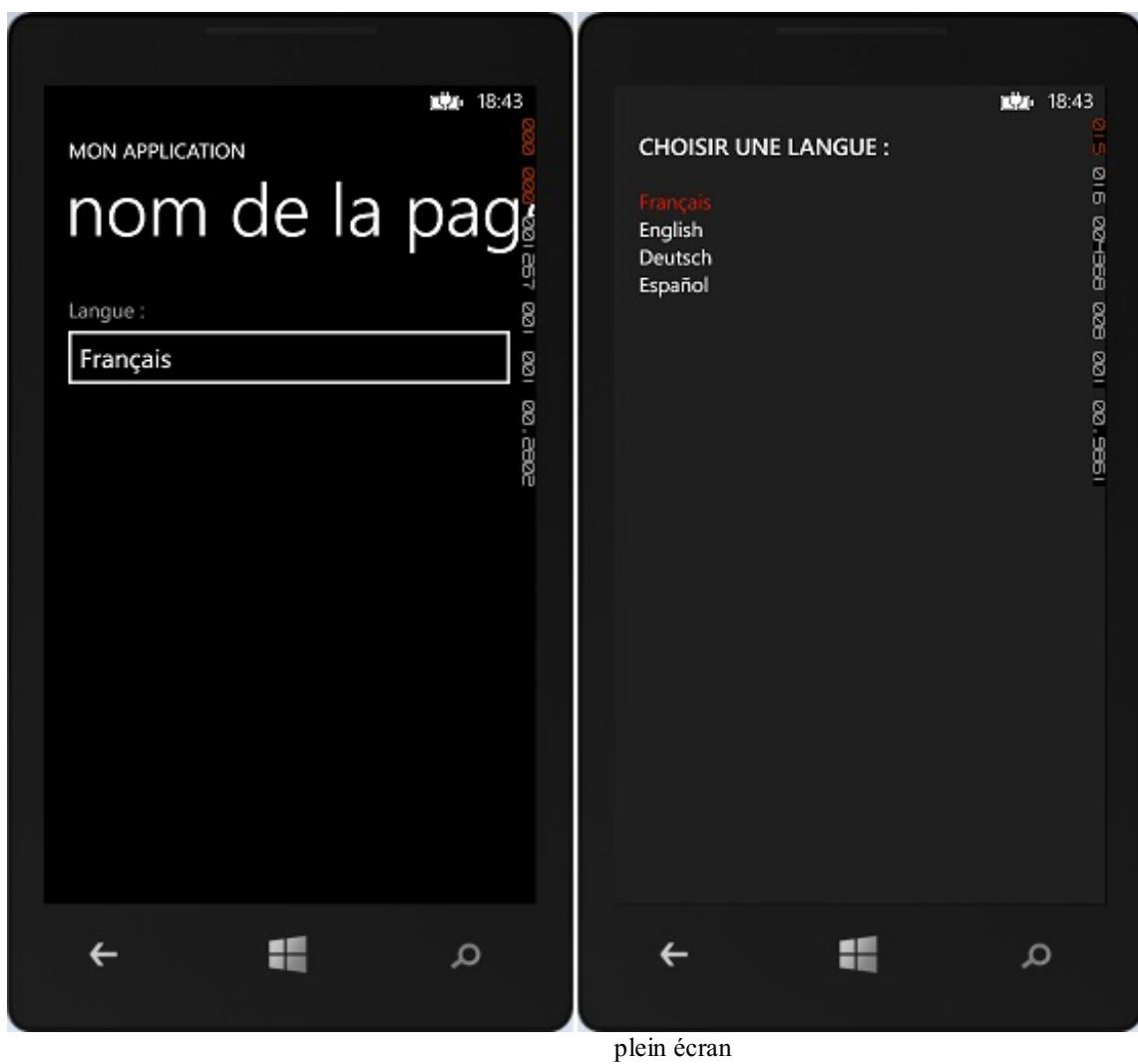
Sélection d'un élément du ListPicker

Dans cet exemple, lorsqu'on clique dans le ListPicker, la liste se déroule pour nous montrer tous les éléments de la liste. Il existe un autre mode de sélection : le mode **plein écran**. Il suffit de changer dans le XAML :

Code : XML

```
<toolkit:ListPicker ItemsSource="{Binding Langues}" Header="Langue"  
: " FullModeHeader="Choisir une langue :"  
ExpansionMode="FullScreenOnly" />
```

Remarquons la propriété ExpansionMode qui permet de forcer la sélection dans un mode plein écran. Nous pouvons également donner un titre à notre ListPicker grâce à la propriété Header, titre qui peut être différent si nous sommes en mode plein écran (voir la figure suivante).



Comme je l'ai déjà dit, il est possible de fournir un modèle pour chaque élément et ceci grâce aux propriétés `ItemTemplate` et `FullModeItemTemplate`. Par exemple, changeons le type de notre liste d'éléments :

Code : C#

```
private List<Element> langues;
public List<Element> Langues
{
    get { return langues; }
    set { NotifyPropertyChanged(ref langues, value); }
}
```

Avec la classe `Element` suivante :

Code : C#

```
public class Element
{
    public string Langue { get; set; }
    public string Code { get; set; }
    public Uri Url { get; set; }
}
```

Que nous alimentons ainsi :

Code : C#

```
Langues = new List<Element>
{
    new Element { Code = "FR", Langue = "Français", Url = new Uri("/Assets/Images/france.png", UriKind.Relative)},
    new Element { Code = "US", Langue = "English", Url = new Uri("/Assets/Images/usa.png", UriKind.Relative)},
    new Element { Code = "DE", Langue = "Deutsch", Url = new Uri("/Assets/Images/allemagne.png", UriKind.Relative)},
    new Element { Code = "ES", Langue = "Español", Url = new Uri("/Assets/Images/espagne.png", UriKind.Relative)},
};
```

sachant que les images correspondant aux drapeaux de chaque langue sont incluses dans le répertoire `Images`, sous le répertoire `Assets`, dans notre solution (en ayant changé l'action de génération à « Contenu »).
Je peux alors modifier le XAML pour avoir :

Code : XML

```
<toolkit:ListPicker ItemsSource="{Binding Langues}" Header="Langue :" FullModeHeader="Choisir une langue :" ExpansionMode="FullScreenOnly">
    <toolkit:ListPicker.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <Border BorderThickness="2" BorderBrush="Beige" Background="Azure">
                    <TextBlock Text="{Binding Code}" Foreground="Blue" />
                    </Border>
                    <TextBlock Margin="20 0 0 0" Text="{Binding Langue}" />
                </StackPanel>
            </DataTemplate>
        </toolkit:ListPicker.ItemTemplate>
        <toolkit:ListPicker.FullModeItemTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Margin="20 0 0 0" Text="{Binding Langue}" Style="{StaticResource PhoneTextTitle1Style}" />
                    <Image Source="{Binding Url}" />
                </StackPanel>
            </DataTemplate>
        </toolkit:ListPicker.FullModeItemTemplate>
    </toolkit:ListPicker>
```

Ce qui donnera la page présentée dans la figure suivante.



Utilisation des templates du ListPicker

Ainsi que dans le mode de sélection plein écran (voir la figure suivante).



Le template plein écran

Ce ListPicker est définitivement bien pratique lorsqu'il s'agit de permettre de choisir entre plusieurs éléments, dans une liste relativement courte. De plus, ses différents modes de sélection offrent une touche supplémentaire à vos applications.

WrapPanel

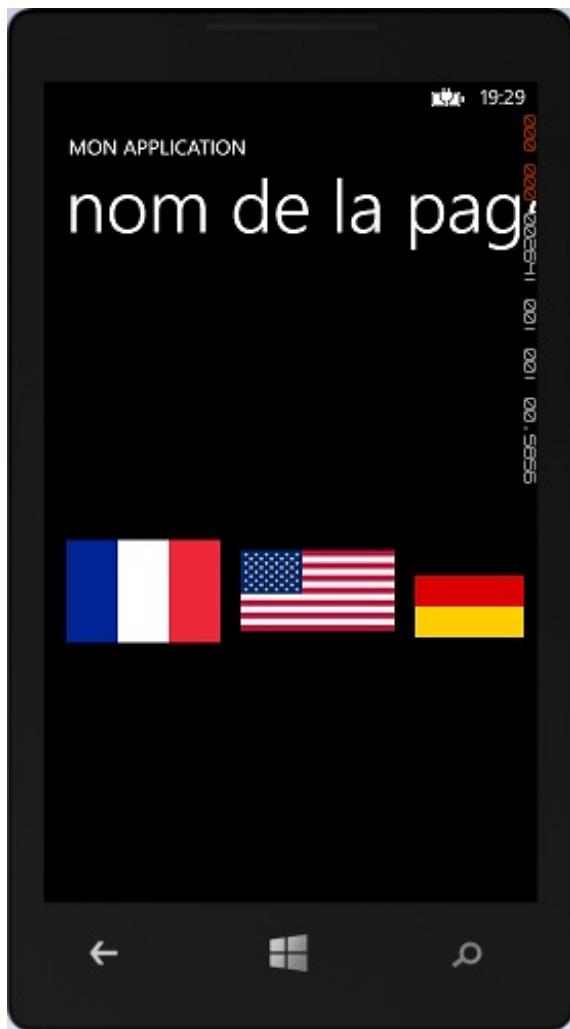
Au même titre que le StackPanel, le WrapPanel du toolkit est un conteneur de contrôles qui a une fonctionnalité intéressante. Il est capable de passer automatiquement à la ligne ou à la colonne suivante si les contrôles présents à l'intérieur dépassent du conteneur.

Prenons par exemple le XAML suivant et réutilisons les images de l'exemple précédent :

Code : XML

```
<StackPanel Orientation="Horizontal">
    <Image Source="/Assets/Images/france.png" Width="150"
    Height="150" Margin="10" />
    <Image Source="/Assets/Images/usa.png" Width="150" Height="150"
    Margin="10" />
    <Image Source="/Assets/Images/allemagne.png" Width="150"
    Height="150" Margin="10" />
    <Image Source="/Assets/Images/espagne.png" Width="150"
    Height="150" Margin="10" />
</StackPanel>
```

Vous pouvez voir le résultat à la figure suivante.



L'écran n'est pas assez large pour voir les 4 images avec un StackPanel

Oh diantre, c'est plutôt embêtant, on ne voit que 3 images sur les 4 promises... ! Eh oui, il n'y a pas assez de place sur l'écran et la troisième image dépasse du conteneur.

Changeons maintenant juste le conteneur et remplaçons-le par un WrapPanel :

Code : XML

```
<toolkit:WrapPanel>
    <Image Source="/Assets/Images/france.png" Width="150"
Height="150" Margin="10" />
    <Image Source="/Assets/Images/usa.png" Width="150" Height="150"
Margin="10" />
    <Image Source="/Assets/Images/allemande.png" Width="150"
Height="150" Margin="10" />
    <Image Source="/Assets/Images/espagne.png" Width="150"
Height="150" Margin="10" />
</toolkit:WrapPanel>
```

Nous aurons la figure suivante.



Le WrapPanel passe automatiquement à la ligne s'il n'y a pas assez de place

Et ô miracle, toutes les images sont désormais présentes.

Vous avez compris que c'est le WrapPanel qui a fait automatiquement passer les images à la ligne afin qu'elles apparaissent à l'écran. Son but : faire en sorte que tout tienne à l'écran !

Cela fonctionne également en orientation verticale :

Code : XML

```
<toolkit:WrapPanel Orientation="Vertical">
    <Image Source="/Assets/Images/france.png" Width="150"
Height="150" Margin="10" />
    <Image Source="/Assets/Images/usa.png" Width="150" Height="150"
Margin="10" />
    <Image Source="/Assets/Images/allemagne.png" Width="150"
Height="150" Margin="10" />
    <Image Source="/Assets/Images/espagne.png" Width="150"
Height="150" Margin="10" />
</toolkit:WrapPanel>
```

Ce qui donnera la figure suivante.



Le WrapPanel en orientation verticale

Le WrapPanel est très pratique combiné avec une ListBox ou même un ListPicker. Dans l'exemple fourni avec le toolkit, on peut le voir utilisé à l'intérieur d'un Pivot.

N'hésitez pas à vous servir de ce conteneur dès que vous aurez besoin d'afficher une liste d'éléments dont vous ne maîtrisez pas forcément le nombre.

Et je ne sais pas si vous vous souvenez, mais je vous ai fait une capture d'écran précédemment lorsque je parlais des différentes résolutions des Windows Phone où j'avais ajouté plein de boutons les uns à la suite des autres. Ils étaient dans un WrapPanel et s'alignaient alors automatiquement, les uns à la suite des autres.

LongListSelector

Avant de s'arrêter sur l'exploration des contrôles de ce toolkit, regardons encore le contrôle LongListSelector. Bien que j'aie choisi de présenter ce contrôle dans le chapitre sur le toolkit Windows Phone, le LongListSelector existe dans le SDK de Windows Phone 8. Eh oui, c'est comme pour le ProgressBar, voici un contrôle qui n'existe pas avec le SDK pour Windows Phone 7, qui a été créé et approuvé par la communauté, et qui trouve naturellement sa place dans le SDK pour Windows Phone 8. Il permet d'améliorer encore la ListBox, notamment quand celle-ci contient énormément de valeurs. Grâce à lui, nous pouvons regrouper des valeurs afin que chacune soit plus facilement accessible. Ce type de contrôle est utilisé par exemple avec les contacts dans notre téléphone, lorsque nous commençons à en avoir beaucoup.

La figure suivante montre le résultat que nous allons obtenir à la fin de ce chapitre, les prénoms sont regroupés suivant leur première lettre.



Les prénoms sont regroupés avec le LongListSelector

Et lors d'un clic sur un groupe, nous obtenons la liste des groupes où nous pouvons sélectionner un élément pour l'atteindre plus rapidement (voir la figure suivante).



La liste des groupes du LongListSelector

Pour démontrer ce fonctionnement, prenons par exemple la liste des 50 prénoms féminins de bébé les plus donnés en 2012 :

Code : C#

```
List<string> liste = new List<string> { "Emma", "Manon", "Chloé",
"Camille", "Léa", "Lola", "Louise", "Zoé", "Jade", "Clara", "Lena",
"Eva", "Anaïs", "Clémence", "Lilou", "Inès", "Juliette", "Maëlys",
"Lucie", "Alice", "Sarah", "Romane", "Jeanne", "Margaux",
"Mathilde", "Elise", "Léonie", "Louna", "Lisa", "Ambre", "Anna",
"Justine", "Lily", "Pauline", "Laura", "Charlotte", "Rose", "Julia",
"Noémie", "Eloïse", "Lou", "Alicia", "Eléna", "Margot", "Elsa",
"Louane", "Elisa", "Nina", "Marie", "Agathe" };
```

Le principe est de faire une liste de liste groupée d'éléments grâce à l'opérateur Linq `join`:

Code : C#

```
IEnumerable<List<string>> prenoms = from prenom in liste
group prenom by prenom[0].ToString() into p
orderby p.Key
select new List<string>(p);

ListePrenoms = prenoms.ToList();
```

Ici, j'ai choisi de regrouper les prénoms en fonction de leur première lettre, mais on pourrait très bien imaginer une liste de villes

regroupées en fonction de leur pays d'appartenance... ou quoi que ce soit d'autre... Pour utiliser le binding avec ce LongListSelector, nous devons avoir la propriété suivante :

Code : C#

```
private List<List<string>> listePrenoms;
public List<List<string>> ListePrenoms
{
    get { return listePrenoms; }
    set { NotifyPropertyChanged(ref listePrenoms, value); }
}
```

Côté XAML c'est un peu plus compliqué. Nous avons premièrement besoin de lier la propriété ItemsSource du LongListSelector à notre propriété ListePrenoms :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}">
</phone:LongListSelector>
```

N'oubliez pas qu'il s'agit d'un contrôle Windows Phone 8, et que son espace de nom est inclus dans la page avec :
xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
Si vous utilisez le SDK pour Windows Phone 7, il vous suffira d'utiliser le contrôle du toolkit, mais il y aura quand même quelques adaptations à faire sur les templates.

Il faut maintenant indiquer plusieurs choses, premièrement que le contrôle est en mode liste, qu'il accepte de grouper les éléments et qu'il a la possibilité d'obtenir la liste des groupes. Cela se fait grâce à ces propriétés :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}"
LayoutMode="List" IsGroupingEnabled="True"
JumpListStyle="{StaticResource LongListSelectorJumpListStyle}">
</phone:LongListSelector>
```

Notez la présence du style LongListSelectorJumpListStyle qui est à définir.

Ensuite, il y a plusieurs modèles à créer. Premièrement le modèle des éléments, c'est-à-dire des prénoms. Il s'agit du modèle ItemTemplate. Ici, un simple TextBlock suffit :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}">
    <phone:LongListSelector.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding}" FontSize="26" Margin="12,-
12,12,6"/>
        </DataTemplate>
    </phone:LongListSelector.ItemTemplate>
</phone:LongListSelector>
```

Ensuite, nous rajoutons un modèle pour les groupes visibles dans la liste. Il s'agit du modèle GroupHeaderTemplate. Ici nous devons afficher la première lettre du groupe, entourée dans un rectangle dont le bord correspond à la couleur

d'accentuation :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}">
    ...
    <phone:LongListSelector.GroupHeaderTemplate>
        <DataTemplate>
            <Border BorderBrush="{Binding Converter={StaticResource BackgroundConverter}}" BorderThickness="2" Width="60" Margin="10" HorizontalAlignment="Left">
                <TextBlock Text="{Binding Converter={StaticResource ListConverter}}" FontSize="40" Foreground="{Binding Converter={StaticResource BackgroundConverter}}"
                    HorizontalAlignment="Center" VerticalAlignment="Center"/>
            </Border>
        </DataTemplate>
    </phone:LongListSelector.GroupHeaderTemplate>
</phone:LongListSelector>
```

Remarquez qu'étant donné que la liaison se fait sur la `List<string>`, j'utilise un converter pour n'afficher que la première lettre. Ce converter est défini classiquement en ressource :

Code : XML

```
<converter:ListConverter x:Key="ListConverter" />
```

Avec l'import d'espace de nom adéquat :

Code : XML

```
xmlns:converter="clr-namespace:DemoToolkit"
```

Le converter quant à lui est très simple, il prend simplement la première lettre du premier élément de la liste :

Code : C#

```
public class ListConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        List<string> liste = (List<string>)value;
        if (liste == null || liste.Count == 0)
            return string.Empty;
        return liste[0][0];
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Notez l'utilisation du converter `BackgroundConverter` qui permet d'obtenir la couleur d'accentuation. Ce converter est

défini dans les ressources et est un converter système. Déclarons-le ainsi que son petit frère, le `ForegroundConverter`, qui permet d'avoir la couleur du thème :

Code : XML

```
<phone:JumpListItemBackgroundConverter x:Key="BackgroundConverter"/>
<phone:JumpListItemForegroundConverter x:Key="ForegroundConverter"/>
```

Enfin, il reste le style de la liste des groupes, qui apparaît lorsqu'on clique sur un groupe. Il s'agit du style `LongListSelectorJumpListStyle` que nous avons précédemment vu, qui est également à mettre en ressources :

Code : XML

```
<Style x:Key="LongListSelectorJumpListStyle"
TargetType="phone:LongListSelector">
    <Setter Property="GridCellSize" Value="113,113"/>
    <Setter Property="LayoutMode" Value="Grid" />
    <Setter Property="ItemTemplate">
        <Setter.Value>
            <DataTemplate>
                <Border Background="{Binding
                    Converter={StaticResource BackgroundConverter}}"
                    Width="113" Height="113" Margin="6">
                    <TextBlock Text="{Binding
                        Converter={StaticResource ListConverter}}"
                        FontFamily="{StaticResource
                            PhoneFontFamilySemiBold}"
                        FontSize="48" Padding="6"
                        Foreground="{Binding
                            Converter={StaticResource ForegroundConverter}}"
                        VerticalAlignment="Center" />
                </Border>
            </DataTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Ici, le principe est le même, on utilise le converter pour afficher la première lettre. Elle sera dans un cadre au fond du thème d'accentuation.

Ce qui nous donne le code XAML complet suivant :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}"
LayoutMode="List" IsGroupingEnabled="True"
JumpListStyle="{StaticResource LongListSelectorJumpListStyle}">
    <phone:LongListSelector.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding}" FontSize="26" Margin="12,-
12,12,6"/>
        </DataTemplate>
    </phone:LongListSelector.ItemTemplate>
    <phone:LongListSelector.GroupHeaderTemplate>
        <DataTemplate>
            <Border BorderBrush="{Binding Converter={StaticResource
                BackgroundConverter}}" BorderThickness="2" Width="60" Margin="10"
                HorizontalAlignment="Left">
                <TextBlock Text="{Binding Converter={StaticResource
                    ListConverter}}" FontSize="40" Foreground="{Binding
                    Converter={StaticResource BackgroundConverter}}"
                    HorizontalAlignment="Center" VerticalAlignment="Center" />
            </Border>
        </DataTemplate>
    </phone:LongListSelector.GroupHeaderTemplate>
</phone:LongListSelector>
```

```
</Border>
</DataTemplate>
</phone:LongListSelector.GroupHeaderTemplate>
</phone:LongListSelector>
```

C'est vrai, je le reconnais, il fait un peu peur ! 😱 Nous allons le simplifier en mettant nos DataTemplate en ressource. Il suffit de les déclarer ainsi :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
...
<DataTemplate x:Key="ModeleElement">
    <TextBlock Text="{Binding}" FontSize="26" Margin="12,-
12,12,6"/>
</DataTemplate>
<DataTemplate x:Key="ModeleGroupe">
    <Border BorderBrush="{Binding Converter={StaticResource
BackgroundConverter}}" BorderThickness="2" Width="60" Margin="10"
HorizontalAlignment="Left">
        <TextBlock Text="{Binding Converter={StaticResource
ListConverter}}" FontSize="40" Foreground="{Binding
Converter={StaticResource BackgroundConverter}}"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Border>
</DataTemplate>
...
</phone:PhoneApplicationPage.Resources>
```

N'oubliez pas qu'il faut impérativement une clé à un élément présent dans les ressources. Nous pourrons alors simplifier l'écriture du contrôle ainsi :

Code : XML

```
<phone:LongListSelector ItemsSource="{Binding ListePrenoms}"
LayoutMode="List" IsGroupingEnabled="True"
JumpListStyle="{StaticResource LongListSelectorJumpListStyle}"
ItemTemplate="{StaticResource ModeleElement}"
GroupHeaderTemplate="{StaticResource ModeleGroupe}" />
```

Ce qui est beaucoup plus court et qui permet également de potentiellement réutiliser les modèles...

Je n'ai pas décrit toutes les possibilités de ce contrôle. Sachez qu'il est possible de définir un modèle pour réaliser un entête de liste et un pied de liste. Elle peut également se consulter « à plat », en utilisant la propriété :

Code : C#

```
IsFlatList="True"
```

Ce contrôle est plutôt pratique. J'ai choisi d'utiliser un converter pour afficher le titre d'un groupe en utilisant la première lettre des prénoms. Ce qui se fait aussi c'est de construire un objet directement lisible possédant une propriété Titre et une liste d'éléments, en général quelque chose comme ça :

Code : C#

```
public class Group<T> : List<T>
{
    public Group(string nom, IEnumerable<T> elements)
        : base(elements)
    {
        this.Titre = nom;
    }

    public string Titre { get; set; }
}
```

Avantages & limites du toolkit

Bon, je m'arrête là pour cette petite présentation limitée du toolkit. Nous aurons l'occasion de revoir des choses du toolkit de temps en temps dans la suite du cours.

Il rajoute des fonctionnalités très intéressantes qui manquaient aux développeurs souhaitant réaliser des applications d'envergure avec Windows Phone. Il ne s'agit bien sûr pas de contrôles officiels, mais ils sont largement reconnus par la communauté. Ce toolkit, notamment dans sa version XAML, sert parfois également de bac à sable pour les développeurs Microsoft afin de fournir des contrôles sans qu'ils ne fassent partie intégrante de la bibliothèque de contrôles officielle.

Il y a tout une communauté active et dynamique qui pousse afin que ces contrôles aient le moins de bug possible, cependant nous ne sommes pas à l'abri d'un comportement indésirable...

Les autres toolkits

Le toolkit pour Windows Phone n'est pas la seule bibliothèque de contrôles du marché. Il en existe d'autres.

Citons par l'exemple la bibliothèque Coding4Fun <http://coding4fun.codeplex.com/> qui est gratuite et qui fournit des contrôles plutôt sympathiques. Notons par exemple un contrôle qui permet de sélectionner une date ou une heure (voir la figure suivante).



Le choix d'une durée grâce au toolkit Coding4Fun

D'autres sont payants et développés par des sociétés tierces, citons par exemple les contrôles de la société Telerik ou encore de Syncfusion. Ils fournissent des contrôles qui nous simplifient la tâche et qui permettent de gagner du temps dans nos développements. N'hésitez pas à y jeter un coup d'œil, c'est souvent plus intéressant d'acheter un contrôle déjà tout fait que de passer du temps (beaucoup de temps !) à le réaliser.

- Le toolkit pour Windows Phone est une bibliothèque de contrôles fournie gratuitement qui permet d'enrichir vos applications avec des contrôles très pratiques.
- Il y a beaucoup de bibliothèques qui existent et qui proposent des contrôles complets, s'adaptant à beaucoup de situation. Certaines sont gratuites, d'autres payantes. N'hésitez pas à les consulter, il y a souvent de bonnes surprises.

Le contrôle de cartes (Map)

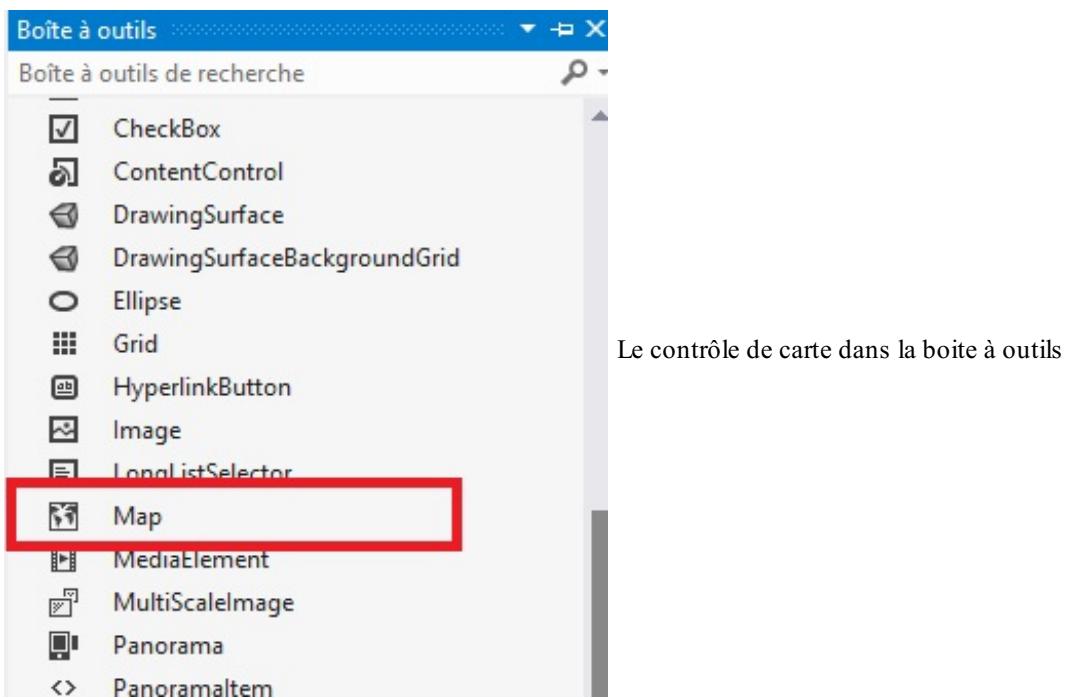
Avant de commencer à regarder le contrôle Map, il faut savoir qu'il est différent entre la version du SDK pour développer pour Windows Phone 7 et celui pour développer pour Windows Phone 8. Le premier est le contrôle de cartes de Microsoft alors que pour Windows Phone 8, c'est celui réalisé par Nokia qui est utilisé. Ils se ressemblent cependant fortement dans leurs utilisations, mais celui de Nokia a une particularité intéressante lui permettant de faire des choses hors connexion que ne permet pas celui de Microsoft. Mais rassurez-vous, nul besoin de posséder un téléphone Nokia pour pouvoir s'en servir, il fonctionne pour tous les téléphones Windows Phone 8.

Je vais présenter ici le contrôle de Windows Phone 8, mais sachez qu'il est également possible d'utiliser l'ancien contrôle avec des applications Windows Phone 8. Il s'agit d'un contrôle très complet et bien pratique : [le contrôle Map](#). Il permet d'embarquer une carte dans notre application. Nous allons pouvoir afficher la carte de France, la carte d'Europe, etc ... définir des positions, calculer des itinéraires ... Bref, tout plein de choses qui peuvent servir à nos téléphones équipés de GPS.

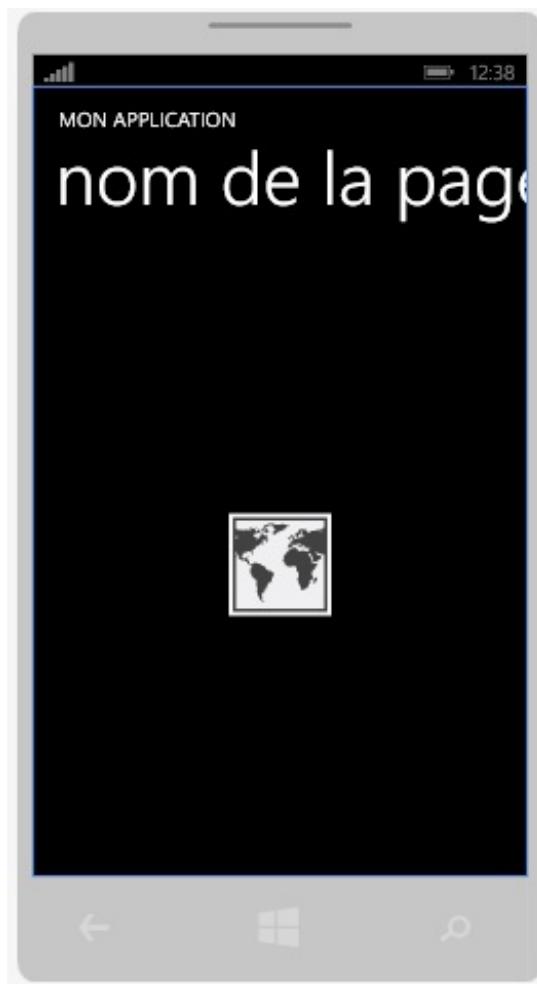
Découvrons à présent ce fameux contrôle.

Présentation et utilisation

Pour utiliser le contrôle Map, le plus simple est de le faire glisser depuis la boîte à outils pour le mettre par exemple à l'intérieur de la grille, comme indiqué à la figure suivante.



Visual Studio nous ajoute le contrôle et nous pouvons voir dans la fenêtre de design une mini carte du monde (voir la figure suivante).



Le contrôle map dans le designer

Après l'ajout du contrôle, si nous regardons le XAML, Visual Studio nous a ajouté l'espace de nom suivant :

Code : XML

```
xmlns:Controls="clr-  
namespace:Microsoft.Phone.Maps.Controls;assembly=Microsoft.Phone.Maps"
```

ainsi que le XAML positionnant le contrôle :

Code : XML

```
<Controls:Map />
```

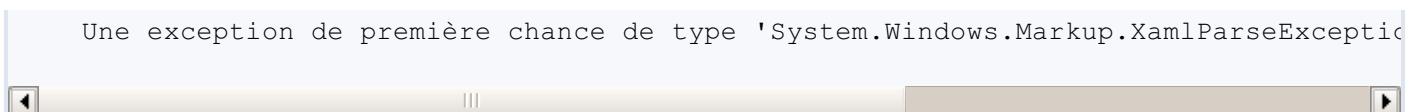
Personnellement, je change le « Controls » en « carte » et je donne le nom « Carte » à mon contrôle :

Code : XML

```
<carte:Map Name="Carte" />
```

Si vous démarrez l'émulateur tout de suite, vous allez avoir un problème. Visual Studio nous lève une exception de type :

Code : Autre

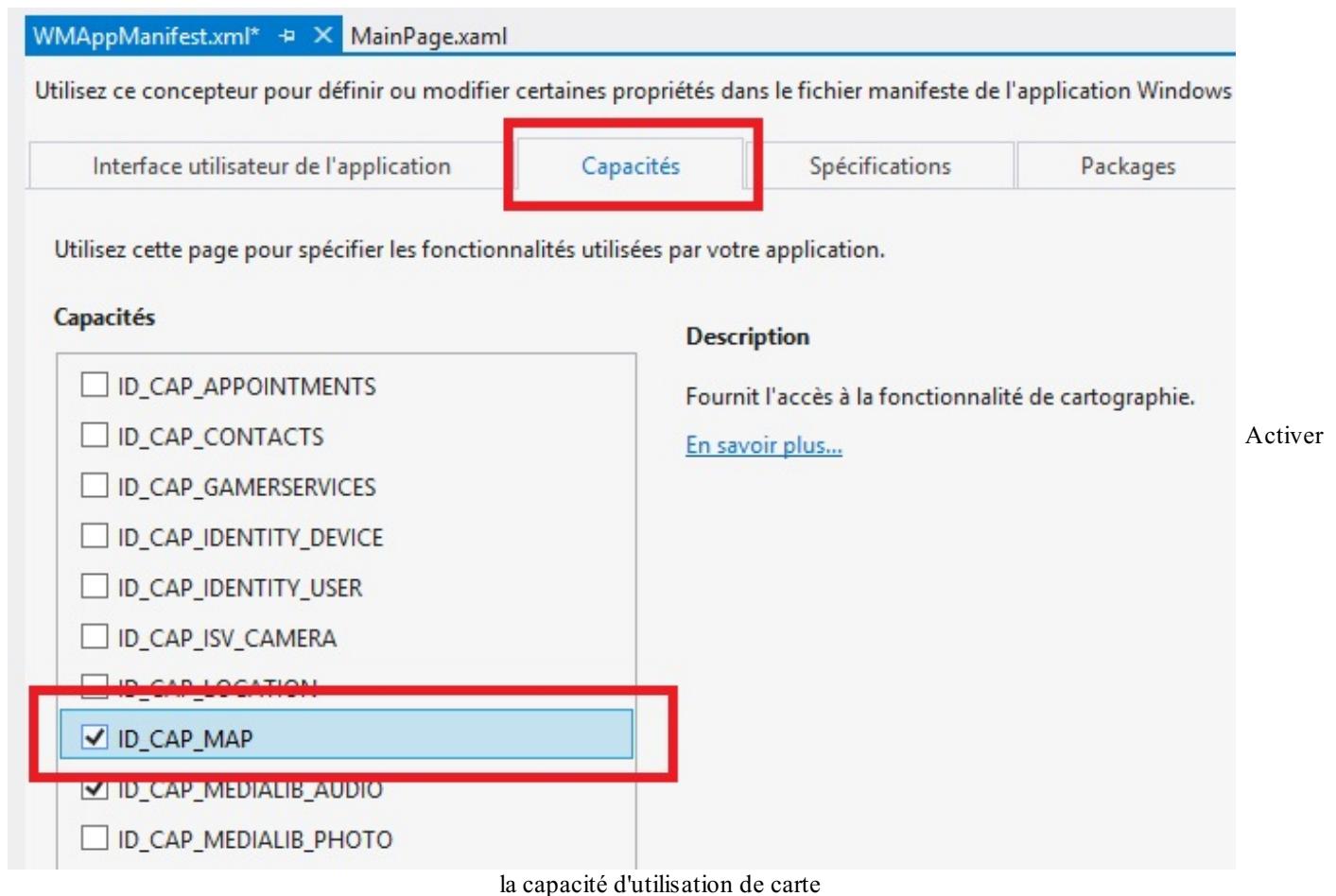


Mais si on fouille un peu dans les `InnerException`, on trouve plutôt :

Code : Autre

```
Access to Maps requires ID_CAP_MAP to be defined in the manifest
```

En fait, pour utiliser le contrôle de cartes, nous devons déclarer notre application comme utilisatrice du contrôle de cartes. Cela permettra notamment aux personnes qui veulent télécharger notre application de savoir qu'elle utilise le contrôle de carte. On appelle cela les capacités. Pour déclarer une capacité, nous allons avoir besoin de double-cliquer sur le fichier `WMAppManifest.xml` (sous `Properties` dans l'explorateur de solutions) et d'aller dans l'onglet `Capacités`. Ensuite, il faut cocher la capacité `ID_CAP_MAP`, comme l'indique la figure suivante.



Et voilà, maintenant en démarrant l'émulateur, nous allons avoir une jolie carte du monde (si vous êtes connectés à Internet) - voir la figure suivante.



La carte s'affiche dans l'émulateur

Interactions avec le contrôle

Et si nous prenions le contrôle de la carte ?

On peut tout faire avec cette carte, comme se positionner à un emplacement précis grâce à des coordonnées GPS, ajouter des marques pour épinglez des lieux, zoomer, dé-zoomer, etc.

Pour illustrer tout cela, nous allons manipuler cette fameuse carte. Première chose à faire, nous allons afficher des boutons pour zoomer et dé-zoomer. Utilisons par exemple une barre d'application pour ce faire et intégrons-y les deux icônes suivantes, présentes dans le répertoire du SDK :

- add.png
- minus.png

N'hésitez pas à aller faire un tour dans le chapitre de la barre d'application si vous avez un doute sur la marche à suivre. Voici donc le XAML de la barre d'application utilisée :

Code : XML

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:AppBar IsVisible="True" IsMenuEnabled="True">
        <shell:AppBarIconButton
            IconUri="/Assets/Icones/add.png" Text="Zoom" Click="Zoom_Click"/>
        <shell:AppBarIconButton
            IconUri="/Assets/Icones/minus.png" Text="Dé-zoom"
            Click="Dezoom_Click"/>
    </shell:AppBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Comme d'habitude, prenez garde au chemin des icônes.

Et le code-behind associé :

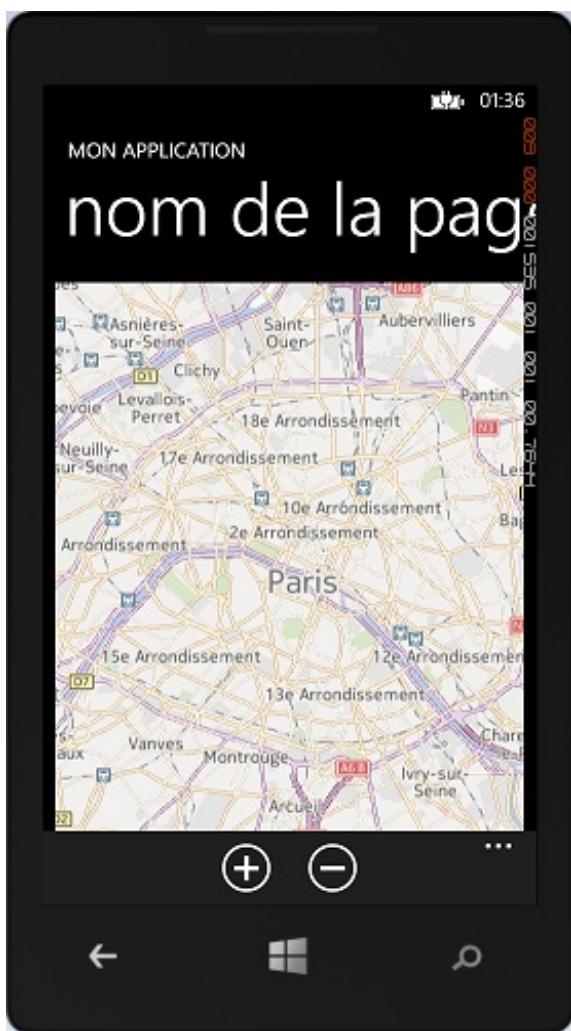
Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void Zoom_Click(object sender, EventArgs e)
    {
        Carte.ZoomLevel++;
    }

    private void Dezoom_Click(object sender, EventArgs e)
    {
        Carte.ZoomLevel--;
    }
}
```

Vous pouvez voir que nous pouvons facilement zoomer ou dé-zoomer en utilisant les boutons de la barre d'application, et tout ça grâce à la propriété `ZoomLevel`. De plus, nous pouvons nous déplacer sur la carte en faisant glisser notre doigt (ou notre souris !), comme indiqué à la figure suivante.



Déplacement et zoom pour afficher la carte de Paris

Tout au long de l'utilisation, sur un téléphone, nous pouvons également zoomer et dé-zoomer grâce au geste qui consiste à ramener ses deux doigts vers le centre de l'écran ou au contraire à les écarter pour dé-zoomer (geste que l'on appelle le *Pinch-to-*

zoom ou le *Stretch-to-zoom*). C'est par contre un peu plus difficile à faire à la souris dans l'émulateur, sachant que nous avons quand même la possibilité de zoomer en double-cliquant... Mais cela sera quand même bien plus simple dans l'émulateur avec la barre d'application. 😊

Nous pouvons aussi centrer la carte à un emplacement précis. Pour cela, on utilise la propriété `Center` qui est du type `GeoCoordinate`. Il s'agit d'une classe contenant des coordonnées GPS, avec notamment la latitude et la longitude. Nous pouvons utiliser des coordonnées pour centrer la carte à un emplacement désiré. Par exemple :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    Carte.ZoomLevel = 17;
    Carte.Center = new GeoCoordinate { Latitude = 48.858115,
    Longitude = 2.294710 };
}
```

qui centre la carte sur la tour Eiffel. On n'oubliera pas d'inclure l'espace de nom permettant d'utiliser le type `GeoCoordinate` :

Code : C#

```
using System.Device.Location;
```

La carte est également disponible en mode satellite (ou aérien), il suffit de changer la propriété `CartographicMode` du contrôle pour passer en mode aérien avec :

Code : C#

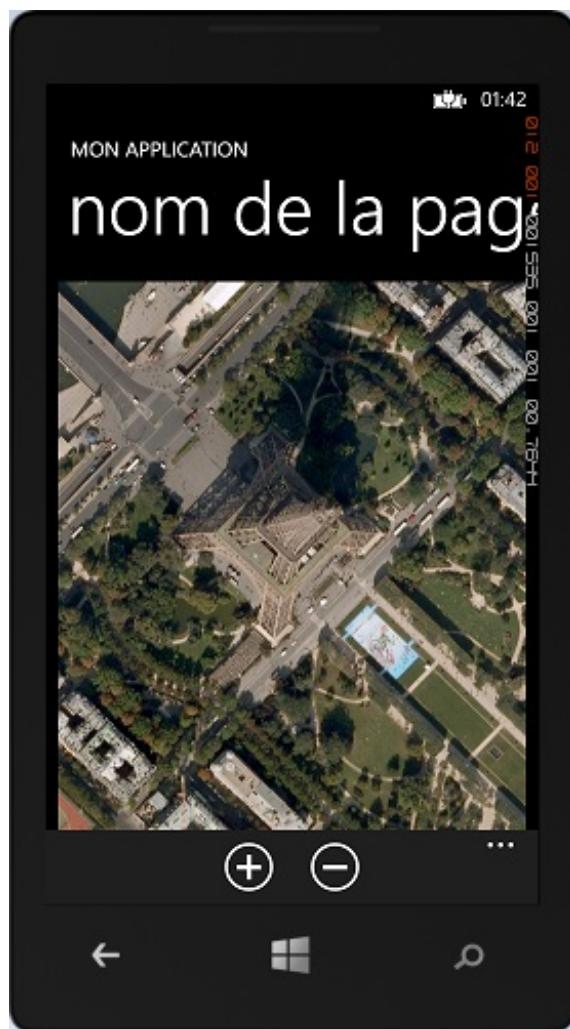
```
Carte.CartographicMode = MapCartographicMode.Aerial;
```

Disponible avec l'import d'espace de nom suivant :

Code : C#

```
using Microsoft.Phone.Maps.Controls;
```

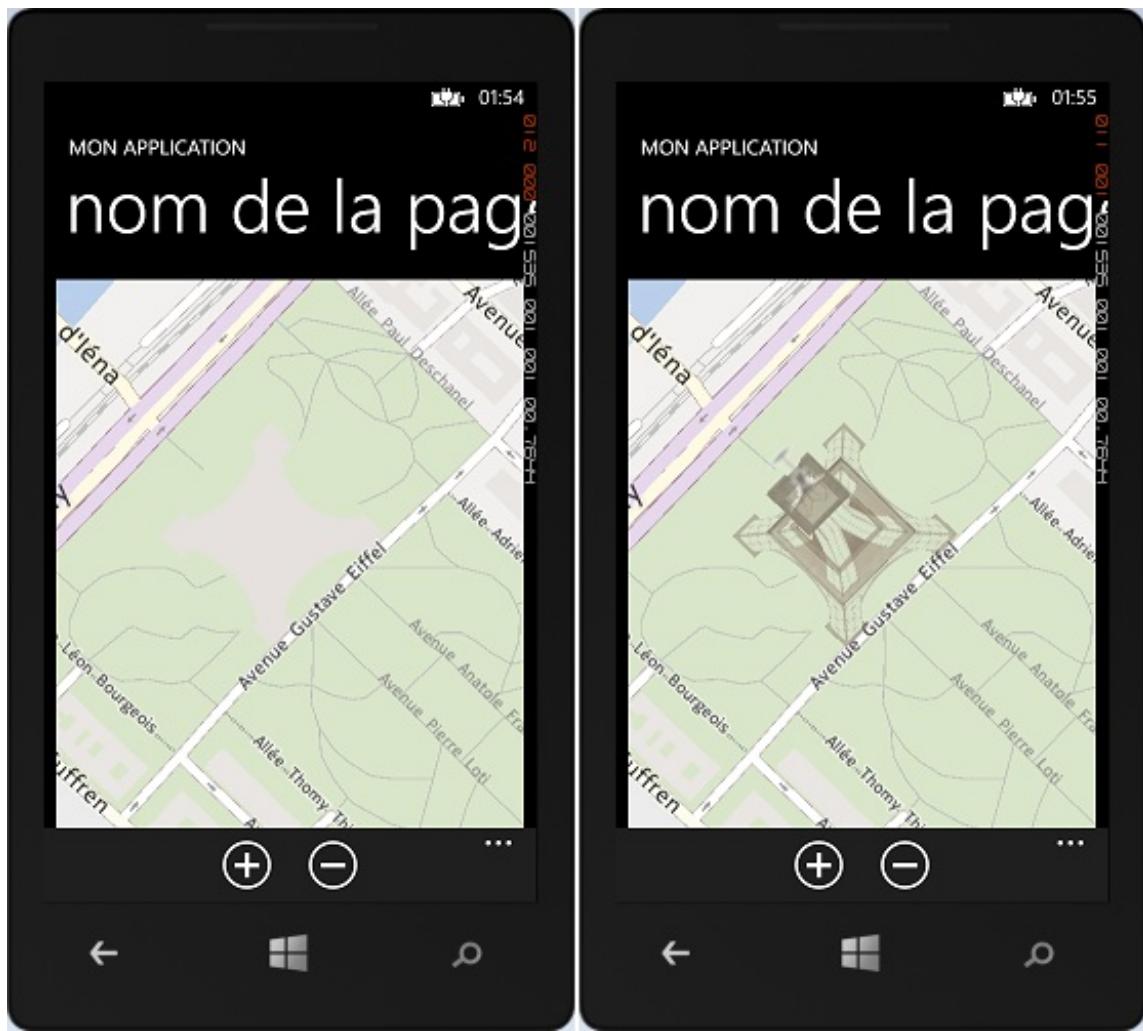
Vous pouvez voir le résultat à la figure suivante.



La carte en mode aérien

Le mode route, par défaut, correspond à la valeur d'énumération `MapCartographicMode.Road`, sachant qu'il existe également la valeur `Terrain` et `Hybrid`.

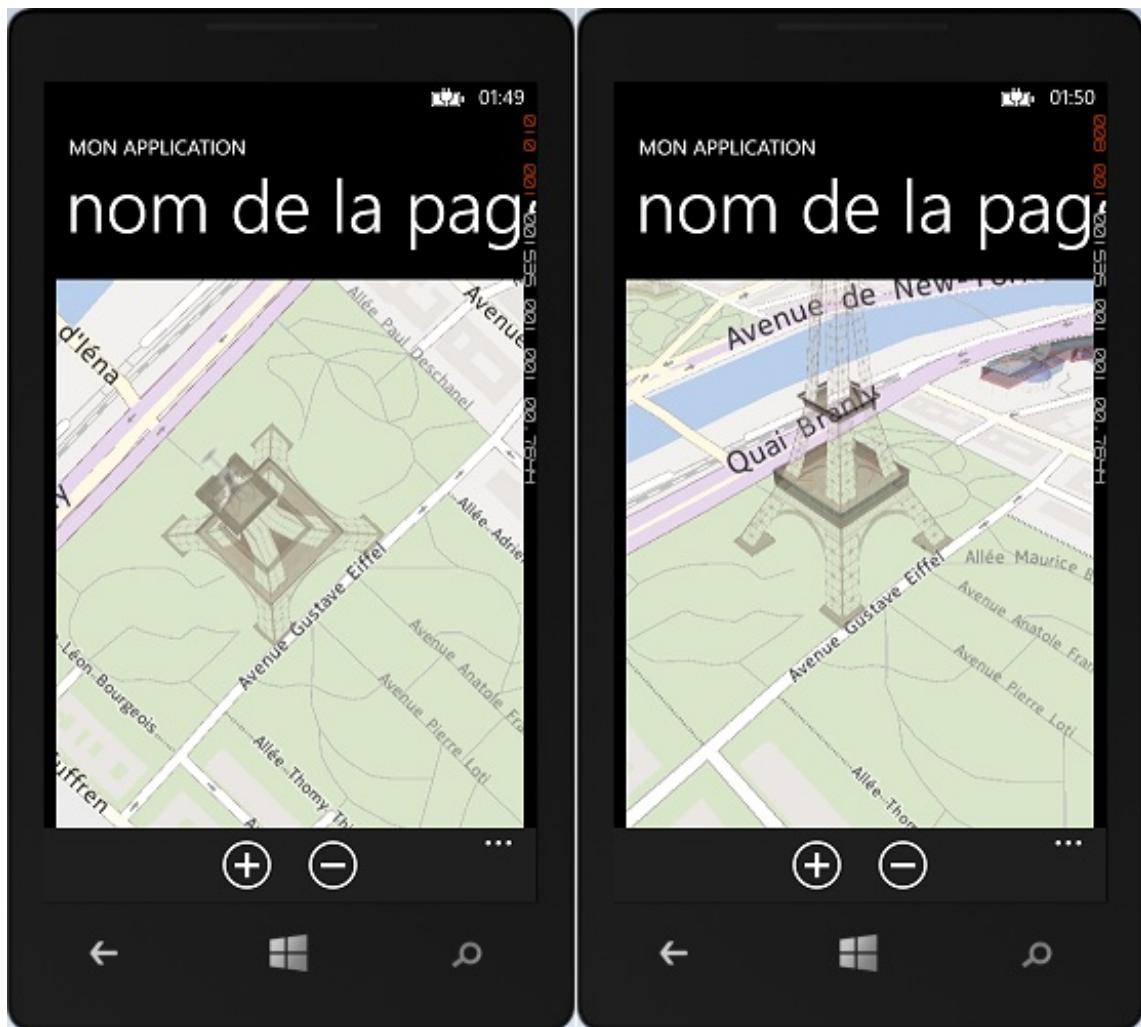
Nous pouvons inclure des points de repères dans la carte grâce à la propriété `LandmarksEnabled` en la passant à `True`. À la figure suivante, la même carte sans et avec points de repères.



La carte peut afficher
des points de repères

Vous noterez la différence 😊.

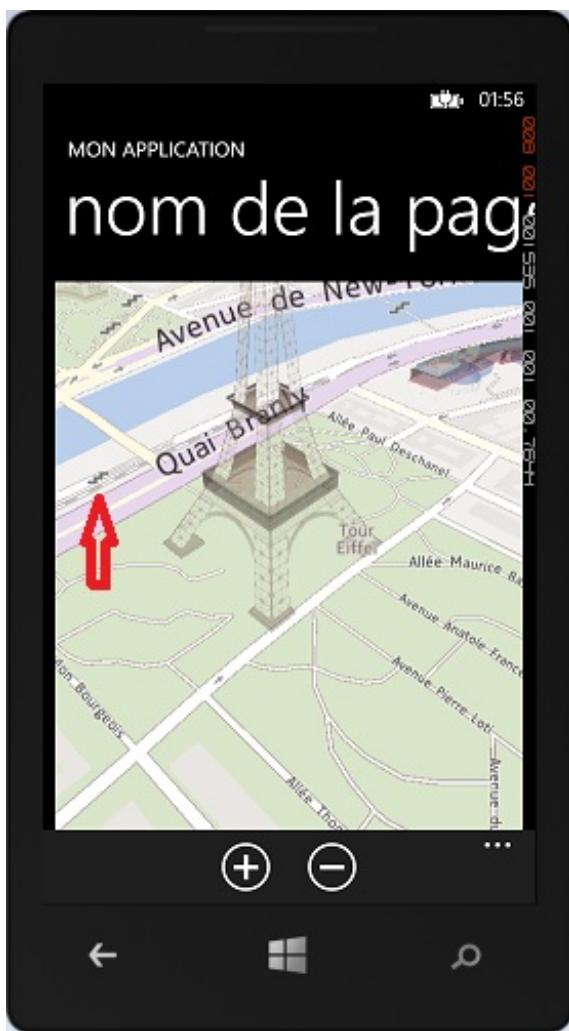
Vous pouvez également indiquer un degré d'inclinaison grâce à la propriété Pitch. Voici par exemple à la figure suivante la carte avec un degré d'inclinaison de 0 et de 45.



La carte sans et avec

un degré d'inclinaison

Nous pouvons également inclure les informations piétonnes, comme les escaliers grâce à la propriété `PedestrianFeaturesEnabled` en la passant à `True` (voir la figure suivante).



La carte avec les informations piétonnes



Le zoom doit être supérieur à 7 et le pitch supérieur à 25 pour avoir accès à cette fonctionnalité.

Epingler des points d'intérêt

Une des grandes forces du contrôle Map est qu'on peut dessiner n'importe quoi par dessus, pour par exemple épingle des points d'intérêts sur la carte. Cela permet de mettre en valeur certains lieux et pourquoi pas afficher une information contextuelle complémentaire. Le principe est simple, il suffit d'ajouter des coordonnées GPS et une punaise apparaît automatiquement à cet emplacement.

Cela ne se fait pas tout seul, mais grâce au Windows Phone Toolkit que nous venons de voir. Ajoutez une référence à celui-ci, comme nous venons de le faire, et utilisons la classe PushPin qui représente une telle épingle. Il faut dans un premier temps importer l'espace de nom :

Code : XML

```
xmlns:toolkitcarte="clr-  
namespace:Microsoft.Phone.Maps.Toolkit;assembly=Microsoft.Phone.Controls.Toolkit"
```

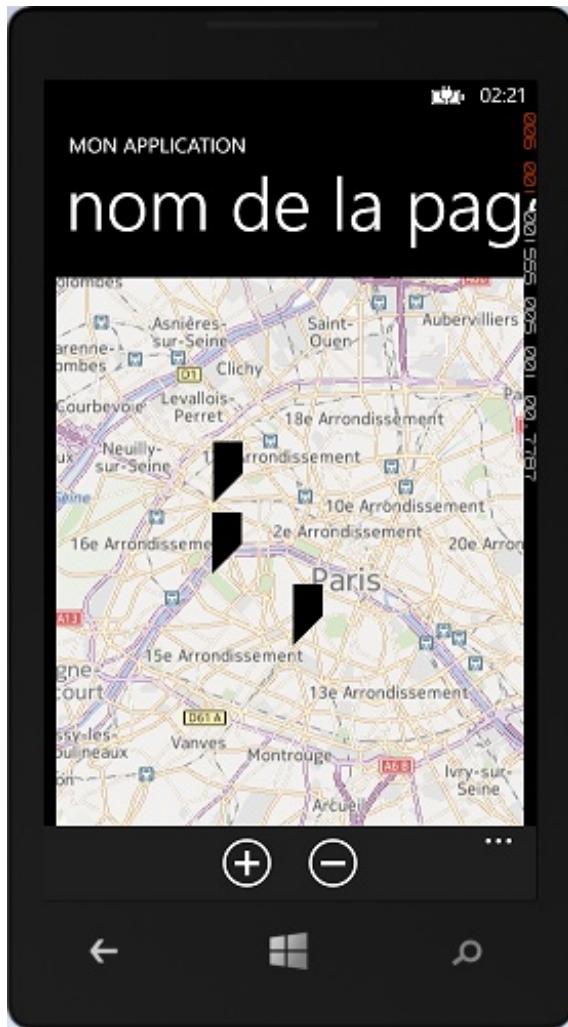
puis ajouter des éléments PushPin :

Code : XML

```
<carte:Map Name="Carte">  
    <toolkitcarte:MapExtensions.Children>  
        <toolkitcarte:Pushpin GeoCoordinate="48.842276, 2.321747" />  
        <toolkitcarte:Pushpin GeoCoordinate="48.858115, 2.294710" />  
        <toolkitcarte:Pushpin GeoCoordinate="48.873783, 2.294930" />
```

```
</toolkitcarte:MapExtensions.Children>
</carte:Map>
```

Et nous aurons ce résultat (voir la figure suivante).



Les punaises pour épingle des points d'intérêts

Pour avoir le même rendu avec le code behind, il suffit d'avoir le XAML suivant :

Code : XML

```
<carte:Map Name="Carte" />
```

Et le code suivant :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    Carte.ZoomLevel = 12;
    Carte.Center = new GeoCoordinate { Latitude = 48.863134,
Longitude = 2.320518 };

    var elts = MapExtensions.GetChildren(Carte);
    MapExtensions.Add(elts, new Pushpin(), new GeoCoordinate {
Longitude = 2.321747, Latitude = 48.842276 });
}
```

```
MapExtensions.Add(elts, new Pushpin(), new GeoCoordinate {
    Longitude = 2.294710, Latitude = 48.858115 });
    MapExtensions.Add(elts, new Pushpin(), new GeoCoordinate {
    Longitude = 2.294930, Latitude = 48.873783 });
}
```

La punaise est bien sûr stylisable à souhait, car celle-là est un peu triste. Prenez par exemple celle-ci, permettant de remplacer la grosse punaise par un petit point bleu :

Code : XML

```
<phone:PhoneApplicationPage.Resources>
    <ControlTemplate x:Key="PushpinControlTemplate"
        TargetType="toolkitcarte:Pushpin">
        <Ellipse Fill="{TemplateBinding Background}"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Width="20"
            Height="20"
            Stroke="{TemplateBinding Foreground}"
            StrokeThickness="3" />
    </ControlTemplate>

    <Style TargetType="toolkitcarte:Pushpin"
        x:Key="PushpinControlTemplateEllipse">
        <Setter Property="Template" Value="{StaticResource
            PushpinControlTemplate}" />
        <Setter Property="Background" Value="Blue" />
        <Setter Property="Foreground" Value="White" />
    </Style>
</phone:PhoneApplicationPage.Resources>
```

Que nous pourrons utiliser ainsi :

Code : XML

```
<carte:Map Name="Carte">
    <toolkitcarte:MapExtensions.Children>
        <toolkitcarte:Pushpin GeoCoordinate="48.842276, 2.321747"
            Style="{StaticResource PushpinControlTemplateEllipse}" />
        <toolkitcarte:Pushpin GeoCoordinate="48.858115, 2.294710"
            Style="{StaticResource PushpinControlTemplateEllipse}" />
        <toolkitcarte:Pushpin GeoCoordinate="48.874956, 2.350690"
            Style="{StaticResource PushpinControlTemplateEllipse}" />
    </toolkitcarte:MapExtensions.Children>
</carte:Map>
```

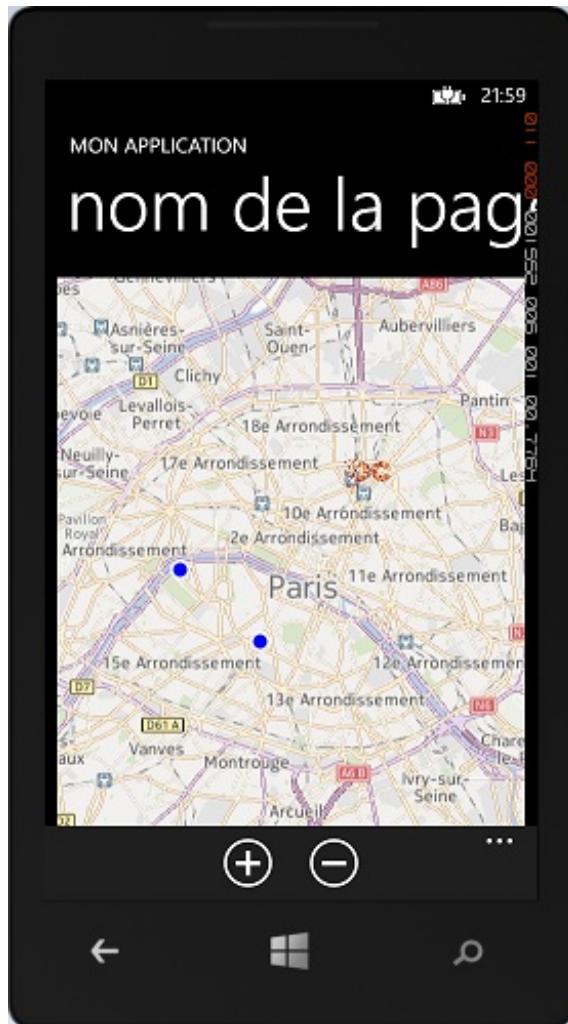
ou la même chose avec le code behind :

Code : C#

```
var elts = MapExtensions.GetChildren(Carte);
MapExtensions.Add(elts, new Pushpin { Style =
    Resources["PushpinControlTemplateEllipse"] as Style }, new
    GeoCoordinate { Longitude = 2.321747, Latitude = 48.842276 });
MapExtensions.Add(elts, new Pushpin { Style =
    Resources["PushpinControlTemplateEllipse"] as Style }, new
    GeoCoordinate { Longitude = 2.294710, Latitude = 48.858115 });
MapExtensions.Add(elts, new Pushpin { Style =
```

```
Resources["PushpinControlTemplateEllipse"] as Style }, new  
GeoCoordinate { Longitude = 2.350690, Latitude = 48.874956 });
```

Ce qui donne le résultat affiché à la figure suivante.



Les punaises ont du style !

D'où vient cette punaise OpenClassrooms ? Simplement de mon autre style utilisant une image comme punaise :

Code : XML

```
<ControlTemplate x:Key="PushpinControlTemplateImage"  
TargetType="toolkitcarte:Pushpin">  
    <Image Source="http://open-e-education-  
2013.openclassrooms.com/img/logos/logo-openclassrooms.png"  
Width="60" Height="60" />  
</ControlTemplate>  
  
<Style TargetType="toolkitcarte:Pushpin"  
x:Key="PushpinControlTemplateImageStyle">  
    <Setter Property="Template" Value="{StaticResource  
PushpinControlTemplateImage}" />  
</Style>
```

Le logo d'OpenClassrooms pour indiquer l'emplacement des locaux d'OpenClassrooms ? C'est pas la classe ça ? 🤪
Ajouter des punaises en spécifiant des coordonnées dans le XAML, c'est bien. Mais nous pouvons également le faire par binding !

Voyez par exemple avec ce XAML :

Code : XML

```
<carte:Map Name="Carte">
    <toolkitcarte:MapExtensions.Children>
        <toolkitcarte:Pushpin GeoCoordinate="{Binding
PositionTourEiffel}" Style="{StaticResource
PushpinControlTemplateEllipse}" />
    </toolkitcarte:MapExtensions.Children>
</carte:Map>
```

La propriété GeoCoordinate de l'objet Pushpin est liée à la propriété PositionTourEiffel, qui sera du genre :

Code : C#

```
private GeoCoordinate positionTourEiffel;
public GeoCoordinate PositionTourEiffel
{
    get { return positionTourEiffel; }
    set { NotifyPropertyChanged(ref positionTourEiffel, value); }
}
```

Que l'on pourra alimenter avec :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    Carte.ZoomLevel = 12;
    Carte.Center = new GeoCoordinate { Latitude = 48.863134,
Longitude = 2.320518 };
    PositionTourEiffel = new GeoCoordinate { Longitude = 2.321747,
Latitude = 48.842276 };
    DataContext = this;
}
```

N'oubliez pas d'implémenter correctement INotifyPropertyChanged, mais bon, vous savez faire maintenant. 😊
On peut même leur rajouter un petit texte grâce à la propriété Content :

Code : XML

```
<toolkitcarte:Pushpin GeoCoordinate="{Binding PositionTourEiffel}"
Content="{Binding Texte}" />
```

Avec :

Code : C#

```
private string texte;
public string Texte
{
    get { return texte; }
    set { NotifyPropertyChanged(ref texte, value); }
```

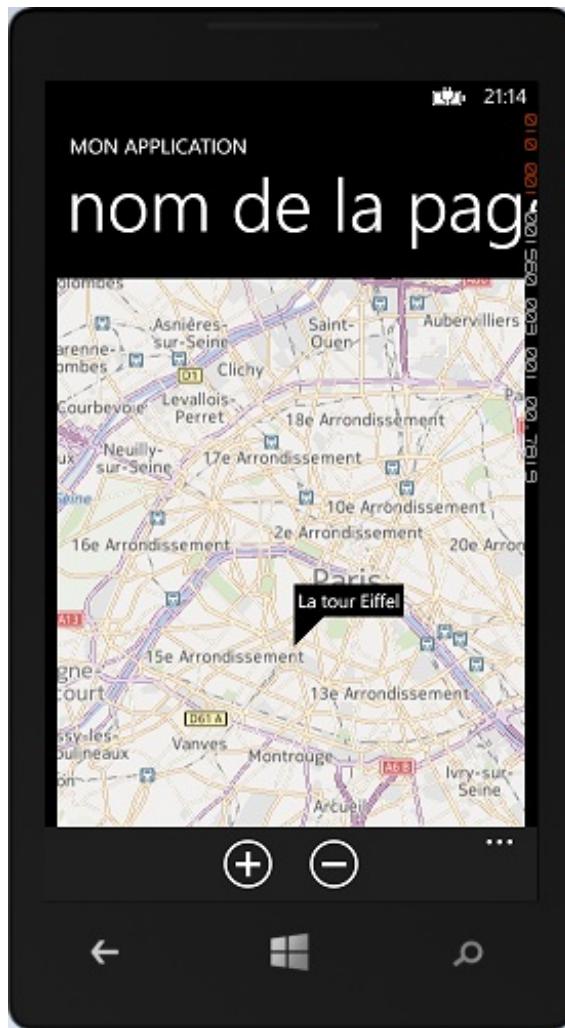
}

Et :

Code : C#

```
Texte = "La tour Eiffel";
```

Ce qui donne la figure suivante.



Légende sur les punaises

Notez que j'ai retiré le style ellipse bleue car ce style ne prenait pas en charge la propriété Content. Pour ce faire, il faudrait modifier le style pour avoir, par exemple :

Code : XML

```
<ControlTemplate x:Key="PushpinControlTemplate"
    TargetType="toolkitcarte:Pushpin">
    <StackPanel>
        <ContentPresenter Content="{TemplateBinding Content}" />
        <Ellipse Fill="{TemplateBinding Background}"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Width="20"
            Height="20"
            Stroke="{TemplateBinding Foreground}"
```

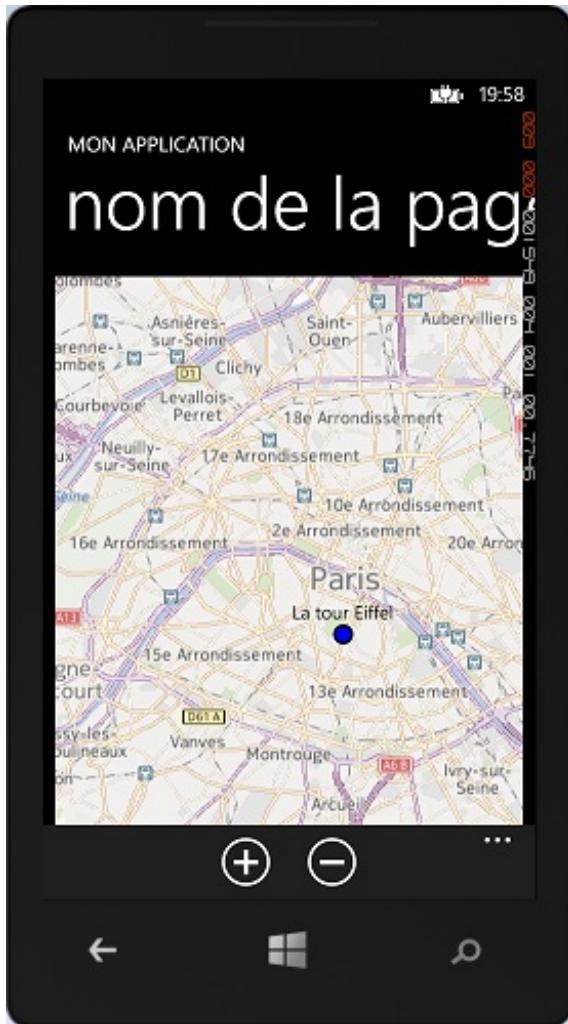
```
        StrokeThickness="3" />
    </StackPanel>
</ControlTemplate>
```

Et le contrôle serait :

Code : XML

```
<carte:Map Name="Carte">
    <toolkitcarte:MapExtensions.Children>
        <toolkitcarte:Pushpin GeoCoordinate="{Binding
PositionTourEiffel}" Style="{StaticResource
PushpinControlTemplateEllipse}" Content="{Binding Texte}"
Foreground="Black" />
    </toolkitcarte:MapExtensions.Children>
</carte:Map>
```

Pour le résultat, observez la figure suivante.



Le style avec une légende

Et si on a plusieurs punaises à lier ? On pourrait être tentés d'utiliser le binding, mais à ce jour ce n'est pas fonctionnel. Peut-être un bug à corriger ?

On peut quand même s'en sortir grâce au code-behind. La première chose à faire est de définir un MapItemsControl possédant un template :

Code : XML

```
<carte:Map Name="Carte">
    <toolkitcarte:MapExtensions.Children>
        <toolkitcarte:MapItemsControl>
            <toolkitcarte:MapItemsControl.ItemTemplate>
                <DataTemplate>
                    <toolkitcarte:Pushpin GeoCoordinate="{Binding}"
Style="{StaticResource PushpinControlTemplateEllipse}" />
                </DataTemplate>
            </toolkitcarte:MapItemsControl.ItemTemplate>
        </toolkitcarte:MapItemsControl>
    </toolkitcarte:MapExtensions.Children>
</carte:Map>
```

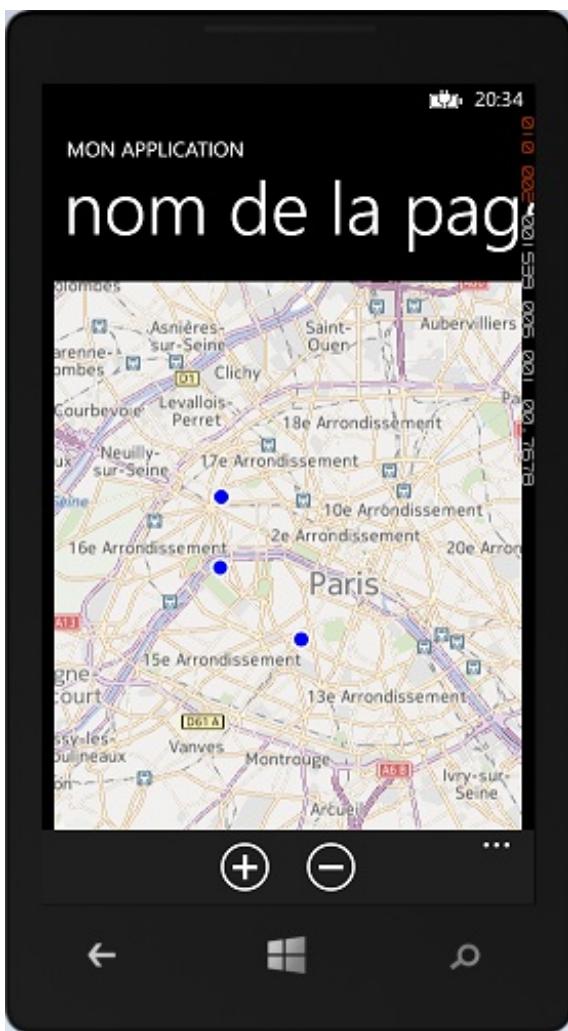
Il faudra ensuite lier par exemple une liste de positions à ce MapItemsControl via code-behind :

Code : C#

```
List<GeoCoordinate> maListeDePositions = new List<GeoCoordinate>
{
    new GeoCoordinate { Longitude = 2.321747, Latitude = 48.842276 },
    new GeoCoordinate { Longitude = 2.294710, Latitude = 48.858115 },
    new GeoCoordinate { Longitude = 2.294930, Latitude = 48.873783 }
};

MapItemsControl mapItemsControl =
MapExtensions.GetChildren(Carte).OfType<MapItemsControl>().FirstOrDefault();
mapItemsControl.ItemsSource = maListeDePositions;
```

Comme ceci, cela fonctionne et donne le résultat que vous voyez sur la figure suivante.



Plusieurs punaises liées par code-behind

Personnellement, j'adore ces punaises ! 😊

Elles peuvent également réagir à un clic, ce qui nous laisse l'opportunité d'afficher par exemple des informations complémentaires sur cette position. On utilisera l'événement Tap du contrôle Pushpin. Sauf qu'en général, si on utilise une liste de positions comme on l'a fait juste au-dessus, nous ne disposons pas d'informations suffisantes pour savoir quelle punaise a été cliquée. Ce qu'on peut faire à ce moment-là, c'est utiliser la propriété Tag du contrôle, qui est une espèce d'objet fourre-tout pour passer des informations.



La propriété Tag est une propriété héritée de la classe `FrameworkElement`. Tous les contrôles possèdent donc cette propriété fourre-tout.

Illustrons ce point en créant une nouvelle classe :

Code : C#

```
public class MaPosition
{
    public GeoCoordinate Position { get; set; }
    public string Informations { get; set; }
}
```

Puis, changeons notre propriété `MaListeDePositions` pour avoir une liste d'objets `MaPosition` :

Code : C#

```

private IEnumerable<MaPosition> maListeDePositions;
public IEnumerable<MaPosition> MaListeDePositions
{
    get { return maListeDePositions; }
    set { NotifyPropertyChanged(ref maListeDePositions, value); }
}

```

Qui sera alimentée de cette façon :

Code : C#

```

MaListeDePositions = new List<MaPosition>
{
    new MaPosition { Position = new GeoCoordinate { Longitude =
2.321747, Latitude = 48.842276 }, Informations = "Tour Eiffel" },
    new MaPosition { Position = new GeoCoordinate { Longitude =
2.294710, Latitude = 48.858115 }, Informations = "Tour
Montparnasse" },
    new MaPosition { Position = new GeoCoordinate { Longitude =
2.294930, Latitude = 48.873783 }, Informations = "Arc de triomphe" }
};

```

Puis changeons le XAML pour avoir :

Code : XML

```

<carte:Map Name="Carte">
    <toolkitcarte:MapExtensions.Children>
        <toolkitcarte:MapItemsControl ItemsSource="{Binding
PositionList}">
            <toolkitcarte:MapItemsControl.ItemTemplate>
                <DataTemplate>
                    <toolkitcarte:Pushpin GeoCoordinate="{Binding
Position}" Style="{StaticResource PushpinControlTemplateEllipse}"
Tag="{Binding Informations}" Tap="Pushpin_Tap" />
                </DataTemplate>
            </toolkitcarte:MapItemsControl.ItemTemplate>
        </toolkitcarte:MapItemsControl>
    </toolkitcarte:MapExtensions.Children>
</carte:Map>

```

Notons que la propriété `GeoCoordinate` de la punaise est liée à la propriété `Position` de notre classe et que la propriété `Tag` est liée à la propriété `Informations`. Ce qui nous permet, dans l'événement associé, de faire :

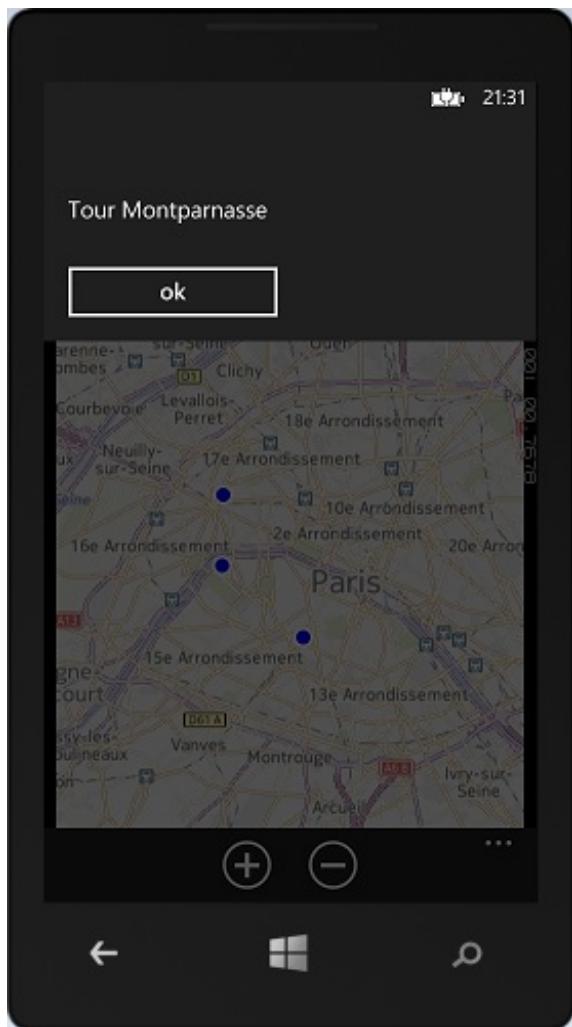
Code : C#

```

private void Pushpin_Tap(object sender, GestureEventArgs e)
{
    MessageBox.Show(((FrameworkElement)sender).Tag.ToString());
}

```

Et nous aurons ce résultat (voir la figure suivante).



Le clic sur une punaise nous déclenche l'affichage d'un message

Pratique.

Afficher un itinéraire

Bonne nouvelle, avec Windows Phone 8, il est très facile de calculer et d'afficher un itinéraire entre deux points. Prenons une carte classique :

Code : XML

```
<maps:Map x:Name="Carte" />
```

Et calculons dans le code-behind un itinéraire entre des coordonnées de départ, disons la Tour Montparnasse, jusqu'aux Champs-Élysées. Il suffit d'utiliser un objet [GeocodeQuery](#) et de faire :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private List<GeoCoordinate> coordonnesTrajet;

    public MainPage()
    {
        InitializeComponent();

        Carte.ZoomLevel = 13;
        Carte.Center = new GeoCoordinate { Latitude = 48.863134,
    Longitude = 2.320518 };
        coordonnesTrajet = new List<GeoCoordinate>();
```

```
        CalculerItineraire();
    }

    private void CalculerItineraire()
    {
        GeoCoordinate positionDepart = new GeoCoordinate(48.858115,
2.294710);
        coordonneesTrajet.Add(positionDepart);

        GeocodeQuery geocodeQuery = new GeocodeQuery
        {
            SearchTerm = "avenue des champs-élysées, Paris",
            GeoCoordinate = positionDepart
        };

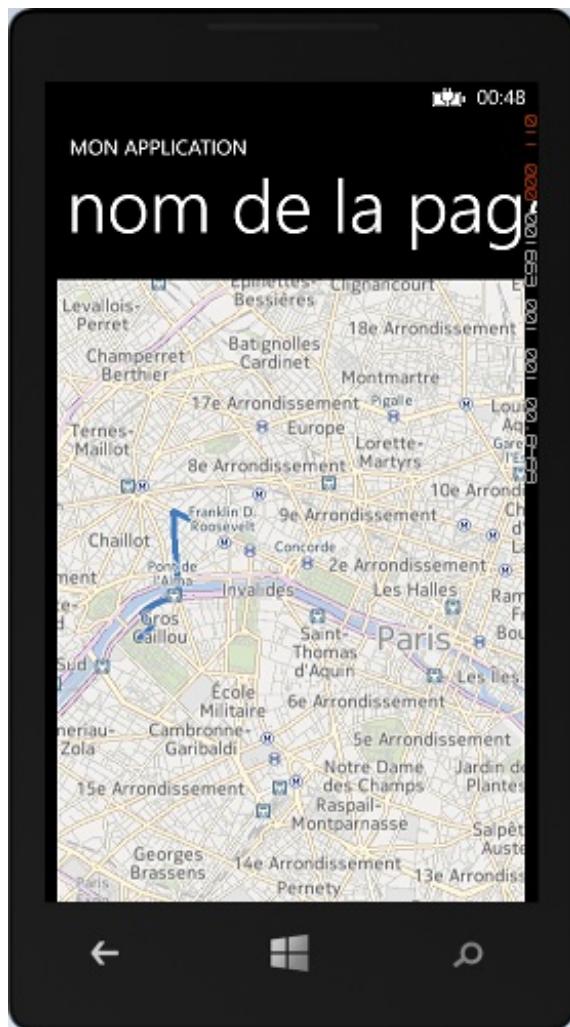
        geocodeQuery.QueryCompleted += geocodeQuery_QueryCompleted;
        geocodeQuery.QueryAsync();
    }

    private void geocodeQuery_QueryCompleted(object sender,
QueryCompletedEventArgs<IList<MapLocation>> e)
{
    if (e.Error == null)
    {
        RouteQuery routeQuery = new RouteQuery();
        coordonneesTrajet.Add(e.Result[0].GeoCoordinate);
        routeQuery.Waypoints = coordonneesTrajet;
        routeQuery.QueryCompleted += routeQuery_QueryCompleted;
        routeQuery.QueryAsync();
    }
}

void routeQuery_QueryCompleted(object sender,
QueryCompletedEventArgs<Route> e)
{
    if (e.Error == null)
    {
        Carte.AddRoute(new MapRoute(e.Result));
    }
}
```

Dans l'objet GeocodeQuery il suffit de renseigner la destination souhaitée et la méthode `QueryAsync()` calcule automatiquement le trajet et nous fournit de quoi construire une route à ajouter à la carte grâce à la méthode `AddRoute`. Remarquez que la destination souhaitée peut également être des coordonnées GPS.

Et nous aurons la figure suivante.



Calcul d'itinéraire entre deux points

Il est bien sûr possible d'appliquer la même technique que pour le WebClient afin de pouvoir utiliser les mots-clés await et async. Il suffit de rajouter ces méthodes dans une classe d'extensions :

Code : C#

```
public static class Extensions
{
    public static Task<IList<MapLocation>> QueryLocationAsync(this
GeocodeQuery geocodeQuery)
    {
        TaskCompletionSource<IList<MapLocation>>
taskCompletionSource = new
TaskCompletionSource<IList<MapLocation>>();
        EventHandler<QueryCompletedEventArgs<IList<MapLocation>>>
queryCompletedHandler = null;
        queryCompletedHandler = (s, e) =>
        {
            geocodeQuery.QueryCompleted -= queryCompletedHandler;
            if (e.Error != null)
                taskCompletionSource.TrySetException(e.Error);
            else
                taskCompletionSource.TrySetResult(e.Result);
        };
        geocodeQuery.QueryCompleted += queryCompletedHandler;
        geocodeQuery.QueryAsync();

        return taskCompletionSource.Task;
    }

    public static Task<Route> QueryRouteAsync(this RouteQuery
```

```
        routeQuery)
    {
        TaskCompletionSource<Route> taskCompletionSource = new
TaskCompletionSource<Route>();
        EventHandler<QueryCompletedEventArgs<Route>>
queryCompletedHandler = null;
        queryCompletedHandler = (s, e) =>
{
    routeQuery.QueryCompleted -= queryCompletedHandler;
    if (e.Error != null)
        taskCompletionSource.TrySetException(e.Error);
    else
        taskCompletionSource.TrySetResult(e.Result);
};

        routeQuery.QueryCompleted += queryCompletedHandler;
        routeQuery.QueryAsync();

        return taskCompletionSource.Task;
}
}
```

Et ensuite de faire :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private List<GeoCoordinate> coordonnesTrajet;

    public MainPage()
    {
        InitializeComponent();

        Carte.ZoomLevel = 13;
        Carte.Center = new GeoCoordinate { Latitude = 48.863134,
Longitude = 2.320518 };
        coordonnesTrajet = new List<GeoCoordinate>();

        CalculerItineraire();
    }

    private async void CalculerItineraire()
    {
        GeoCoordinate positionDepart = new GeoCoordinate(48.858115,
2.294710);
        coordonnesTrajet.Add(positionDepart);

        GeocodeQuery geocodeQuery = new GeocodeQuery
        {
            SearchTerm = "avenue des champs-élysées, Paris",
            GeoCoordinate = positionDepart
        };

        try
        {
            var resultat = await geocodeQuery.QueryLocationAsync();

            RouteQuery routeQuery = new RouteQuery();
            coordonnesTrajet.Add(resultat[0].GeoCoordinate);
            routeQuery.Waypoints = coordonnesTrajet;
            var route = await routeQuery.QueryRouteAsync();
            Carte.AddRoute(new MapRoute(route));
        }
        catch (Exception)
        {
```

```
        }
    }
```

Nous pouvons même avoir les instructions de déplacement. Rajoutons une ListBox dans le XAML :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <maps:Map x:Name="Carte" />
    <ListBox x:Name="ListeDirections" Grid.Row="1">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding}" />
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Grid>
```

Que nous pouvons lier à la liste d'instruction, obtenues ainsi :

Code : C#

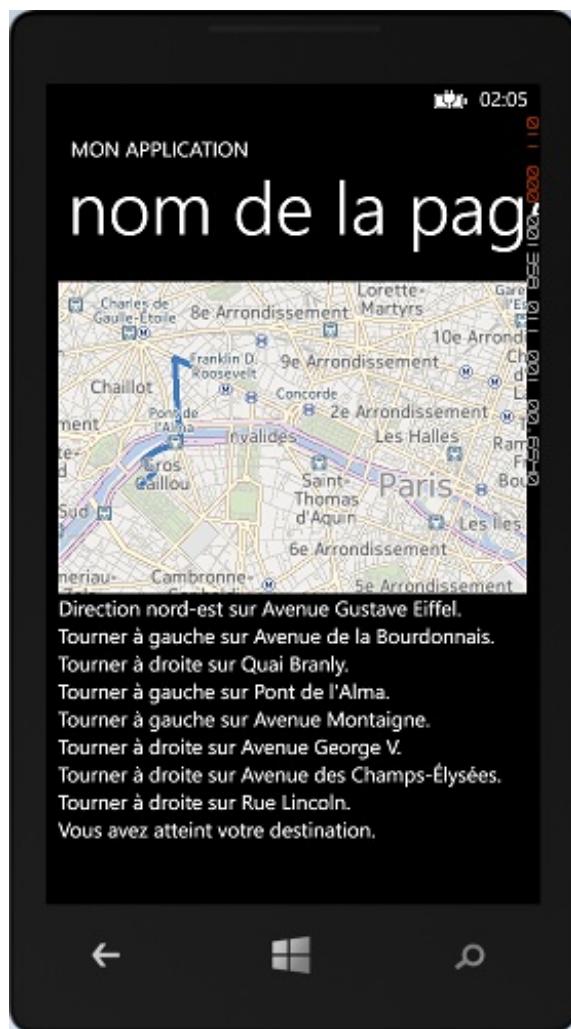
```
try
{
    var resultat = await geocodeQuery.QueryLocationAsync();

    RouteQuery routeQuery = new RouteQuery();
    coordonneesTrajet.Add(resultat[0].GeoCoordinate);
    routeQuery.Waypoints = coordonneesTrajet;
    var route = await routeQuery.QueryRouteAsync();
    Carte.AddRoute(new MapRoute(route));

    List<string> routeList = new List<string>();
    foreach (RouteLeg leg in route.Legs)
    {
        foreach (RouteManeuver maneuver in leg.Maneuvers)
        {
            routeList.Add(maneuver.InstructionText);
        }
    }

    ListeDirections.ItemsSource = routeList;
}
catch (Exception)
{}
```

Ce qui donne le résultat affiché à la figure suivante.



Les instructions de l'itinéraire

À noter que si vous souhaitez calculer un itinéraire piéton, vous pouvez changer le mode de calcul en modifiant l'objet `RouteQuery`:

Code : C#

```
routeQuery.TravelMode = TravelMode.Walking;
```

- Le contrôle Map est un contrôle très puissant qui nous permet d'exploiter les cartes du monde entier dans nos applications.
- Il est possible d'épingler des points d'intérêts grâce aux objets PushPin du toolkit Windows Phone.
- Le contrôle de carte fournit de quoi calculer un itinéraire, que l'on peut afficher sur la carte.

TP : Une application météo

Bienvenue dans ce nouveau TP. Nous allons mettre en pratique les derniers éléments que nous avons appris, mais aussi des éléments déjà vus. Eh oui ! Étant donné qu'ils vont faire partie intégrante de beaucoup de vos futures applications, vous devez les maîtriser.

Bref, le but de ce TP sera de réaliser une petite application météo tout à fait fonctionnelle, que vous pourrez exhiber devant vos amis : regardez, c'est moi qui l'ai fait !

Allez, passons sans plus attendre à l'énoncé du TP.

Instructions pour réaliser le TP

Il s'agit donc de réaliser une application météo. Cette application fournira les prévisions d'une ville grâce au service web de météo de **worldweatheronline**. Pourquoi celui-là ? Parce que je le trouve très facile à utiliser, vous le verrez par vous-même et que cela vous forcera à manipuler du JSON, ce qui ne fait jamais de mal. Il nécessite cependant une petite inscription préalable afin de disposer d'une clé d'API, mais rassurez-vous, tout est gratuit.



Notez que je n'ai pas d'actions chez eux, vous n'êtes bien sûr pas obligés de vous inscrire (ou utilisez un email poubelle). Au tout début de la rédaction de ce cours, j'utilisais le service météo de Google, mais ils ont décidé de l'arrêter, me forçant à en trouver un autre.

Allez sur <http://www.worldweatheronline.com/register.aspx> et remplissez le formulaire avec votre nom et votre email, ainsi que le captcha (voir la figure suivante).

Weather API Sign Up

Register to get an API Key to access our Free Global Weather API and Free Marine / Surfing Weather API for accurate and reliable weather forecast.

Having Problems?

- [Resend Verification Email](#)
- [Forgotten your API Key?](#)

Note: You only need to register once and use the same API Key to access both Global API and Marine API weather data.

Note: All the fields are required!

Your Name:

Email:

Confirm Email:

Are you human?

cucumbers eshell

Saisissez les deux mots :

 reCAPTCHA™ stop spam. read books.

I have read and agree to [T&C's](#) and [Usage Policy](#).

Generate API Key

Le formulaire de création de compte

Après l'inscription, vous recevez un mail pour vérifier votre compte, ainsi qu'une clé d'API une fois le mail vérifié. Avec cette clé d'API, vous pourrez ensuite construire votre requête. Par exemple pour obtenir la météo de Bordeaux à 5 jours, j'appellerai l'URL suivante :

Code : Autre

```
http://free.worldweatheronline.com/feed/weather.ashx?  
q=Bordeaux&format=json&num_of_days=5&key=MA_CLE_API
```

On peut donc préciser le nom de la ville dans le paramètre q, le type de format souhaité et le nombre de jours d'informations météos souhaités (maxi 5).

J'obtiens un JSON en retour, du genre :

Code : Autre

```
{
  "data" : { "current_condition" : [ ...abrégré... ],
    "request" : [ { "query" : "Bordeaux, France",
      "type" : "City"
    } ],
    "weather" : [ { "date" : "2012-11-14",
      "tempMaxC" : "17",
      "tempMinC" : "8",
      "weatherDesc" : [ { "value" : "Sunny" } ],
      "weatherIconUrl" : [ { "value" : "http://www.worldweatheronline.com/...épuré..." }
    },
    { "date" : "2012-11-15",
      [...abrégré...]
    },
    { "date" : "2012-11-16",
      [...abrégré...]
    },
    { "date" : "2012-11-17",
      [...abrégré...]
    },
    { "date" : "2012-11-18",
      [...abrégré...]
    }
  ]
}
```

Nous pouvons voir quelques informations intéressantes, comme la température mini, la température maxi, une image qui illustre le temps prévu, la description du temps, etc. Ah oui tiens, la description du temps est en anglais, il pourra être judicieux de se faire une petite matrice de traduction grâce à la liste que l'on trouve ici : [http://www.worldweatheronline.com/feed \[...\] tionCodes.xml](http://www.worldweatheronline.com/feed [...] tionCodes.xml).

Vous avez l'habitude maintenant du format JSON. Nous allons donc devoir exploiter ces informations.

L'application sera composée de 3 pages. La première page présentera les différentes conditions météos dans un Pivot qui nous permettra de consulter les conditions météo du jour et des jours suivants. Vous afficherez le nom de la ville, et dans le pivot toutes les informations que nous possédons, de la façon que vous le souhaitez.

Pendant le chargement des informations de météo, vous mettrez une barre de progression indéterminée afin que l'utilisateur ne soit pas perturbé et ne croie l'application inactive.

Cette page contiendra une barre d'application contenant deux icônes qui renverront vers une page permettant d'ajouter une ville et vers une autre page permettant de sélectionner une ville avec le ListPicker parmi la liste de toutes les villes précédemment ajoutées.

Bien sûr, l'application retiendra la liste de toutes les villes ajoutées ainsi que la dernière ville sélectionnée afin d'afficher directement les conditions météo de cette ville lors de la prochaine ouverture de l'application.

Vous vous sentez prêt ? Vous avez tout ce qu'il faut ? Alors, allez-y et créez une belle application météo.

Bon courage

Correction

Ah voilà une petite application qu'elle est sympathique. Et utile en plus ! C'est toujours pratique de pouvoir savoir s'il vaut mieux prendre son parapluie ou ses tongues.

Pour réaliser cette correction, nous allons commencer par créer la page qui permet d'ajouter une ville, je l'appelle Ajouter.xaml. Voici le XAML :

Code : XML

```
<phone:PhoneApplicationPage
  x:Class="TpApplicationMeteo1.Ajouter"
  ...
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
```

```

        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="Météo en direct" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Ajouter une ville" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle2Style}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="Nom de la ville" />
            <TextBox x:Name="NomVille" />
            <Button Content="Ajouter" Tap="Button_Tap" />
        </StackPanel>
    </Grid>
</Grid>
</phone:PhoneApplicationPage>

```

Elle n'est pas très compliquée, nous avons une zone de saisie permettant d'indiquer une ville et un bouton permettant d'ajouter la ville. Le code-behind est :

Code : C#

```

public partial class Ajouter : PhoneApplicationPage
{
    public Ajouter()
    {
        InitializeComponent();
    }

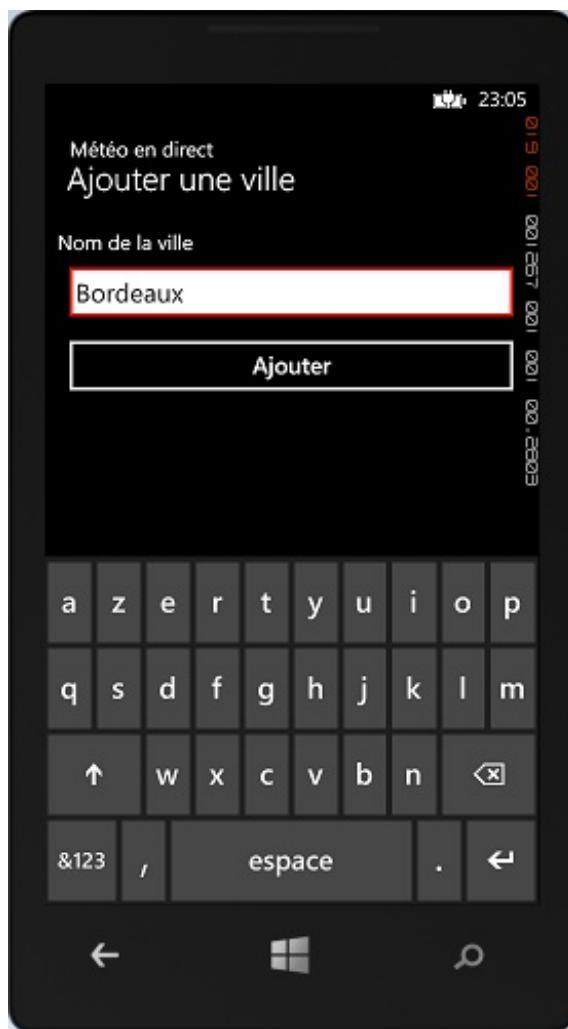
    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        if (string.IsNullOrEmpty(NomVille.Text))
        {
            MessageBox.Show("Veuillez saisir un nom de ville");
        }
        else
        {
            IsolatedStorageSettings.ApplicationSettings["DerniereVille"] =
NomVille.Text;
            List<string> nomVilles;
            if
(IsolatedStorageSettings.ApplicationSettings.Contains("ListeVilles"))
                nomVilles =
(List<string>)IsolatedStorageSettings.ApplicationSettings["ListeVilles"];
            else
                nomVilles = new List<string>();
            nomVilles.Add(NomVille.Text);
            IsolatedStorageSettings.ApplicationSettings["ListeVilles"] =
nomVilles;

            if (NavigationService.CanGoBack)
                NavigationService.GoBack();
        }
    }
}

```

Après un test pour vérifier qu'il y a bien un élément dans la zone de saisie, j'enregistre la ville dans le répertoire local, en tant que dernière ville consultée et dans la liste totale des villes déjà enregistrées. Bien sûr, si cette liste n'existe pas, je la crée.

Je m'autorise même une petite navigation arrière après l'enregistrement, fainéant comme je suis, pour éviter d'avoir à appuyer sur le bouton de retour arrière (voir la figure suivante).



L'écran d'ajout de ville

Bon, cette page est plutôt simple à faire. Passons à la page qui permet de choisir une ville déjà enregistrée, je l'appelle ChoisirVille.xaml. Le XAML sera :

Code : XML

```
<phone:PhoneApplicationPage  
    x:Class="TpApplicationMeteo1.ChoisirVille"  
    ...  
    xmlns:toolkit="clr-  
    namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit">  
  
    <Grid x:Name="LayoutRoot" Background="Transparent">  
        <Grid.RowDefinitions>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition Height="*"/>  
        </Grid.RowDefinitions>  
  
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">  
            <TextBlock x:Name="ApplicationTitle" Text="Météo en direct"  
                Style="{StaticResource PhoneTextNormalStyle}"/>  
            <TextBlock x:Name="PageTitle" Text="Choisir une ville" Margin="9,-  
                7,0,0" Style="{StaticResource PhoneTextTitle2Style}"/>  
        </StackPanel>  
  
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">  
            <toolkit:ListPicker x:Name="Liste" ItemsSource="{Binding  
                ListeVilles}"  
                Header="Ville choisie :">
```

```
        CacheMode="BitmapCache">
    </toolkit:ListPicker>
</Grid>
</Grid>
</phone:PhoneApplicationPage>
```

Nous notons l'utilisation du ListPicker et sa liaison à la propriété ListeVilles. Il a donc fallu importer l'espace de nom du toolkit ainsi que référencer l'assembly du toolkit. Le code-behind sera :

Code : C#

```
public partial class ChoisirVille : PhoneApplicationPage,
INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private List<string> listeVilles;
    public List<string> ListeVilles
    {
        get { return listeVilles; }
        set { NotifyPropertyChanged(ref listeVilles, value); }
    }

    public ChoisirVille()
    {
        InitializeComponent();
        DataContext = this;
    }

    protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        if
(!IsolatedStorageSettings.ApplicationSettings.Contains("ListeVilles"))
        {
            MessageBox.Show("Vous devez ajouter des villes");
            if (NavigationService.CanGoBack)
                NavigationService.GoBack();
        }
        else
        {
            ListeVilles =
(List<string>) IsolatedStorageSettings.ApplicationSettings["ListeVilles"];
            if (ListeVilles.Count == 0)
            {
                MessageBox.Show("Vous devez ajouter des villes");
                if (NavigationService.CanGoBack)
```

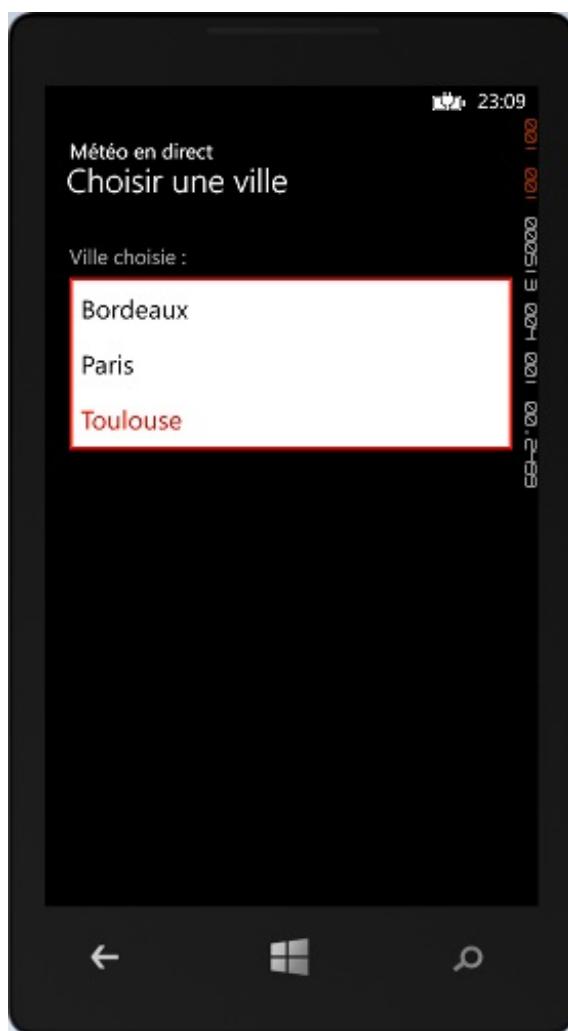
```
        NavigationService.GoBack();
    }

    if
(IsoaltedStorageSettings.ApplicationSettings.Contains("DerniereVille"))
{
    string ville =
(string)IsoaltedStorageSettings.ApplicationSettings["DerniereVille"];
    int index = ListeVilles.IndexOf(ville);
    if (index >= 0)
        Liste.SelectedIndex = index;
}
Liste.SelectionChanged += Liste_SelectionChanged;
}
base.OnNavigatedTo(e);
}

protected override void
OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
    Liste.SelectionChanged -= Liste_SelectionChanged;
    base.OnNavigatedFrom(e);
}

private void Liste_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (Liste.SelectedItem != null)
    {
        IsoaltedStorageSettings.ApplicationSettings["DerniereVille"]
= (string)Liste.SelectedItem;
    }
}
}
```

On commence par un petit test, s'il n'y a pas de ville à choisir alors, nous n'avons rien à faire ici. Ensuite nous associons la propriété `ListeVilles` au contenu du répertoire local et nous récupérons la dernière ville afin de présélectionner le `ListPicker` avec la ville déjà choisie. Attention, lorsque nous sélectionnons un élément ainsi, l'événement de changement de sélection est levé. Ce qui ne m'intéresse pas. C'est pour cela que je me suis abonné à cet événement après avoir modifié la propriété `SelectedIndex`. Pour la propriété du code, ceci implique que je me désabonne de ce même événement lorsque je quitte la page. Enfin, en cas de changement de sélection, j'enregistre la ville sélectionnée dans le répertoire local. Pas très compliqué non plus, à part peut-être la petite astuce pour éviter que l'événement de sélection ne soit levé. De toute façon, ce genre de chose se voit très rapidement lorsque nous testons notre application, comme vous pouvez le constater sur la figure suivante.



L'écran de choix d'une ville

Enfin, il reste la page affichant les conditions météo. Voici ma classe Meteo utilisée, ainsi que les classes générées pour le mapping des données JSON :

Code : C#

```
public class Meteo
{
    public string Date { get; set; }
    public string TemperatureMin { get; set; }
    public string TemperatureMax { get; set; }
    public Uri Url { get; set; }
    public string Temps { get; set; }
}

public class WeatherDesc
{
    public string value { get; set; }
}

public class WeatherIconUrl
{
    public string value { get; set; }
}

public class CurrentCondition
{
    public string cloudcover { get; set; }
    public string humidity { get; set; }
    public string observation_time { get; set; }
    public string precipMM { get; set; }
    public string pressure { get; set; }
}
```

```
public string temp_C { get; set; }
public string temp_F { get; set; }
public string visibility { get; set; }
public string weatherCode { get; set; }
public List<WeatherDesc> weatherDesc { get; set; }
public List<WeatherIconUrl> weatherIconUrl { get; set; }
public string winddir16Point { get; set; }
public string winddirDegree { get; set; }
public string windspeedKmph { get; set; }
public string windspeedMiles { get; set; }
}

public class Request
{
    public string query { get; set; }
    public string type { get; set; }
}

public class WeatherDesc2
{
    public string value { get; set; }
}

public class WeatherIconUrl2
{
    public string value { get; set; }
}

public class Weather
{
    public string date { get; set; }
    public string precipMM { get; set; }
    public string tempMaxC { get; set; }
    public string tempMaxF { get; set; }
    public string tempMinC { get; set; }
    public string tempMinF { get; set; }
    public string weatherCode { get; set; }
    public List<WeatherDesc2> weatherDesc { get; set; }
    public List<WeatherIconUrl2> weatherIconUrl { get; set; }
    public string winddir16Point { get; set; }
    public string winddirDegree { get; set; }
    public string winddirection { get; set; }
    public string windspeedKmph { get; set; }
    public string windspeedMiles { get; set; }
}

public class Data
{
    public List<CurrentCondition> current_condition { get; set; }
    public List<Request> request { get; set; }
    public List<Weather> weather { get; set; }
}

public class RootObject
{
    public Data data { get; set; }
}
```

Voyons à présent le XAML de la page, qui sera donc MainPage.xaml :

Code : XML

```
<phone:PhoneApplicationPage
    x:Class="TpApplicationMeteo1.MainPage"
    ...>
```

```
<phone:PhoneApplicationPage.Resources>
    <converter:VisibilityConverter x:Key="VisibilityConverter"
/>
</phone:PhoneApplicationPage.Resources>

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="Météo en direct" Style="{StaticResource PhoneTextNormalStyle}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ProgressBar IsIndeterminate="{Binding ChargementEnCours}" Visibility="{Binding ChargementEnCours, Converter={StaticResource VisibilityConverter}}"/>
        <TextBlock Text="{Binding NomVille}" Style="{StaticResource PhoneTextTitle2Style}"/>
        <phone:Pivot Grid.Row="1" ItemsSource="{Binding ListeMeteo}">
            <phone:Pivot.HeaderTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Date}"/>
                </DataTemplate>
            </phone:Pivot.HeaderTemplate>
            <phone:Pivot.ItemTemplate>
                <DataTemplate>
                    <StackPanel>
                        <TextBlock Text="{Binding TemperatureMin}"/>
                        <TextBlock Text="{Binding TemperatureMax}"/>
                        <TextBlock Text="{Binding Temps}"/>
                        <Image Source="{Binding Url}" Width="200" Height="200" Margin="0 50 0 0" HorizontalAlignment="Center"/>
                    </StackPanel>
                </DataTemplate>
            </phone:Pivot.ItemTemplate>
        </phone:Pivot>
        <TextBlock Text="Ajoutez une ville avec les boutons en bas" Visibility="Collapsed" x:Name="Information"/>
    </Grid>
</Grid>
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:AppBar IsVisible="True">
        <shell:AppBarIconButton IconUri="/Assets/Icones/add.png" Text="Ajouter" Click="Ajouter_Click"/>
        <shell:AppBarIconButton IconUri="/Assets/Icones/feature.settings.png" Text="Choisir" Click="Choisir_Click"/>
    </shell:AppBar>
</phone:PhoneApplicationPage.ApplicationBar>
</phone:PhoneApplicationPage>
```

Tout d'abord, regardons la barre d'application tout en bas. Elle possède deux boutons avec deux icônes. Il faudra bien sûr rajouter ces icônes dans notre application, en action de génération égale à contenu et en copie si plus récent. Les deux boutons permettent de naviguer vers nos deux pages, créées précédemment. Ensuite, nous avons une barre de progression, liée à la propriété `ChangementEnCours`, que ce soit sa propriété `IsIndeterminate` ou sa propriété `Visibility`. Nous avons également le Pivot, lié à la propriété `ListeMeteo`. L'entête du Pivot sera le nom du jour et les éléments du corps du pivot sont les diverses propriétés de l'objet `Meteo`.

Accompagnant le XAML, nous aurons le code-behind suivant :

Code : C#

```
public partial class MainPage : PhoneApplicationPage,
INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    private bool NotifyPropertyChanged<T>(ref T variable, T valeur,
[CallerMemberName] string nomPropriete = null)
    {
        if (object.Equals(variable, valeur)) return false;

        variable = valeur;
        NotifyPropertyChanged(nomPropriete);
        return true;
    }

    private List<Meteo> listeMeteo;
    public List<Meteo> ListeMeteo
    {
        get { return listeMeteo; }
        set { NotifyPropertyChanged(ref listeMeteo, value); }
    }

    private bool changementEnCours;
    public bool ChangementEnCours
    {
        get { return changementEnCours; }
        set { NotifyPropertyChanged(ref changementEnCours, value); }
    }

    private string nomVille;
    public string NomVille
    {
        get { return nomVille; }
        set { NotifyPropertyChanged(ref nomVille, value); }
    }

    public MainPage()
    {
        InitializeComponent();
        DataContext = this;
    }

    protected async override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
    {
        if
(IsolatedStorageSettings.ApplicationSettings.Contains("DerniereVille"))
        {
            Information.Visibility = Visibility.Collapsed;
            ChangementEnCours = true;
        }
    }
}
```

```
NomVille =
(string) IsolatedStorageSettings.ApplicationSettings["DerniereVille"];
WebClient client = new WebClient();
try
{
    ChargementEnCours = false;
    string resultatMeteo = await
client.DownloadStringTaskAsync(new
Uri(string.Format("http://free.worldweatheronline.com/feed/weather.ashx?
q={0}&format=json&num_of_days=5&key=MA_CLE_API", NomVille.Replace(' ', '+'))), UriKind.Absolute));

    RootObject resultat =
JsonConvert.DeserializeObject<RootObject>(resultatMeteo);
    List<Meteo> liste = new List<Meteo>();
    foreach (Weather temps in
resultat.data.weather.OrderBy(w => w.date))
    {
        Meteo meteo = new Meteo { TemperatureMax =
temps.tempMaxC + " °C", TemperatureMin = temps.tempMinC + " °C" };
        DateTime date;
        if (DateTime.TryParse(temps.date, out date))
        {
            meteo.Date = date.ToString("ddd dd MMMM");
            meteo.Temps = GetTemps(temps.weatherCode);
            WeatherIconUrl2 url =
temps.weatherIconUrl.FirstOrDefault();
            if (url != null)
            {
                meteo.Url = new Uri(url.value,
UriKind.Absolute);
            }
        }
        liste.Add(meteo);
    }
    ListeMeteo = liste;
}
catch (Exception)
{
    MessageBox.Show("Impossible de récupérer les
informations de météo, vérifiez votre connexion internet");
}
else
    Information.Visibility = Visibility.Visible;
    base.OnNavigatedTo(e);
}

private string GetTemps(string code)
{
// à compléter ...
switch (code)
{
    case "113":
        return "Clair / Ensoleillé";
    case "116":
        return "Partiellement nuageux";
    case "119":
        return "Nuageux";
    case "296":
        return "Faible pluie";
    case "353":
        return "Pluie";
    default:
        return "";
}
}

private void Ajouter_Click(object sender, EventArgs e)
```

```
{  
    NavigationService.Navigate(new Uri("/Ajouter.xaml",  
UriKind.Relative));  
}  
  
private void Choisir_Click(object sender, EventArgs e)  
{  
    NavigationService.Navigate(new Uri("/ChoisirVille.xaml",  
UriKind.Relative));  
}  
}
```

On commence par tester si la dernière ville consultée existe bien. Si ce n'est pas le cas, nous affichons un petit message pour dire quoi faire. Puis nous démarrons le téléchargement des conditions de météo, sans oublier d'animer la barre de progression. Après avoir attendu la fin du téléchargement (mot clé `await`), nous pouvons utiliser JSON.NET pour extraire les conditions météo et les mettre dans la propriété liée au Pivot. Nous remarquons au passage ma technique hautement élaborée pour obtenir une traduction de la description du temps... Quoi il manque des traductions ? N'hésitez pas à compléter avec les vôtres ... Enfin, les deux méthodes de clic sur les boutons de la barre d'application appellent simplement le service de navigation (voir la figure suivante).



Et si vous gériez l'orientation multiple maintenant ? 😊

Dans cette partie, nous avons étudiés plusieurs contrôles qui font partie intégrante du développement d'applications pour Windows Phone.

Nous avons dans un premier temps étudié le contrôle Panorama et le contrôle Pivot, qui sont des contrôles indispensables pour offrir une expérience utilisateur intéressante lorsque l'on a des données à présenter. Nous avons également vu comment afficher des pages web grâce au contrôle WebBrowser. Puis nous avons appris à gérer les différentes orientations et résolutions qu'un Windows Phone peut prendre.

Nous avons pu voir également comment fonctionnait la barre d'application qui permet d'avoir une espèce de menu accessible n'importe quand. Nous avons aperçu quelques contrôles issus de la bibliothèque gratuite Windows Phone. Enfin, nous avons également découvert la puissance du contrôle Map pour afficher des cartes, des itinéraires et les extensions du toolkit pour afficher des points d'intérêts.

Tous ces contrôles nous permettent d'enrichir nos applications. Vous avez pu le voir lors de la réalisation de notre application météo. Ils nous fournissent tous les éléments dont nous avons besoin pour réaliser de belles applications fonctionnelles. Usez-en, abusez-en, ils sont là pour ça 😊.

Partie 4 : Un téléphone ouvert vers l'extérieur

Il est temps maintenant de pousser le téléphone dans ses retranchements. Même si jusqu'à présent nous avons fait plein de choses, il ne faut pas oublier que ces smartphones sont des jouets technologiques, avec plein de gadgets. Un écran tactile, un accéléromètre, un GPS, ... Nous nous devons d'étudier ceci absolument. Nous allons également découvrir des solutions pour donner des informations à notre utilisateur, même s'il n'utilise pas notre application. Il y a encore beaucoup de choses que savent faire nos téléphones, et nous ne pourrons pas toutes les étudier.

Entamons alors cette dernière ligne droite et exploitons jusqu'au bout notre téléphone. Avec tout ce que vous allez apprendre, vous allez être capable de réaliser des applications dignes de votre imagination et vous serez prêt à les partager au monde entier.

Et ça, c'est quand même le but ultime 🎉

La gestuelle

Au contraire de la génération précédente, Windows Mobile, les Windows Phone sont utilisables exclusivement avec les doigts. Cela peut paraître évident, mais un doigt est beaucoup plus large que le pointeur d'une souris. Pour les développeurs qui sont habitués à créer des applications ou des sites web utilisables avec une souris, il faut prendre conscience que les zones qui sont touchables par un doigt doivent être taillées en conséquence.

De plus, les écrans des Windows Phone sont multipoints, c'est-à-dire que nous pouvons exercer plusieurs points de pressions simultanés, avec notamment plusieurs doigts. Ce qui offre tout une gamme de nouvelles façons d'appréhender l'interaction avec l'utilisateur.

Malheureusement, il est difficile de faire du multipoint avec une souris dans l'émulateur. Il est préférable dans ce cas d'utiliser directement un téléphone.

Le simple toucher

Nous l'avons vu à plusieurs reprises, c'est le mode d'interaction le plus pratique et le plus naturel pour l'utilisateur. Il utilise un doigt pour toucher l'écran. Geste très classique qui ressemble très fortement à un clic d'une souris. Le doigt est utilisé pour sélectionner un élément.

En général, les contrôles qui ont besoin d'être sélectionnés par une pression exposent un événement Tap. C'est le cas par exemple des boutons que nous avons vu :

Code : XML

```
<Button x:Name="MonBouton" Content="Cliquez-moi" Tap="MonBouton_Tap" />
```

Ils exposent également un événement Click :

Code : XML

```
<Button x:Name="MonBouton" Content="Cliquez-moi" Click="MonBouton_Click_1" />
```

Nous l'avons vu par exemple sur la barre d'application, mais il est présent un peu partout. Cet événement est hérité de Silverlight pour PC, il reste cependant utilisable mais il est déconseillé pour des raisons notamment de performance. Nous lui préférerons l'événement Tap, comme on l'a déjà vu.

Il existe d'autres événements, toujours hérités de Silverlight, comme l'événement MouseLeftButtonDown :

Code : XML

```
<Rectangle Width="200" Height="200" Fill="Aqua" MouseLeftButtonDown="Rectangle_MouseLeftButtonDown" />
```

Il correspond à l'événement qui est levé lorsque l'on touche un contrôle. L'événement MouseLeftButtonUp est quant à lui

levé quand le doigt est relevé.

Mais le toucher simple ne sert pas qu'à « cliquer », il permet également d'arrêter un défilement. Voyez par exemple lorsque vous faites défiler une ListBox bien remplie, si vous touchez la ListBox et que vous lancez le doigt vers le bas, la liste défile vers le bas en fonction de la vitesse à laquelle vous avez lancé le doigt (ce mouvement s'appelle le « Flick »). Si vous retouchez la ListBox, le défilement s'arrêtera.

Tous ces toucher sont gérés nativement par beaucoup de contrôles XAML. Il est alors très simple de réagir à un toucher.

Les différents touchers

D'autres touchers sont utilisables, notamment le double-toucher, que l'on peut rapprocher du double-clic bien connu des utilisateurs de Windows. Il correspond à l'événement DoubleTap. Généralement peu utilisé, il peut servir à effectuer un zoom, comme dans Internet Explorer.

Nous connaissons un autre toucher, que l'on utilise pour faire défiler une ListBox par exemple. Il s'appelle le « pan » et consiste à toucher l'écran et à maintenir le toucher tout en bougeant le doigt dans n'importe quelle direction. Cette gestuelle ressemble au *drag & drop* que l'on connaît sous Windows.

Dans le même genre, il existe le « flick » qui correspond à un toucher puis à un mouvement rapide dans une direction. C'est ce mouvement que l'on utilise dans le contrôle Pivot par exemple, et qui ressemble à un tourner de page. Ce flick est également utilisé dans la ListBox pour effectuer un fort défilement.

Notons encore un autre toucher, le *touch and hold* qui correspond à un « clic long ». On maintient le doigt appuyé pendant un certain temps. En général, cela fait apparaître un menu contextuel.

Enfin, nous avons le *pinch* et le *stretch* qui consistent, avec deux doigts à rapprocher ou écarter ses doigts. C'est ce mouvement qui est utilisé pour zoomer et dé-zoomer.

Il n'y a pas de support pour ces gestuelles évoluées dans les contrôles Windows Phone, aussi si nous souhaitons les utiliser nous allons devoir implémenter par nous-même ces gestuelles. Certaines sont plus ou moins faciles. Pour ceux par exemple qui ont déjà implémenté un *drag & drop* dans des applications Windows, il devient assez facile d'implémenter le Pan grâce aux événements MouseLeftButtonDown, MouseLeftButtonUp et MouseMove (qui correspondent à la souris qui bouge).

Gestuelle avancée

D'autres événements un peu plus complexes sont également disponibles sur nos contrôles, il s'agit des événements ManipulationStarted, ManipulationDelta, et ManipulationCompleted. Ce qui est intéressant dans ces événements c'est qu'ils fournissent un paramètre avec beaucoup d'informations sur le toucher.

L'événement ManipulationStarted est levé au démarrage de la manipulation fournissant la position d'un ou plusieurs points de pression. Au fur et à mesure de la gestuelle, c'est l'événement ManipulationDelta qui est levé fournissant des informations par exemple sur les translations opérées. Enfin, à la fin de la manipulation, c'est l'événement ManipulationCompleted qui est levé. Par exemple, on pourrait se servir de ces événements pour déterminer la gestuelle du « flick » car cet événement fournit la vitesse finale du mouvement ainsi que la translation totale. La translation nous renseigne sur la direction du mouvement et la vitesse nous permet de savoir si il s'agit réellement d'un flick et sa « puissance ».

Bref, ces événements peuvent fournir beaucoup d'informations sur la gestuelle en cours, mais c'est à nous de fournir du code pour interpréter ces mouvements, ce qui n'est pas toujours simple...

Je pourrais vous faire une petite démonstration, mais ... il y a mieux 😊

Le toolkit à la rescousse

Heureusement, d'autres personnes ont réalisés ces calculs pour nous. Ouf !

Même si cela pourrait être très intéressant de prendre en compte des considérations de moteur physique pour déterminer la puissance d'un flick, il s'avère que c'est une tâche fastidieuse. C'est là qu'intervient à nouveau le toolkit et ses contrôles de gestuelle.

Prenez justement le Flick, il suffit de faire dans le XAML :

Code : XML

```
<Rectangle Width="300" Height="300" Fill="Aqua">
    <toolkit:GestureService.GestureListener>
        <toolkit:GestureListener
            Flick="GestureListener_Flick" />
    </toolkit:GestureService.GestureListener>
</Rectangle>
```

On utilise la propriété attachée GestureListener et on s'abonne à l'événement Flick. Ainsi, dans le code-behind on pourra avoir :

Code : C#

```
private void GestureListener_Flick(object sender,
FlickGestureEventArgs e)
{
    switch (e.Direction)
    {
        case System.Windows.Controls.Orientation.Horizontal:
            MessageBox.Show("Flick horizontal, angle : " + e.Angle);
            break;
        case System.Windows.Controls.Orientation.Vertical:
            MessageBox.Show("Flick vertical, angle : " + e.Angle);
            break;
    }
}
```

Ce qui est quand même super simple pour interpréter un flick.



N'hésitez pas à tester ce code, la gestuelle est difficile à faire passer avec des copies d'écrans. 😊

D'autres gestuelles sont gérées avec les événements suivants :

| Événement | Description |
|--|---|
| Tap | Simple toucher |
| DoubleTap | Deux touchers rapprochés |
| Flick | Mouvement rapide dans une direction |
| DragStarted, DragDelta, DragCompleted | Utilisés pour le mouvement du Pan |
| Hold | Représente le toucher long |
| PinchStarted, PinchDelta, PinchCompleted | Pour le mouvement du zoom |
| GestureBegin, GestureCompleted | Classe de base pour toutes les gestuelles |

Chaque événement possède un paramètre qui fournit des informations complémentaires sur la gestuelle. Par exemple, le PinchDelta fournit des informations sur l'angle de rotation ou sur la distance parcourue lors du mouvement.

Pour finir, nous allons illustrer l'événement DragDelta pour déplacer notre rectangle grâce à une transformation :

Code : XML

```
<Rectangle Width="300" Height="300" Fill="Aqua">
    <Rectangle.RenderTransform>
        <CompositeTransform x:Name="Transformation"/>
    </Rectangle.RenderTransform>
    <toolkit:GestureService.GestureListener>
        <toolkit:GestureListener
            DragDelta="GestureListener_DragDelta" />
    </toolkit:GestureService.GestureListener>
</Rectangle>
```

Et dans le code behind :

Code : C#

```
private void GestureListener_DragDelta(object sender,  
DragDeltaGestureEventArgs e)  
{  
    Transformation.TranslateX += e.HorizontalChange;  
    Transformation.TranslateY += e.VerticalChange;  
}
```

Difficile d'illustrer ce mouvement avec une copie d'écran, mais n'hésitez pas à tester cet exemple. Vous verrez que le rectangle bouge en fonction de votre mouvement de type Pan.

Merci le Windows Phone Toolkit ! 😊

- Les Windows Phone étant utilisables exclusivement avec les doigts, il est important de bien gérer les gestuelles en utilisant les événements des contrôles.
- Le Windows Phone Toolkit nous aide fortement dans l'implémentation de ces gestuelles.

L'accéléromètre

Chaque Windows Phone est équipé d'un accéléromètre. Il s'agit d'un capteur qui permet de mesurer l'accélération linéaire suivant les trois axes dans l'espace. Ainsi, à tout moment, on peut connaître la direction et l'accélération de la pesanteur qui s'exerce sur le téléphone.

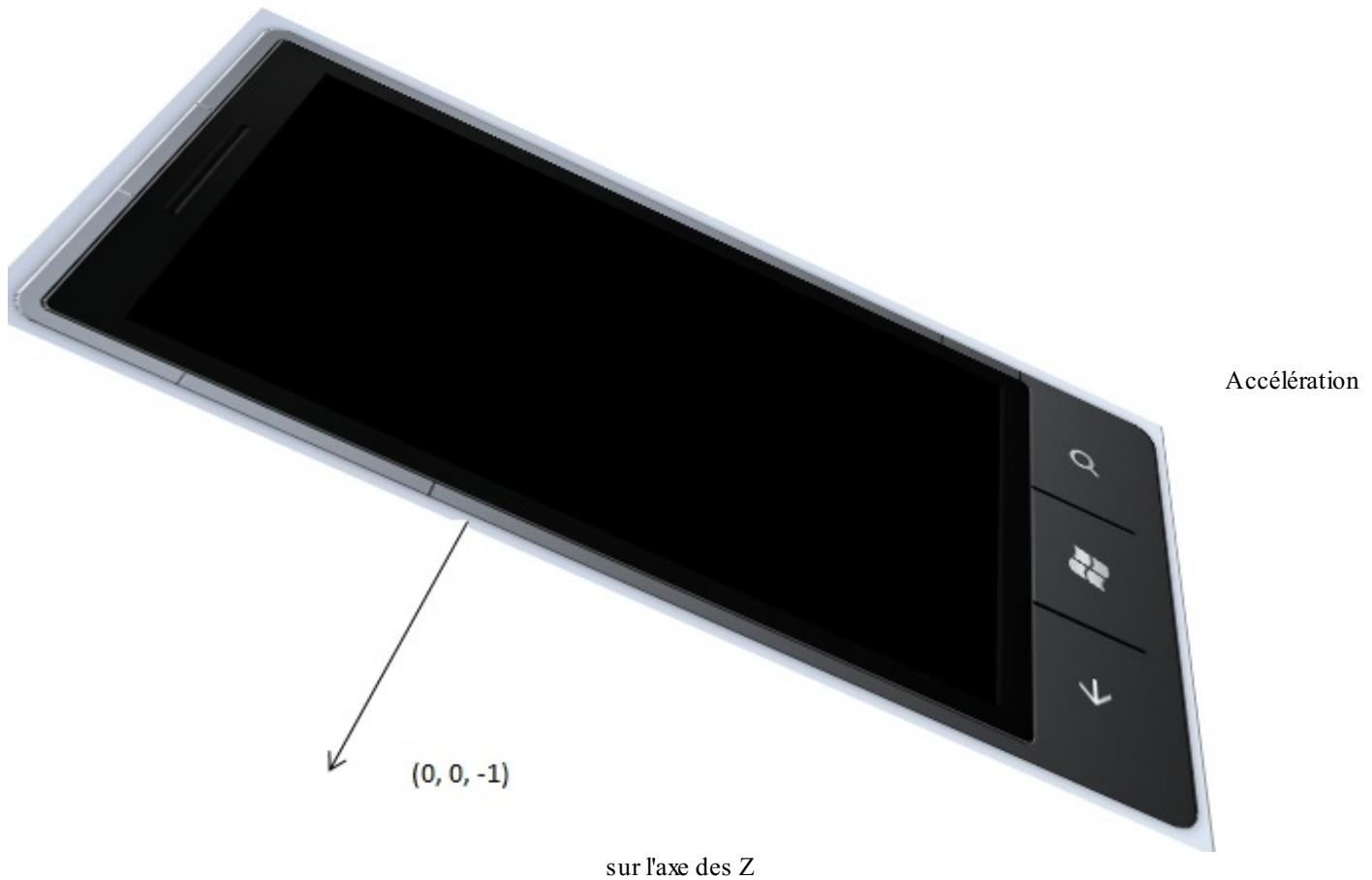
Ceci peut nous permettre de réaliser des petites applications sympathiques en nous servant de l'orientation du téléphone comme d'une interface avec l'utilisateur.

D'autres capteurs facultatifs existent sur un Windows Phone, comme le gyroscope ou le compas.

Voyons comment nous en servir.

Utiliser l'accéléromètre

Si le téléphone est posé sur la table et que la table est bien horizontale, nous allons pouvoir détecter une accélération d'1g sur l'axe des Z, qui correspond à la force de l'apesanteur. Si l'écran est tourné vers le haut, alors l'accélération sera de $(0, 0, -1)$ - voir la figure suivante.



Bien sûr, l'accéléromètre n'est pas figé dans cette position, la valeur oscille sans arrêt et vous aurez plus vraisemblablement une valeur comme $(0,02519062, -0,05639198, -0,994348)...$

Pour utiliser l'accéléromètre, vous allez devoir importer l'espace de nom :

Code : C#

```
using Microsoft.Devices.Sensors;
```

Étant donné que l'accéléromètre fournit un objet de type `Vector3`, qui fait partie de la bibliothèque XNA, nous aurons besoin d'ajouter une référence à l'assembly `Microsoft.Xna.Framework.dll` (uniquement dans les versions 7.X car pour le SDK 8, la référence est déjà ajoutée).

Il suffit ensuite de s'abonner à l'événement de changement de valeur et de démarrer l'accéléromètre :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private Accelerometer accelerometre;
    public MainPage()
    {
        InitializeComponent();
        accelerometre = new Accelerometer();
        accelerometre.CurrentValueChanged +=
accelerometre_CurrentValueChanged;
        accelerometre.Start();
    }

    private void accelerometre_CurrentValueChanged(object sender,
SensorReadingEventArgs<AccelerometerReading> e)
    {
        Dispatcher.BeginInvoke(() => Valeurs.Text =
e.SensorReading.Acceleration.X + ", " +
e.SensorReading.Acceleration.Y + ", " +
e.SensorReading.Acceleration.Z);
    }
}
```

On utilisera le Dispatcher pour pouvoir mettre à jour notre contrôle dans le thread dédié à l'interface :

Code : XML

```
<TextBlock x:Name="Valeurs" />
```

Cet événement nous fournit les 3 valeurs des différents axes. Par contre, à partir du moment où nous démarrons l'accéléromètre, nous recevrons un paquet de positions très régulièrement. Il est important de pouvoir faire du tri là-dedans. Ne vous inquiétez pas si vos valeurs oscillent dans une petite fourchette, c'est normal. Même si vous êtes le plus immobile possible. 😊 Suivant vos besoins, vous aurez peut-être besoin de lisser les informations obtenus, par exemple en faisant une moyenne sur les X dernières valeurs. Une autre solution est d'utiliser une formule un peu plus compliquée, issue du traitement du signal. Je ne rentrerai pas dans ces détails car nous n'en aurons pas besoin mais vous pouvez retrouver quelques informations en anglais à cette adresse.

Utiliser l'accéléromètre avec l'émulateur

Alors, l'accéléromètre, c'est très bien avec un téléphone, mais comment faire lorsque nous n'avons pas encore appris à déployer une application sur notre téléphone ?

Cela semble difficile d'orienter notre PC pour faire croire à l'émulateur que nous sommes en train de bouger... Heureusement, il existe une autre solution : les outils de l'émulateur. On peut les démarrer en cliquant sur le dernier bouton de sa barre à droite, comme indiqué sur la figure suivante.



Accéder aux outils de l'émulateur

Dans le premier onglet des outils, nous avons de quoi simuler l'accéléromètre (voir la figure suivante).

Outils supplémentaires

Accéléromètre Localisation Capture d'écran Réseau

X : 0 Y : 0,8485 Z : -0,5292

Orientation

Paysage vertical

Portrait vertical

Paysage vertical

Données enregistrées

shake

Lire

En déplaçant le rond orange, nous simulons une accélération du téléphone

Il suffit de sélectionner le petit point orange pour simuler une accélération du téléphone. Vous pouvez utiliser la liste déroulante en bas à gauche pour changer l'orientation du téléphone afin de faciliter l'utilisation de l'accéléromètre. De même, la liste en bas à droite permet de simuler un secouage de téléphone... 😊

Exploiter l'accéléromètre

Maintenant que nous savons titiller l'accéléromètre de l'émulateur, utilisons dès à présent les valeurs brutes de l'accéléromètre pour réaliser une petite application où nous allons faire bouger une balle en bougeant notre téléphone.

Nous avons dit que lorsqu'on tient le téléphone à plat, écran vers le haut, nous avons une accélération de (0,0,-1).

Lorsqu'on incline le téléphone vers la gauche, on tend vers l'accélération suivante (-1,0,0). Lorsqu'on incline vers la droite, on tend vers l'accélération (1,0,0).

De la même façon, lorsqu'on incline le téléphone vers l'avant, on tend vers (0,1,0) et lorsqu'on incline vers nous, on tend vers (0,-1,0).

On peut donc utiliser la force des composantes du vecteur pour faire bouger notre balle. Utilisons un Canvas et ajoutons un cercle dedans :

Code : XML

```
<phone:PhoneApplicationPage
    x:Class="DemoAccelerometre.MainPage"
    ...>

    <Canvas x:Name="LayoutRoot" Background="Transparent" Width="480"
    Height="800" >
        <Ellipse x:Name="Balle" Fill="Blue" Width="50" Height="50"
    />
    </Canvas>

</phone:PhoneApplicationPage>
```

Dans le code-behind, nous prendrons gare à démarrer et à arrêter l'accéléromètre, puis il suffira de récupérer les coordonnées de la balle et de les modifier en fonction des composantes du vecteur :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private Accelerometer accelerometre;

    public MainPage()
    {
        InitializeComponent();

        Balle.SetValue(Canvas.LeftProperty, LayoutRoot.Width / 2);
        Balle.SetValue(Canvas.TopProperty, LayoutRoot.Height / 2);

        accelerometre = new Accelerometer();
        accelerometre.CurrentValueChanged +=
accelerometre_CurrentValueChanged;
    }

    private void accelerometre_CurrentValueChanged(object sender,
SensorReadingEventArgs<AccelerometerReading> e)
    {
        Dispatcher.BeginInvoke(() =>
        {
            double x =
(double)Balle.GetValue(Canvas.LeftProperty) +
e.SensorReading.Acceleration.X;
            double y =
(double)Balle.GetValue(Canvas.TopProperty) -
e.SensorReading.Acceleration.Y;

            if (x <= 0)
```

```

        x = 0;
        if (y <= 0)
            y = 0;
        if (x >= LayoutRoot.Width - Balle.Width)
            x = LayoutRoot.Width - Balle.Width;
        if (y >= LayoutRoot.Height - Balle.Height)
            y = LayoutRoot.Height - Balle.Height;

        Balle.SetValue(Canvas.LeftProperty, x);
        Balle.SetValue(Canvas.TopProperty, y);
    });
}

protected override void
OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
    accelerometre.Stop();
    base.OnNavigatedFrom(e);
}

protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    accelerometre.Start();
    base.OnNavigatedTo(e);
}
}

```

Et voilà, notre cercle bouge en fonction de l'orientation du téléphone. Remarquez que pour avoir un mouvement naturel, nous avons inversé l'axe des Y en réalisant une soustraction.

Bon, c'est bien mais la balle n'avance pas très vite. Il pourrait être judicieux d'appliquer un facteur de vitesse, par exemple :

Code : C#

```

double vitesse = 3.5;
double x = (double)Balle.GetValue(Canvas.LeftProperty) +
e.SensorReading.Acceleration.X * vitesse;
double y = (double)Balle.GetValue(Canvas.TopProperty) -
e.SensorReading.Acceleration.Y * vitesse;

```

La balle va cette fois-ci un peu plus vite. Mais elle pourrait avoir un peu plus d'inertie, et notamment freiner lorsqu'on redresse le téléphone. N'oubliez pas que plus le téléphone est plat et plus Z tend vers -1.

On peut donc trouver une formule où le fait que le téléphone soit plat ralentisse la balle, en divisant par exemple par la valeur absolue de Z :

Code : C#

```

double facteur = e.SensorReading.Acceleration.Z == 0 ? 0.00001 :
Math.Abs(e.SensorReading.Acceleration.Z);
double vitesse = 3.5;
double x = (double)Balle.GetValue(Canvas.LeftProperty) +
e.SensorReading.Acceleration.X * vitesse / facteur;
double y = (double)Balle.GetValue(Canvas.TopProperty) -
e.SensorReading.Acceleration.Y * vitesse / facteur;

```

Bon, on est loin d'un vrai moteur physique simulant l'accélération, mais c'est déjà un peu mieux. 😊



Attention à faire en sorte que Z soit toujours différent de zéro, sinon c'est la méga honte. Je crois que diviser par zéro c'est vraiment la pire des exceptions que l'on peut avoir. 🍪

Attention, n'oubliez pas d'arrêter l'accéléromètre quand vous avez fini avec lui grâce à la méthode `Stop()` afin d'économiser la batterie. C'est ce que je fais dans la méthode `OnNavigatedFrom()`.

Les autres capteurs facultatifs

L'accéléromètre est le seul capteur obligatoire dans les spécifications d'un Windows Phone. Mais il y a d'autres capteurs facultatifs qui peuvent faire partie d'un Windows Phone. Il y a notamment le compas (ou le magnétomètre). Il permet de connaître l'emplacement du pôle nord magnétique et peut servir à faire une boussole par exemple. Étant facultatif, il faudra penser à vérifier s'il est présent avec :

Code : C#

```
if (!Compass.IsSupported)
{
    MessageBox.Show("Votre téléphone ne possède pas de compas");
}
```

De même, le gyroscope est facultatif sur les Windows Phone. Il sert à mesurer la vitesse de rotation suivant les trois axes. Il est différent de l'accéléromètre. Pour voir la différence entre les deux, imaginez-vous debout avec le téléphone en position portrait devant vous, face au nord. Si vous faites un quart de tour vers la droite, vous vous retrouvez face à l'est, le téléphone n'a pas changé de position dans vos mains, mais il a subi une rotation suivant un axe. Cette rotation, c'est le gyroscope qui va être en mesure de vous la fournir. Pour l'accéléromètre, pensez à notre petite application de balle réalisée plus haut. Pour tester sa présence, vous pourrez faire :

Code : C#

```
if (!Gyroscope.IsSupported)
{
    MessageBox.Show("Votre téléphone ne possède pas de gyroscope");
}
```

Il vous faudra peut-être adapter le comportement de votre application en conséquence, ou prévenir l'utilisateur qu'elle est inutilisable sans gyroscope.

La motion API

Travailler avec tous ces capteurs en même temps est plutôt complexe. De plus, si jamais il manque au téléphone un capteur que vous souhaitez utiliser, il vous faut revoir vos calculs pour adapter votre application.

C'est là qu'intervient la *motion API*, que l'on peut traduire en API de mouvement. Elle permet de gérer toute la logique mathématique associée à ces trois capteurs afin de fournir des valeurs facilement exploitables. Au final, on arrive très facilement à déterminer comment le téléphone est orienté dans l'espace. Cela permet par exemple de transposer le téléphone dans un monde 3D virtuel, permettant les applications de réalité augmentée, les jeux et pourquoi pas des applications auxquelles nous n'avons pas encore pensé...

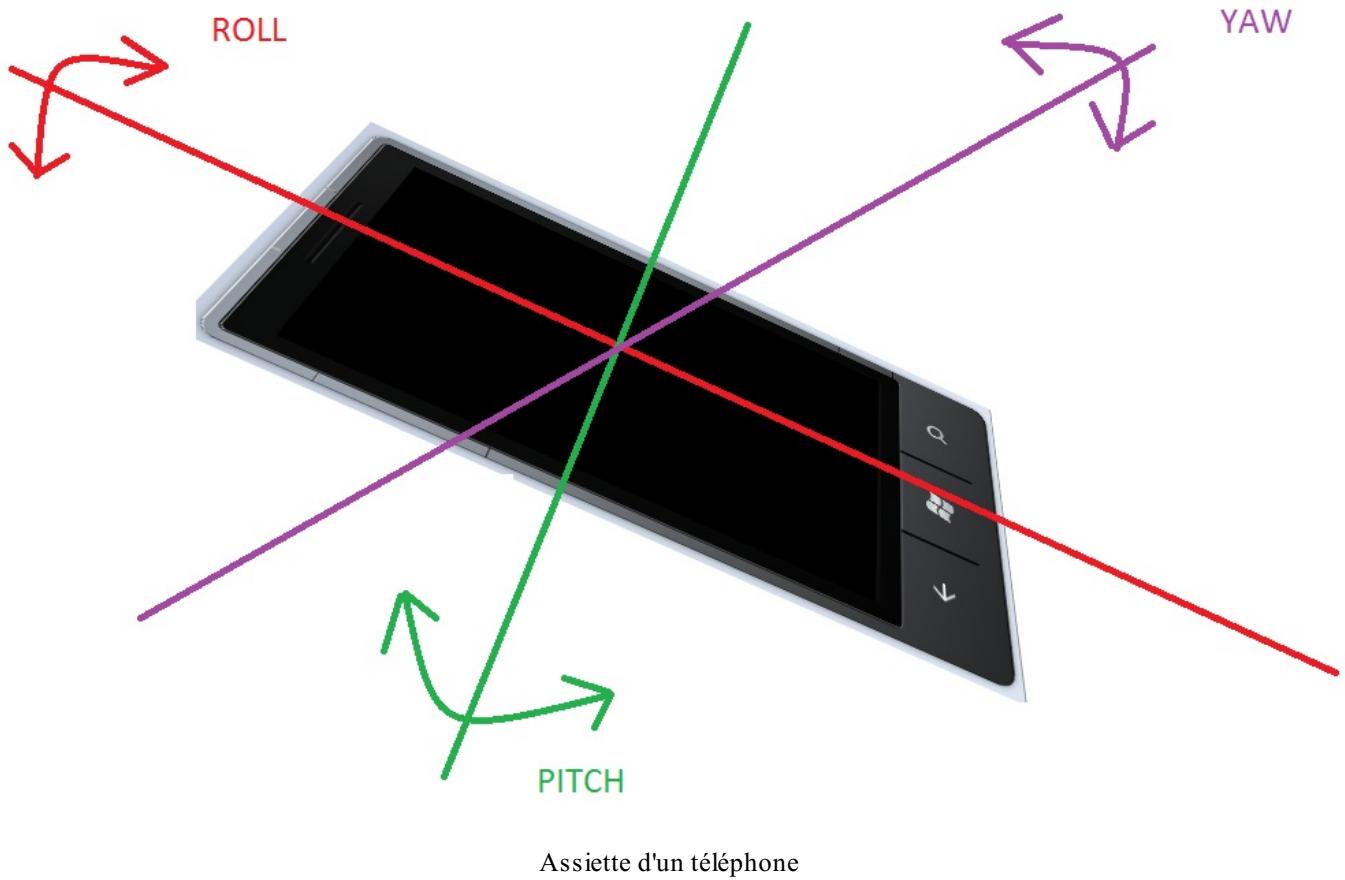
La motion API s'utilise grossièrement comme l'accéléromètre. Elle nous fournit plusieurs valeurs avec notamment une propriété `Attitude` que l'on peut traduire en « assiette », dans le langage de l'aviation, qui permet d'obtenir l'orientation de l'avion dans l'espace. Cette propriété nous fournit plusieurs valeurs intéressantes qui, toujours dans le domaine de l'aviation, sont :

- `Yaw` : qui indique la direction par rapport au nez de l'avion, afin de savoir si l'avion tourne à droite ou à gauche ;
- `Pitch` : qui indique si le nez de l'avion monte ou descend ;
- `Roll` : qui permet de savoir si l'avion bascule sur la droite ou la gauche.

Grosso modo, imaginons que vous ayez le téléphone posé sur la table devant vous, avec les boutons proches de vous :

- Si vous faites faire une rotation sur la table, tout en le conservant posé sur la table, alors vous changez le `Yaw`.
- Si vous levez le téléphone pour l'avoir en position verticale, alors vous changez le `pitch`.
- Si vous basculez le téléphone pour le mettre sur sa tranche, alors vous changez le `roll`.

On peut le représenter ainsi (voir la figure suivante).



Assiette d'un téléphone

Même si je ne doute pas de la qualité de mon dessin (XD), le mieux pour comprendre est de l'expérimenter sur un vrai téléphone.

Pour obtenir ces valeurs, il vous suffit de déclarer un objet Motion et de vous abonner à l'événement CurrentValueChanged. Notez que vous pouvez utiliser un petit Helper venant du framework XNA afin d'obtenir un angle à partir de cette valeur.

Utilisez donc le XAML suivant :

Code : XML

```
<TextBlock x:Name="Yaw" />
<TextBlock x:Name="Pitch" />
<TextBlock x:Name="Roll" />
```

Avec le code-behind :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    Motion motion;

    public MainPage()
    {
        InitializeComponent();
    }

    protected override void
```

```
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    if (!Motion.IsSupported)
    {
        MessageBox.Show("L'API Motion n'est pas supportée sur ce
téléphone.");
        return;
    }

    if (motion == null)
    {
        motion = new Motion { TimeBetweenUpdates =
TimeSpan.FromMilliseconds(20) };
        motion.CurrentValueChanged +=
motion_CurrentValueChanged;
    }

    try
    {
        motion.Start();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Impossible de démarrer l'API.");
    }
}

private void motion_CurrentValueChanged(object sender,
SensorReadingEventArgs<MotionReading> e)
{
    Dispatcher.BeginInvoke(() =>
    {
        if (motion.IsValid)
        {
            float yaw =
MathHelper.ToDegrees(e.SensorReading.Attitude.Yaw);
            float pitch =
MathHelper.ToDegrees(e.SensorReading.Attitude.Pitch);
            float roll =
MathHelper.ToDegrees(e.SensorReading.Attitude.Roll);

            Yaw.Text = "Yaw : " + yaw + " °";
            Pitch.Text = "Pitch : " + pitch + " °";
            Roll.Text = "Roll : " + roll + " °";
        }
    });
}
}
```

Pour utiliser le MathHelper, il faudra inclure l'espace de nom suivant :

Code : C#

```
using Microsoft.Xna.Framework;
```

L'objet Motion est quant à lui disponible avec :

Code : C#

```
using Microsoft.Devices.Sensors;
```



Vous ne pourrez malheureusement pas tester l'API motion dans l'émulateur, celle-ci n'est pas supportée. Et vous ne savez pas encore comment déployer une application sur votre téléphone, sauf si vous êtes allés vers la fin du cours en avance !

En plus de l'assiette, l'API motion fournit d'autres valeurs :

- `DeviceAcceleration`, qui est la même chose que ce que l'on obtient avec l'accéléromètre.
- `DeviceRotationRate`, qui est la même chose que ce que l'on obtient avec le gyroscope.
- `Gravity`, qui renvoie un vecteur en direction du centre de la terre, pour représenter la gravité, comme avec l'accéléromètre.

Vous avez compris que ces valeurs peuvent être obtenues avec les méthodes propres aux capteurs et que ce qui est vraiment intéressant, ce sont les valeurs calculées de l'assiette.

- L'accéléromètre est un capteur obligatoire sur tout téléphone Windows Phone qui nous offre de nouvelles façons d'interagir avec notre utilisateur.
- Il est facile à émuler grâce aux outils de l'émulateur Windows Phone.
- D'autres capteurs facultatifs existent, comme le gyroscope ou le compas.
- La motion API nous simplifie grandement la détection de l'orientation du téléphone dans l'espace et nous ouvre les portes de la réalité augmentée.

TP : Jeux de hasard (Grattage et secouage)

Ahhh, ça commence à devenir sympa ce qu'on peut faire ! L'écran tactile et la gestuelle associée, ainsi que l'accéléromètre sont des outils vraiment intéressant à utiliser. Et puis cela nous oblige à sortir de la classique souris et à imaginer des nouveaux types d'interactions avec l'utilisateur.

Nous allons donc mettre en pratique ces derniers éléments dans ce nouveau TP où nous allons créer une petite application de jeu de hasard, découpée en deux petits jeux qui vont utiliser la gestuelle et l'accéléromètre.

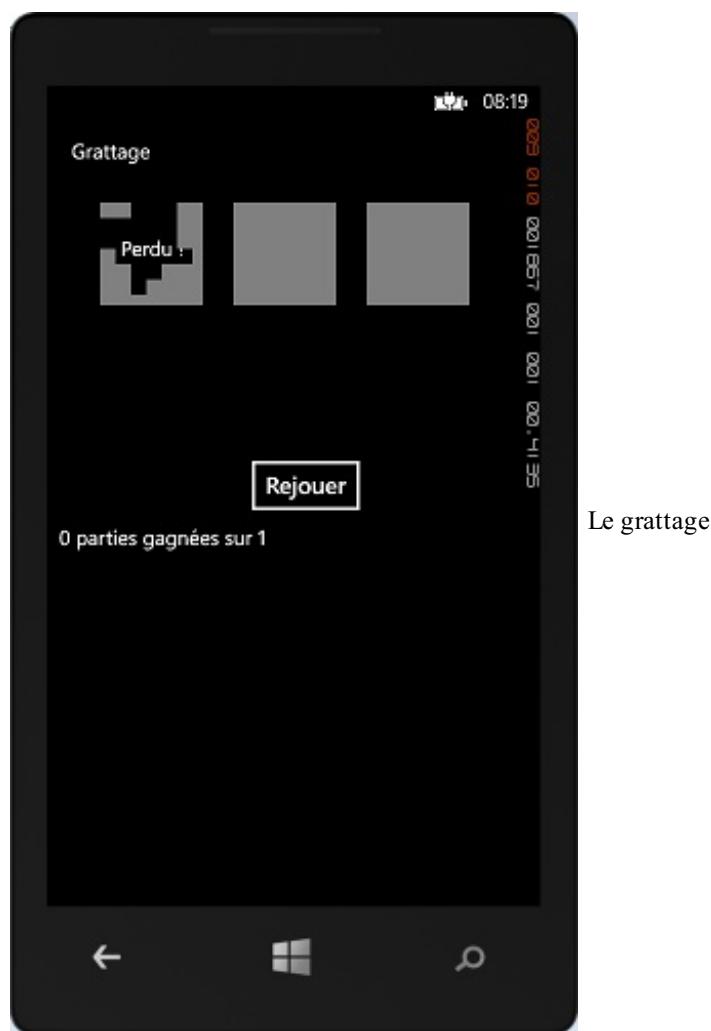
Instructions pour réaliser le TP

Créez dans un premier temps une page de menu qui renverra vers une page où nous aurons un jeu de grattage et une autre page où nous aurons un jeu de secouage...

Le jeu en lui-même ne sera pas super évolué et peu esthétique car je souhaite que vous vous concentriez sur les techniques étudiées précédemment, mais rien ne vous empêche de laisser courir votre imagination et de réaliser la prochaine killer-app. 😊

Donc, le grattage, je propose qu'il s'agisse d'afficher 3 rectangles que l'on peut gratter. Une fois ces rectangles grattés, ils découvrent si nous avons gagné ou pas. Un tirage aléatoire est fait pour déterminer le rectangle gagnant et une fois que nous avons commencé à gratter un rectangle, il n'est plus possible d'en gratter un autre. Pas besoin de gratter tout le rectangle, vous pourrez afficher la victoire ou la défaite de l'utilisateur une fois un certain pourcentage du rectangle gratté. N'oubliez pas d'offrir à l'utilisateur la possibilité de rejouer et de voir le nombre de parties gagnées.

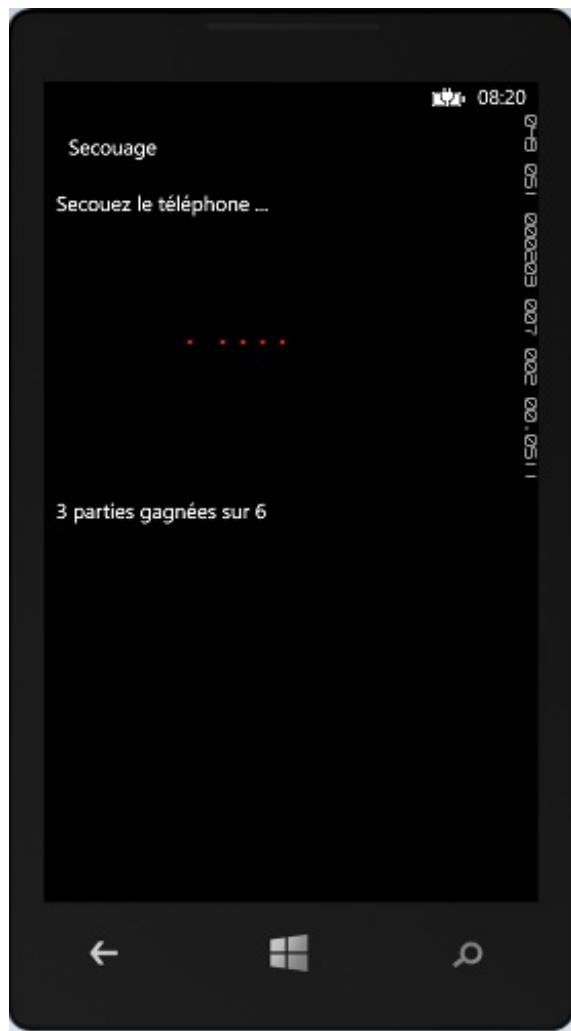
Voici à la figure suivante le résultat que je vous propose d'atteindre.



Le grattage

Passons maintenant au secouage. Le principe est de détecter via l'accéléromètre lorsque l'utilisateur secoue son téléphone. À ce moment-là, nous pourrons générer un nombre aléatoire avec une chance sur 3 de gagner. Pourquoi ne pas faire mariner un peu l'utilisateur en lui affichant une barre de progression indéterminée et en attendant deux secondes pour afficher le résultat. 😊

Voici à la figure suivante le résultat que je vous propose d'atteindre.



Le secouage

Alors, si vous vous le sentez, n'hésitez pas à vous lancer directement. Sinon, je vais vous proposer quelques pistes de réflexion pour démarrer sereinement le TP.

Tout d'abord, au niveau du grattage. Il y a plusieurs solutions envisageables. Celle que je vous propose est de ne pas avoir réellement un unique rectangle à gratter, mais plutôt plein de petits rectangles qui recouvrent un `TextBlock` contenant un texte affichant si c'est gagné ou perdu. Chaque `TextBlock` sera à l'écoute d'un événement de manipulation, j'ai choisi pour ma part la gestuelle du *drap & drop* du toolkit. Il faut ensuite arriver à déterminer quel élément est concerné lorsque nous touchons l'écran. Pour cela, j'utilise une méthode du framework .NET : [FindElementsInHostCoordinates](#). Par exemple, pour récupérer le `TextBlock` choisi lors du premier contact, je pourrais faire :

Code : C#

```
private void GestureListener_DragStarted(object sender,
    DragStartedGestureEventArgs e)
{
    Point position = e.GetPosition(Application.Current.RootVisual);
    IEnumerable<UIElement> elements =
    VisualTreeHelper.FindElementsInHostCoordinates(new Point(position.X,
    position.Y), Application.Current.RootVisual);
    TextBlock textBlockChoisi =
    elements.OfType<TextBlock>().FirstOrDefault();
}
```

La méthode `GetPosition` nous fournit la position du doigt par rapport à la page courante, que nous pouvons obtenir grâce à la propriété [RootVisual de l'application](#). Ainsi, il sera possible de déterminer les éléments qui sont sélectionnés et les supprimer de devant le `TextBlock`.

Passons maintenant au secouage. Comment détecter que l'utilisateur secoue son téléphone ? Il y a plusieurs solutions. Celle que

j'ai choisi de consister à détecter un écart significatif entre les deux dernières accélérations du téléphone. Si cet écart se reproduit plusieurs fois, alors je peux considérer qu'il s'agit d'un secouage. Par contre, si l'écart passe sous un certain seuil, alors je dois arrêter d'imaginer un potentiel secouage.

Allez, c'est à vous de jouer.

Correction

Alors, vous avez trouvé comment ? Facile ? Difficile ?

Ce n'est pas toujours facile de se confronter directement à ce genre de situations, surtout lorsqu'on a l'habitude de réaliser des applications clientes lourdes ou web, ou même lorsqu'on a pas du tout l'habitude de réaliser des applications. 😊

Voici la correction que je propose. Tout d'abord le menu, vous savez faire, il s'agit de ma page MainPage.xaml qui renvoie vers deux autres pages :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP Jeux de hasard" Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="Menu" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <Button Content="Grattage ..." Tap="Button_Tap" />
            <Button Content="Secouage ..." Tap="Button_Tap_1" />
        </StackPanel>
    </Grid>
</Grid>
```

C'est très épuré, le code-behind sera :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        NavigationService.Navigate(new Uri("/Grattage.xaml",
UriKind.Relative));
    }

    private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        NavigationService.Navigate(new Uri("/Secouage.xaml",
UriKind.Relative));
    }
}
```

Une utilisation très classique du service de navigation. Passons maintenant à la page Grattage.xaml :

Code : XML

```
<phone:PhoneApplicationPage
    ...
    xmlns:toolkit="clr-
    namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit">

    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Grattage"
Style="{StaticResource PhoneTextNormalStyle}"/>
        </StackPanel>
        <Canvas x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
            <TextBlock x:Name="TextBlock1" Width="100" Height="100"
Canvas.Top="10" Canvas.Left="40" Margin="20 30 0 0">
                <toolkit:GestureService.GestureListener>
                    <toolkit:GestureListener
DragStarted="GestureListener_DragStarted"
DragDelta="GestureListener_DragDelta"
DragCompleted="GestureListener_DragCompleted" />
                </toolkit:GestureService.GestureListener>
            </TextBlock>
            <TextBlock x:Name="TextBlock2" Width="100" Height="100"
Canvas.Top="10" Canvas.Left="170" Margin="20 30 0 0">
                <toolkit:GestureService.GestureListener>
                    <toolkit:GestureListener
DragStarted="GestureListener_DragStarted"
DragDelta="GestureListener_DragDelta"
DragCompleted="GestureListener_DragCompleted" />
                </toolkit:GestureService.GestureListener>
            </TextBlock>
            <TextBlock x:Name="TextBlock3" Width="100" Height="100"
Canvas.Top="10" Canvas.Left="300" Margin="20 30 0 0">
                <toolkit:GestureService.GestureListener>
                    <toolkit:GestureListener
DragStarted="GestureListener_DragStarted"
DragDelta="GestureListener_DragDelta"
DragCompleted="GestureListener_DragCompleted" />
                </toolkit:GestureService.GestureListener>
            </TextBlock>
            <StackPanel Canvas.Top="250" Width="480">
                <Button Content="Rejouer" HorizontalAlignment="Center"
Tap="Button_Tap" />
                <TextBlock x:Name="Resultat" />
            </StackPanel>
        </Canvas>
    </Grid>
</phone:PhoneApplicationPage>
```

Comme je l'ai proposé, j'ai simplement ajouté trois TextBlock dans un Canvas. Remarquez que j'ai spécifié exhaustivement les largeurs et les hauteurs de chacun. Sur chaque TextBlock, je suis également à l'écoute des trois événements de drag & drop.

Rien de bien compliqué.

C'est coté code-behind que cela se complique.

Code : C#

```
public partial class Grattage : PhoneApplicationPage
{
    private const int largeurRectangle = 15;
```

```
private int nbRects;
private TextBlock textBlockChoisi;
private Random random;
private bool aGagne;
private int nbParties;
private int nbPartiesGagnées;

public Grattage()
{
    InitializeComponent();
    random = new Random();
}

protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    Init();

    base.OnNavigatedTo(e);
}
}
```

Nous déclarons dans un premier temps plusieurs variables qui vont nous servir dans la classe. Il y a notamment un générateur de nombre aléatoires, initialisé dans le constructeur. Ensuite, cela se passe dans la méthode `Init()`. Il faut dans un premier temps créer plein de petits rectangles à afficher par-dessus chaque `TextBlock`:

Code : C#

```
private void Init()
{
    textBlockChoisi = null;
    TextBlock[] textBlocks = new[] { TextBlock1, TextBlock2,
TextBlock3 };

    int gagnant = random.Next(0, 3);
    for (int cpt = 0; cpt < textBlocks.Length; cpt++)
    {
        TextBlock tb = textBlocks[cpt];
        if (cpt == gagnant)
            tb.Text = "Gagné !";
        else
            tb.Text = "Perdu !";
    }

    foreach (TextBlock textBlock in textBlocks)
    {
        double x = (double)textBlock.GetValue(Canvas.LeftProperty);
        double y = (double)textBlock.GetValue(Canvas.TopProperty);

        nbRects = 0;
        for (double j = 0; j < textBlock.Height; j +=
largeurRectangle)
        {
            for (double i = 0; i < textBlock.Width; i +=
largeurRectangle)
            {
                double width = largeurRectangle;
                double height = largeurRectangle;
                if (i + width > textBlock.Width)
                    width = textBlock.Width - i;
                if (j + height > textBlock.Height)
                    height = textBlock.Height - j;
                Rectangle r = new Rectangle { Fill = new
SolidColorBrush(Colors.Gray), Width = width, Height = height, Tag =
textBlock };
                r.SetValue(Canvas.LeftProperty, i + x);
            }
        }
    }
}
```

```
        r.SetValue(Canvas.TopProperty, j + y);
        ContentPanel.Children.Add(r);
        nbRects++;
    }
}
}
```

Le principe est de récupérer la position de chaque `TextBlock` dans le `Canvas` puis de générer plein de petits rectangles qui couvrent la surface du `TextBlock`. Chaque rectangle est positionné dans le `Canvas` et ajouté à celui-ci. Notez que j'en profite pour rattacher chaque rectangle à son `TextBlock` grâce à la propriété `Tag`, cela sera plus simple par la suite pour déterminer quel rectangle on peut supprimer. Remarquons au passage que nous utilisons le générateur de nombre aléatoire pour déterminer quel `TextBlock` est le gagnant.

Viens ensuite le grattage en lui-même. C'est lors du démarrage de la gestuelle drag & drop que nous allons pouvoir déterminer le `TextBlock` choisi. Comme je l'avais indiqué, j'utilise la méthode `FindElementsInHostCoordinates` à partir des coordonnées du doigt. Puis, je peux utiliser le même principe lors de l'exécution de la gestuelle, récupérer le Rectangle à cette position et l'enlever du `Canvas`.

Code : C#

```
public partial class Grattage : PhoneApplicationPage
{
    [...]
    private void GestureListener_DragStarted(object sender,
DragStartedGestureEventArgs e)
    {
        if (textBlockChoisi == null)
        {
            aGagne = false;
            Point position =
e.GetPosition(Application.Current.RootVisual);
            IEnumerable<UIElement> elements =
VisualTreeHelper.FindElementsInHostCoordinates(new Point(position.X,
position.Y), Application.Current.RootVisual);
            textBlockChoisi =
elements.OfType<TextBlock>().FirstOrDefault();
        }
    }

    private void GestureListener_DragDelta(object sender,
DragDeltaGestureEventArgs e)
    {
        if (textBlockChoisi != null)
        {
            Point position =
e.GetPosition(Application.Current.RootVisual);

            IEnumerable<UIElement> elements =
VisualTreeHelper.FindElementsInHostCoordinates(new Point(position.X,
position.Y), Application.Current.RootVisual);
            foreach (Rectangle element in
elements.OfType<Rectangle>().Where(r => r.Tag == textBlockChoisi))
            {
                ContentPanel.Children.Remove(element);
            }
        }
    }

    private void GestureListener_DragCompleted(object sender,
DragCompletedGestureEventArgs e)
    {
        if (textBlockChoisi != null)
        {
            double taux = nbRects / 3.0;
            double nbRectangles =
ContentPanel.Children.OfType<Rectangle>().Count(r => r.Tag ==
```

```
textBlockChoisi);
        if (nbRectangles <= taux)
        {
            if (textBlockChoisi.Text == "Perdu !")
                MessageBox.Show("Vous avez perdu");
            else
            {
                aGagne = true;
                MessageBox.Show("Félicitations");
            }
        }
    }
}
```

Enfin, quand la gestuelle est terminée, je peux déterminer combien il reste de rectangles qui recouvrent le TextBlock choisi, et au-dessous d'un certain taux, je peux afficher si c'est gagné ou pas. Reste plus qu'à réinitialiser tout pour offrir la possibilité de rejouer :

Code : C#

```
public partial class Grattage : PhoneApplicationPage
{
    [...]
    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        if (aGagne)
            nbPartiesGagnees++;
        nbParties++;
        Resultat.Text = nbPartiesGagnees + " parties gagnées sur " +
nbParties;
        Init();
    }
}
```

Et voilà, le grattage est terminé. Passons maintenant à la partie secouage. Coté XAML c'est plutôt simple :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0"
Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="Secouage"
Style="{StaticResource PhoneTextNormalStyle}"/>
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Grid.RowDefinitions>
            <RowDefinition Height="100" />
            <RowDefinition Height="100" />
            <RowDefinition Height="100" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Text="Secouez le téléphone ..." />
        <ProgressBar x:Name="Barre" Grid.Row="1"
Visibility="Collapsed" />
        <TextBlock x:Name="Resultat" Grid.Row="2"

```

```
        HorizontalAlignment="Center" />
    <TextBlock x:Name="Total" Grid.Row="3" />
</Grid>
</Grid>
```

Tout se passe dans le code-behind, la première chose est d'initialiser l'accéléromètre et le générateur de nombres aléatoires. J'utilise également un DispatcherTimer pour faire mariner un peu l'utilisateur avant de lui fournir le résultat :

Code : C#

```
public partial class Secouage : PhoneApplicationPage
{
    private Accelerometer accelerometre;
    private const double ShakeThreshold = 0.7;
    private DispatcherTimer timer;
    private Random random;
    private int nbParties;
    private int nbPartiesGagnees;
    private Vector3 derniereAcceleration;
    private int cpt;

    public Secouage()
    {
        InitializeComponent();
        accelerometre = new Accelerometer();
        accelerometre.CurrentValueChanged +=
accelerometre_CurrentValueChanged;
        timer = new DispatcherTimer { Interval =
TimeSpan.FromSeconds(2) };
        random = new Random();
    }

    protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    accelerometre.Start();
    timer.Tick += timer_Tick;
    base.OnNavigatedTo(e);
}

    protected override void
OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
    accelerometre.Stop();
    timer.Tick -= timer_Tick;
    base.OnNavigatedFrom(e);
}
}
```

 Attention, n'oubliez pas d'arrêter l'accéléromètre lorsque nous n'en avons plus besoin afin d'économiser la batterie.

Il ne reste plus qu'à gérer la détection du secouage. Comme je vous l'ai indiqué, il suffit de comparer la différence d'accélération suivant un axe entre deux mesures. Si celle-ci dépasse un certain seuil, alors il y a un premier secouage, si par contre elle repasse sous un autre seuil, la détection est interrompue. J'ai fixé arbitrairement ces valeurs à 0.8 et 0.1 car je trouve qu'elles simulent des valeurs acceptables. Pour mesurer la différence d'accélération, je prends la valeur absolue de la différence entre la dernière valeur d'accélération sur un axe et la précédente. Si un changement brutal est détecté, alors on incrémenté le compteur. S'il dépasse 4 alors nous considérons qu'il s'agit d'un vrai secouage de téléphone.

Code : C#

```
public partial class Secouage : PhoneApplicationPage
{
    [...]
    private void accelerometre_CurrentValueChanged(object sender,
SensorReadingEventArgs<AccelerometerReading> e)
    {
        Dispatcher.BeginInvoke(() =>
        {
            Vector3 accelerationEnCours =
e.SensorReading.Acceleration;

            if (derniereAcceleration == null)
            {
                derniereAcceleration = accelerationEnCours;
                return;
            }
            double seuilMax = 0.8;
            double seuilMin = 0.1;
            int maxSecouage = 3;
            if (cpt <= maxSecouage && (
                Math.Abs(accelerationEnCours.X -
derniereAcceleration.X) >= seuilMax ||
                Math.Abs(accelerationEnCours.Y -
derniereAcceleration.Y) >= seuilMax ||
                Math.Abs(accelerationEnCours.Z -
derniereAcceleration.Z) >= seuilMax))
            {
                Resultat.Text = string.Empty;
                cpt++;
                if (cpt > maxSecouage)
                {
                    cpt = 0;
                    Barre.Visibility = Visibility.Visible;
                    Barre.Indeterminate = true;
                    timer.Start();
                }
            }
            else
            {
                if (Math.Abs(accelerationEnCours.X -
derniereAcceleration.X) >= seuilMin ||
                    Math.Abs(accelerationEnCours.Y -
derniereAcceleration.Y) >= seuilMin ||
                    Math.Abs(accelerationEnCours.Z -
derniereAcceleration.Z) >= seuilMin)
                {
                    cpt = 0;
                }
            }
            derniereAcceleration = accelerationEnCours;
        });
    }
}
```

À ce moment-là, je démarre le timer ainsi que l'animation de la barre de progression. Il ne reste plus qu'à gérer la suite du jeu :

Code : C#

```
public partial class Secouage : PhoneApplicationPage
{
    [...]
    private void timer_Tick(object sender, EventArgs e)
    {
        timer.Stop();
        Barre.Visibility = Visibility.Collapsed;
        Barre.Indeterminate = false;
```

```
int nombre = random.Next(0, 3);
if (nombre == 1)
{
    // nombre gagnant
    Resultat.Text = "Gagné !";
    nbPartiesGagnees++;
}
else
{
    Resultat.Text = "Perdu !";
}
nbParties++;
Total.Text = nbPartiesGagnees + " parties gagnées sur " +
nbParties;
}
```

Et voilà. Un petit jeu qui nous a permis de plonger doucement dans la gestuelle et dans l'utilisation de l'accéléromètre !

Aller plus loin

Ici, je suis resté très sobre sur l'interface (qui a dit très moche ?). Mais il serait tout à fait pertinent d'améliorer cet aspect-là de l'application. Pourquoi ne pas utiliser des images de dés et des transformations pour réaliser un lancer de dés majestueux ?

De même, plutôt que d'utiliser des rectangles lors du grattage, on pourrait utiliser des images ainsi que la classe [WriteableBitmap](#) pour effacer des éléments de l'image...

Je n'en parle pas ici car cela sort de la portée de ce cours, mais c'est une idée à creuser.

En ce qui concerne le secouage, j'ai proposé un algorithme ultra simpliste permettant de détecter ce genre de mouvement. Il existe une bibliothèque développée par des personnes à Microsoft qui permet de détecter les secouages. Il s'agit de la Shake Gesture Library, que [vous pouvez télécharger ici](#). Nous aurons tout à fait intérêt à l'utiliser ici. L'algorithme de détection est plus pertinent et pourquoi réinventer la roue alors que d'autres personnes l'ont déjà fait pour nous.

La géolocalisation

De plus en plus de matériels technologiques sont équipés de systèmes de localisation, notamment grâce aux GPS. C'est également le cas des téléphones équipés de Windows Phone dont les spécifications imposent la présence d'un GPS. Mais le système de localisation est un peu plus poussé que le simple GPS, qui a ses limites. En effet, on se trouve souvent à l'intérieur d'un immeuble ou entourés par des gros murs qui peuvent bloquer le signal du GPS... heureusement, la triangulation à base des réseaux WIFI peut prendre la relève pour indiquer la position d'un téléphone.

Nous allons pouvoir exploiter la position de notre utilisateur et réaliser ainsi des applications géolocalisées, nous allons voir dans ce chapitre comment faire.

Avec Windows Phone 7, on utilisait une première version du service de localisation grâce au [GeoCoordinateWatcher](#). Windows 8 a proposé une refonte du service de localisation et c'est tout naturellement que Windows Phone 8 en a profité. Les deux fonctionnent globalement de la même façon, mais celui de Windows Phone 8 supporte par défaut l'asynchronisme avancé avec `await` et `async`. C'est celui-ci que nous allons étudier. Sachez quand même que si vous projetez de faire en sorte que votre application fonctionne avec Windows Phone 7.X et 8, vous aurez intérêt à utiliser la première version du service de localisation.

Remarquez que vous devrez indiquer dans votre application un descriptif de votre politique d'utilisation des données géolocalisées, soit directement dans une page, soit en donnant un lien vers cette politique. De même, votre application devra fournir un moyen de désactiver explicitement l'utilisation du service de localisation.

Déterminer sa position

Il est très facile de récupérer sa position, on parle de géolocalisation. Pour ce faire, on pourra utiliser [la classe Geolocator](#) qui est le point d'accès au service de localisation. Nous aurons besoin dans un premier temps d'indiquer que notre application utilise le service de localisation en rajoutant la capacité `ID_CAP_LOCATION`. Il suffit de la sélectionner en double-cliquant sur le fichier `WMAppManifest.xml`, comme indiqué sur la figure suivante.

WMAppManifest.xml* ➔ X MainPage.xaml

Utilisez ce concepteur pour définir ou modifier certaines propriétés dans le fichier manifeste de l'application Windows Phone.

Interface utilisateur de l'application Capacités Spécifications Packages

Utilisez cette page pour spécifier les fonctionnalités utilisées par votre application.

| Capacités | Description | |
|---|---|------------|
| <input type="checkbox"/> ID_CAP_APPOINTMENTS | Fournit l'accès aux services d'emplacement. | Activer la |
| <input type="checkbox"/> ID_CAP_CONTACTS | | |
| <input type="checkbox"/> ID_CAP_GAMERSERVICES | | |
| <input type="checkbox"/> ID_CAP_IDENTITY_DEVICE | | |
| <input type="checkbox"/> ID_CAP_IDENTITY_USER | | |
| <input type="checkbox"/> ID_CAP_ISV_CAMERA | | |
| <input checked="" type="checkbox"/> ID_CAP_LOCATION | | |
| <input type="checkbox"/> ID_CAP_MAP | | |
| <input checked="" type="checkbox"/> ID_CAP_MEDIALIB_AUDIO | | |

capacité de géolocalisation

Ensuite, il faut avoir une instance de la classe `Geolocator` et potentiellement indiquer des paramètres comme la précision. Utilisons dans un premier temps ce XAML :

Code : XML

```
<StackPanel>
    <Button Click="Button_Click_1" Content="Obtenir la position du"
```

```
téléphone"/>
    <TextBlock x:Name="Latitude"/>
    <TextBlock x:Name="Longitude"/>
</StackPanel>
```

Et dans le code-behind, nous pouvons obtenir notre position avec :

Code : C#

```
private async void Button_Click_1(object sender, RoutedEventArgs e)
{
    Geolocator geolocator = new Geolocator();
    geolocator.DesiredAccuracyInMeters = 50;

    try
    {
        Geoposition geoposition = await
geolocator.GetGeopositionAsync(TimeSpan.FromMinutes(5),
TimeSpan.FromSeconds(10));

        Dispatcher.BeginInvoke(() =>
        {
            Latitude.Text =
geoposition.Coordinate.Latitude.ToString();
            Longitude.Text =
geoposition.Coordinate.Longitude.ToString();
        });
    }
    catch (UnauthorizedAccessException)
    {
        MessageBox.Show("Le service de location est désactivé dans
les paramètres du téléphone");
    }
    catch (Exception ex)
    {
    }
}
}
```

Grâce à la méthode `GetGeopositionAsync` nous obtenons la position du téléphone, et ce, de manière asynchrone (notez l'emploi des mots-clés `await` et `async`), ce qui évite de bloquer l'interface utilisateur. Nous pouvons ensuite exploiter l'objet `Geoposition` obtenu en retour et notamment ses propriétés `Latitude` et `Longitude`.

Dans cet objet, nous obtenons également d'autres informations, comme l'altitude (en mètres au-dessus du niveau de la mer), l'orientation (en degrés par rapport au nord), la vitesse (en mètres par secondes) ; cette dernière donnée ayant peu de sens lorsque la géolocalisation est demandée une unique fois.

Sauf qu'une position, ça change au fur et à mesure ! Eh oui, en voiture, dans le train, etc. On peut avoir besoin d'être notifié d'un changement de position ; ce qui peut être très pratique dans une application de navigation ou pour enregistrer un parcours de course à pied. Dans ce cas, il faut s'abonner à l'événement `PositionChanged` qui nous fournira une nouvelle position à chaque fois. Nous aurons également besoin de nous abonner à l'événement `StatusChanged` qui nous permettra de connaître les changements de statut du matériel :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private Geolocator geolocator;

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

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    geolocator = new Geolocator { DesiredAccuracy =
PositionAccuracy.High, MovementThreshold = 20 };
    geolocator.StatusChanged += geolocator_StatusChanged;
    geolocator.PositionChanged += geolocator_PositionChanged;

    base.OnNavigatedTo(e);
}

protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    geolocator.PositionChanged -= geolocator_PositionChanged;
    geolocator.StatusChanged -= geolocator_StatusChanged;
    geolocator = null;

    base.OnNavigatedFrom(e);
}

private void geolocator_StatusChanged(Geolocator sender,
StatusChangedEventArgs args)
{
    string status = "";

    switch (args.Status)
    {
        case PositionStatus.Disabled:
            status = "Le service de localisation est désactivé
dans les paramètres";
            break;
        case PositionStatus.Initializing:
            status = "En cours d'initialisation";
            break;
        case PositionStatus.Ready:
            status = "Service de localisation prêt";
            break;
        case PositionStatus.NotAvailable:
        case PositionStatus.NotInitialized:
        case PositionStatus.NoData:
            break;
    }

    Dispatcher.BeginInvoke(() =>
{
    Statut.Text = status;
});
}

private void geolocator_PositionChanged(Geolocator sender,
PositionChangedEventArgs args)
{
    Dispatcher.BeginInvoke(() =>
{
    Latitude.Text =
args.Position.Coordinate.Latitude.ToString();
    Longitude.Text =
args.Position.Coordinate.Longitude.ToString();
});
}
```

Il faut toujours vérifier la bonne disponibilité du GPS, car il peut arriver que l'utilisateur veuille explicitement le désactiver. C'est le cas par exemple lorsque le statut est égal à Disabled.
Pour afficher le statut, j'ai légèrement modifié mon XAML :

Code : XML

```
<TextBlock x:Name="Statut"/>
<TextBlock x:Name="Latitude"/>
<TextBlock x:Name="Longitude"/>
```

Remarque : il n'y a pas de méthode pour démarrer le service de localisation comme il pouvait y avoir la méthode `Start()` avec le service de localisation de Windows Phone 7. Le démarrage s'effectue lorsqu'on s'abonne à l'événement `PositionChanged`. Ce qui implique que lorsqu'on n'en aura plus besoin, il sera possible d'arrêter l'utilisation du service de localisation en se désabonnant de ce même événement. Cela permet d'économiser la batterie. C'est ce que j'ai fait dans mon exemple en démarrant le service de navigation lorsque j'arrive sur la page et en le désactivant lorsque je quitte la page.

Et voilà, dans la méthode `geolocator_PositionChanged`, nous recevons un objet nous fournissant les coordonnées de la position du téléphone, comme vu plus haut, et nous avons des relevés successifs de la position du téléphone.

Remarquons qu'il est possible de définir l'effort que doit faire le service de localisation pour déterminer notre position. Il suffit de créer le `Geolocator` en affectant sa propriété `DesiredAccuracy`. Il existe deux modes : précision forte ou précision par défaut. Vous aurez compris que la précision forte s'obtient avec :

Code : C#

```
DesiredAccuracy = PositionAccuracy.High
```

Et que la précision par défaut s'obtient avec

Code : C#

```
DesiredAccuracy = PositionAccuracy.Default
```

En mode par défaut, le téléphone utilise essentiellement les bornes wifi et les antennes radios pour faire une triangulation, ce qui est relativement peu consommateur de batterie. En mode de précision forte, le téléphone va utiliser l'ensemble des moyens dont il dispose pour nous géolocaliser en négligeant la consommation de batterie. Il utilisera notamment la puce GPS pour un repérage par satellite.

À noter que l'on peut indiquer à partir de quelle distance (en mètres par rapport à la dernière position trouvée) le service de localisation envoie un événement pour nous indiquer un changement de position. Cela se fait via la propriété `MovementThreshold` qui est à 0 mètre par défaut, ce qui implique qu'un événement de changement de position est envoyé à chaque changement de position. Nous avons également à notre disposition la propriété `DesiredAccuracyInMeters` qui permet d'indiquer le niveau de précision souhaité (en mètres) pour les données renvoyées depuis le service de localisation.

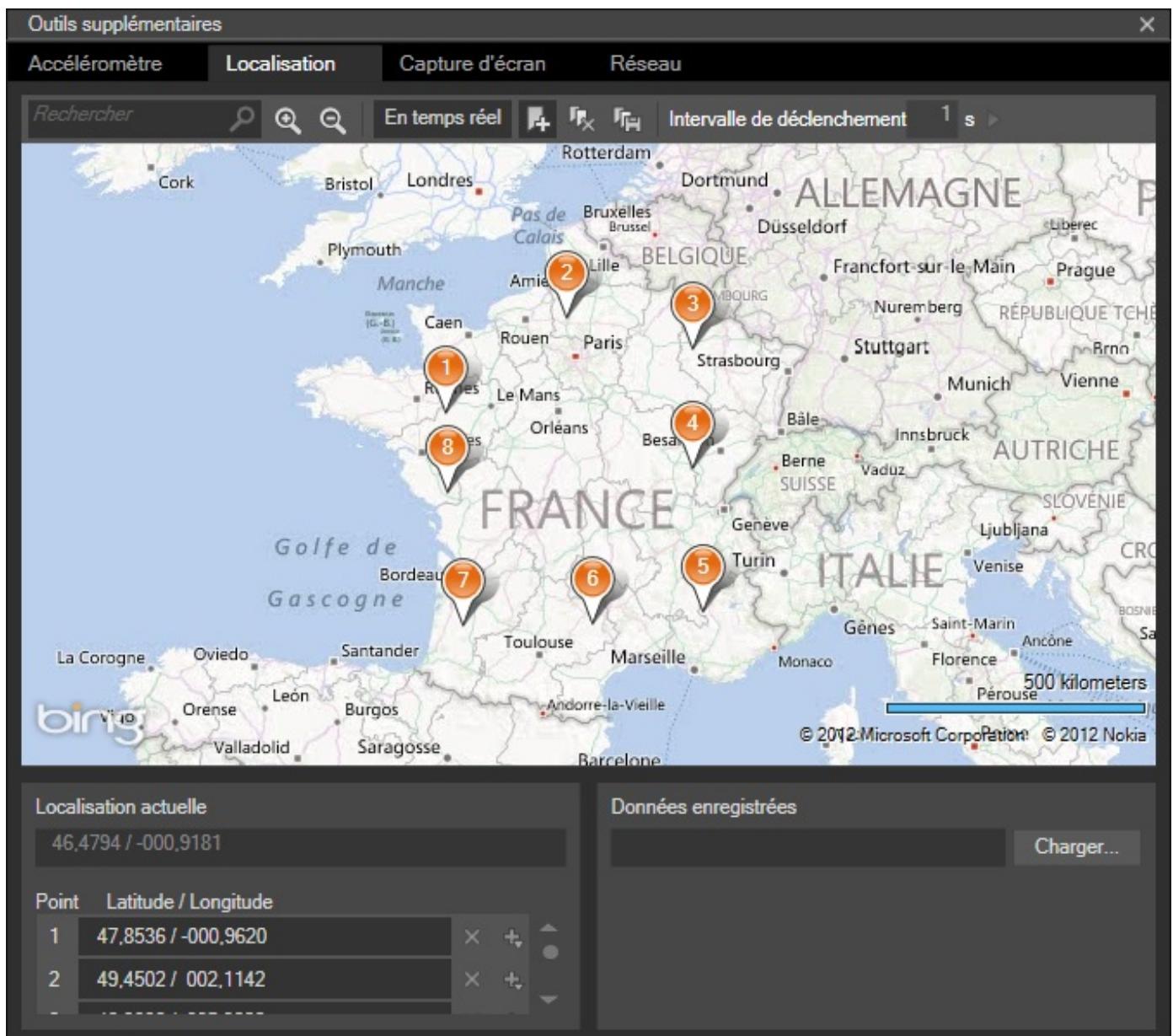
Utiliser la géolocalisation dans l'émulateur



Oui mais un GPS, c'est très bien sur un téléphone... mais sur un émulateur ? Comment on fait ? Il peut quand même nous fournir une position ?

Pour le vérifier, démarrons l'émulateur et récupérons la position comme nous l'avons vu. Nous obtenons une latitude de 47,669444 et une longitude de -122,123889, qui correspond à la position... des bâtiments de Microsoft. Hasard ? 😐

En fait, ceci se passe dans les outils supplémentaires de l'émulateur, le dernier bouton en bas de la barre située à droite de l'émulateur. Il est possible de changer les valeurs et même de définir des itinéraires pour simuler des déplacements, cela se passe dans l'onglet `Localisation`, comme indiqué à la figure suivante.



Simuler la géolocalisation dans les outils de l'émulateur

En quelques clics, je définis ma position et mon tour de France. Il sera alors très facile de faire changer l'émulateur de position pendant son exécution, soit manuellement, soit en jouant un itinéraire préalablement enregistré... Pratique pour simuler des positions ou des changements de position tout en restant confortablement installé sur son PC avec son débogueur.

Utiliser la géolocalisation avec le contrôle Map

La géolocalisation prend tout son intérêt utilisée concomitamment avec le contrôle Map vu dans la partie précédente. Étant donné que le service de localisation fournit un objet avec une latitude et une longitude (de type [Geocoordinate](#)), il est très facile d'utiliser les coordonnées GPS de notre utilisateur sur la carte, par exemple ici je centre la carte sur la position de l'utilisateur :

Code : C#

```
private void geolocator_PositionChanged(Geolocator sender,
PositionChangedEventArgs args)
{
    Dispatcher.BeginInvoke(() =>
    {
        Latitude.Text =
args.Position.Coordinate.Latitude.ToString();
        Longitude.Text =
args.Position.Coordinate.Longitude.ToString();

        Carte.Center = new
```

```
GeoCoordinate(args.Position.Coordinate.Latitude,  
args.Position.Coordinate.Longitude);  
});  
}
```



Attention, il y a une différence entre la classe `Geocoordinate`, issue du service de localisation, et la classe `GeoCoordinate` que nous utilisons avec le contrôle de carte. Notez la subtile différence de casse et les espaces de noms différents, à savoir respectivement `Windows.Devices.Geolocation` et `System.Device.Location`.

Remarquons qu'il faut utiliser le dispatcher pour revenir sur le thread d'interface afin de mettre à jour cette dernière.

- Le service de localisation est accessible à partir du moment où il a été déclaré et accepté par l'utilisateur.
- La classe `Geolocator` permet d'obtenir les informations issues du service de localisation.
- On peut utiliser le contrôle de carte pour afficher sa position, en utilisant la classe `GeoCoordinate`.

Les Tasks du téléphone

Utiliser les capteurs du téléphone n'est pas le seul moyen d'accéder aux fonctionnalités internes du téléphone. Les Task, que l'on peut traduire en tâches, sont une solution permettant d'accéder à d'autres applications du téléphone. Il s'agit par exemple de la possibilité de sélectionner un contact dans le répertoire de l'utilisateur, d'envoyer un SMS, de prendre une photo, etc. Nous allons découvrir dans ce chapitre qu'il y a deux sortes de tâches, les choosers et les launchers, elles sont situées dans l'espace de nom: `using Microsoft.Phone.Tasks;`

Les choosers

Les choosers, comme le nom le suggère aux anglophones, permettent de choisir une information. Plus précisément, il s'agira de démarrer une fonctionnalité qui va nous renvoyer quelque chose d'exploitable, par exemple un contact dans le répertoire. Voici une liste des choosers des Windows Phone 8 :

| Chooser | Description |
|-------------------------|---|
| AddressChooserTask | Démarre l'application Contacts pour choisir une adresse physique |
| AddWalletItemTask | Démarrer l'application Wallet (portefeuille) afin de permettre d'y ajouter un produit |
| CameraCaptureTask | Démarre l'appareil photo pour pouvoir prendre une photo |
| EmailAddressChooserTask | Démarre l'application Contacts pour choisir une adresse email |
| GameInviteTask | Permet d'inviter des autres joueurs à participer à un jeu en ligne |
| PhoneNumberChooserTask | Démarre l'application Contacts pour choisir un numéro de téléphone |
| PhotoChooserTask | Permet de sélectionner une photo du téléphone |
| SaveContactTask | Démarre l'application Contacts pour enregistrer un contact |
| SaveEmailAddressTask | Démarre l'application Contacts pour enregistrer un email |
| SavePhoneNumberTask | Démarre l'application Contacts pour enregistrer un numéro de téléphone |
| SaveRingtoneTask | Démarre l'application Sonneries |

Je ne vais pas tous vous les présenter en détail, car ce serait un peu fastidieux, d'autant plus que la description est globalement assez explicite. Sachez cependant que l'émulateur ne permet pas de simuler toutes les tâches et que vous aurez parfois besoin d'un téléphone pour vous rendre vraiment compte du résultat.

L'important est que les choosers renvoient un élément à exploiter. Donc globalement, il faudra instancier la tâche, s'abonner à l'événement de fin de tâche et exploiter le résultat. Par exemple, pour sélectionner une photo disponible dans le téléphone, nous pourrons utiliser le code suivant :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private PhotoChooserTask photoChooserTask;

    public MainPage()
    {
        InitializeComponent();
        photoChooserTask = new PhotoChooserTask();
        photoChooserTask.Completed += photoChooserTask_Completed;
    }

    private void photoChooserTask_Completed(object sender,
    PhotoResult e)
    {
        if (e.TaskResult == TaskResult.OK)
        {
            BitmapImage image = new BitmapImage();
            image.SetSource(e.ChosenPhoto);
            Photo.Source = image;
        }
    }
}
```

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    photoChooserTask.Show();  
}  
}
```

Comme vous pouvez le constater, la photo sélectionnée est dans le paramètre de l'événement. Celui-ci possède également un résultat de tâche, par exemple, nous n'aurons une valeur dans la propriété ChosenPhoto que si l'utilisateur est allé au bout de la tâche, et qu'également le résultat soit OK.

Côté XAML, nous pourrons avoir :

Code : XML

```
<StackPanel>  
    <Button Content="Choisir la photo" Tap="Button_Tap" />  
    <Image x:Name="Photo" />  
</StackPanel>
```

Dans l'émulateur, nous avons quelques valeurs bouchonnées pour cette tâche, voyez plutôt sur la figure suivante.



Illustrons encore les choosers avec la tâche CameraCaptureTask qui permet de prendre une photo depuis votre application

Code : C#

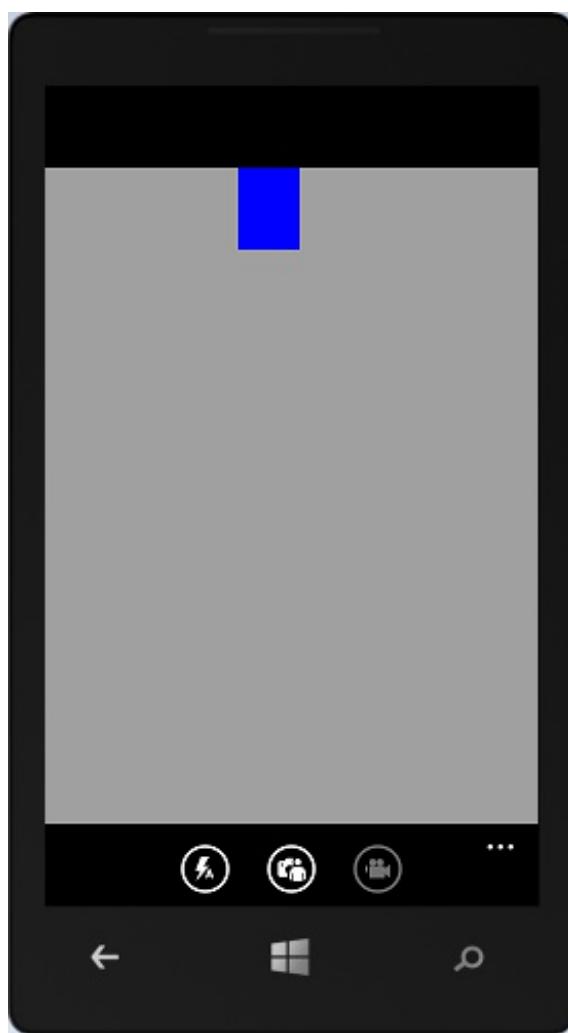
```
public partial class MainPage : PhoneApplicationPage
{
    private CameraCaptureTask cameraCaptureTask;

    public MainPage()
    {
        InitializeComponent();
        cameraCaptureTask = new CameraCaptureTask();
        cameraCaptureTask.Completed += cameraCaptureTask_Completed;
    }

    private void cameraCaptureTask_Completed(object sender,
PhotoResult e)
    {
        if (e.TaskResult == TaskResult.OK)
        {
            BitmapImage image = new BitmapImage();
            image.SetSource(e.ChosenPhoto);
            Photo.Source = image;
        }
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        cameraCaptureTask.Show();
    }
}
```

Ici aussi, l'émulateur fonctionne en mode bouchon et nous propose une image factice pour simuler la prise d'une photo, comme on peut le voir sur la figure suivante.



Simulation de prise de photo

Vous pourrez prendre la photo depuis l'émulateur en cliquant avec le bouton droit.



Attention, vous ne pourrez pas utiliser la tâche CameraCaptureTask sur votre téléphone lorsque celui-ci est branché à votre PC.

Les launchers

Les launchers permettent de démarrer une application sur le téléphone, par exemple envoyer un mail ou un SMS. Ils fonctionnent comme les choosers mais n'attendent pas d'informations en retour.

| Launcher | Description |
|------------------------|---|
| BingMapsDirectionsTask | Permet de démarrer l'application carte de Windows Phone 7 pour afficher un itinéraire |
| BingMapTask | Permet de démarrer la carte de Windows Phone 7, centrée sur une position |
| ConnectionSettingsTask | Permet d'afficher la page de configuration du réseau |
| EmailComposeTask | Permet de composer un nouvel email via l'application de messagerie |
| MapDownloaderTask | Permet de télécharger les cartes pour une utilisation hors ligne |
| MapsDirectionsTask | Permet de démarrer l'application de carte pour afficher un itinéraire entre deux points |
| MapsTask | Permet de démarrer l'application de carte centrée sur un emplacement |
| MapUpdaterTask | Permet de télécharger les mises à jour de cartes |
| MarketplaceDetailTask | Permet d'afficher le détail d'une application dans le Marketplace |
| MarketplaceHubTask | Démarre l'application Marketplace |
| MarketplaceReviewTask | Permet d'atteindre la fiche des avis d'une application |

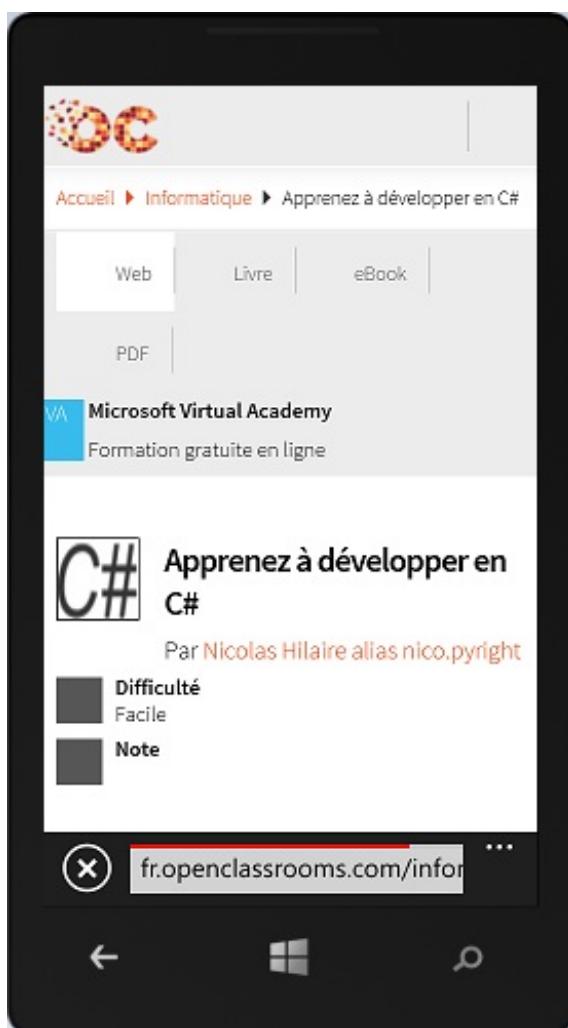
| | |
|-----------------------|---|
| MarketplaceSearchTask | Affiche les résultats d'une recherche Marketplace |
| MediaPlayerLauncher | Démarre le lecteur de média |
| PhoneCallTask | Permet de passer des appels |
| SaveAppointmentTask | Permet d'enregistrer un rendez-vous |
| SearchTask | Démarre une recherche sur le web |
| ShareLinkTask | Partage un lien sur un réseau social |
| ShareMediaTask | Permet de partager une photo ou vidéo avec les applications qui se sont enregistrées pour les exploiter |
| ShareStatusTask | Partage un message de statut sur un réseau social |
| SmsComposeTask | Permet de composer un SMS |
| WebBrowserTask | Démarre le navigateur web |

Ici aussi, les descriptions parlent d'elles-mêmes. Dans notre application de lecteur de flux RSS, il aurait par exemple été possible d'utiliser le WebBrowserTask au lieu d'embarquer le contrôle WebBrowser dans nos pages. Il s'utilise ainsi :

Code : C#

```
WebBrowserTask webBrowserTask = new WebBrowserTask { Uri = new Uri("http://fr.openclassrooms.com/informatique/cours/apprenez-a-developper-en-c", UriKind.Absolute) };
webBrowserTask.Show();
```

qui démarre donc Internet Explorer avec la page demandée (voir la figure suivante).



Le launcher WebBrowserTask affiche une page web

i Il est possible d'utiliser le `WebBrowserTask` pour demander à Internet Explorer de nous afficher un document Word par exemple.

Tous les launchers fonctionnent de la même façon, par exemple pour pré-remplir un SMS avant envoi :

Code : C#

```
SmsComposeTask smsComposeTask = new SmsComposeTask { To =
"+33123456789", Body = "Bon anniversaire !" };
smsComposeTask.Show();
```

À la figure suivante, vous pouvez observer le rendu dans l'émulateur.



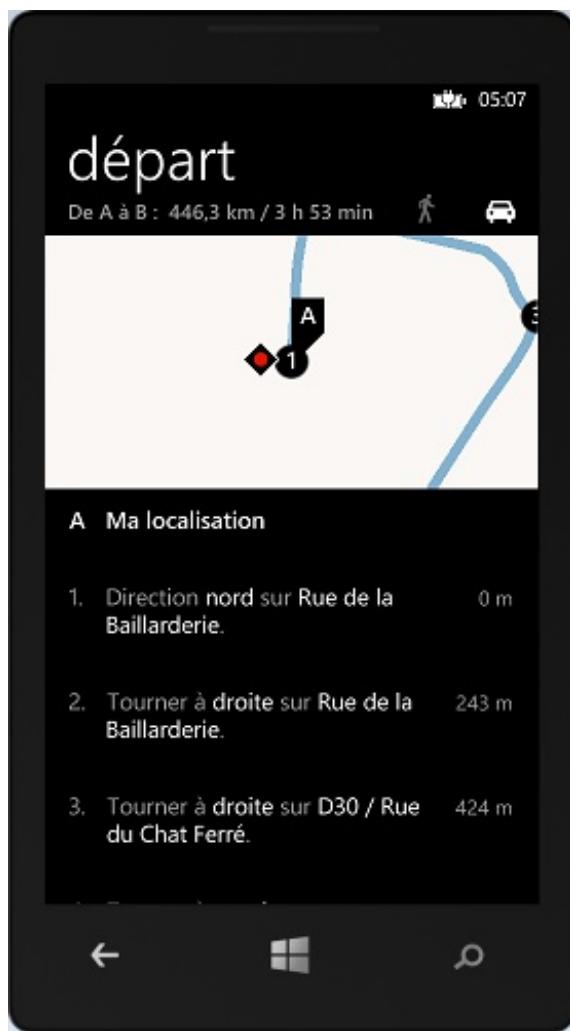
Composition d'un SMS pré-rempli

Voyons enfin un autre launcher bien pratique qui nous permet d'afficher un itinéraire sans passer par les API de la carte que nous avons précédemment vues :

Code : C#

```
MapsDirectionsTask mapsDirectionsTask = new MapsDirectionsTask();
LabeledMapLocation emplacement = new LabeledMapLocation("Tour
Eiffel", null);
mapsDirectionsTask.End = emplacement;
mapsDirectionsTask.Show();
```

Regardez la figure suivante pour le rendu.



Calcul d'itinéraire grâce au launcher

À noter que dans ce cas, il prend la position de l'utilisateur comme point de départ pour calculer l'itinéraire. Vous pouvez affecter la propriété `mapsDirectionsTask.Start` afin de préciser un point de départ différent de la position courante de l'utilisateur. Vous pouvez également spécifier des coordonnées GPS comme point d'arrivée, et dans ce cas-là, la chaîne passée est utilisée comme étiquette :

Code : C#

```
GeoCoordinate tourEiffel = new GeoCoordinate { Latitude = 48.858115,  
Longitude = 2.294710 };  
LabeledMapLocation emplacement = new LabeledMapLocation("Tour  
Eiffel", tourEiffel);
```



Il est de bon ton d'encadrer la méthode `Show()` d'un launcher ou d'un chooser par un `try/catch`, surtout si l'appel à la méthode `Show()` est fait dans un bouton. Si vous démarrez deux tâches en même temps, par exemple en appuyant deux fois rapidement sur le bouton, vous aurez une exception qui fera planter l'application.

Etat de l'application

Comme nous l'avons déjà appris, lorsqu'est démarrée une nouvelle application, l'application en cours passe en mode suspendu et peut potentiellement être terminée. Cela veut dire que lorsque nous démarrons un chooser ou un launcher, il est possible que votre application soit arrêtée. Il faut donc que vous veilliez à enregistrer l'état de votre application au cas où celle-ci serait arrêtée.

Mais en plus, dans le cas du chooser, il pourrait arriver que votre application soit terminée avant d'avoir récupéré la réponse du chooser, ce qui pose un problème. Pour éviter cela, en instanciant un chooser, Windows Phone est capable de vérifier la présence d'une information dans celui-ci. Cela implique que l'événement de fin de choix soit défini sur une variable d'instance de la classe et non dans une méthode, comme ce que j'ai montré dans les exemples de chooser. On ne doit pas déclarer un chooser par exemple dans le constructeur de notre page. L'instanciation correcte du chooser doit donc être :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private EmailAddressChooserTask emailAddressChooserTask;

    public MainPage()
    {
        InitializeComponent();
        emailAddressChooserTask = new EmailAddressChooserTask();
        emailAddressChooserTask.Completed +=
emailAddressChooserTask_Completed;
    }

    private void emailAddressChooserTask_Completed(object sender,
EmailResult e)
    {
        if (e.TaskResult == TaskResult.OK)
            MessageBox.Show("Adresse choisie : " + e.Email);
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        emailAddressChooserTask.Show();
    }
}
```

afin d'avoir une chance de récupérer la sélection.

- Les launchers et les choosers offrent une solution pour interagir avec d'autres applications du téléphone très facilement.
- Ils permettent d'avoir une expérience utilisateur propre et cohérente pour utiliser les applications internes du téléphone.
- Un chooser doit être déclaré dans la portée d'une page.

Les tuiles

Les tuiles, ce sont les icônes représentant nos applications présentes sur la page d'accueil d'un Windows Phone. Elles sont un raccourci pour démarrer les applications présentes dans notre téléphone, un peu comme les icônes présentes sur le bureau de nos vieux Windows. Pour l'instant, rien d'extraordinaire vous me direz !

Mais elles sont un peu plus qu'une simple icône permettant de démarrer une application. Elles peuvent avoir une icône mais également du texte fournit des informations sur l'application ou son contexte. Elles peuvent également être animées et sont une des grandes forces de Windows Phone. D'ailleurs, Windows 8 s'est empressé de se les récupérer 😊.

Présentes en une seule taille avec Windows Phone 7.5, elles se déclinent en trois tailles à partir de Windows Phone 7.8, pour le plus grand plaisir des utilisateurs qui les ont adoptées avec intérêt.

Découvrons à présent ces fameuses tuiles.

Que sont les tuiles ?

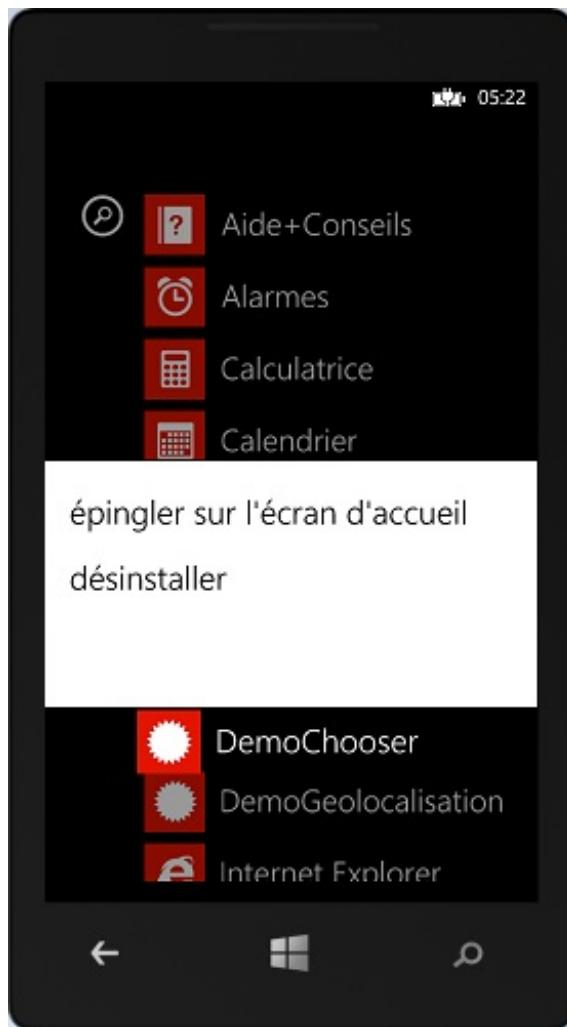
Les tuiles, ce sont les icônes représentant nos applications présentes sur la page d'accueil d'un Windows Phone (ou sur la page d'accueil de l'émulateur, comme à la figure suivante) :



Les tuiles de la page d'accueil dans l'émulateur

Lorsque l'on clique dessus, notre application est lancée. Lorsque l'on télécharge (ou développe 😊) une application, la tuile n'est pas tout de suite présente sur l'accueil. C'est à l'utilisateur de choisir explicitement de l'y mettre en l'épinglant sur la page d'accueil. Cela se fait en allant dans la liste des applications et en appuyant longtemps sur l'application. Cela fait apparaître un menu contextuel qui propose de supprimer l'application, de l'épingler sur la page d'accueil ou d'aller la noter sur le Windows Phone Store.

Il est également possible d'épingler une application depuis l'émulateur suivant le même principe, en cliquant longtemps sur l'application. Par exemple dans la figure suivante, pour épingler l'application DemoChooser, je clique dessus et un menu déroulant apparaît.



Épingler une application



Note : Pour accéder à la liste des applications de l'émulateur, le plus simple est de cliquer sur le bouton de menu et de faire bouger l'écran sur la droite.

Dans l'émulateur, on ne peut bien sûr pas noter les applications et d'autant plus celles que nous sommes en train de développer 😞.

Une fois l'application épingle, nous pouvons aller sur l'écran d'accueil et nous aurons une magnifique tuile (voir figure suivante).



Notre application est épingle et nous voyons sa tuile sur la page d'accueil

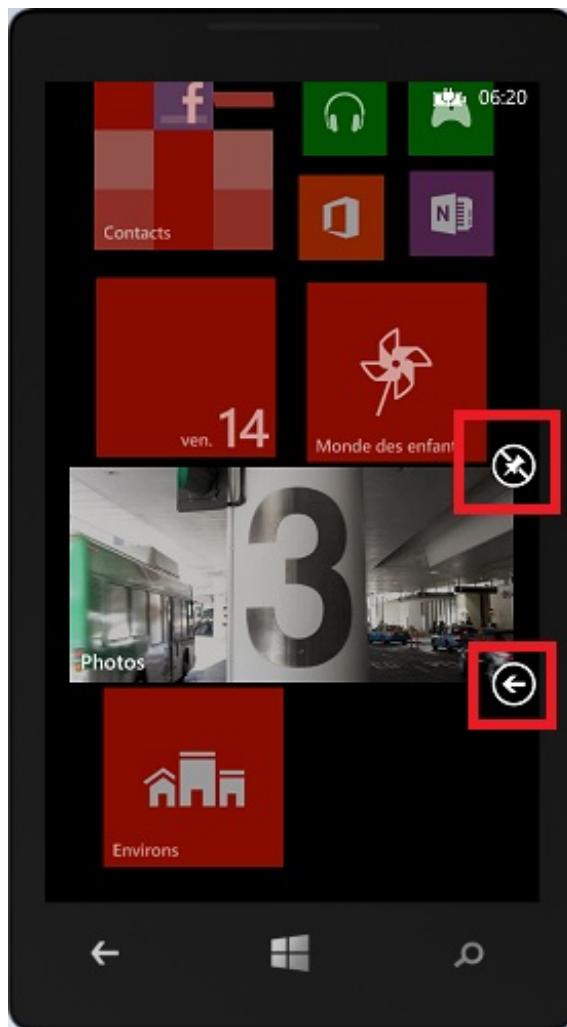
Des tuiles pour tous les goûts

Les tuiles existent en différentes tailles. Il y a les petites, les moyennes et les larges. Si les applications l'ont prévu, il est possible de redimensionner ces tuiles sur notre écran d'accueil. Voici par exemple dans la figure suivante la tuile permettant d'accéder aux images en taille large.



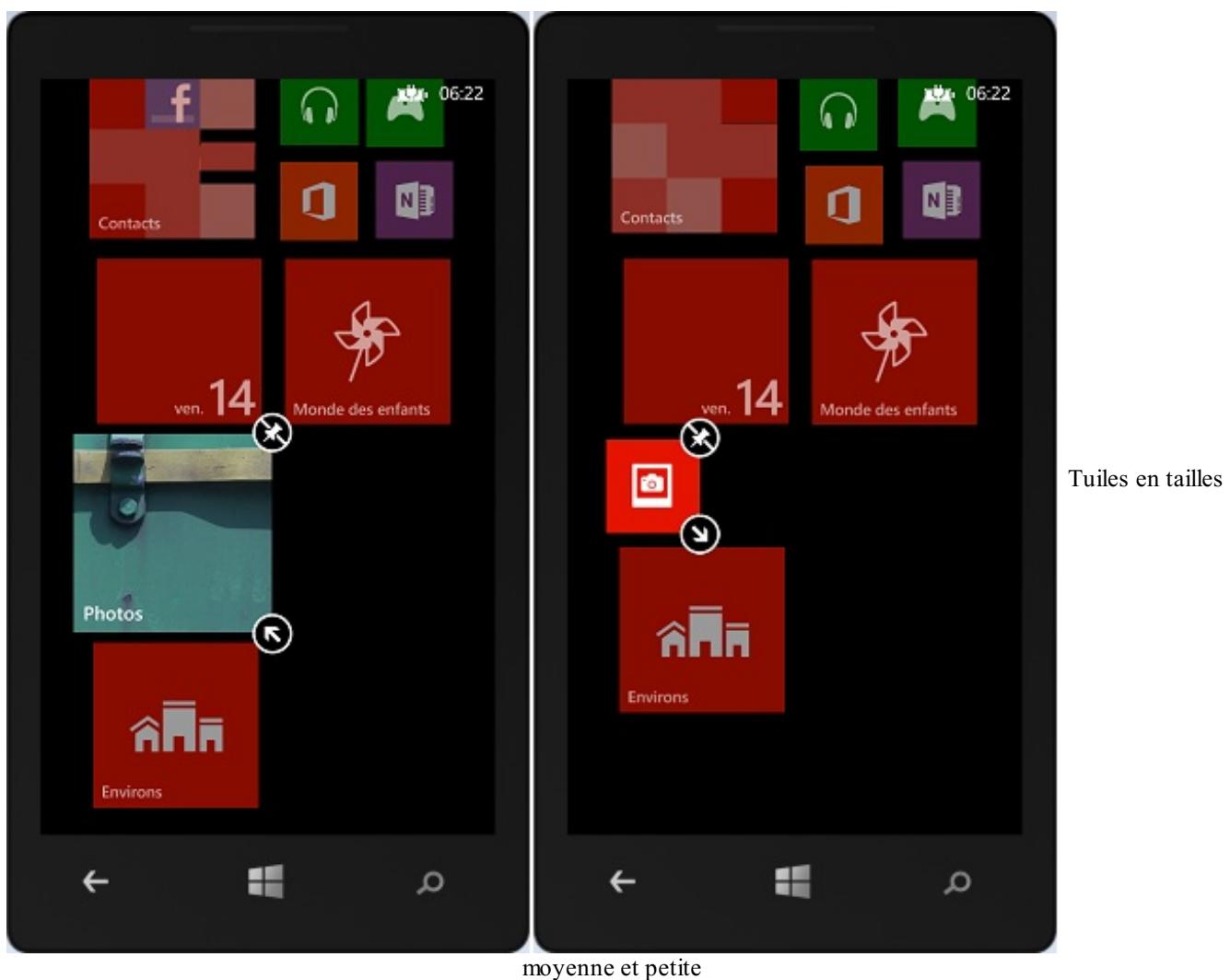
Une tuile en taille large

Pour redimensionner la tuile, il faut faire un toucher long sur celle-ci. Deux icônes apparaissent alors à droite de la tuile, comme le montre la figure suivante.



Redimensionnement des tuiles

La première icône, en haut à droite de la tuile, permet de désépingler la tuile. Mais ici, c'est la seconde, en bas à droite de la tuile, qui nous intéresse et qui permet de passer dans la taille inférieure, ici la taille moyenne, puis la petite taille dans la figure suivante.



On peut constater que chaque taille offre une expérience différente fournissant plus ou moins d'informations, voire une information différente en fonction des souhaits de l'utilisateur.

Si vous essayez de redimensionner d'autres tuiles, vous verrez qu'elles ne supportent pas toutes les trois tailles, c'est une question de choix du créateur de l'application.

De même, les tuiles peuvent être de plusieurs styles différents. Nous venons de voir la tuile de l'application images qui, dans son format large et moyen, affiche des images qui défilent sur la tuile. On appelle ce format le « cycle tile template », que l'on peut traduire en modèle de tuile cyclique. Elle permet de faire défiler entre 1 et 9 images.

Il existe trois modèles en tout que nous allons découvrir dans ce chapitre :

- Le modèle de tuile cyclique (cycle tile template)
- Le modèle de tuile qui se retourne (flip tile template)
- Le modèle de tuile icône (iconic tile template)

Les tuiles ont les tailles suivantes :

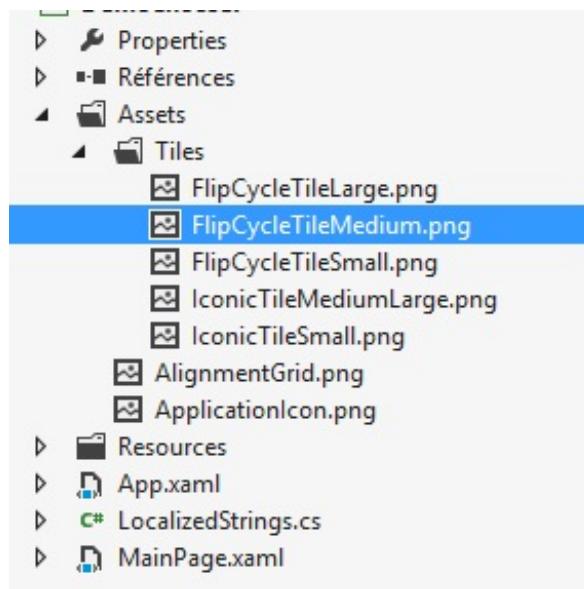
| - | Flip et cycle | Iconic |
|---------|---------------|---------|
| Petite | 159x159 | 110x110 |
| Moyenne | 336x336 | 202x202 |
| Large | 691x336 | - |

Nul besoin de vous soucier des différentes résolutions, Windows Phone s'occupe de redimensionner tout pour nous, vous n'avez besoin que d'ajouter les images pour la résolution XWGA.

Personnaliser les tuiles par défaut

Nous venons de voir comment épinglez notre application sur la page d'accueil. Celle-ci prend par défaut l'apparence de l'image

se trouvant dans le répertoire du projet Assets\Tiles\FlipCycleTileMedium.png, que l'on peut retrouver dans l'explorateur de solutions illustré dans la figure suivante.



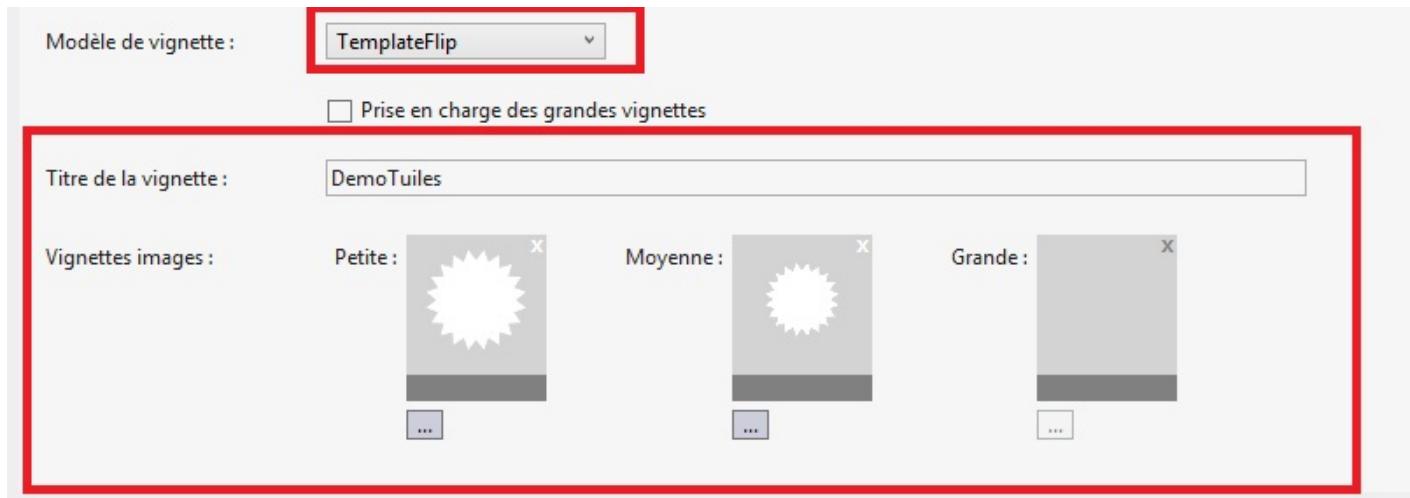
Les images par défaut des tuiles

Cette tuile s'enrichit du titre de l'application (voir figure suivante).



Image de la tuile par défaut

Plus précisément, elle est construite grâce aux paramètres présents dans le fichier WMAppManifest.xml, que l'on obtient en dépliant le répertoire *properties* du projet et en double-cliquant dessus (voir prochaine figure).



Paramétrage de la tuile par défaut, dans WMAppManifest.xml

Nous pouvons voir que le modèle de la vignette est Flip et que l'application ne supporte que les tuiles de taille petite et moyenne. D'ailleurs, si vous essayez de redimensionner la tuile sur la page d'accueil, vous ne pourrez pas l'avoir en large. Pour cela, vous pouvez cocher la case « prise en charge des grandes vignettes » et choisir une image pour la grande taille, par exemple FlipCycleTileLarge.png qui est (comme par hasard) à la bonne taille et qui se trouve dans le répertoire Assets/Tiles avec les autres images de tuiles.



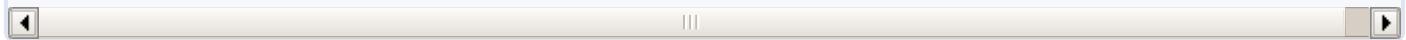
Note : pour voir les changements dans l'émulateur, vous allez devoir dés-épingler la tuile et la ré-épingler.



Mais l'éditeur du fichier WMAppManifest.xml n'est pas complet. Enfin, disons qu'on peut faire plus de choses directement en modifiant le contenu du fichier XML. Si vous l'ouvrez, vous verrez notamment la section Tokens contenant la définition des tuiles :

Code : XML

```
<Tokens>
  <PrimaryToken TokenID="DemoTuilesToken" TaskName="_default">
    <TemplateFlip>
      <SmallImageURI IsRelative="true"
        IsResource="false">Assets\Tiles\FlipCycleTileSmall.png</SmallImageURI>
      <Count>0</Count>
      <BackgroundImageURI IsRelative="true"
        IsResource="false">Assets\Tiles\FlipCycleTileMedium.png</BackgroundImageURI>
      <Title>DemoTuiles</Title>
      <BackContent>
        </BackContent>
      <BackBackgroundImageURI>
        </BackBackgroundImageURI>
      <BackTitle>
        </BackTitle>
      <LargeBackgroundImageURI IsRelative="true"
        IsResource="false">Assets\Tiles\FlipCycleTileLarge.png</LargeBackgroundImageURI>
      <LargeBackContent />
      <LargeBackBackgroundImageURI IsRelative="true" IsResource="false">
        </LargeBackBackgroundImageURI>
      <DeviceLockImageURI>
        </DeviceLockImageURI>
      <HasLarge>True</HasLarge>
    </TemplateFlip>
  </PrimaryToken>
</Tokens>
```



On y retrouve les diverses urls relatives vers nos images de tuiles, mais aussi d'autres choses. Modifions par exemple la balise count pour mettre la valeur 3 à la place de 0 :

Code : XML

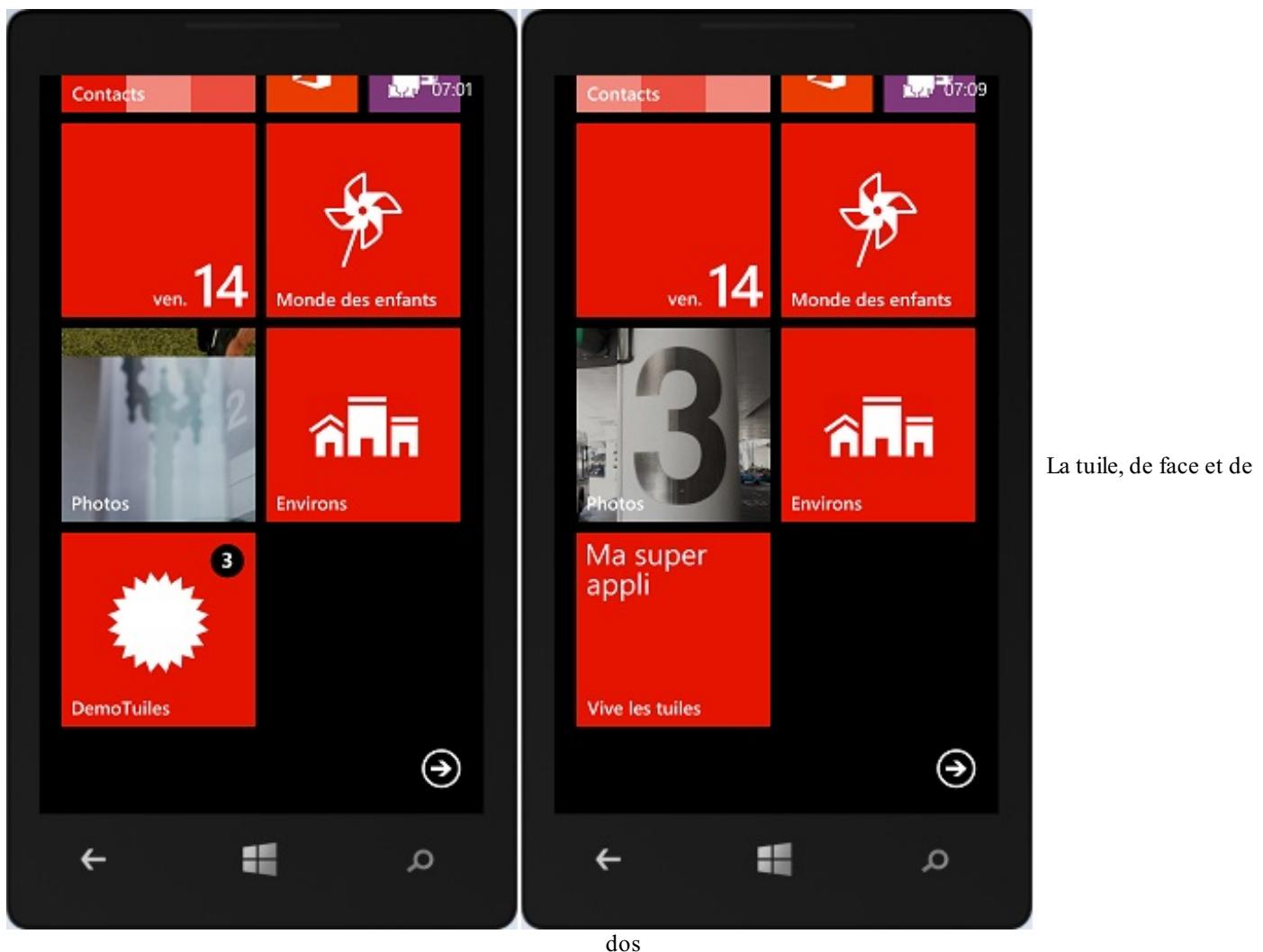
```
<Count>3</Count>
```

Modifions également les balises BackContent et BackTitle :

Code : XML

```
<BackContent>Ma super appli</BackContent>
<BackTitle>Vive les tuiles</BackTitle>
```

Nous obtenons le résultat dans la figure qui suit.



Sur la partie gauche de l'image, nous pouvons voir un petit 3, qui correspond à la balise Count. Cela permet, pourquoi pas, d'indiquer qu'il y a 3 éléments nouveaux à venir consulter dans notre application. La partie droite de l'image correspond au dos de la tuile. Rappelez-vous, notre application utilise le modèle de tuile « Flip » qui se retourne pour fournir d'autres informations. Ici, nous avons affiché du texte, mais il est également possible d'afficher une image grâce à la balise BackBackgroundImageURI. Notez que si vous changez la taille de la tuile, vous verrez que le dos de la tuile ne s'affiche qu'en grande ou moyenne taille et qu'il ne s'affiche pas en petite taille.

Mais tout ceci est bien statique ... notamment par rapport au chiffre en haut à droite. On ne peut pas envisager de déterminer qu'il y aura 3 nouveaux éléments tout le temps, dès la création de l'application. Heureusement, il est également possible de modifier la tuile pour mettre à jour quelques informations lorsque l'application se ferme, et rappeler à l'utilisateur où il en était lorsqu'il la rouvre. Pour effectuer ces mises à jour, il faut accéder à la tuile concernée via la classe statique `ShellTile`, avec le code suivant :

Code : C#

```
public MainPage()
{
    InitializeComponent();

    ShellTile tuileParDefaut = ShellTile.ActiveTiles.First();

    if (tuileParDefaut != null)
    {
        FlipTileData flipTileData = new FlipTileData
        {
            Title = "Ma tuile",
            Count = 4,
            BackTitle = "Nouveau titre",
            BackContent = "Ouvrez vite !",
        };
    }
}
```

```
        tuileParDefaut.Update(flipTileData);  
    }  
}
```

Une fois la mise à jour de la tuile faite, nous pourrons avoir un nouveau rendu, illustré dans la figure suivante.



La tuile créée par code,

en grande taille

Du coup, je vous montre la tuile en grande taille 😊.

Notez la présence du chiffre 4 en haut à droite qui correspond à la propriété Count. Mettre 0 ou null dans la propriété effacera le nombre qui apparaît.

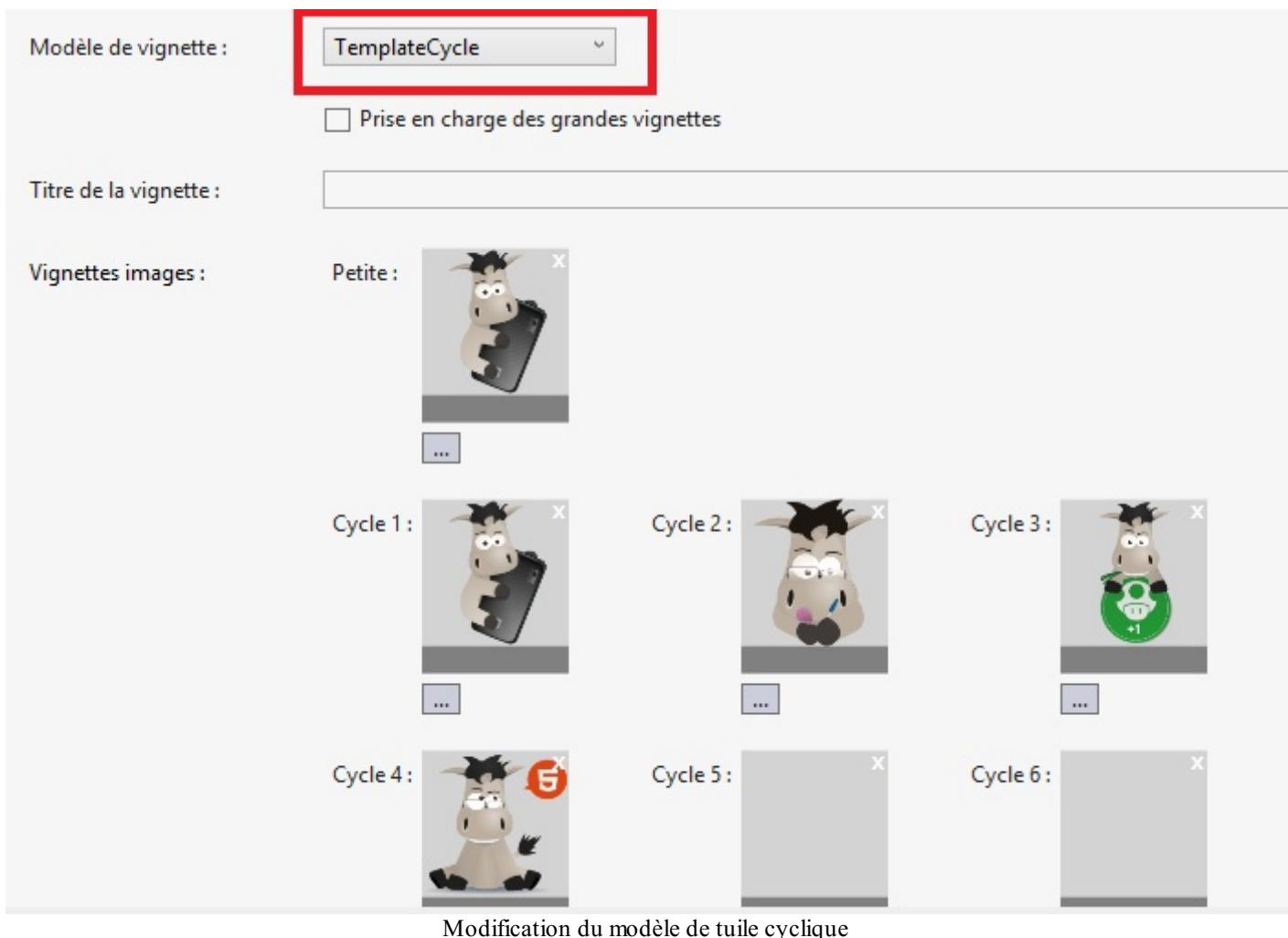


Attention : le fait de mettre à jour la tuile par défaut par code n'épingle pas pour autant l'application sur l'écran d'accueil. Il faut que ce soit une action volontaire de l'utilisateur. Cela changera juste son apparence.

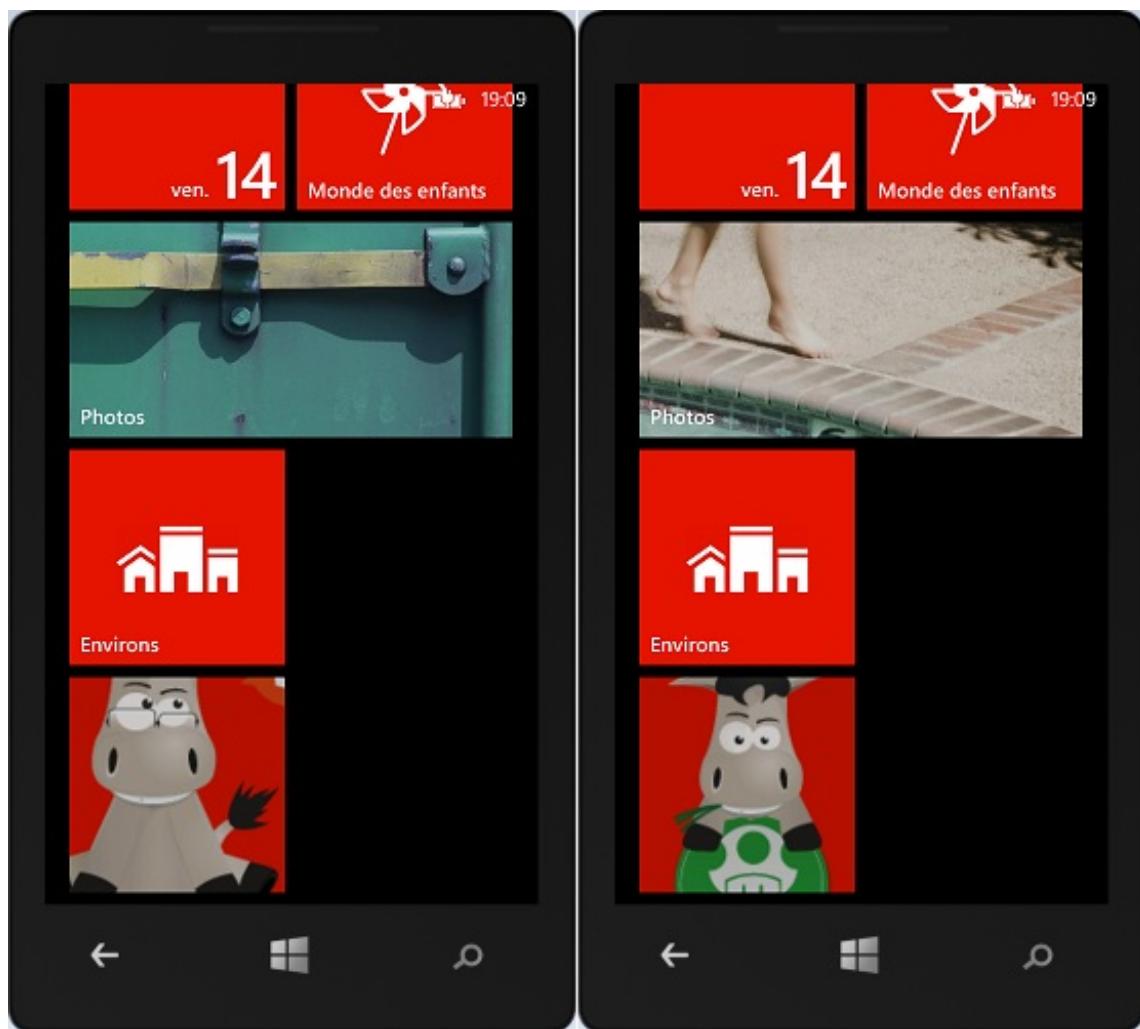


Et les autres modèles de tuiles alors ?

J'y arrive. Car effectivement, nous n'avons vu que les tuiles qui se retournent (« flip »), il reste encore les tuiles cycliques (« cycle ») et les tuiles icônes (« iconic »). Sachez dès à présent que le principe de création est globalement le même. Pour les tuiles cycliques (« cycle »), vous pouvez faire défiler de 1 à 9 images. Changez le modèle dans l'éditeur et vous pourrez voir que vous allez pouvoir remplir jusqu'à 9 images (voir prochaine figure).



J'en ai profité pour mettre quelques images de notre mascotte préférée dans la figure suivante. Une fois la mascotte épinglée, nous pourrons voir les images défiler.

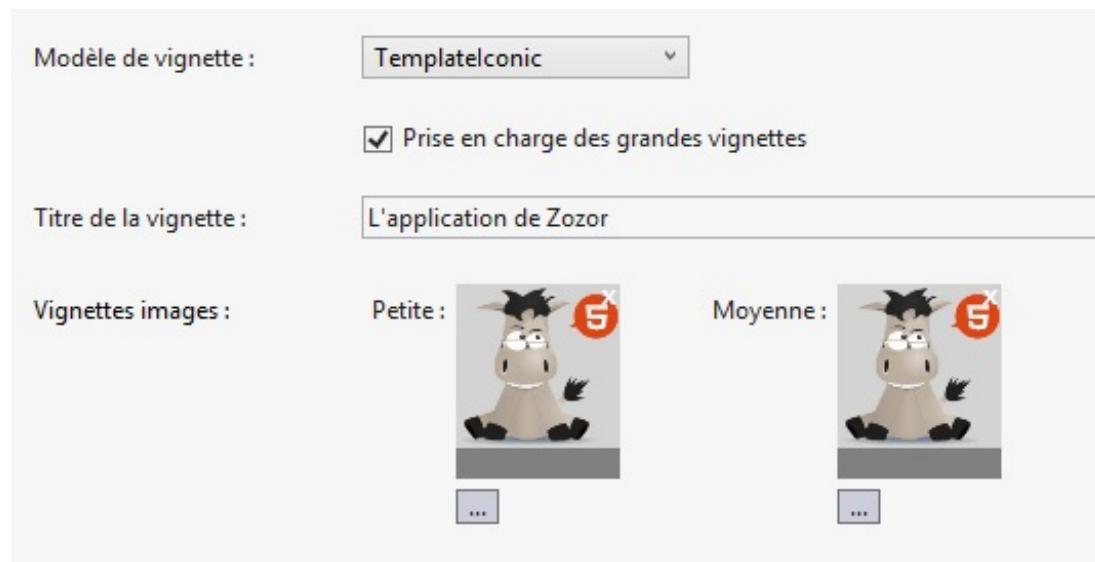


Les images de Zozor

défilent dans la tuile

Pour mettre à jour l'image qui s'affiche par défaut sur la tuile, on utilisera la classe [CycleTileData](#) et notamment la propriété CycleImages qui est un tableau d'Uri pointant sur des images à afficher. Notez que la propriété Count est également disponible pour ce modèle de tuile.

Reste les tuiles icônes (« iconic », voir figure suivante).



tuiles icônes

Le résultat des tuiles icônes en grande taille est illustré dans la figure qui suit.



Tuile icône en grande taille

Remarque, ici j'ai modifié la balise Count et la balise Message :

Code : XML

```
<Count>1</Count>
<Message>Venez vite découvrir Zozor !</Message>
```

À noter qu'on utilisera la classe `IIconicTileData` pour créer la tuile icône par code.



Attention : il n'est pas possible de changer dynamiquement (en utilisant du code) le modèle de tuile par défaut que nous venons de changer par l'interface. Si vous souhaitez le faire une fois votre application terminée et déployée sur un téléphone, il vous faudra mettre à jour votre application et la déployer à nouveau.

Créer des tuiles secondaires

La tuile principale permet de lancer l'application. Il n'y a qu'une façon pour les obtenir : épinglez l'application depuis le menu, comme nous l'avons vu.

Les tuiles secondaires permettent de démarrer l'application d'une façon particulière. Elles servent en général à accéder à des fonctionnalités de l'application, comme un raccourci. Ainsi, il est possible de créer des tuiles qui vont naviguer directement sur une page précise de l'application, avec pourquoi pas des paramètres.

Reprenez par exemple notre dernier TP, qui contenait un jeu de grattage et un jeu de secouage ... Quand j'ai réalisé ces jeux, c'était un peu insupportable de devoir passer à chaque fois par le menu pour lancer une application et tester une petite modification. Que je suis triste de ne pas avoir eu une telle fonctionnalité de tuiles secondaires ! Avec elles, j'aurai pu facilement créer des raccourcis vers les pages de mon choix ...

Alors, réparons tout de suite cette erreur et rajoutons sur la page principale deux boutons permettant de créer nos tuiles secondaires :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackPanel>
        <Button Content="Grattage ..." Tap="Button_Tap" />
        <Button Content="Secouage ..." Tap="Button_Tap_1" />
    </StackPanel>
    <StackPanel Grid.Row="1" VerticalAlignment="Bottom">
        <Button Content="Créer tuile grattage" Tap="Button_Tap_2" />
        <Button Content="Créer tuile secouage" Tap="Button_Tap_3" />
    </StackPanel>
</Grid>
```

Nous utiliserons grossièrement le même code et le même principe pour créer une tuile secondaire :

Code : C#

```
private void Button_Tap_2(object sender,
System.Windows.Input.GestureEventArgs e)
{
    ShellTile tuileGrattage =
ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("Grattage.xaml"));
    if (tuileGrattage == null)
    {
        FlipTileData tuile = new FlipTileData
        {
            SmallBackgroundImage = new Uri("/Assets/Tiles/gratter-
petit.png", UriKind.Relative),
            BackgroundImage = new Uri("/Assets/Tiles/gratter-
moyen.png", UriKind.Relative),
            Title = "Grattage",
            BackContent = "Accès direct au grattage",
        };

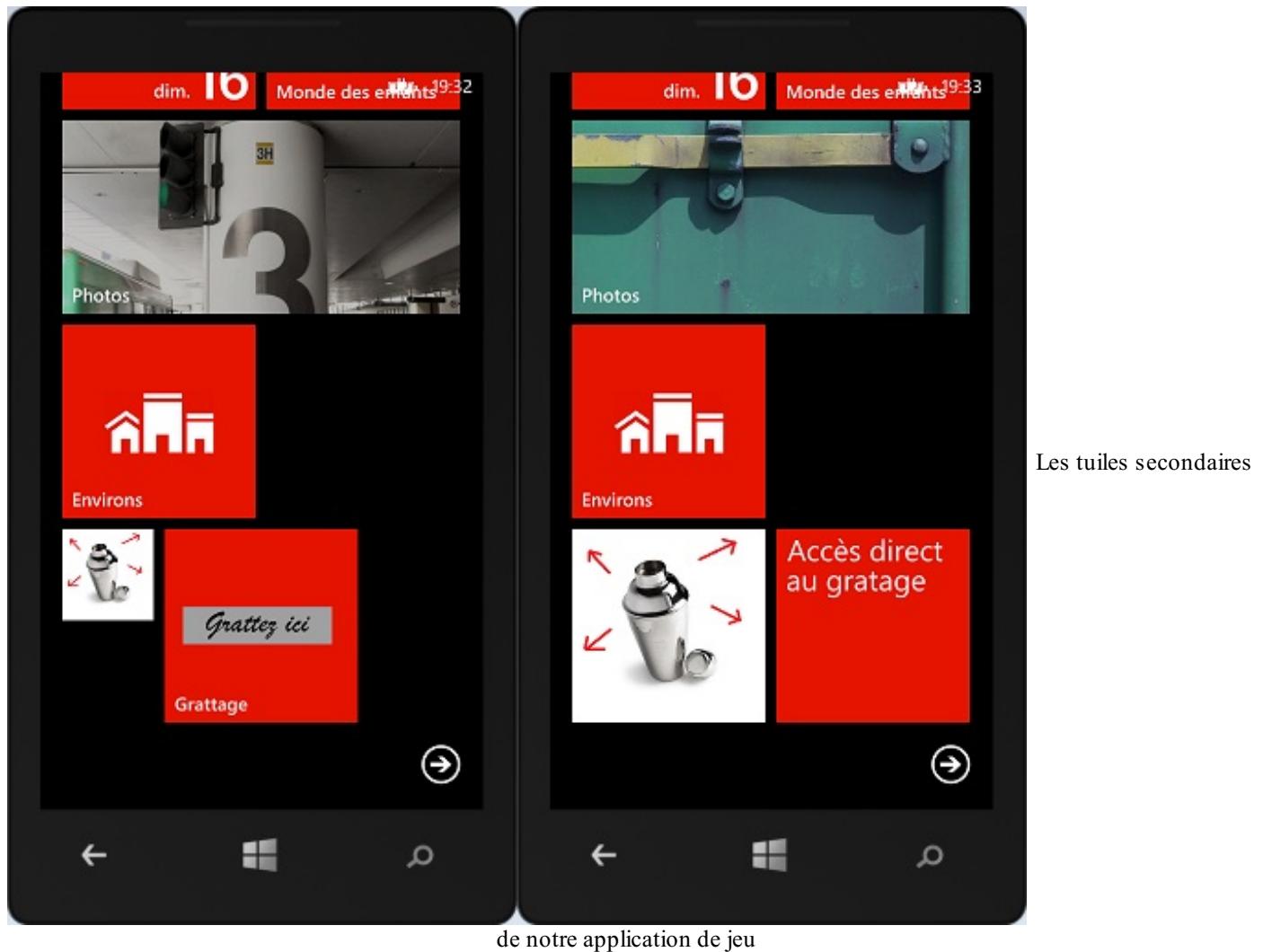
        ShellTile.Create(new Uri("/Grattage.xaml",
UriKind.Relative), tuile, false);
    }
}

private void Button_Tap_3(object sender,
System.Windows.Input.GestureEventArgs e)
{
    ShellTile tuileSecouage =
ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("Secouage.xaml"));
    if (tuileSecouage == null)
    {
        FlipTileData tuile = new FlipTileData
        {
            SmallBackgroundImage = new Uri("/Assets/Tiles/secouer-
petit.png", UriKind.Relative),
            BackgroundImage = new Uri("/Assets/Tiles/secouer-
moyen.png", UriKind.Relative),
            Title = "Secouage",
            BackContent = "Accès direct au secouage",
        };

        ShellTile.Create(new Uri("/Secouage.xaml",
UriKind.Relative), tuile, false);
    }
}
```

Pour éviter d'ajouter plusieurs fois la même tuile, on pourra se baser sur la propriété `NavigationUri` des tuiles existantes, afin de vérifier si elles ont déjà été ajoutées.

Il ne reste plus qu'à cliquer sur nos boutons. Vous verrez que lorsque la tuile secondaire est créée, nous sommes renvoyés sur la page d'accueil pour voir le résultat, ce qui implique que l'application passe dans un mode désactivé. Pour la réactiver, il faudra soit faire un retour arrière, soit pourquoi pas utiliser nos nouvelles tuiles.



Et ainsi, nous accédons directement à la page XAML indiquée lors de la création de la tuile secondaire, qui permet de naviguer directement sur la page. Pratique non ?

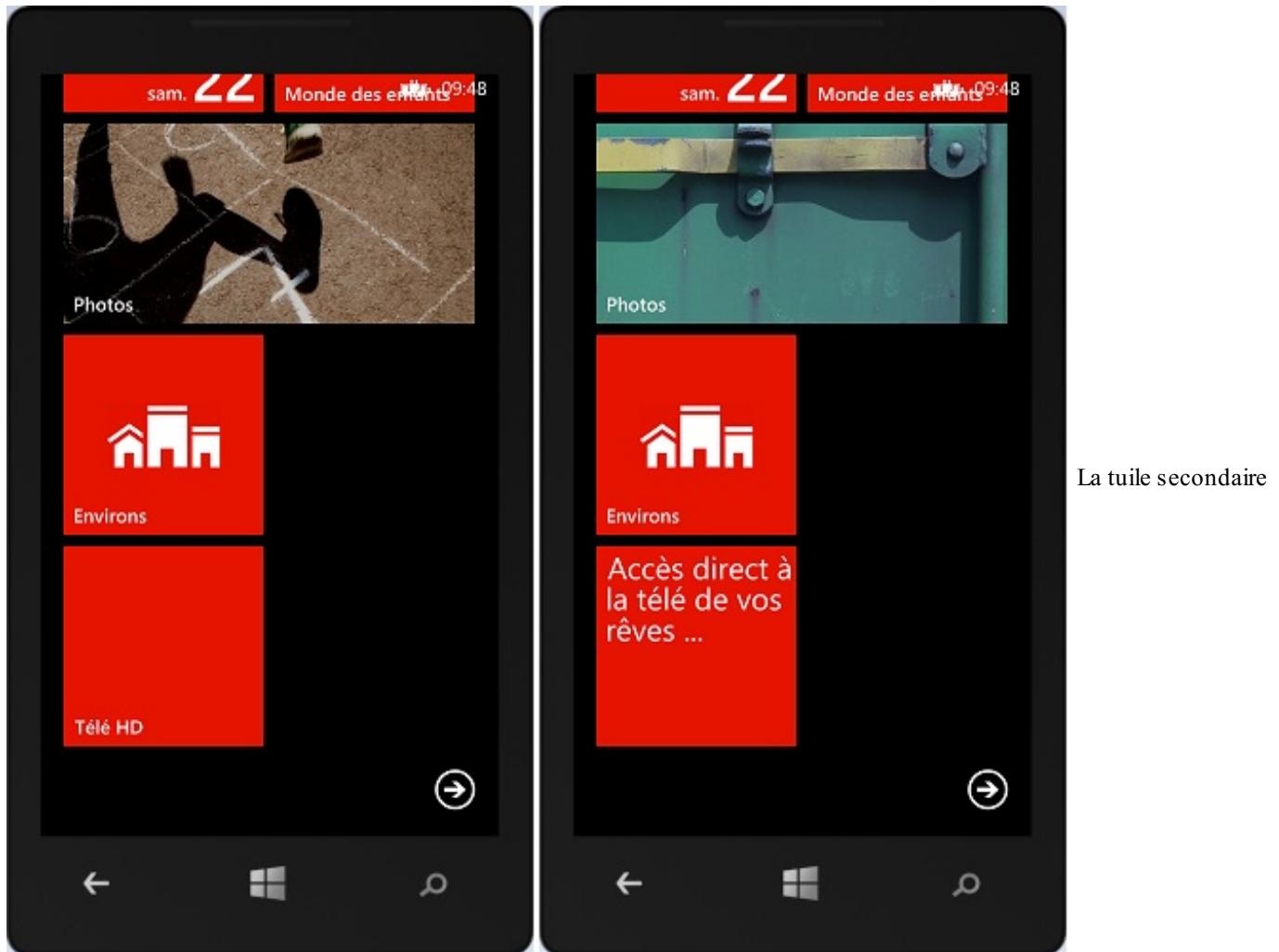
Remarquez que vous pouvez également passer des paramètres à la page, comme ce que nous avions fait dans le chapitre sur la navigation. Ici, ce n'est pas utile car nous avons deux pages distinctes, mais nous pourrions avoir la même page qui affiche une fiche produit d'une application de vente que l'on souhaiterait charger avec des produits différents ... Par exemple :

Code : C#

```
string idProduit = "telehd";
ShellTile tuileProduit = ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("idproduit=" + idProduit));
if (tuileProduit == null)
{
    FlipTileData tuile = new FlipTileData
    {
        Title = "Télé HD",
        BackContent = "Accès direct à la télé de vos rêves ...",
    };
    ShellTile.Create(new Uri("/VoirProduit.xaml?idproduit=" +
idProduit, UriKind.Relative), tuile, false);
```

}

Vous pouvez voir ce que donne ce bout de code sur la figure suivante.



La tuile secondaire

Je vous renvoie au chapitre sur la navigation pour exploiter la query string, avec :

Code : C#

```
NavigationContext.QueryString.TryGetValue("idproduit", out valeur);
```

Bien sûr, vous pouvez créer des tuiles secondaires dans tous les modèles que vous souhaitez. Illustrons par exemple le modèle cyclique :

Code : C#

```
ShellTile secondeTuile = ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("VieZozor.xaml"));
if (secondeTuile == null)
{
    CycleTileData tuile = new CycleTileData
    {
        CycleImages = new Uri[]
        {
            new Uri("/zozor_01.png", UriKind.Relative),
            new Uri("/zozor_02.png", UriKind.Relative),
            new Uri("/zozor_03.png", UriKind.Relative),
            new Uri("/zozor_04.png", UriKind.Relative)
        }
    };
    secondeTuile.Update(tuile);
}
```

```
        new Uri("/zozor_03.png", UriKind.Relative),
        new Uri("/zozor_04.png", UriKind.Relative),
    },
    Title = "Accédez à la vie de zozor"
};

ShellTile.Create(new Uri("/VieZozor.xaml", UriKind.Relative),
tuile, false);
}
```

Modifier et supprimer une tuile

Nous avons vu qu'il était possible de modifier des tuiles via la méthode Update. Ceci est valable pour la tuile principale, mais également pour les tuiles secondaires. Par exemple, le code ci-dessous montre comment mettre à jour une tuile secondaire d'une application de vente en ligne en affichant le changement de stock de l'un de ses produits :

Code : C#

```
ShellTile tuileProduit = ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("idproduit=" + idProduit));
if (tuileProduit != null)
{
    FlipTileData tuile = new FlipTileData
    {
        Title = "Télé HD",
        BackContent = "Plus que 2 produits en stock !!!!",
    };
    tuileProduit.Update(tuile);
}
```

Il n'est par contre pas possible de modifier l'URL de la page à afficher. Dans ce cas, il faudra la supprimer puis la recréer. Supprimer une tuile secondaire est très facile, il suffit d'utiliser la méthode Delete. Par exemple, si nous souhaitons supprimer la tuile du secouage, nous pourrons écrire :

Code : C#

```
ShellTile tuileSecouage = ShellTile.ActiveTiles.FirstOrDefault(elt
=> elt.NavigationUri.ToString().Contains("Secouage.xaml"));
if (tuileSecouage == null)
    tuileSecouage.Delete();
```

L'utilisateur pourra bien sûr faire la même chose tout seul depuis l'écran d'accueil.



Attention, on peut supprimer une tuile secondaire, par contre il est impossible de supprimer la tuile principale par code.



Remarque, il est très courant de mettre à jour une tuile via un agent qui tourne en tâche de fond. Cela offre la possibilité d'informer l'utilisateur que quelque chose de nouveau s'est passé dans son application et qu'il serait bien de lancer l'application pour le découvrir, comme l'application de mail qui affiche le nombre de mails non-lus. Nous découvrirons les agents tournant en tâche de fond dans un chapitre ultérieur.

- Les tuiles se trouvent sur la page d'accueil d'un Windows Phone et sont un raccourci évolué permettant de démarrer une application.
- Elles sont disponibles en plusieurs tailles : petite, moyenne ou grande.
- Il existe plusieurs modèles de tuiles : tuile icône (« iconic »), tuile cyclique (« cyclic »), tuile qui se retourne (« flip »).
- Il est possible de créer des tuiles secondaires, qui permettent d'accéder à une application à l'aide de raccourcis supplémentaires.

Les notifications

Le principe des notifications n'est pas vraiment nouveau mais est globalement plutôt simple. Imaginons que j'installe une application qui me permette de consulter les cours de la plateforme OpenClassrooms. Ca y est, j'ai lu tous les cours qui m'intéressent et j'ai hâte que de nouveaux cours voient le jour pour que je puisse assouvir ma soif d'apprendre. Sauf que je ne vais pas démarrer l'application toutes les 5 minutes histoire de voir si un nouveau cours est en ligne ... ça serait bien si quelqu'un me prévenait dès qu'un nouveau cours est mis en ligne. Et tout ça, même si l'application n'est pas démarrée ou épinglee ...

Voilà, le principe de la notification. Cela consiste en la réception d'un petit message sur notre téléphone avec une information. La réception du message peut être signalée par un bruit ou une vibration, puis le message reste affiché quelques secondes pour nous donner le temps de le lire. Mieux, si nous cliquons sur le message, notre application est automatiquement démarrée. En revanche, un des inconvénients est justement que la notification ne s'affiche que quelques secondes et si on est loin du téléphone, il n'y a aucun moyen pour voir le contenu si on l'a raté (en tout cas à l'heure où j'écris ces lignes, car il y aurait des rumeurs de création de centre de notification ...).

Mais les notifications sont tout de même bien pratiques, donc plongeons nous tout de suite dans le vif du sujet pour créer nos propres alertes et rester au courant des nouveautés de nos applications préférées !

Le principe d'architecture des notifications

Pour recevoir une notification, il faut que quelqu'un l'envoie. Forcément !

Et pour que ce quelqu'un l'envoie, il faut qu'on lui dise que ça nous intéresse de recevoir des notifications.

Donc, le principe global peut se résumer dans les étapes suivantes :

- L'application Windows Phone déclare un canal pour recevoir des notifications.
- L'application Windows Phone indique à l'émetteur (le créateur de l'application) qu'elle souhaite recevoir des notifications et lui donne l'identifiant unique du téléphone sur lequel elle tourne ainsi que le canal.
- L'émetteur enregistre l'identifiant et le canal et l'ajoute à la liste des toutes les applications souhaitant recevoir des notifications.
- L'émetteur souhaite envoyer un message : pour chaque téléphone souhaitant recevoir des notifications, le message est envoyé au service de notification de Microsoft (le Microsoft Push Notification Service) grâce au canal.
- Le service de notification envoie le message à chaque téléphone.
- Le message s'affiche sur le téléphone de l'utilisateur.

Il y a donc trois intervenants dans ce mécanisme :

- Le téléphone, qui reçoit les notifications, ainsi que l'application installée dessus (votre application !),
- Le service de notification, créé par le développeur qui souhaite envoyer des notifications (VOUS !),
- Le service de notification de Microsoft, qui s'occupe d'envoyer vraiment le message au téléphone.

Sachez que le message peut être de plusieurs types.

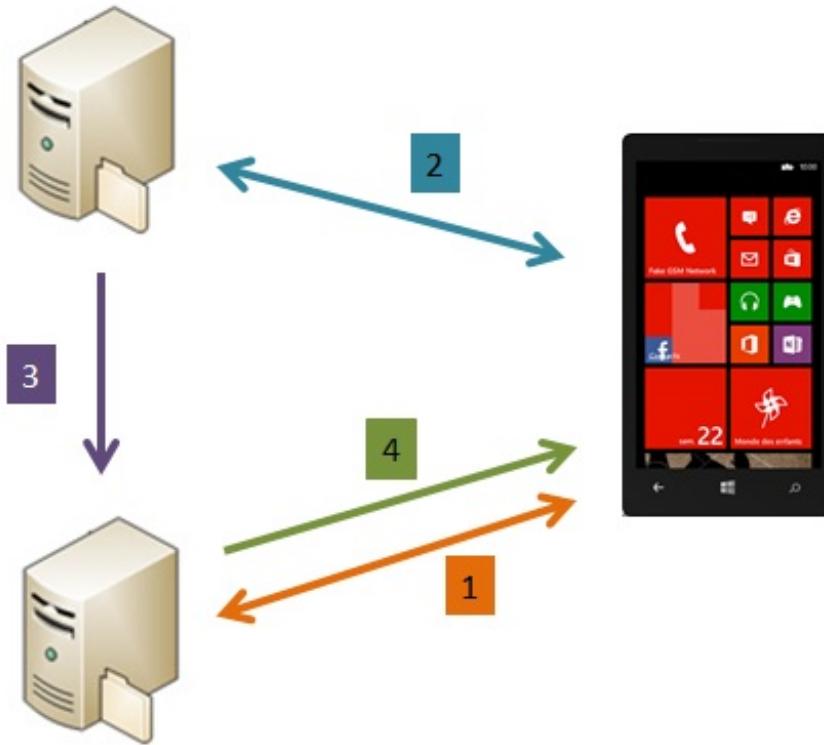
- Cela peut être un message système qui apparaît même si l'application n'a pas démarrée. Il apparaît en haut du téléphone et si l'on clique dessus alors l'application démarre. Il s'agit des notifications **Toast**.
- Cela peut être un message qui va modifier l'apparence d'une tuile. Ce message a peu d'intérêt depuis Mango car les tuiles peuvent être pilotées par code facilement, comme nous l'avons vu précédemment, mais il peut quand même trouver son intérêt. Il s'agit des notifications **Tile**.
- Enfin, cela peut être un message qui apparaît alors que l'application est en cours d'utilisation. Si l'application n'est pas lancée, alors le message est ignoré. Il s'agit des notifications **Raw**.

On peut résumer l'architecture avec le schéma suivant, qui comprend 4 grandes étapes :

- 1. La connexion s'établit et une URL est créée pour la communication.
- 2. Communication de l'URL.
- 3. Envoi du message.
- 4. Envoi du message sur le téléphone.

Service de notification

créé par le développeur



Architecture des notifications

Service de notification Microsoft

Le principe de création du serveur de notification

Je vais décrire ici le principe de création du serveur de notification sachant que nous n'allons pas le réaliser car il sort de la portée de ce cours. Mais ne vous inquiétez pas, dans ce chapitre nous ferons comme si le serveur avait été créé et nous nous occuperons de ce qui se passe juste après un envoi de notification.

Ce serveur de notification consiste en général en un site web ou des services web, car ceux-ci doivent être toujours disponibles en ligne, à n'importe quel moment. Celui-ci peut-être bien sûr un serveur à base de produits Microsoft (avec un logiciel de serveur IIS et des composants WCF), mais il peut tout aussi bien être en PHP, Ruby ou ce que vous maîtrisez.

Le principe est de fournir une URL permettant d'enregistrer un identifiant unique de téléphone ainsi que le canal (en fait, seul le canal est vraiment indispensable). Un téléphone souhaitant recevoir des notifications va s'enregistrer auprès du service de notification de Microsoft et recevra une URL identifiant le canal en retour. L'URL est de la forme :

Code : Autre

```
http://db3.notify.live.net/throttledthirdparty/01.00/AE17kNJdrSwXXXXXXXXXXXXXX
```

Le téléphone appelle ensuite le serveur de notification pour lui fournir cette URL ainsi que son identifiant unique. Le serveur devra être capable de stocker cet identifiant ainsi que l'URL et devra également permettre de modifier l'URL si l'identifiant est déjà stocké. Le serveur va donc servir à stocker une liste d'identifiants associés à des URL et à la maintenir à jour. Puis, lorsque nous voudrons envoyer une notification à tous les utilisateurs de notre application, il suffira de parcourir cette liste et d'envoyer un message au serveur de Microsoft, via l'URL du canal, contenant le message. C'est ensuite le serveur de Microsoft qui s'occupe d'envoyer la notification à chaque téléphone, grâce à ce canal.

Et voilà, en fait c'est très simple.

Dans l'idéal, le serveur devra également permettre de désabonner un téléphone ne souhaitant plus recevoir de notifications, car il est indispensable que votre application propose à son utilisateur un moyen d'arrêter de recevoir des notifications. De même,

lorsque le serveur de notification envoie la requête à Microsoft permettant l'envoi du message, ce dernier lui répond en lui indiquant notamment si le téléphone est injoignable. Cela pourra correspondre au fait que l'application est désinstallée par exemple. À ce moment-là, il faudra mettre à jour la liste des identifiants et des URL afin de supprimer le couple identifiant-canal, pour ne plus tenter de lui envoyer de notifications.

N'hésitez pas à créer votre propre serveur de notification, avec le langage de votre choix, et pourquoi pas à proposer les sources d'un tel serveur afin que chacun puisse s'en servir pour héberger son serveur de notification.

Les différents messages de notifications

Nous avons vu les différents types de notification :

- Toast,
- Tile,
- Raw.

Pour déclencher une telle notification, il faut envoyer un message au serveur de Microsoft contenant du XML, ou du texte simple, en fonction du type de notification. Ce message devra être envoyé avec la méthode POST, dans un format XML (content-type="text/xml") et contenir des en-têtes HTTP personnalisées. Il y a plusieurs éléments à passer dans les en-têtes HTTP, plus ou moins obligatoires. Par exemple, il est possible et facultatif de fournir un identifiant de message via l'en-tête HTTP MessageID. En revanche, nous devons obligatoirement fournir dans l'en-tête le type de notification. Cela se fait avec l'attribut X-Windows-Phone-Target. Par exemple, on ajoute le code ci-dessous dans l'en-tête pour une notification de type toast :

Code : Autre

```
X-WindowsPhone-Target:type
```

Le message de notification envoyé au serveur de Microsoft peut être de trois types différents :

- Message de type toast, pour les notifications toast.
- Message de type token, pour les notifications tile.
- Message de type raw, pour les notifications raw.



Remarque : si le type de notification n'est pas précisé, alors la notification sera de type raw.

Nous devons également indiquer le délai d'envoi, qui varie selon le type de notification comme le montre le tableau suivant.

| - | Envoi immédiat | Délai de 450 secondes (7 minutes et demie) | Délai de 15 minutes |
|-------|----------------|--|---------------------|
| Tile | 1 | 11 | 21 |
| Toast | 2 | 12 | 22 |
| Raw | 3 | 13 | 23 |

Par exemple, pour envoyer une notification **Toast** immédiatement, je devrais avoir l'en-tête HTTP suivante :

Code : Autre

```
X-WindowsPhone-Target:toast  
X-NotificationClass:2
```

Ensuite, reste le corps du message.

- Pour envoyer un message de type **raw**, c'est très simple. Il suffit d'envoyer le message tel quel, pas besoin d'XML.
- Pour envoyer un message de type **toast**, nous aurons besoin de démarrer l'application dans un contexte particulier. Pour

cela, il nous faudra exécuter une requête query string dans une page XAML de l'application. Il faudra envoyer le XML suivant avec cette page XAML comme Paramètre :

Code : XML

```
<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
  <wp:Toast>
    <wp:Text1>Titre</wp:Text1>
    <wp:Text2>Sous titre</wp:Text2>
    <wp:Param>paramètre</wp:Param>
  </wp:Toast>
</wp:Notification>
```

- Enfin, pour envoyer un message de type **tile**, nous aurons le XML suivant :

Code : XML

```
<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
  <wp:Tile>
    <wp:BackgroundImage>Url image de fond</wp:BackgroundImage>
    <wp:Count>Numéro en haut à droite</wp:Count>
    <wp:Title>Titre</wp:Title>
    <wp:BackBackgroundImage>Url image de fond du dos de la
tuile</wp:BackBackgroundImage>
    <wp:BackTitle>Titre du dos de la tuile</wp:BackTitle>
    <wp:BackContent>Contenu du dos de la tuile</wp:BackContent>
  </wp:Tile>
</wp:Notification>
```

Nous allons voir dans le chapitre suivant comment envoyer ces beaux messages et surtout ce qu'il faut faire côté Windows Phone pour créer un canal et réagir aux notifications.



Note : Si vous devez mettre des caractères spéciaux dans vos messages, alors vous devez les encoder en XML, par exemple < devient <

Création du client Windows Phone recevant la notification

Avant toute chose, vous devez déclarer que votre application utilise des notifications. Cela se fait en utilisant la capacité ID_CAP_PUSH_NOTIFICATION que nous allons voir à l'œuvre dans cette section.
Il faut ensuite créer ou utiliser un canal depuis notre application Windows Phone. Un canal est identifié par son nom. La première chose est de tenter de le récupérer s'il existe déjà, sinon il faut le créer.
Voici comment utiliser un canal pour recevoir une notification **Toast** :

Code : C#

```
public MainPage()
{
    InitializeComponent();

    string nomCanal = "TestCanalOpenClassroomsNico";
    HttpNotificationChannel canal =
    HttpNotificationChannel.Find(nomCanal);

    if (canal == null)
    {
        // si le canal n'est pas trouvé, on le crée
        canal = new HttpNotificationChannel(nomCanal);

        canal.ChannelUriUpdated += canal_ChannelUriUpdated;
        canal.ErrorOccurred += canal_ErrorOccurred;
```

```
// On s'abonne à cet événement si on veut être averti de la
reception de la notification toast lorsque l'application
// est ouverte, sinon on ne la reçoit pas
canal.ShellToastNotificationReceived +=
canal_ShellToastNotificationReceived;

canal.Open();

// Permet de déclarer le canal pour recevoir les toasts
canal.BindToShellToast();
}

else
{
    // le canal est déjà créé, on l'utilise
    canal.ChannelUriUpdated +=canal_ChannelUriUpdated;
    canal.ErrorOccurred +=canal_ErrorOccurred;
    canal.ShellToastNotificationReceived +=
canal_ShellToastNotificationReceived;

    // On affiche le canal dans la console de sortie pour
    // pouvoir l'exploiter ensuite dans notre appli de test d'envoi
    // normalement, on l'aurait envoyé à notre serveur de
    // notification, avec l'identifiant unique du téléphone
    Debug.WriteLine(canal.ChannelUri.ToString());
}
}
```

Si le canal existe déjà, alors on obtient son URL qui normalement devra être envoyée à notre serveur de notification. Ici, je vais simplement l'afficher dans la console de sortie pour éviter de devoir créer un serveur de notification. Je vais ainsi pouvoir récupérer l'URL et l'utiliser dans une application tierce qui enverra la requête au serveur de notification de Microsoft. Pour l'instant, implémentez les événements ainsi :

Code : C#

```
private void canal_ShellToastNotificationReceived(object sender,
NotificationEventArgs e)
{
    // optionnel, cet événement est levé quand l'appli est ouverte
    // et qu'on reçoit la notif
}

private void canal_ErrorOccurred(object sender,
NotificationChannelErrorEventArgs e)
{
    Dispatcher.BeginInvoke(() => MessageBox.Show(string.Format("Une
erreur est survenue {0}, {1} ({2}) {3}", e.ErrorType, e.Message,
e.ErrorCode, e.ErrorAdditionalData)));
}

private void canal_ChannelUriUpdated(object sender,
NotificationChannelUriEventArgs e)
{
    // On affiche le canal dans la console de sortie pour pouvoir
    // l'exploiter ensuite dans notre appli de test d'envoi
    // normalement, on l'aurait envoyé à notre serveur de
    // notification, avec l'identifiant unique du téléphone
    Debug.WriteLine(e.ChannelUri.ToString());
}
```

Vous aurez besoin de :

Code : C#

```
using Microsoft.Phone.Notification;
```

Puis démarrez l'application en debug, l'URL s'affiche dans la fenêtre de sortie de Visual Studio comme dans la figure suivante.

The screenshot shows the Visual Studio interface with the MainPage.xaml.cs file open. In the code editor, two event handlers are shown:

```

private void canal_ErrorOccurred(object sender, NotificationChannelErrorEventArgs e)
{
    Dispatcher.BeginInvoke(() => MessageBox.Show(string.Format("Une erreur est survenue {0}, {1} ({2}) {3}", e.ErrorType, e.ErrorCode, e.ErrorMessage)));
}

private void canal_ChannelUriUpdated(object sender, NotificationChannelUriEventArgs e)
{
    // On affiche le canal dans la console de sortie pour pouvoir l'exploiter ensuite dans notre appli de test d'envoi
    // normalement, on l'aurait envoyé à notre serveur de notification, avec l'identifiant unique du téléphone
    Debug.WriteLine(e.ChannelUri.ToString());
}

```

In the Output window (Sortie), the URL is displayed:

```

Afficher la sortie à partir de: Déboguer
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\windows\system32\System.Xml.dll'. Chargé
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\Data\Programs\{62D4DAE7-1C4F-485B-BF7C-8E
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\windows\system32\Microsoft.Phone.ni.dll'.
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\windows\system32\Microsoft.Phone.Interop.
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\windows\system32\en-US\mscorlib.debug.res
'taskHost.exe' (CLR C:\windows\system32\coreclr.dll: Silverlight AppDomain) : Chargé 'C:\windows\system32\System.Core.dll'. Chargé
http://db3.notify.live.net/throttledthirdparty/01.00/AAHmJDXpe47LT10ct6NNXXXXXX

```

The URL `http://db3.notify.live.net/throttledthirdparty/01.00/AAHmJDXpe47LT10ct6NNXXXXXX` is highlighted with a red box.

Url du canal dans la fenêtre de sortie

Je l'ai un peu masquée ici, mais elle a la forme :

Code : Autre

```
http://db3.notify.live.net/throttledthirdparty/01.00/AAHmJDXpe47LT10ct6NNXXXXXX
```

Sauvegardez cette URL dans un coin, nous allons en avoir besoin. Nous pouvons arrêter l'application mais sans arrêter l'émulateur. Le canal est créé. Remarquez que si vous redémarrez l'application, nous récupérons le canal déjà créé et obtenons à nouveau la même URL.

Si vous vous souvenez de ce dont nous avons parlé juste avant, c'est cette URL qui devra être utilisée pour envoyer un message au serveur de notification de Microsoft.

Par souci de simplicité, ici je vais utiliser directement l'URL pour envoyer ma requête en POST, mais vous aurez compris que le rôle de notre serveur est de stocker ces URL dans une liste et d'envoyer le message à toutes les URL que nous avons stockées, correspondant aux utilisateurs qui souhaitent recevoir nos notifications.

Nous devons donc envoyer désormais une requête en POST à cette url, pour cela nous allons créer une petite application console grâce à Visual Studio Express 2012 pour Windows Desktop, que vous pouvez [télécharger gratuitement à cet emplacement](#) (je vous fait grâce de l'installation qui ne devrait pas poser de problème).



Note : Vous pouvez bien sûr implémenter un vrai serveur de notification si vous le souhaitez, mais pour la démonstration ce sera plus simple d'envoyer un simple message en POST ainsi.

Ici, vu que je me suis abonné aux notifications **toast** (grâce à `canal.BindToShellToast()`), il faut que j'envoie un message toast, par exemple :

Code : XML

```
<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
    <wp:Toast>
        <wp:Text1>Texte 1 youpi</wp:Text1>
        <wp:Text2>Texte 2 joie</wp:Text2>
        <wp:Param>/ MainPage.xaml?cle=parametre</wp:Param>
    </wp:Toast>
</wp:Notification>
```

Le principe est donc de faire une requête POST, avec les bons headers, contenant le message de type **toast**, et de le poster à l'url récupérée. Voici le code C# permettant de réaliser ceci dans une application console :

Code : C#

```
try
{
    // url où envoyer le message
    // normalement, il faudrait récupérer la liste des urls stockées sur le serveur
    // et envoyer le message à toutes ces urls
    string urlOuEnvoyerMessage =
"http://db3.notify.live.net/throttledthirdparty/01.00/AAHmJDXpe47LTI0ct6NNXXXXX"

    HttpWebRequest requete = (HttpWebRequest)WebRequest.Create(urlOuEnvoyerMessage);
    requete.Method = "POST"; // envoi en POST
    string messageToastAEnvoyer = @"<?xml version=""1.0"" encoding=""utf-8""?>
        <wp:Notification xmlns:wp=""WPNotification"">
            <wp:Toast>
                <wp:Text1>Texte 1 youpi</wp:Text1>
                <wp:Text2>Texte 2 joie</wp:Text2>
                <wp:Param>/ MainPage.xaml?cle=parametre</wp:Param>
            </wp:Toast>
        </wp:Notification>";

    byte[] message = Encoding.Default.GetBytes(messageToastAEnvoyer);
    requete.ContentLength = message.Length;
    requete.ContentType = "text/xml";
    requete.Headers.Add("X-WindowsPhone-Target", "toast"); // notification de type toast
    requete.Headers.Add("X-NotificationClass", "2"); // envoi immédiat

    using (Stream requestStream = requete.GetRequestStream())
    {
        requestStream.Write(message, 0, message.Length);
    }

    // envoi de la notification, et récupération du retour
    HttpWebResponse response = (HttpWebResponse)requete.GetResponse();
    string statutNotification = response.Headers["X-NotificationStatus"];
    string statutCanal = response.Headers["X-SubscriptionStatus"];
    string statutMateriel = response.Headers["X-DeviceConnectionStatus"];

    // affiche le résultat de la requête
    Console.WriteLine(statutNotification + " | " + statutMateriel + " | " + statutCanal);

    // gestion d'erreur ultra simplifiée :
    if (string.Compare(statutNotification, "Dropped", StringComparison.CurrentCulture) == 0)
    {
        // il faut arrêter de tenter d'envoyer des messages à ce téléphone
    }
}
catch (Exception ex)
{
    Console.WriteLine("Erreur d'envoi : " + ex);
}
```

Et nous verrons la notification Toast s'afficher sur notre Windows Phone (voir figure suivante).



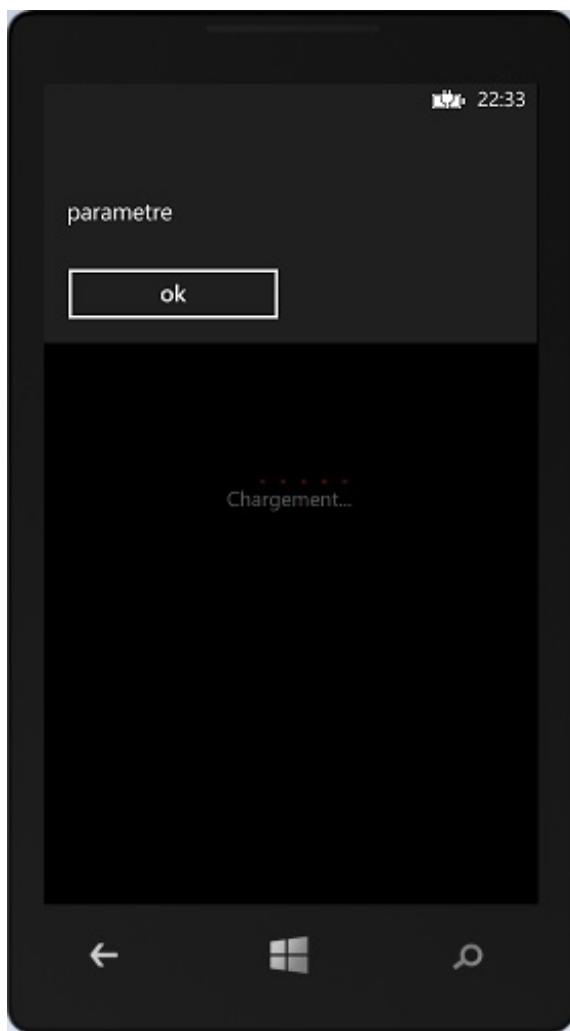
La notification Toast s'affiche dans l'émulateur

Et ce qui est intéressant, c'est que si l'on clique sur la notification toast, alors notre application se lance et navigue sur la page passée en paramètre. Ceci nous permet également de récupérer les paramètres de la query string (voir code et figure qui suivent).

Code : C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    string valeur;
    if (NavigationContext.QueryString.TryGetValue("cle", out valeur))
    {
        MessageBox.Show(valeur);
    }
    base.OnNavigatedTo(e);
}
```

Pour avoir :



Utilisation de la query string

Attention, il est important de traiter le retour du message en fonction des codes d'erreurs que nous obtenons en réponse de l'envoi de la notification. Les erreurs peuvent être de diverses sortes. On peut par exemple détecter si l'utilisateur a désinstallé l'application, cela voudra dire qu'il n'est plus nécessaire de lui envoyer de message et qu'il va falloir enlever cette url de la liste de notre serveur de notification. C'est ce que j'ai fait dans l'exemple de code précédent, de manière minimale je le reconnaiss. Vous pouvez consulter les codes d'erreurs à cet emplacement.

À noter que si l'application est ouverte, alors le message toast ne s'affiche pas en haut de l'écran. L'application peut cependant être notifiée de la réception d'un message toast en s'abonnant à l'événement ShellToastNotificationReceived et ainsi faire ce qu'elle désire, par exemple afficher le message :

Code : C#

```
private void canal_ShellToastNotificationReceived(object sender,  
NotificationEventArgs e)  
{  
    // optionnel, cet événement est levé quand l'appli est ouverte  
    et qu'on reçoit la notif  
    string message = e.Collection["wp:Text1"] +  
e.Collection["wp:Text2"];  
    Dispatcher.BeginInvoke(() => MessageBox.Show(message));  
}
```

Ici, nous utilisons simplement la boîte de message MessageBox.Show pour afficher le message de la notification reçue. Vous pouvez en faire ce que vous voulez, l'afficher dans un TextBlock, ou juste traiter le message.

Pour les autres types de notifications, le principe est le même. Veuillez toujours à faire attention au type de notifications que

vous envoyez dans les en-têtes HTTP, ainsi qu'au délai d'envoi. De même, pour qu'un téléphone puisse recevoir les notifications d'un certain type, pensez à déclarer le canal comme tel. Par exemple, nous avons vu que pour recevoir des notifications **toasts**, nous utilisions :

Code : C#

```
canal.BindToShellToast();
```

Pour les notifications **tiles**, il faudra utiliser :

Code : C#

```
canal.BindToShellTile();
```

Pour les notifications raw, il n'y a rien de spécial à faire sur le canal. Il faudra par contre s'abonner à l'événement `HttpNotificationReceived` :

Code : C#

```
[...]
canal.HttpNotificationReceived += canal_HttpNotificationReceived;
[...]
private void canal_HttpNotificationReceived(object sender,
HttpNotificationEventArgs e)
{
    using (System.IO.StreamReader reader = new
System.IO.StreamReader(e.Notification.Body))
    {
        string message = reader.ReadToEnd();
        Dispatcher.BeginInvoke(() => MessageBox.Show("Notification
raw reçue : " + message));
    }
}
```



Remarque : Il est demandé de pouvoir désactiver les notifications depuis l'application. Pensez à rajouter une interface qui permet à l'utilisateur de se désabonner.

- Les notifications permettent d'envoyer un message ou une demande de mise à jour de tuile.
- La notification de type toast peut-être reçue même si l'application n'est pas démarrée.
- Le développeur aura besoin de créer un serveur de notification, accessible à tout moment, afin de maintenir une liste de canaux de communication avec les téléphones intéressés pour recevoir des notifications.

TP : Améliorer l'application météo avec géolocalisation et tuiles

Que de nouvelles fonctionnalités découvertes. C'est le moment rêvé pour les mettre en pratique et améliorer notre superbe application météo de la partie précédente.

Nous allons jouer avec les tuiles et la géolocalisation. Vous êtes prêt ?
Alors c'est parti !

Instructions pour réaliser le TP

Franchement, une application météo où il faut absolument saisir le nom de la ville où nous nous trouvons, vous conviendrez avec moi que ce n'est pas terrible ! Alors que nos téléphones possèdent un GPS intégré... Nous allons donc proposer à l'utilisateur d'afficher la météo correspondant à sa position.

Donc première chose à faire, rajouter un bouton dans la barre d'application qui affichera la météo à votre position. Ensuite, imaginons que je veuille rapidement connaître la météo de Paris et de Toulouse, histoire de savoir où il vaut mieux que je passe le week-end. Comme c'est quelque chose que je fais régulièrement, j'ai besoin de la possibilité de rajouter un raccourci vers ces villes. Mon application va donc permettre d'épingler des villes depuis la page de choix de villes. Si la ville a déjà été choisie, il faut que le bouton soit grisé. Bien sûr, le démarrage de l'application via des tuiles secondaires affichera directement la météo de la ville choisie. La tuile contiendra forcément le nom de la ville. Je vous laisse libre de choisir le modèle de tuile qui vous fait plaisir.

Voici un programme intéressant.

Attention avant de vous lancer, il va vous manquer quelque chose.

Le GPS fournit une position avec une longitude et une latitude alors que nous interrogions le service web de météo avec un nom de ville en entrée. Donc, soit il vous faut donc une solution pour transformer une position GPS en un nom de ville (on appelle cela du géocodage inversé), soit il faudrait que le service web accepte d'utiliser des coordonnées GPS.

Et vous savez quoi ? Il l'accepte. C'est la classe !

C'est le même principe, il suffit de remplacer le nom de la ville par les coordonnées sous la forme ci-dessous, avec la première coordonnée, la latitude, et la seconde, la longitude.

Code : Autre

```
http://free.worldweatheronline.com/feed/weather.ashx?q=44.839073,-0.579113&format=json&num_of_days=5&key=MA_CLE_API
```

En résumé, je vous propose de réaliser les éléments suivants :

1. Utiliser le GPS pour permettre de vous géolocaliser,
2. Rajouter un bouton dans la barre d'application pour afficher la météo à votre position,
3. Rajouter deux tuiles de raccourci affichant la météo à Paris et à Toulouse.

Voilà, vous avez tout. À vous de jouer.

Correction

Je suis certain que vous vous en êtes très bien sorti avec ce petit exercice sympathique.

Voici ma correction, je ne vais pas tout re-détailler mais simplement les choses qui ont changé par rapport au TP précédent d'application météo.

1. Utiliser le GPS pour vous géolocaliser

Nous avons premièrement besoin d'utiliser le GPS afin de nous géolocaliser. Vous devez utiliser la classe Geolocator, via par exemple une variable privée :

Code : C#

```
private Geolocator geolocator;
```

Vous initialisez cette variable geolocator dans le constructeur :

Code : C#

```
public MainPage()
{
    InitializeComponent();
    geolocator = new Geolocator();
    DataContext = this;
}
```



Ici, nul besoin d'avoir une précision importante, la localisation approximative suffit.

2. Rajouter un bouton d'application pour afficher la météo à votre position

On rajoute également un bouton dans la barre d'application. Il possède une image (gps.png) et un texte (Ma position), et appelle une méthode (Position_Click) lorsqu'on clique dessus :

Code : XML

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True">
        <shell:ApplicationBarIconButton
            IconUri="/Assets/Icones/gps.png" Text="Ma position"
            Click="Position_Click"/>
        <shell:ApplicationBarIconButton
            IconUri="/Assets/Icones/add.png" Text="Ajouter"
            Click="Ajouter_Click"/>
        <shell:ApplicationBarIconButton
            IconUri="/Assets/Icones/feature.settings.png" Text="Choisir"
            Click="Choisir_Click"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Pour la peine, je me suis réalisé une petite image pour symboliser le GPS. Vous ne manquerez pas d'admirer ma maîtrise de Paint (image dans le chapitre suivant) ! N'oubliez pas de changer les propriétés de l'image pour mettre l'action de génération à « Contenu », et la propriété « Copier dans le répertoire » à « Copier si plus récent ».

Lorsque l'on clique sur le bouton, une méthode est appelée pour démarrer le service de localisation :

Code : C#

```
private async void Position_Click(object sender, EventArgs e)
{
    ChargementEnCours = true;
    try
    {
        Geoposition geoposition = await
geolocator.GetGeopositionAsync(TimeSpan.FromMinutes(5),
TimeSpan.FromSeconds(10));
        string position =
geoposition.Coordinate.Latitude.ToString(NumberFormatInfo.InvariantInfo) +
"," +
geoposition.Coordinate.Longitude.ToString(NumberFormatInfo.InvariantInfo);
        WebClient client = new WebClient();
        string resultatMeteo = await client.DownloadStringTaskAsync(new
Uri(string.Format("http://free.worldweatheronline.com/feed/weather.ashx?
q={0}&format=json&num_of_days=5&key=MA_CLE_API", position,
UriKind.Absolute)));
        TraiteResultats(resultatMeteo);
        NomVille = position;
    }
    catch (Exception)
    {
        MessageBox.Show("Vous devez activer le GPS pour pouvoir utiliser
"
```

```

        cette fonctionnalité");
        ChargementEnCours = false;
    }
}

```



N'oubliez pas d'activer la capacité ID_CAP_LOCATION, qui sert à autoriser l'utilisation du GPS.

Notez la présence du mot clé `async` dans la signature de la méthode. Il y a également un petit `try/catch` pour vérifier que le GPS est utilisable. Et puis j'appelle le service de localisation pour obtenir la position de l'utilisateur, que je fournis au service web. La méthode `TraiteResultats` contient la logique d'affichage qui a été factorisée, car utilisée à deux endroits :

Code : C#

```

private void TraiteResultats(string resultats)
{
    RootObject resultat =
JsonConvert.DeserializeObject<RootObject>(resultats);
    List<Meteo> liste = new List<Meteo>();
    foreach (Weather temps in resultat.data.weather.OrderBy(w =>
w.date))
    {
        Meteo meteo = new Meteo { TemperatureMax = temps.tempMaxC +
" °C", TemperatureMin = temps.tempMinC + " °C" };
        DateTime date;
        if (DateTime.TryParse(temps.date, out date))
        {
            meteo.Date = date.ToString("ddd dd MMMM");
            meteo.Temps = GetTemps(temps.weatherCode);
            WeatherIconUrl2 url =
tempo.weatherIconUrl.FirstOrDefault();
            if (url != null)
            {
                meteo.Url = new Uri(url.value, UriKind.Absolute);
            }
        }
        liste.Add(meteo);
    }
    ListeMeteo = liste;
    ChargementEnCours = false;
}

```

3. Rajouter deux tuiles secondaires affichant la météo à Paris et à Toulouse

Reste maintenant à gérer l'épinglage des villes. Pour ajouter un bouton, modifions la page XAML `ChoisirVille` que nous avons construite dans le TP précédent :

Code : XML

```

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <toolkit:ListPicker x:Name="Liste" ItemsSource="{Binding
ListeVilles}" Header="Ville choisie :"
CacheMode="BitmapCache">
    </toolkit:ListPicker>
    <Button Grid.Row="1" VerticalAlignment="Bottom"
Content="Epingler" x:Name="BoutonEpingle" Tap="BoutonEpingle_Tap" />

```

Le clic sur le bouton ajoutera une tuile secondaire :

Code : C#

```
private void BoutonEpingle_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    string ville =
(string)IsolatedStorageSettings.ApplicationSettings["DerniereVille"];
    ShellTile tuileVille = ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("ville=" + ville));
    if (tuileVille == null)
    {
        FlipTileData tuile = new FlipTileData
        {
            Title = "Météo " + ville,
        };

        ShellTile.Create(new Uri("/ MainPage.xaml?ville=" + ville,
UriKind.Relative), tuile, false);
        BoutonEpingle.IsEnabled = false;
    }
}
```

Bien sûr, dans la query string, on indique le nom de la ville afin de pouvoir l'exploiter au démarrage de l'application. Dans l'événement permettant de choisir sa ville, on en profite pour désactiver les boutons des villes qui ont déjà une tuile de raccourci.

Code : C#

```
private void Liste_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (Liste.SelectedItem != null)
    {
        IsolatedStorageSettings.ApplicationSettings["DerniereVille"] =
(string)Liste.SelectedItem;
        ShellTile tuileVille =
ShellTile.ActiveTiles.FirstOrDefault(elt =>
elt.NavigationUri.ToString().Contains("ville=" +
(string)Liste.SelectedItem));
        BoutonEpingle.IsEnabled = tuileVille == null;
    }
}
```

Il ne reste plus qu'à exploiter l'information passée en query string au démarrage de l'application afin de charger la météo de la bonne ville. On peut le faire avec la méthode `OnNavigatedTo` par exemple :

Code : C#

```
protected async override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    string ville;
    if (NavigationContext.QueryString.TryGetValue("ville", out ville))
    {
        NavigationContext.QueryString.Remove("ville");
        IsolatedStorageSettings.ApplicationSettings["DerniereVille"] =
```

```

ville;
}

if
(IsolatedStorageSettings.ApplicationSettings.Contains("DerniereVille"))
{
    Information.Visibility = Visibility.Collapsed;
    ChargementEnCours = true;
    NomVille =
(string)IsolatedStorageSettings.ApplicationSettings["DerniereVille"];
    WebClient client = new WebClient();
    try
    {
        ChargementEnCours = false;
        string resultatMeteo = await
client.DownloadStringTaskAsync(new
Uri(string.Format("http://free.worldweatheronline.com/feed/weather.ashx?
q={0}&format=json&num_of_days=5&key=MA_CLE_API", NomVille.Replace(' ', '+'))), UriKind.Absolute));
        TraiteResultats(resultatMeteo);
    }
    catch (Exception)
    {
        MessageBox.Show("Impossible de récupérer les informations de
météo, vérifiez votre connexion internet");
    }
}
else
    Information.Visibility = Visibility.Visible;
base.OnNavigatedTo(e);
}

```



Remarquez qu'une fois que j'ai extrait l'éventuelle ville passée en paramètre dans la query string, je la supprime afin que l'application ne soit pas bloquée sur cette ville.

Et le tour est joué. Voici une belle application météo qui exploite les infos de géolocalisation et qui nous permet même d'avoir des raccourcis vers nos villes favorites. Pas mal comme TP, non ?

Aller plus loin

Vous aurez remarqué que lorsqu'on consulte la météo par rapport à ses coordonnées GPS, on ne dispose plus du nom de la ville. En général, ce n'est pas trop grave car nous savons où nous sommes ... mais on pourrait améliorer notre application pour afficher le nom de la ville où nous nous trouvons, et pas les coordonnées GPS.

Il suffit d'utiliser le géocodage inversé — j'en ai parlé en introduction du TP — et il se trouve qu'il existe des services gratuits de géocodage inversé. Prenons par exemple celui de Google qui est assez facile à utiliser : si vous naviguez sur <http://maps.googleapis.com/maps/api/ge...3&sensor=true> vous pourrez obtenir du code JSON qui nous indique notamment dans quelle ville nous nous trouvons. Il n'y a plus qu'à modifier notre code pour faire l'appel à ce service web et nous pourrons obtenir la ville où nous sommes, ce qui nous permettra d'ailleurs de la stocker dans le dossier local :

Code : C#

```

private async void Position_Click(object sender, EventArgs e)
{
    ChargementEnCours = true;
    WebClient client = new WebClient();
    string position;
    try
    {
        Geoposition geoposition = await
geolocator.GetGeopositionAsync(TimeSpan.FromMinutes(5),
TimeSpan.FromSeconds(10));
        position =
geoposition.Coordinate.Latitude.ToString(NumberFormatInfo.InvariantInfo) +
", " +

```

```
geoposition.Coordinate.Longitude.ToString(NumberFormatInfo.InvariantInfo);
        string resultatMeteo = await client.DownloadStringTaskAsync(new
Uri(string.Format("http://free.worldweatheronline.com/feed/weather.ashx?
q={0}&format=json&num_of_days=5&key=MA_CLE_API", position,
UriKind.Absolute)));
        TraiteResultats(resultatMeteo);
    }
    catch (Exception)
    {
        MessageBox.Show("Vous devez activer le GPS pour pouvoir utiliser
        cette fonctionnalité");
        ChargementEnCours = false;
        return;
    }
    try
    {
        string json = await client.DownloadStringTaskAsync(new
Uri(string.Format("http://maps.googleapis.com/maps/api/geocode/json?
latlng={0}&sensor=true", position), UriKind.Absolute));

        GeocodageInverse geocodageInverse =
JsonConvert.DeserializeObject<GeocodageInverse>(json);
        if (geocodageInverse.status == "OK")
        {
            AddressComponent adresse = (from result in
geocodageInverse.results
            from addressComponent in
result.address_components
            from type in
addressComponent.types
            where type == "locality"
            select
addressComponent).FirstOrDefault();
            if (adresse == null)
            {
                MessageBox.Show("Impossible de déterminer le géocodage
inversé");
                ChargementEnCours = false;
            }
            else
            {
                NomVille = adresse.long_name;

IsolatedStorageSettings.ApplicationSettings["DerniereVille"] =
adresse.long_name;
            }
        }
        catch (Exception)
        {
            MessageBox.Show("Impossible de déterminer le géocodage inversé");
        }
    }
}
```



Vous commencez à avoir l'habitude du langage JSON maintenant. Bien sûr, vous allez devoir générer les classes correspondantes mais faites attention, car vous avez déjà un objet RootObject qui a été généré juste avant... Il faudra donc le renommer.

Le résultat est montré dans la figure suivante.



Géocodage inversé sur l'application météo

Moi, je suis fan ... et vous ?

Une application fluide = une application propre !

La performance est un point crucial à prendre en compte lors du développement d'applications pour Windows Phone. Les ressources du téléphone sont beaucoup moins importantes que nos PC de développement. Aussi, il est important d'y faire attention afin de faire en sorte que son application soit réactive et ne paraisse pas bloquée. Vous devez bien sûr veiller à ce que vos algorithmes soient un minimum optimisés et ne pas vous dire « oh, ce n'est pas grave, le processeur du téléphone va m'optimiser tout ça... ». De même, vous devez comprendre le mécanisme des threads pour pouvoir tirer le meilleur de votre Windows Phone et obtenir l'application la plus fluide possible.

Un thread, c'est quoi ?

On peut traduire Thread par « fil d'exécution » ou « tâche ». Il s'agit d'un processus qui peut exécuter du code dans notre application, en l'occurrence le thread principal est celui que nous utilisons pour exécuter notre code C#. Il y aussi le thread principal d'interface, que l'on nomme *UI Thread*. Windows Phone possède un autre thread en relai du principal, c'est le thread de composition, appelé *Composition Thread* (ou *Compositor Thread*).

D'autres threads sont à notre disposition, ce sont des threads qui tournent en arrière-plan, de manière asynchrone. Nous nous en sommes déjà servis sans le savoir en utilisant la programmation asynchrone, par exemple lorsque nous téléchargeons des données avec les classes WebClient ou HttpWebRequest. Ces opérations asynchrones se font sur un thread secondaire.

Le thread d'interface

Le thread d'interface (*UI Thread*) va servir à mettre à jour l'interface ; ce qu'on voit à l'écran. En l'occurrence, il va servir à créer les objets depuis le XAML et dessiner tous les contrôles. Il gère également toutes les interactions avec l'utilisateur, notamment tous les touches. Il est donc très important que ce thread soit le moins chargé possible afin que l'application reste réactive, notamment aux actions de l'utilisateur. Si ce thread contient une longue série de codes à exécuter, alors l'interface sera bloquée et l'utilisateur ne pourra plus rien faire, ce qui est fortement déplaisant et risque de le faire très vite désinstaller votre application... Essayez plutôt de répartir les tâches, en imaginant que vous avez deux boutons dans votre page : l'un qui fait une action longue et l'autre qui affiche simplement un message dans une boîte. Voici le code de la boîte contenant les deux boutons :

Code : XML

```
<StackPanel>
    <Button Content="Lancer le calcul" Tap="Button_Tap" />
    <Button Content="Cliquez-moi" Tap="Button_Tap_1" />
</StackPanel>
```

Voici le code-behind des deux boutons :

Code : C#

```
private void Button_Tap(object sender, RoutedEventArgs e)
{
    List<int> nombrePremiers = new List<int>();
    for (int i = 0; i < 2000000; i++)
    {
        if (EstNombrePremier(i))
            nombrePremiers.Add(i);
    }
}

private void Button_Tap_1(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Clic");
}

private bool EstNombrePremier(int nombre)
{
    if ((nombre % 2) == 0)
        return nombre == 2;
    int racine = (int)Math.Sqrt(nombre);
    for (int i = 3; i <= racine; i+=2)
    {
        if (nombre % i == 0)
            return false;
    }
}
```

```
    return nombre != 1;  
}
```

Le premier bouton permettra de déterminer les nombres premiers de 0 jusqu'à 2000000, le deuxième affichera un simple message, dans une boîte de dialogue.

Si vous démarrez l'application et cliquez sur le premier bouton, vous ne pourrez pas cliquer sur le deuxième bouton tant que le premier calcul n'est pas terminé. De plus, on voit à l'état du bouton que celui-ci reste cliqué tant que le traitement long n'est pas terminé.

Nous avons donc bloqué le thread UI en effectuant un calcul trop long. De ce fait, l'application n'est plus capable de traiter correctement les entrées utilisateurs, comme le clic sur le deuxième bouton, étant donné que le thread UI est surchargé par le long calcul.

Afin que le code soit plus court, nous allons remplacer le long calcul par une mise en veille du thread courant grâce à la méthode Thread.Sleep(), que nous retrouverons dans l'espace de noms :

Code : C#

```
using System.Threading;
```

Ceci nous permet de simuler un traitement long tout en économisant des lignes de codes, d'où le code-behind devient :

Code : C#

```
private void Button_Tap(object sender, RoutedEventArgs e)  
{  
    Thread.Sleep(TimeSpan.FromSeconds(4));  
}  
  
private void Button_Tap_1(object sender, RoutedEventArgs e)  
{  
    MessageBox.Show("Clic");  
}
```

Ceci me permet de simuler un traitement qui dure 4 secondes.

Ok, c'est bien beau, mais notre interface semble toujours bloquée et incapable de traiter le clic sur le deuxième bouton.

Utiliser un thread d'arrière-plan

Une solution pour résoudre ce problème serait d'utiliser un thread d'arrière-plan. Ce ne sera donc plus le thread UI qui va gérer le calcul mais un thread qui tourne en arrière-plan. C'est un peu le même principe qu'avec une opération asynchrone, comme lorsque nous effectuons un téléchargement, notre code qui s'exécute utilisera une partie de la mémoire pour fonctionner de manière plus ou moins parallèle au thread UI, ce qui lui permettra de continuer à pouvoir traiter les actions de l'utilisateur. On pourra utiliser pour cela la classe Thread. Il suffit d'inclure une méthode dans le thread (ici je passe une expression lambda, qui le met en pause pendant 4 secondes), d'appeler la méthode Start, et Windows Phone s'occupera d'exécuter notre méthode dans un thread d'arrière-plan. Cela donnera :

Code : C#

```
public partial class MainPage : PhoneApplicationPage  
{  
    public MainPage()  
    {  
        InitializeComponent();  
    }  
  
    private void Button_Tap(object sender, RoutedEventArgs e)  
    {  
        Thread thread = new Thread(() =>  
        Thread.Sleep(TimeSpan.FromSeconds(4));  
        thread.Start();  
    }  
}
```

```
        thread.Start();
    }

    private void Button_Tap_1(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Clic");
    }
}
```



Note : Cette façon de créer des threads n'est pas la plus optimale, nous verrons d'autres solutions pour créer des threads.

Maintenant, si vous démarrez votre application, vous pourrez voir que l'exécution du code long ne bloque plus le traitement du clic sur l'autre bouton. Ô joie, merci les threads !

En fait, notre code est un peu bête ! On fait des calculs mais ils ne nous servent à rien ici ... Je suis sûr que, comme moi, vous seriez très curieux de connaître le plus grand nombre premier inférieur à 10 millions, n'est-ce pas ? Ok, ressortons notre méthode, utilisons notre thread et affichons le résultat dans un TextBlock :

Code : XML

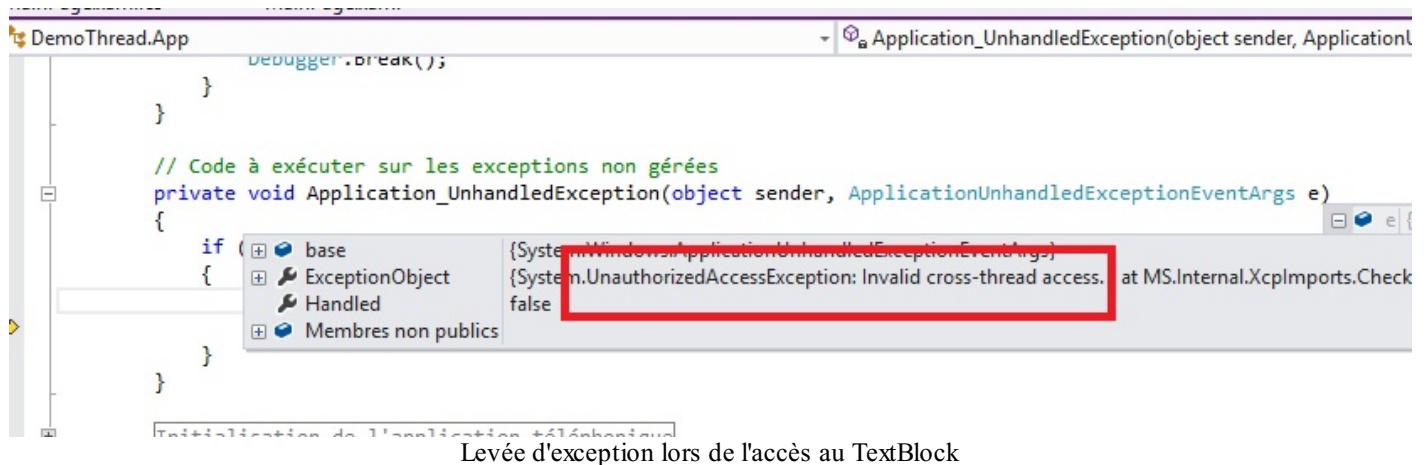
```
<StackPanel>
    <Button Content="Lancer le calcul" Tap="Button_Tap" />
    <Button Content="Cliquez-moi" Tap="Button_Tap_1" />
    <TextBlock x:Name="Resultat" />
</StackPanel>
```

Le calcul sera fait ainsi :

Code : C#

```
private void Button_Tap(object sender, RoutedEventArgs e)
{
    Thread thread = new Thread(() =>
    {
        int max = 0;
        for (int i = 0; i < 10000000; i++)
        {
            if (EstNombrePremier(i))
                max = i;
        }
        Resultat.Text = "Résultat : " + max;
    });
    thread.Start();
}
```

Sauf que, si vous démarrez l'application, que vous lancez le calcul, votre application va se planter avec une belle erreur du nom de UnauthorizedAccessException (que l'on peut apercevoir dans la figure qui suit).



```

    }
}

// Code à exécuter sur les exceptions non gérées
private void Application_UnhandledException(object sender, ApplicationUnhandledEventArgs e)
{
    if (base != null)
    {
        ExceptionObject = (System.Windows.ApplicationUnhandledExceptionEventArgs)e;
        {System.UnauthorizedAccessException: Invalid cross-thread access. at MS.Internal.XcpImports.CheckForIllegalCrossThreadCalls+Wrapper<T>.ctor(T)}
        Handled = false;
    }
}

```

Levée d'exception lors de l'accès au TextBlock



Pourquoi cette erreur ?

Pour une simple et bonne raison et je crois qu'on peut la mettre en avertissement :

 Seul le thread UI a le droit de mettre à jour l'interface. Si nous tentons de mettre à jour un élément de l'interface depuis un autre thread, comme un thread d'arrière-plan, nous aurons une erreur.

Utiliser le Dispatcher

Nous avons déjà rapidement vu comment résoudre ce problème lorsque nous avons utilisé des opérations asynchrones (HttpWebRequest et WebClient) et que nous avons dû mettre à jour l'interface. Nous avons résolu le problème grâce au Dispatcher.

Ce dispatcher permet d'exécuter des actions sur le thread auquel il est associé, grâce à sa méthode BeginInvoke. En l'occurrence, chaque DependencyObject possède un dispatcher et donc, la PhoneApplicationPage possède également un dispatcher par héritage, qui a été créé depuis le thread UI. Ainsi, l'appel à BeginInvoke depuis un thread d'arrière-plan sur le dispatcher de la page exécutera automatiquement l'action sur le thread UI. Voyons ceci dans le code et le résultat illustré suivants.

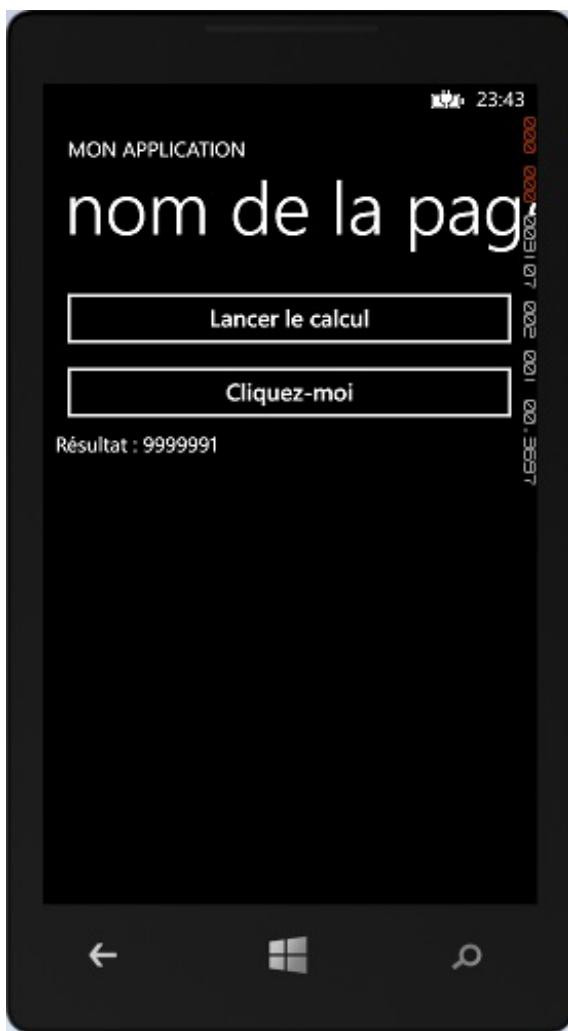
Code : C#

```

private void Button_Tap(object sender, RoutedEventArgs e)
{
    Thread thread = new Thread(() =>
    {
        int max = 0;
        for (int i = 0; i < 10000000; i++)
        {
            if (EstNombrePremier(i))
                max = i;
        }
        Dispatcher.BeginInvoke(() => Resultat.Text = "Résultat : " +
            max);
    });
    thread.Start();
}

```

Ce qui donne :



Affichage correct grâce au Dispatcher

Et voilà, plus de problèmes d'accès interdit entre les threads.

i Note : c'est une bonne idée d'afficher une barre de progression pendant les traitements longs. Nous avons déjà utilisé la ProgressBar, n'hésitez pas à l'employer à chaque fois qu'il y a un traitement potentiellement long, afin de toujours montrer à l'utilisateur qu'il se passe quelque chose.

Utiliser un BackgroundWorker

Je vous ai dit qu'utiliser directement la classe Thread n'était pas la solution la plus élégante pour créer des threads. C'est vrai (je ne mens jamais) ! Il existe d'autres classes particulièrement adaptées pour réaliser des traitements longs sur un thread d'arrière-plan, c'est le cas par exemple de la classe [BackgroundWorker](#). Elle permet notamment de connaître l'état d'avancement du thread, et de l'annuler si besoin. Pour l'utiliser, vous aurez besoin d'inclure l'espace de nom suivant :

Code : C#

```
using System.ComponentModel;
```

Vous aurez également besoin d'avoir une variable représentant le BackgroundWorker. En général, on utilise une variable membre de la classe :

Code : C#

```
private BackgroundWorker bw;
```

Au moment d'instancier le BackgroundWorker, il faudra lui passer les paramètres dont il a besoin.

Code : C#

```
public MainPage()
{
    InitializeComponent();

    bw = new BackgroundWorker { WorkerSupportsCancellation = true,
    WorkerReportsProgress = true };
    bw.DoWork += bw_DoWork;
    bw.ProgressChanged += bw_ProgressChanged;
    bw.RunWorkerCompleted += bw_RunWorkerCompleted;
}
```

Ici, nous lui indiquons qu'il est autorisé d'annuler le thread et que celui-ci peut témoigner de son avancement. Ensuite nous déclarons des événements. DoWork contiendra le long traitement à effectuer. ProgressChanged sera un événement levé lorsque le thread témoigne de son avancement. Enfin, RunWorkerCompleted sera levé lorsque le thread aura terminé son travail.

Voyons la méthode de travail :

Code : C#

```
private void bw_DoWork(object sender, DoWorkEventArgs e)
{
    BackgroundWorker worker = (BackgroundWorker)sender;

    int max = 0;
    int dernièreValeur = 0;
    for (int i = 0; i < 10000000; i++)
    {
        if (worker.CancellationPending)
        {
            e.Cancel = true;
            break;
        }
        int pourcentage = i * 100 / 10000000;
        if (pourcentage != dernièreValeur)
        {
            dernièreValeur = pourcentage;
            worker.ReportProgress(pourcentage);
        }
        if (EstNombrePremier(i))
            max = i;
    }
    e.Result = max;
}
```

On peut retrouver notre long calcul des nombres premiers. On y trouve également plusieurs choses, comme de vérifier s'il n'aurait pas été demandé à notre thread de se terminer prématurément. Cela se fait en testant la propriété CancellationPending. Si elle vaut vrai, alors on peut arrêter le calcul et marquer le BackgroundWorker comme étant annulé, grâce à l'argument de l'événement.

Ensuite, la méthode ReportProgress nous offre l'opportunité d'indiquer l'avancement du calcul. Ici, je lui indique le pourcentage d'avancement par rapport au max à calculer. Remarquez que je n'appelle la méthode ReportProgress que si le pourcentage d'avancement a changé car sinon je l'appelleraï énormément de fois inutilement, ce qui ralentirait considérablement mon calcul. Enfin, je peux utiliser la propriété Result pour stocker le résultat de mon calcul.

Vous avez donc compris que l'événement ProgressChanged était levé lorsque nous appelions la méthode ReportProgress. Cela nous permet de mettre à jour un affichage par exemple pour montrer l'avancement du thread :

Code : C#

```
private void bw_ProgressChanged(object sender,
ProgressChangedEventArgs e)
{
    Resultat.Text = e.ProgressPercentage + "%";
}
```

Notez qu'ici, il n'y a pas besoin de Dispatcher pour mettre à jour l'interface, malgré l'utilisation d'un thread d'arrière-plan. Tout cela est géré par le BackgroundWorker

Enfin, il reste la méthode qui est appelée lorsque le calcul est terminé, RunWorkerCompleted :

Code : C#

```
private void bw_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
{
    if (e.Cancelled)
        Resultat.Text = "Vous avez annulé le calcul !";
    else if (e.Error != null)
        Resultat.Text = "Erreur : " + e.Error.Message;
    else
        Resultat.Text = "Fini " + e.Result;
}
```

Ici, nous avons plusieurs cas de figure :

- Le thread a été annulé. Dans ce cas, la propriété Cancelled de l'argument est Vraie (*True*).
- Il y a une erreur. Dans ce cas, la propriété Error est différente de null.
- Ou alors tout s'est bien passé. Dans ce cas, on peut récupérer le résultat dans la propriété Result.

Comment annuler le thread ? Modifions notre XAML pour ajouter un bouton dont le rôle sera de stopper le thread :

Code : XML

```
<StackPanel>
    <Button Content="Lancer le calcul" Tap="Button_Tap" />
    <Button Content="Arrêter le calcul" Tap="Button_Tap2" />
    <Button Content="Cliquez-moi" Tap="Button_Tap_1" />
    <TextBlock x:Name="Resultat" />
</StackPanel>
```

Le code associé pour arrêter un thread est simplement :

Code : C#

```
private void Button_Tap2(object sender,
System.Windows.Input.GestureEventArgs e)
{
    if (bw.WorkerSupportsCancellation)
        bw.CancelAsync();
}
```

On utilise la méthode CancelAsync.

Pour démarrer le calcul, c'est le même principe, il suffit d'utiliser la méthode RunWorkerAsync :

Code : C#

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    if (!bw.IsBusy)  
        bw.RunWorkerAsync();  
}
```

Et voilà. Lorsque vous démarrez le calcul, non seulement l'interface ne sera pas bloquée, mais vous pourrez également voir l'avancement du thread ainsi que l'arrêter en cours de route (voir figure suivante).



d'exécution puis arrêté

Enchaîner les threads dans un pool

Une autre solution pour démarrer des threads consiste à utiliser la classe `ThreadPool`. Elle est très pratique lorsque nous avons besoin d'enchaîner beaucoup de traitements d'arrière-plan. Grâce au pool de thread nous pourrons empiler les différents threads que nous souhaitons exécuter en arrière-plan, et c'est le système qui va se débrouiller pour les enchaîner les uns après les autres ou potentiellement en parallèle, sans que nous n'ayons quelque chose de particulier à faire.

C'est très pratique. Imaginons par exemple une application qui doit lire beaucoup de flux RSS. Si nous démarrons tous les téléchargements en même temps, il y a fort à parier que nous allons obtenir un timeout. Dans ce cas, la bonne solution est de pouvoir les enchaîner séquentiellement. C'est un parfait candidat pour un `ThreadPool`.

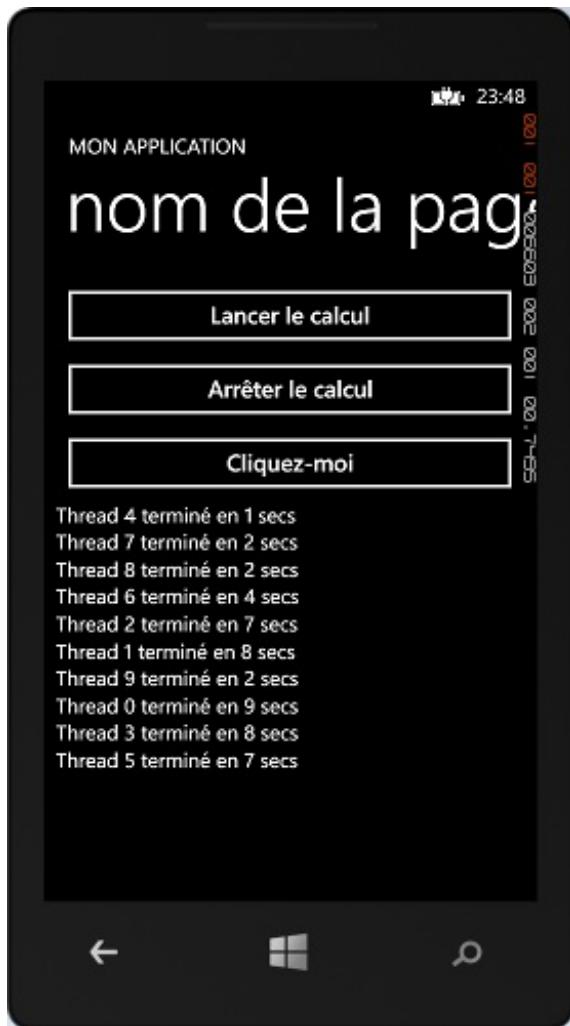
Pour l'illustrer, nous allons faire un peu plus simple et empiler des threads qui auront une durée d'exécution aléatoire :

Code : C#

```
private void Button_Tap(object sender,  
System.Windows.Input.GestureEventArgs e)  
{  
    Random random = new Random();  
    for (int i = 0; i < 10; i++)
```

```
{  
    int numThread = i;  
    ThreadPool.QueueUserWorkItem(o =>  
    {  
        int temps = random.Next(1, 10);  
        Thread.Sleep(TimeSpan.FromSeconds(temps));  
        Dispatcher.BeginInvoke(() => Resultat.Text += "Thread "  
        + numThread + " terminé en " + temps + " secs" +  
        Environment.NewLine);  
    });  
}  
}
```

Rien de bien compliqué, mais il y a cependant une petite astuce pour utilisateurs avancés. Il s'agit de l'utilisation de la variable numThread. On pourrait croire qu'elle ne sert à rien et qu'on pourrait utiliser juste la variable i à la place, mais que nenni. Si vous l'enlevez, vous n'aurez que des threads numérotés 10. Tout ça, à cause d'une histoire de *closure* que je vais vous épargner, mais si vous êtes curieux, vous pouvez trouver une explication en anglais [ici](#). Vous trouverez le résultat en images dans la prochaine figure.



L'exécution des threads a été prise en charge par le pool de thread

Attention, notez le retour en force du Dispatcher pour mettre à jour l'interface.



Le DispatcherTimer

Nous avons déjà utilisé cette classe précédemment mais sans l'avoir vraiment décrite. La classe `DispatcherTimer` va nous permettre d'appeler une méthode à intervalles réguliers. L'intérêt va être de pouvoir exécuter une tâche en arrière-plan et de manière répétitive. Imaginons par exemple une tâche de synchronisation, qui toutes les 5 minutes va enregistrer le travail de l'utilisateur sur le cloud ...

L'exemple le plus classique est la création d'une horloge, dont l'heure est mise à jour toutes les secondes :

Code : XML

```
<TextBlock x:Name="Heure" />
```

La mise à jour de ce contrôle se fera périodiquement grâce au DispatcherTimer :

Code : C#

```
private DispatcherTimer dispatcherTimer;

public MainPage()
{
    InitializeComponent();

    dispatcherTimer = new DispatcherTimer { Interval =
TimeSpan.FromSeconds(1) };
    dispatcherTimer.Tick += dispatcherTimer_Tick;
    dispatcherTimer.Start();
}

private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    Heure.Text = DateTime.Now.ToString();
}
```

Voilà notre TextBlock qui est mis à jour toutes les secondes.

L'avantage de ce timer c'est qu'il utilise la file d'attente du dispatcher, donc nul besoin d'utiliser le Dispatcher pour pouvoir mettre à jour l'interface, ceci est pris en charge automatiquement. L'inconvénient, c'est que rien ne nous garantit que la méthode soit effectivement appelée exactement toutes les secondes. Vous pourrez observer quelques variations en fonction de l'existence d'autres timers ou d'éléments dans le Dispatcher. Si vous laissez tourner un peu votre application, vous verrez que, de temps en temps, il se décale d'une seconde, ce qui n'est pas dramatique, et au pire, nous pouvons augmenter la fréquence de mise à jour. Mais le timer va nous permettre aussi d'illustrer un autre point important de Windows Phone, utilisons-le par exemple pour faire bouger un rectangle sur notre écran. Le XAML est le suivant :

Code : XML

```
<StackPanel>
    <Button Content="Lancer le calcul" Tap="Button_Tap" />
    <Canvas>
        <Rectangle x:Name="Rect" Width="40" Height="40" Fill="Green" />
    </Canvas>
</StackPanel>
```

Voici le code-behind correspondant :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private DispatcherTimer dispatcherTimer;
    private int direction = 1;

    public MainPage()
    {
        InitializeComponent();
```

```

        dispatcherTimer = new DispatcherTimer { Interval =
TimeSpan.FromMilliseconds(10) };
        dispatcherTimer.Tick += dispatcherTimer_Tick;
        dispatcherTimer.Start();
    }

private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    double x = (double)Rect.GetValue(Canvas.LeftProperty);
    int pas = 10;
    if (x > 480 - Rect.Width)
        direction = -1;
    if (x < 0)
        direction = 1;
    x += pas * direction;
    Rect.SetValue(Canvas.LeftProperty, x);
}

private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    Thread.Sleep(TimeSpan.FromSeconds(5));
}
}

```

Si vous démarrez l'application, vous pourrez voir le rectangle qui s'anime horizontalement. Ceci pourrait être une bonne solution pour animer un objet, sauf que ... cliquez voir un peu sur le bouton qui bloque le thread UI en simulant un long calcul ... et paf ! Ça ne bouge plus.

Ah bravo ! Franchement, ça ne se fait pas de bloquer le thread UI !!!

Thread de composition

Vous me direz, il suffit de mettre le calcul dans un thread et je vous répondrai que oui, cela fonctionne sans problème. Mais il y a d'autres solutions pour faire en sorte qu'une animation fonctionne malgré un long calcul. Vous vous rappelez comment on définit une animation via un StoryBoard ?

Code : XML

```

<StackPanel>
    <Button Content="Lancer le calcul" Tap="Button_Tap" />
    <Canvas>
        <Canvas.Resources>
            <Storyboard x:Name="MonStoryboard">
                <DoubleAnimation From="0" To="440" Duration="0:0:2"
RepeatBehavior="Forever" AutoReverse="True"
                    Storyboard.TargetName="Rect"
                    Storyboard.TargetProperty="(Canvas.Left)"/>
            </Storyboard>
        </Canvas.Resources>
        <Rectangle x:Name="Rect" Width="40" Height="40" Fill="Green" />
    </Canvas>
</StackPanel>

```

Puis dans le code-behind :

Code : C#

```

public partial class MainPage : PhoneApplicationPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}

```

```
        MonStoryboard.Begin();  
    }  
  
    private void Button_Tap(object sender,  
    System.Windows.Input.GestureEventArgs e)  
    {  
        Thread.Sleep(TimeSpan.FromSeconds(5));  
    }  
}
```

Et là, lorsque vous démarrez l'application, le fait de bloquer le thread UI ne bloque pas l'animation. Diantre, comment cela se fait-il ?

C'est grâce au thread de composition !

Le thread de composition va servir à alléger le thread UI. Il va pouvoir décharger le thread UI de certaines tâches qui lui incombent. Alors que le thread UI va être utilisé pour créer le premier rendu des contrôles, le thread de composition va travailler directement avec ces éléments qui sont mis en cache dans la mémoire sous forme d'images bitmaps et utiliser en préférence le processeur graphique. Les animations par exemple sont prises en charge par ce thread de composition et mises en cache pour un affichage rapide. Le processeur graphique étant indépendant il pourra continuer à traiter des informations même si le processeur du téléphone est fortement sollicité. Lorsque vous demandez la mise à jour d'un contrôle qui nécessite un changement (changement de couleur, taille, ...) alors ce contrôle est supprimé du cache et redessiné par le thread UI.

Cette utilisation de cache apporte beaucoup de performance et de plus, certaines des opérations peuvent être faites directement sur les objets en cache, par exemple une animation qui fait une rotation.

Comme nous l'avons vu, la meilleure solution pour corriger l'animation précédente est de créer l'animation en XAML et d'utiliser les contrôles adéquats afin qu'elle soit prise en charge par ce fameux thread de composition et utilise mécanisme de mise en cache. C'est toujours le thread UI qui est en charge de mettre à jour l'interface, sauf qu'il ne fait plus aucun calcul et affiche directement des bitmaps mis en cache. C'est le thread de composition qui fait les calculs en s'aidant du processeur graphique, qui est particulièrement doué pour ça.

D'une manière générale, il vaut mieux laisser le système gérer lui-même le cache, mais dans certains cas il peut être utile de positionner soi-même la propriété **CacheMode** d'un objet **BitmapCache** sur un contrôle, mais il vaut mieux savoir ce que l'on fait et tester les performances avec ou sans, car le cache augmente considérablement la consommation de mémoire des processeurs du téléphone.

Vous vous rappelez de l'application PerformanceProgressBar pour Windows Phone 7.5 ou de l'application ProgressBar pour Windows Phone 8 ? Leur principe est justement de faire en sorte que ce soit le thread de composition qui prenne en charge l'animation et non le thread UI.

Les outils pour améliorer l'application

Il existe plusieurs outils pour améliorer la réactivité de son application.

Il y a notamment le Windows Phone Performance Analysis tool, qui est un outil de profilage qui s'installe avec l'environnement de développement SDK de Windows Phone. Il permet de mesurer la consommation de mémoire du processeur central CPU (*Central Processing Unit*), mais aussi le nombre de rafraîchissements par seconde (le frame-rate). Je ne vais pas le décrire ici car on sort un peu de l'apprentissage de Windows Phone, mais sachez que cet outil existe et qu'il peut vous aider à optimiser vos applications.

Pour une description, en anglais, vous pouvez consulter : [http://msdn.microsoft.com/en-us/library \[...\] =vs.105\).aspx](http://msdn.microsoft.com/en-us/library [...] =vs.105).aspx).

Vous avez également à votre disposition des compteurs. Ce sont ces compteurs que l'on voit apparaître dans notre émulateur sur la droite. Ils permettent de connaître entre autres le taux de rafraîchissement. Plus d'informations ici [http://msdn.microsoft.com/fr-fr/library \[...\] v=vs.92\).aspx](http://msdn.microsoft.com/fr-fr/library [...] v=vs.92).aspx).

Vous avez aussi la possibilité de voir facilement quels sont les contrôles qui sont mis à jour. Il suffit de positionner une variable booléenne précise et vous verrez dans vos applications s'il y a un contrôle qui se met à jour alors qu'il ne le devrait pas. Voir à cet emplacement :

[http://msdn.microsoft.com/fr-fr/library \[...\] v=VS.95\).aspx](http://msdn.microsoft.com/fr-fr/library [...] v=VS.95).aspx).

- Les threads sont importants à maîtriser afin d'avoir une application fluide et réactive.
- Il n'est pas possible de mettre à jour l'interface depuis un thread d'arrière-plan. On utilisera le Dispatcher.
- Il existe des classes puissantes du framework .NET pour gérer les threads d'arrière-plan.

Utiliser des tâches Background Agent

Nous avons vu que Windows Phone ne gérait pas vraiment le multitâches entre les applications... Une application peut être soit démarée, soit en pause. Si on démarre une nouvelle application, la précédente s'arrête ; les deux ne tournent pas en même temps. Windows Phone propose un système différent de gestion du multitâches. Ce n'est pas du multitâches à proprement parler, mais la possibilité de faire des choses de manière périodique en tâche de fond, et ce, lorsque l'application est désactivée, en pause. Il s'agit des *Background Agent*, qui sont donc des tâches qui s'exécutent en arrière-plan. Il existe deux types de tâches de fond, les périodiques (*Periodic*) qui vont pouvoir faire des petites tâches régulièrement, et celles qui vont avoir besoin de plus de ressources (*Resource Intensive*) avec fatallement des limitations.

Créer un Background Agent pour une tâche périodique

Une tâche périodique doit être exécutée rapidement et doit faire quelque chose de simple, comme mettre à jour une liste de mails, mettre à jour une tuile, ou mettre à jour une coordonnée GPS. Ces tâches sont exécutées toutes les 30 minutes (plus ou moins précisément, car c'est le système d'exploitation qui gère ces tâches et peut éventuellement en regrouper plusieurs), sont limitées en nombre par téléphone (cela dépend de la configuration du téléphone) et ne dépassent pas 25 secondes d'exécution.

Tandis que les tâches aux ressources intensives peuvent durer jusqu'à 10 minutes, mais ne sont exécutées que lorsque le téléphone est branché à son chargeur, dispose d'une connexion WIFI, à sa batterie rechargée à plus de 90% et que l'écran est verrouillé. C'est typiquement le cas par exemple lorsqu'on recharge le téléphone la nuit...

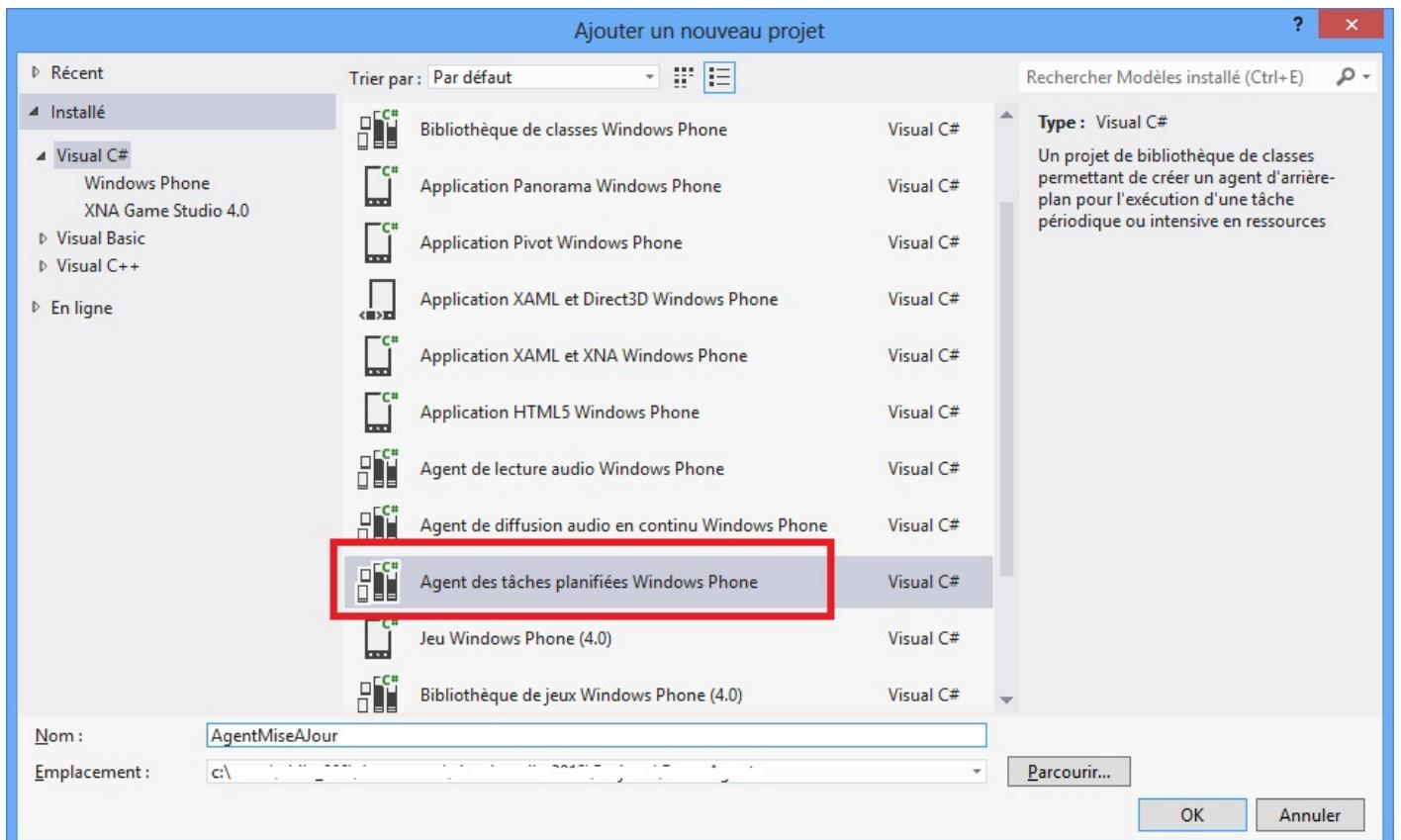
Un exemple classique d'utilisation de tâche périodique est la mise à jour d'une tuile afin d'informer l'utilisateur qu'il y a quelque chose de nouveau dans l'application à aller consulter, un nouveau mail, des nouveaux flux RSS à lire, etc. Pour les tâches aux ressources intensives, il s'agira plutôt de mettre à jour des gros volumes de données, par exemple télécharger la mise à jour d'une carte pour notre logiciel de navigation GPS ou bien faire des synchronisations lorsque nous avons travaillé en hors-ligne, etc.

Notez que nous n'avons aucune garantie que la tâche soit bien exécutée, c'est le cas par exemple si le téléphone est en mode économie de batterie.

Voyons à présent comment cela fonctionne.

Pour créer une tâche de fond, vous aurez besoin de deux projets. Un premier projet classique contenant votre application classique et un projet spécial contenant la tâche de fond. Pour la démonstration, je vais créer un petit programme dont le but sera d'avoir une tuile qui affiche l'heure de la dernière exécution de la tâche de fond, ainsi qu'un nombre aléatoire.

Créons donc un nouveau projet de type Application Windows Phone que nous nommons DemoAgent, puis ajoutons un nouveau projet à la solution fraîchement créée de type « Agent des tâches planifiées Windows Phone », que nous appelons par exemple AgentMiseAJour (voir figure suivante).



Création du projet de type Agent des tâches planifiées

C'est dans ce projet que nous allons créer notre tâche de fond qui aura pour but de mettre à jour la tuile. La tâche a d'ailleurs déjà été créée, via la classe `ScheduledAgent`, qui hérite de `ScheduledTaskAgent`. Elle possède notamment une méthode `OnInvoke` qui est le point d'entrée de notre agent. C'est donc dans cette méthode que nous allons mettre à jour la tuile :

Code : C#

```
protected override void OnInvoke(ScheduledTask task)
{
    ShellTile tuileParDefaut = ShellTile.ActiveTiles.First();

    if (tuileParDefaut != null)
    {
        FlipTileData flipTileData = new FlipTileData
        {
            BackContent = "Dernière MAJ " +
DateTime.Now.ToShortTimeString(),
            Count = new Random().Next(0, 20),
        };

        tuileParDefaut.Update(flipTileData);
    }

    NotifyComplete();
}
```

Vous pouvez constater que nous affichons simplement un message sur le dos de la tuile avec l'heure de dernière mise à jour de la tuile. De même, on détermine un nombre aléatoire qu'on affichera dans le cercle en haut à droite de la tuile.

Mais ceci ne suffit pas. Il va falloir au moins un premier lancement de l'application afin de pouvoir démarrer la tâche et éventuellement permettre de l'arrêter. Dans mon application de démo, je vais donc créer deux boutons permettant de respectivement de démarrer la tâche et de l'arrêter. Voici le XAML :

Code : XML

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <StackPanel>
        <Button Content="Démarrer" Tap="Button_Tap" />
        <Button Content="Arrêter" Tap="Button_Tap_1" />
    </StackPanel>
</Grid>
```

Maintenant, il faut démarrer et arrêter la tâche. On utilise pour cela la classe `ScheduledActionService`. Voyons tout d'abord comment arrêter la tâche, car c'est le plus simple. Il suffit de retrouver la tâche par son nom et de la supprimer du service :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private const string NomTache = "MajHeure";
    public MainPage()
    {
        InitializeComponent();
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {

    }

    private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        if (ScheduledActionService.Find(NomTache) != null)
        {
            ScheduledActionService.Remove(NomTache);
        }
    }
}
```

La constante nous sert à identifier notre tâche de manière unique. Maintenant, pour démarrer la tâche, nous aurons besoin d'instancier un objet de la classe `PeriodicTask`, qui comme son nom l'indique, est une tâche périodique :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private const string NomTache = "MajHeure";
    public MainPage()
    {
        InitializeComponent();
    }

    private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        StopTache();

        PeriodicTask task = new PeriodicTask(NomTache);
        task.Description = "Cette description apparaitra dans les
paramètres du téléphone";
        try
        {
            ScheduledActionService.Add(task);
            // utilisé à des fins de débogages, pour tenter de

```

```
démarrer la tâche immédiatement
    ScheduledActionService.LaunchForTest(NomTache, new
TimeSpan(0, 0, 1));
}
catch (InvalidOperationException)
{
    MessageBox.Show("Impossible d'ajouter la tâche
périodique, car il y a déjà trop d'agents dans le téléphone");
}
}

private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
{
    StopTache();
}

public void StopTache()
{
    if (ScheduledActionService.Find(NomTache) != null)
    {
        ScheduledActionService.Remove(NomTache);
    }
}
```

Il suffit d'instancier la classe PeriodicTask avec le nom de la tâche, de lui donner une description et de l'ajouter au service. Faites attention à toujours encadrer l'ajout d'une tâche d'un try/catch car une exception peut être levée si nous avons atteint le nombre maximum d'agents que votre téléphone peut supporter.

Ensuite, pour nous faciliter le débogage, il est possible de démarrer la tâche immédiatement, ceci se fait grâce à la méthode LaunchForTest. Cette méthode ne doit pas être appelée directement lorsque l'application est terminée et prête à être publiée.



Remarque : il pourrait être tout à fait judicieux d'entourer cette instruction des instructions conditionnelles de débug :

Code : C#

```
#if DEBUG
    ScheduledActionService.LaunchForTest(NomTache, new TimeSpan(0,
0, 1));
#endif
```

Vous aurez remarqué qu'avant de démarrer la tâche, je commence par l'arrêter, si jamais elle est déjà lancée.



N'oubliez pas de rajouter une référence au projet contenant la tâche depuis votre application, sinon le projet ne pourra pas utiliser l'agent et celui-ci ne fonctionnera pas.

Il ne reste plus qu'à démarrer notre application, à cliquer sur le bouton démarrer et à attendre. Si vous mettez un point d'arrêt dans la tâche, vous pourrez voir que vous vous y arrêterez au bout de quelques secondes.

Puis, vous pourrez arrêter l'application, l'épingler dans l'émulateur et attendre à nouveau. La tuile se mettra à jour à peu près toutes les demi-heures, comme le montre la figure suivante.

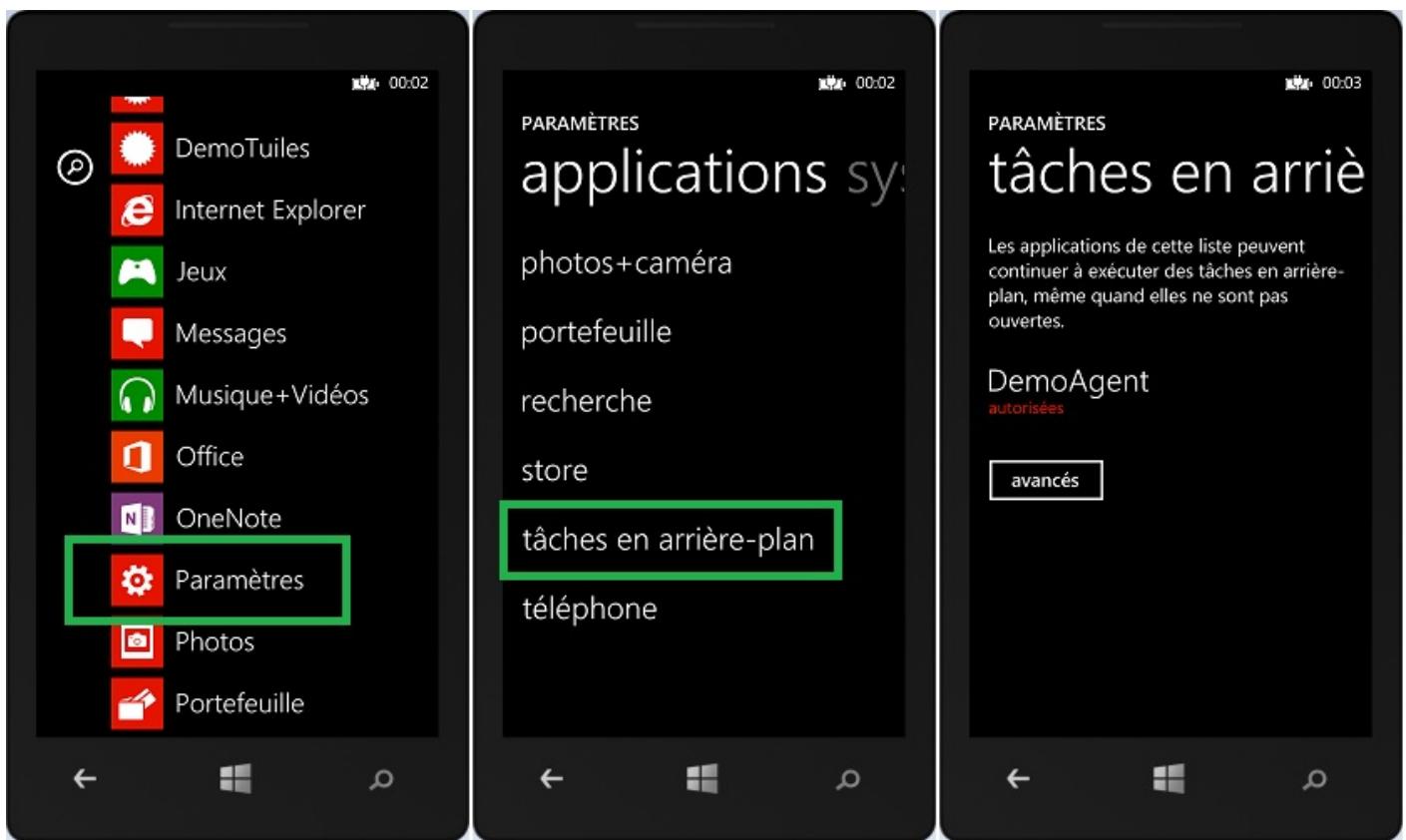


Mise à jour de la tuile

via le background agent



Remarque : vous pouvez accéder à la liste des tâches qui s'exécutent en arrière-plan en allant dans les paramètres de l'émulateur (ou de votre téléphone), applications, tâches en arrière-plan. Vous y trouverez notamment notre tâche (voir figure suivante).



Les tâches d'arrière plan dans les paramètres de l'émulateur

Si vous cliquez sur le nom de l'agent, vous pourrez voir la description de la tâche que nous avons saisie. C'est l'endroit idéal pour expliquer à l'utilisateur ce que fait l'agent afin de l'encourager à ne pas le désactiver. Jetez un œil à la figure qui suit.



Description de l'agent

Et voilà pour les tâches périodiques.

Vous vous rappelez de notre application météo ? Il s'agit d'une application idéale pour utiliser une tâche en arrière-plan afin de mettre à jour les informations de météo et les indiquer directement dans la tuile, sans avoir à lancer l'application. Ainsi, d'un coup d'œil, nous pourrions voir ce que prévoit la météo pour le temps de la journée.

Je ne vais pas illustrer ceci ici, mais je vous encourage à tester par vous-même cette possibilité dans le cadre d'un petit TP en dehors de ce cours.

Créer une tâche aux ressources intensives

Pour créer une tâche aux ressources intensives, c'est exactement la même chose que pour les tâches périodiques.

La seule différence vient du fait qu'on utilisera la classe [ResourceIntensiveTask](#).

Par extension, on pourra donc déterminer dans notre méthode `OnInvoke` le type de tâche en testant le type de la classe :

Code : C#

```
protected override void OnInvoke(ScheduledTask task)
{
    if (task is PeriodicTask)
    {
        // tâche périodique
    }
    else
    {
        // tâche aux ressources intensives
    }

    NotifyComplete();
}
```

Remarques générales sur les tâches

Gardez à l'esprit qu'une tâche peut ne pas être exécutée du tout si jamais le téléphone ne se retrouve pas dans les bonnes conditions d'exécution de la tâche.

De même, vous aurez intérêt à faire en sorte que vos tâches s'exécutent le plus rapidement possible car elles peuvent être terminées à tout moment si jamais une des conditions d'exécution n'est plus garantie (on débranche le téléphone, on reçoit un coup de fil, etc.).

N'utilisez pas trop de mémoire non plus (moins de 5 Mo) sous peine de voir votre tâche terminée.

Sachez enfin qu'une tâche a une durée de vie de 2 semaines. À chaque fois que votre application est lancée, vous avez l'opportunité de renouveler ces 2 semaines. C'est pour cette raison que nous commençons par supprimer la tâche avant de la rajouter, lorsque nous cliquons sur le bouton démarrer. Ceci implique que si votre application n'est pas utilisée pendant 2 semaines, votre tâche sera automatiquement désactivée.



Remarque : une tâche doit pouvoir être désactivée par l'utilisateur depuis l'application qui l'a créé. Il a également la possibilité de la désactiver depuis les paramètres du téléphone.

Envoyer une notification avec un agent d'arrière-plan

Vous vous rappelez des notifications ? Nous avions mis en place toute une architecture complexe pour permettre d'envoyer des notifications sur un téléphone, afin que l'utilisateur puisse être averti de quelque chose sans forcément devoir ouvrir l'application.

Vous vous doutez que l'agent d'arrière-plan est une solution intéressante pour simplifier l'envoi de notifications, en tenant compte bien sûr des limitations de celui-ci. Il suffit de faire en sorte que son agent d'arrière-plan envoie une notification toast locale, après par exemple qu'il soit allé télécharger des informations sur internet. Tout d'abord, créons un agent basique :

Code : C#

```
public class ScheduledAgent : ScheduledTaskAgent
{
    [... code abrégé pour plus de clarté ...]

    protected override void OnInvoke(ScheduledTask task)
    {
        ShellToast toast = new ShellToast
        {
            Title = "Des nouvelles infos !",
            NavigationUri = new Uri("/ MainPage.xaml?nouvelleinfo=" +
DateTime.Now.ToShortTimeString(), UriKind.Relative),
            Content = "Il est " + DateTime.Now.ToShortTimeString()
        };

        toast.Show();

        NotifyComplete();
    }
}
```

Vous voyez, on ne fait rien de formidable, juste créer un nouvel objet de type ShellToast, y mettre l'heure et envoyer la notification. Ce serait bien sûr l'emplacement idéal pour aller voir sur internet s'il y a des nouvelles informations à envoyer. Ensuite, l'enregistrement de l'agent se fait de manière classique dans l'application :

Code : C#

```
public partial class MainPage : PhoneApplicationPage
{
    private const string NomTache = "EnvoiNotifHeure";

    public MainPage()
    {
        InitializeComponent();

        StopTache();

        PeriodicTask task = new PeriodicTask(NomTache);
        task.Description = "Agent permettant l'envoi de
notifications";
```

```
try
{
    ScheduledActionService.Add(task);
    // utilisé à des fins de débogages, pour tenter de
    // démarrer la tâche immédiatement
#if DEBUG
    ScheduledActionService.LaunchForTest(NomTache, new
TimeSpan(0, 0, 1));
#endif
}
catch (InvalidOperationException)
{
    MessageBox.Show("Impossible d'ajouter la tâche
périodique, car il y a déjà trop d'agents dans le téléphone");
}

public void StopTache()
{
    if (ScheduledActionService.Find(NomTache) != null)
    {
        ScheduledActionService.Remove(NomTache);
    }
}
```

Ça peut être plutôt pratique et c'est quand même simple à mettre en place, sans serveur de notification (voir la notification d'arrière-plan dans la figure qui suit).



- Les Background Agent permettent d'exécuter des tâches en arrière-plan, des tâches périodiques ou des tâches aux

ressources intensives.

- Ces tâches peuvent nous permettre facilement de mettre à jour une tuile ou d'envoyer une notification sans que l'utilisateur n'ait besoin de démarrer l'application.
- Une tâche a une durée de vie de 2 semaines, renouvelable à chaque fois que l'application est démarrée.

Utiliser Facebook dans une application mobile

Avec l'avènement des réseaux sociaux, une application Windows Phone va devoir être capable de travailler avec eux ! On ne cite plus bien sûr Twitter ou Facebook. Chacun des grands réseaux sociaux offre une solution pour interagir avec eux. Facebook ne déroge pas à cette règle et propose diverses solutions pour que nos applications tirent parti de la puissance du social.

Nous allons à présent voir comment réaliser une petite application fonctionnant avec Facebook.

Créer une application Facebook

La première chose à faire est de créer une application Facebook. C'est une étape gratuite mais indispensable afin de pouvoir établir une relation de confiance entre votre application Windows Phone et Facebook. Pour ce faire, rendez-vous sur <http://www.facebook.com/developers/createapp.php>.

Si vous n'êtes pas connecté, c'est le moment de le faire.

Créez ensuite une application, comme à la figure suivante.

The screenshot shows the Facebook Developers homepage. At the top, there's a navigation bar with links for Documentation, Assistance, Blog, and Applications. Below the navigation, there's a search bar and a link to 'Recherche'. The main content area is titled 'Applications' and features a large button labeled '+ Créez une application' which is highlighted with a red box. At the bottom of the page, there are links for 'Facebook © 2012 · Français (France)', 'À propos de', 'Conditions d'utilisation de la plateforme', and 'Politique de confidentialité'.

Création d'une application Facebook

Donnez-lui au moins un nom (voir figure suivante).

The screenshot shows the 'Créer une application' (Create an Application) form. The 'Nom de l'application' (Application name) field is filled with 'WP8'. The 'Espace de nom de l'application' (Application namespace) field is set to 'Facultatif' (Optional). The 'Hébergement' (Hosting) section has a checkbox 'Oui, je voudrais un hébergement gratuit proposé par Heroku (Learn More)' which is unchecked. At the bottom, there's a note 'En continuant, vous acceptez les Règlements de la plate-forme Facebook' (By continuing, you accept the Platform Terms of Service) and two buttons: 'Continuer' (Continue) and 'Annuler' (Cancel). A note on the right says 'Donner' (Give).

un nom à l'application Facebook

Finissez la création (voir figure suivante).

Applications ▶ WP8 ▶ Essentiel

WP8

App ID: 3... App Secret: 9... b7 (réinitialiser)

Infos générales

Display Name: [?] WP8

Namespace: [?]

Adresse électronique de contact: [?]

App Domains: [?] Enter your site domains and press enter

Hosting URL: [?] You have not generated a URL through one of our partners (Get one)

Mode bac à sable: [?] Activé Désactivé

Sélectionnez comment votre application est intégrée à Facebook

Site web avec une authentification Facebook Log in to my website using Facebook.

Application sur Facebook Use my app inside Facebook.com.

Site Web Mobile Bookmark my web app on Facebook mobile.

Application iOS native Publish from my iOS app to Facebook.

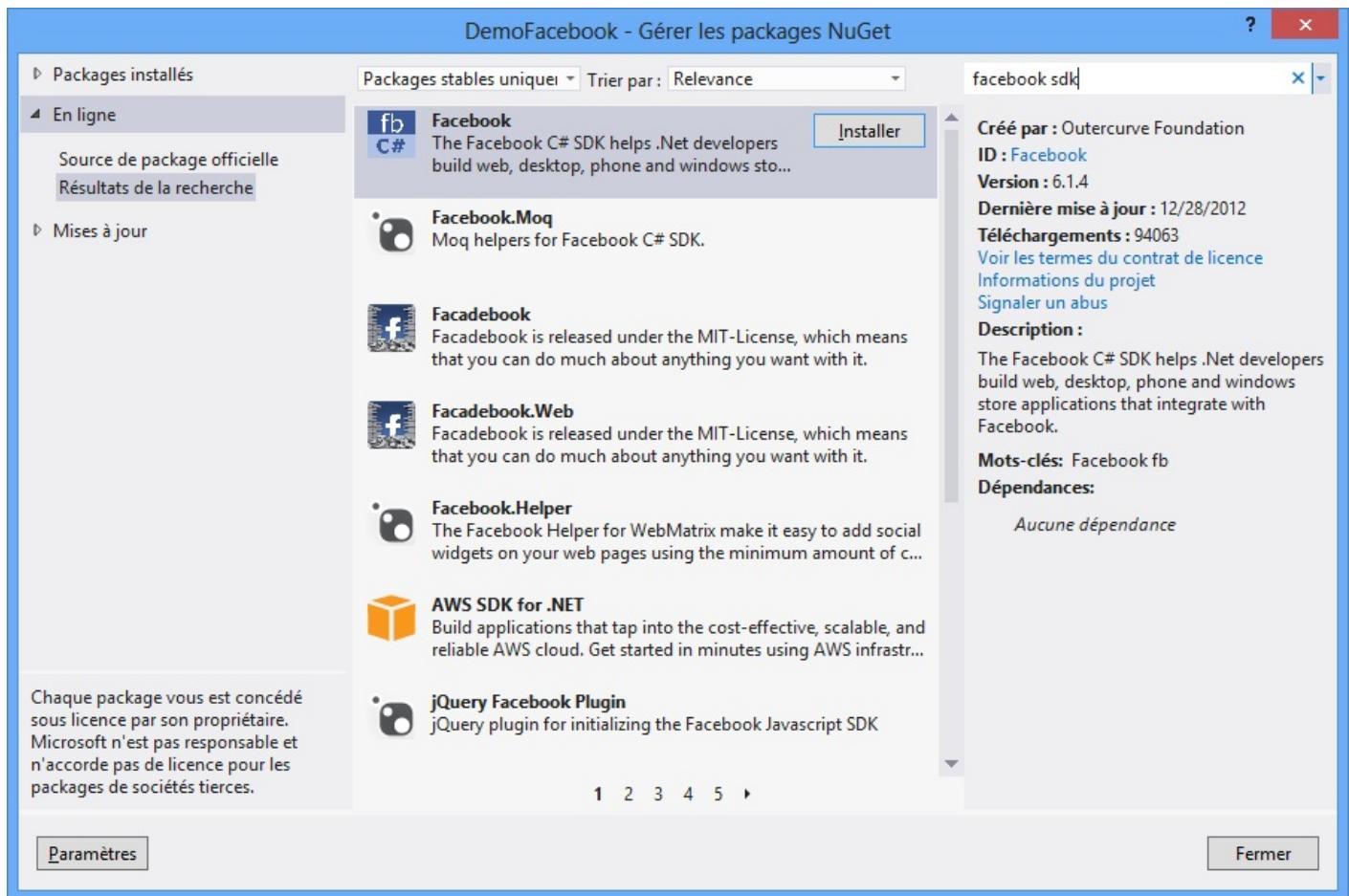
Finalisation de la création de l'application Facebook

Vous obtenez un identifiant d'application, ainsi qu'une clé d'API. Il s'agit du point d'entrée permettant d'exploiter les données issues de Facebook. L'identifiant d'application et la clé vont vous permettre d'utiliser les méthodes de l'API et d'identifier de manière unique votre application afin de pouvoir établir la relation entre le téléphone et un compte Facebook.

Simplifier les connexions à Facebook avec l'API

L'API Facebook est utilisable en REST. Il est possible de faire tous nos appels à Facebook depuis notre application avec REST, mais il existe un projet open source qui encapsule les appels REST afin de nous simplifier la tâche. Personne ici ne rêve de se compliquer la vie, alors nous allons utiliser ce fameux projet. Il s'agit du « Facebook C# SDK », autrefois situé sur la plateforme open source CodePlex, et qui se trouve désormais sur <http://facebooksdk.net/>.

Mais vous n'avez même pas besoin d'y aller pour le récupérer, car les concepteurs nous ont encore simplifié la tâche en le rendant disponible via un package Nuget (voir figure qui suit).



Le SDK Facebook via NuGet

Et voilà, le SDK est référencé !

Sécuriser l'accès à Facebook avec OAuth

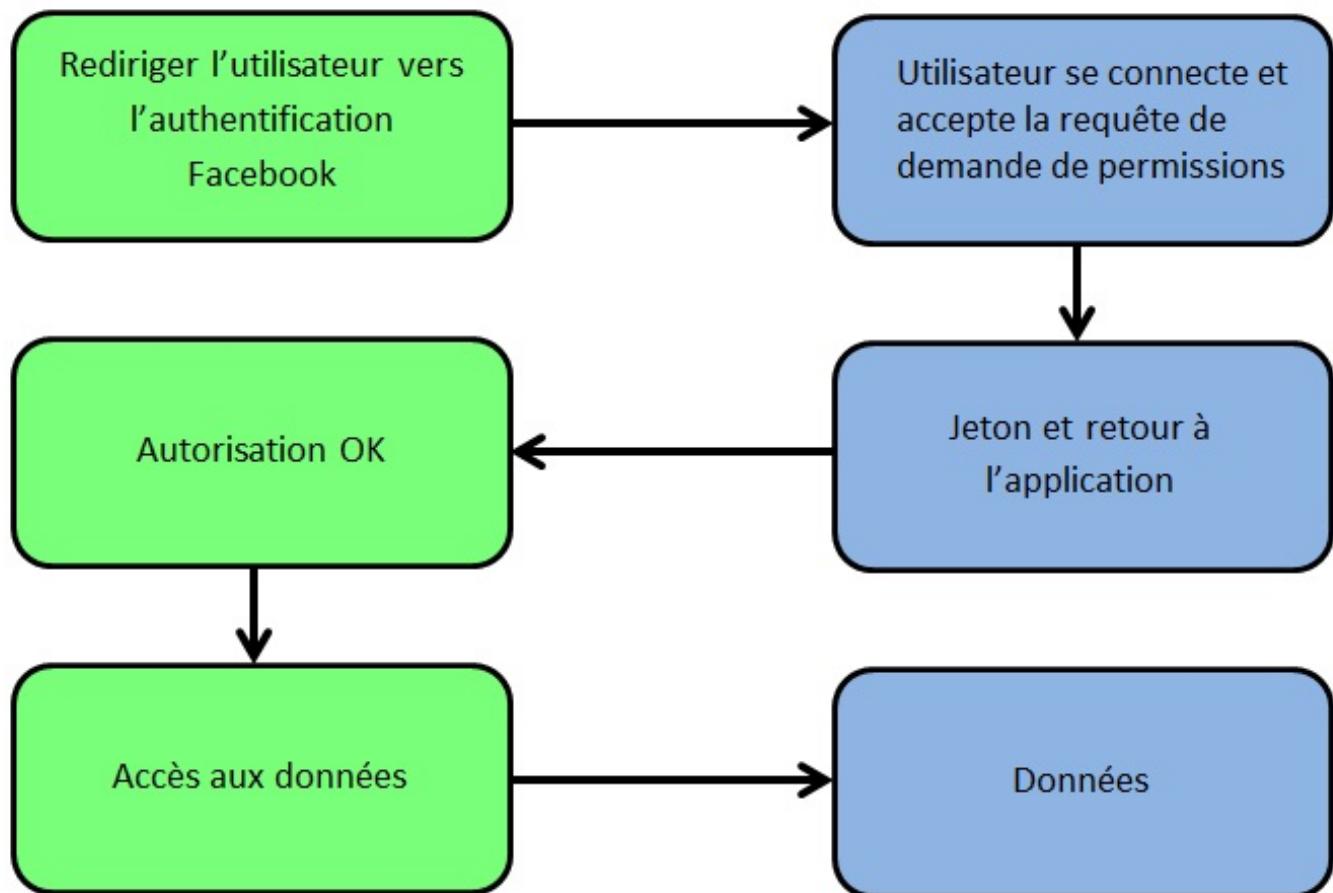
OAuth ? Kézako, pourquoi tu parles de ça ?



Eh bien parce qu'il s'agit du système d'authentification à l'API de Facebook. OAuth est un protocole qui permet l'accès à des données de manière sécurisée et en fonction des restrictions d'accès appliquées par l'utilisateur. Lorsqu'elle aura besoin d'accéder à certaines informations sur l'utilisateur, notre application va donc utiliser OAuth pour demander à Facebook d'authentifier cet utilisateur. Une fois que l'utilisateur est authentifié, Facebook nous fournit un jeton permettant d'accéder à son API.

Application Windows Phone

Facebook



Description du principe de fonctionnement d'OAuth

Regardons dans la pratique comment cela fonctionne...

Se connecter à Facebook

Pour authentifier un utilisateur, nous allons devoir naviguer sur la page d'authentification de Facebook. Mais comment accéder à cette fameuse page ?

Pour le savoir, nous allons utiliser le package SDK, mais avant ça, nous aurons besoin de rajouter un WebBrowser dans notre page XAML :

Code : XML

```
<phone:WebBrowser x:Name="wb" Visibility="Collapsed"
    Navigated="wb_Navigated" />
```

Celui-ci est masqué au chargement de la page (Visibility à Collapsed). Nous avons également associé une méthode à l'événement de navigation terminé, qui nous permettra de traiter l'authentification et de récupérer le jeton.

Puis nous allons créer une variable privée dans notre page, du type FacebookClient pour représenter l'utilisateur que l'on souhaite authentifier :

Code : C#

```
private FacebookClient client;
```

Vous aurez besoin de l'espace de noms Facebook pour accéder aux fonctionnalités de Facebook :

Code : C#

```
using Facebook;
```

Enfin, nous allons naviguer sur l'URL obtenue grâce au package SDK. Pour ce faire, nous allons devoir spécifier un certain nombre de paramètres en créant un dictionnaire d'objets :

Code : C#

```
public MainPage()
{
    InitializeComponent();

    Dictionary<string, object> parameters = new Dictionary<string,
object>();
    parameters["response_type"] = "token";
    parameters["display"] = "touch";
    parameters["scope"] = "user_about_me, friends_about_me,
user_birthday, friends_birthday, publish_stream";
    parameters["redirect_uri"] =
"https://www.facebook.com/connect/login_success.html";
    parameters["client_id"] = "votre id d'application";

    client = new FacebookClient();
    Uri uri = client.GetLoginUrl(parameters);

    wb.Visibility = Visibility.Visible;
    wb.Navigate(uri);
}
```

Dans ces paramètres, on trouve :

- **Le paramètre response_type** : on lui attribue la valeur *token* car nous avons besoin d'un jeton d'accès.
- **Le paramètre display** : on lui attribue la valeur *touch*. Il s'agit de l'interface que Facebook va proposer pour notre authentification. La valeur *touch* est celle la plus adaptée aux périphériques mobiles (les autres modes sont disponibles sur <https://developers.facebook.com/docs/reference/dialogs/>).
- Enfin, et c'est le plus important, **le paramètre scope** : il correspond aux informations que nous souhaitons obtenir de la part de l'utilisateur. Il s'agit des permissions que l'on peut retrouver à cet emplacement : [https://developers.facebook.com/docs/r \[...\] /permissions](https://developers.facebook.com/docs/r [...] /permissions).

Par exemple, j'ai choisi les permissions suivantes :

- user_about_me, qui permet d'obtenir les informations de base d'un utilisateur.
- friends_about_me, qui permet d'obtenir les informations de base des amis de l'utilisateur.
- user_birthday, qui permet d'obtenir la date de naissance d'un utilisateur.
- friends_birthday, qui permet d'obtenir la date de naissance des amis de l'utilisateur.
- publish_stream, qui va me permettre de poster un message sur le mur de l'utilisateur.

Attention, demander ces permissions n'est pas anodin. Cela permet d'accéder aux informations personnelles de l'utilisateur. On ne peut pas faire cela dans son dos, il faut que l'utilisateur autorise notre application à accéder à ces informations. Cela se fait juste après la connexion de l'utilisateur (nous verrons une copie d'écran illustrant ce propos). Ce qui fait que si on demande trop d'informations, notre utilisateur risque de prendre peur et de rejeter nos demandes. On se retrouvera ainsi sans la moindre possibilité d'exploiter les données de l'utilisateur. Il faut donc que nos demandes de permissions soient le plus proche possible des besoins de l'application !

Puis nous fournissons l'URL de redirection Facebook ainsi que l'identifiant de notre application, dans **le paramètre client_id**. Ainsi, après l'approbation de nos permissions par l'utilisateur, la relation de confiance entre le compte Facebook et notre

application Facebook va pouvoir se créer.

Enfin, nous instancions un objet FacebookClient. La méthode GetLoginUrl() va nous retourner l'URL de la page de connexion adéquate sur Facebook afin que notre utilisateur puisse s'y connecter avec son compte. Cette méthode ne fait rien d'extraordinaire à part concaténer tous les paramètres. Nous nous retrouvons avec une URL de la sorte :

Code : Autre

```
https://www.facebook.com/dialog/oauth?  
response_type=token&display.touch&scope=user_about_me%20friends_about_me%20
```

et il ne reste plus qu'à naviguer sur cette URL avec le WebBrowser...
Nous arrivons sur une page ressemblant à celle présentée dans la figure suivante.



Connexion à Facebook

C'est bien une page du site de Facebook. C'est donc Facebook qui nous authentifie grâce à sa fenêtre de login. Entrons nos informations de connexion et validons. Une fois que l'on est connecté, Facebook nous demande si nous acceptons la demande de permission de l'application Facebook (voir prochaine figure).



Installation de l'application

Une fois la demande de permission acceptée, nous sommes redirigés sur une page blanche marquée success.

Parfait tout ça... mais, le jeton ? Comment on le récupère ?



Eh bien cela se fait dans l'événement de navigation auquel nous nous sommes abonnés au début. Dans cet événement, nous allons utiliser la méthode TryParseOAuthCallbackURL de la classe FacebookOAuthResult pour extraire le jeton :

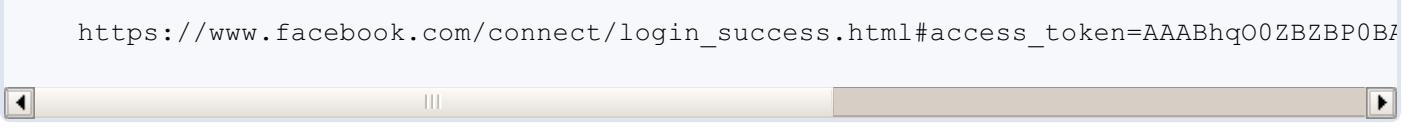
Code : C#

```
private string token;

private void wb_Navigated(object sender, NavigationEventArgs e)
{
    FacebookOAuthResult result;
    if (client.TryParseOAuthCallbackUrl(e.Uri, out result))
    {
        if (result.IsSuccess)
        {
            token = result.AccessToken;
        }
        wb.Visibility = Visibility.Collapsed;
    }
}
```

En fait, cette méthode TryParseOAuthCallbackURL ne fait rien de compliqué. Elle récupère juste l'URL dans laquelle le jeton est disponible. Voici l'allure de cet URL :

Code : Autre



https://www.facebook.com/connect/login_success.html#access_token=AAABhqO0ZBZBP0BZ...



Remarque : il est normal que vous n'ayez pas besoin de ressaisir vos informations à chaque fois, le navigateur de l'émulateur conserve les cookies de session.

Toujours est-il qu'une fois que l'URL de ce jeton est récupérée, nous allons pouvoir faire tout ce que nous voulons. Chouette. Commençons par masquer le WebBrowser, nous n'en aurons plus besoin.

À ce moment-là, je trouve qu'il est plus propre de changer de page en stockant le jeton dans le dictionnaire d'état et en le récupérant à la page suivante.

Code : C#

```
private void wb_Navigated(object sender, NavigationEventArgs e)
{
    FacebookOAuthResult result;
    if (client.TryParseOAuthCallbackUrl(e.Uri, out result))
    {
        if (result.IsSuccess)
        {
            PhoneApplicationService.Current.State["Jetton"] =
result.AccessToken;
        }
        wb.Visibility = Visibility.Collapsed;
        NavigationService.Navigate(new Uri("/Page1.xaml",
UriKind.Relative));
    }
}
```

Remarquez que le jeton a une durée de vie limitée. Il est possible de le renouveler régulièrement avec une requête qui nous retourne un nouveau jeton, voire le même si la requête a déjà été faite dans la journée. Il s'agit de la requête suivante, que l'on peut faire lorsque l'on arrive sur la Page1.xaml :

Code : C#

```
public partial class Page1 : PhoneApplicationPage
{
    private FacebookClient facebookClient;
    private string token;

    public Page1()
    {
        InitializeComponent();
    }

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        token = (string)PhoneApplicationService.Current.State["Jetton"];
        facebookClient = new FacebookClient(token);
        facebookClient.GetCompleted += facebookClient_GetCompleted;

        Dictionary<string, object> parameters = new Dictionary<string,
object>();
        parameters["client_id"] = "votre clé API";
        parameters["client_secret"] = "votre clé secrète";
        parameters["grant_type"] = "fb_exchange_token";
        parameters["fb_exchange_token"] = facebookClient.AccessToken;
```

```
facebookClient.GetAsync("https://graph.facebook.com/oauth/access_token",
parameters, "MisAJourToken");

    base.OnNavigatedTo(e);
}
}
```

On pourra extraire le nouveau token lorsque l'événement GetCompleted est levé :

Code : C#

```
private void facebookClient_GetCompleted(object sender,
FacebookEventArgs e)
{
    if (e.Error == null)
    {
        if ((string)e.UserState == "MisAJourToken")
        {
            JsonObject data = (JsonObject)e.GetResultData();
            token = (string)data["access_token"];
            facebookClient.AccessToken = token;
            PhoneApplicationService.Current.State["token"] = token;
        }
    }
}
```

C'est une bonne idée de lancer une requête de ce genre à chaque connexion de l'utilisateur afin de prolonger la durée de vie du jeton, ou pourquoi pas dans une tâche périodique que nous avons vu dans la partie précédente...



Oui mais moi, j'aime bien les méthodes asynchrones async et await ...

Qu'à cela ne tienne, il suffit de se faire une petite méthode d'extension :

Code : C#

```
public static class Extensions
{
    public static Task<JsonObject> GetAsyncEx(this FacebookClient
facebookClient, string uri, object parameters)
    {
        TaskCompletionSource<JsonObject> taskCompletionSource = new
TaskCompletionSource<JsonObject>();
        EventHandler<FacebookEventArgs> getCompletedHandler =
null;
        getCompletedHandler = (s, e) =>
        {
            facebookClient.GetCompleted -= getCompletedHandler;
            if (e.Error != null)
                taskCompletionSource.TrySetException(e.Error);
            else
                taskCompletionSource.TrySetResult((JsonObject)e.GetResultData());
        };
        facebookClient.GetCompleted += getCompletedHandler;
        facebookClient.GetAsync(uri, parameters);

        return taskCompletionSource.Task;
    }
}
```

```
}
```

On pourra remplacer le code précédent par :

Code : C#

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
{
    token = (string)PhoneApplicationService.Current.State["Jeton"];
    facebookClient = new FacebookClient(token);

    Dictionary<string, object> parameters = new Dictionary<string,
object>();
    parameters["client_id"] = "votre id d'application";
    parameters["client_secret"] = "votre clé d'application";
    parameters["grant_type"] = "fb_exchange_token";
    parameters["fb_exchange_token"] = facebookClient.AccessToken;

    try
    {
        JsonObject data = await
facebookClient.GetAsyncEx("https://graph.facebook.com/oauth/access_token",
parameters);
        token = (string)data["access_token"];
        facebookClient.AccessToken = token;
        PhoneApplicationService.Current.State["token"] = token;
    }
    catch (Exception)
    {
        MessageBox.Show("Impossible de renouveler le token");
    }

    base.OnNavigatedTo(e);
}
```

Pratique, non ?

Exploiter le graphe social avec le SDK

Le graphe social représente le réseau de connexions et de relations entre les utilisateurs sur Facebook. Les connexions peuvent être entre les utilisateurs (amis, famille,...) ou entre des objets via des actions (un utilisateur **aime** une page, un utilisateur **écoute** de la musique,...). Le graphe social se base sur le protocole **Open Graph** pour modéliser ces relations et ces actions, l'action la plus connue étant le fameux « J'aime ».

L'API du graphe permet d'exploiter ces informations. Cette API est utilisable en REST mais encore une fois, le package SDK propose une façon de simplifier son utilisation.

La documentation de référence de cette API est disponible à [cet emplacement](#). Voyons à présent comment s'en servir...

Récupérer des informations

Si vous n'avez pas rafraîchi le token, il va vous falloir instancier un objet FacebookClient avec le jeton passé dans le dictionnaire d'état. Sinon, vous avez déjà tout ce qu'il faut et vous êtes prêt à interroger l'API du graph social avec une nouvelle requête. Commençons par quelque chose de simple : récupérer des informations sur l'utilisateur en cours.

Premièrement, le XAML. Nous allons afficher l'image de l'utilisateur, son nom et prénom, ainsi que sa date de naissance :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid>
        <Grid.ColumnDefinitions>
```

```

        <ColumnDefinition Width="100" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
    </Grid.RowDefinitions>
    <Image x:Name="ImageUtilisateur" Grid.RowSpan="2" />
    <TextBlock x:Name="NomUtilisateur" Grid.Column="1" />
    <TextBlock x:Name="DateNaissanceUtilisateur" Grid.Row="1"
        Grid.Column="1" />
</Grid>
</Grid>

```

Pour obtenir ces infos, nous allons interroger le graphe social. On utilise pour cela la méthode Get de l'objet FacebookClient, que nous avons déjà utilisée (ou plutôt la méthode d'extension que nous avons créée). Le principe est de faire un appel REST à la ressource <https://graph.facebook.com/me> et de récupérer le résultat. Ce résultat s'obtient avec la méthode GetResultData qui retourne un JsonObject. Nous pourrons alors accéder aux informations contenues dans cet objet, comme l'identifiant, le nom, le prénom ou la date de naissance.

Code : C#

```

try
{
    JsonObject data = await
facebookClient.GetAsyncEx("https://graph.facebook.com/me", null);
    string id = (string) data["id"];
    string prenom = (string) data["first_name"];
    string nom = (string) data["last_name"];
    Dispatcher.BeginInvoke(() =>
    {
        ImageUtilisateur.Source = new BitmapImage(new
Uri("https://graph.facebook.com/" + id + "/picture"));
        NomUtilisateur.Text = prenom + " " + nom;
        DateNaissanceUtilisateur.Text =
ObtientDateDeNaissance(data);
    });
}
catch (Exception)
{
    MessageBox.Show("Impossible d'obtenir les informations sur
moi");
}

```

Voici le détail de la méthode ObtientDateDeNaissance utilisée pour récupérer la date de naissance de l'utilisateur :

Code : C#

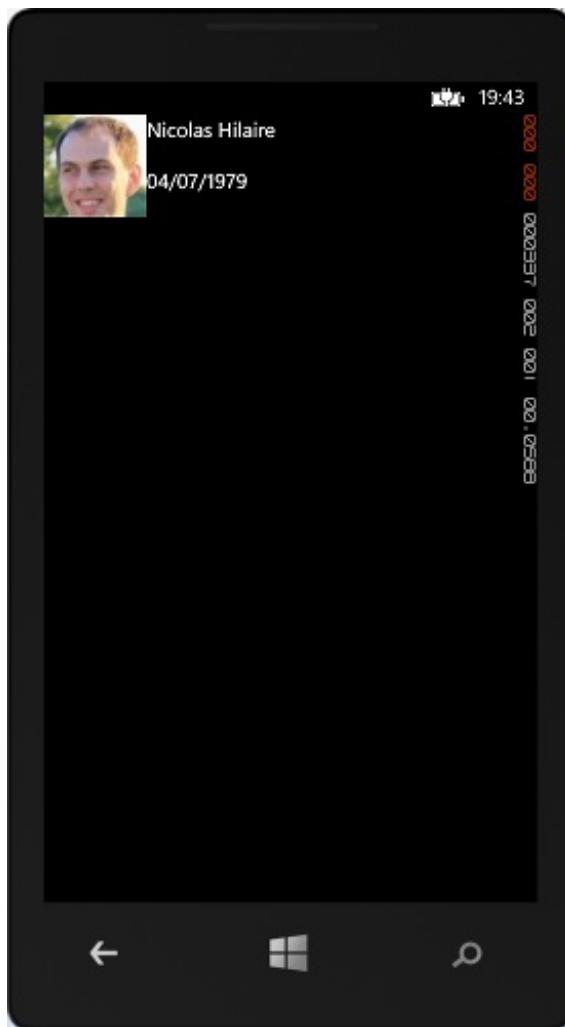
```

private string ObtientDateDeNaissance(JsonObject data)
{
    if (data.ContainsKey("birthday"))
    {
        DateTime d;
        CultureInfo enUS = new CultureInfo("en-US");
        if (DateTime.TryParseExact((string) data["birthday"],
"MM/dd/yyyy", enUS, System.Globalization.DateTimeStyles.None, out
d))
            return d.ToShortDateString();
    }
    return string.Empty;
}

```

Remarquez que vous n'aurez une valeur dans le « champ de date de naissance » que si vous avez demandé la permission user_birthday, d'où la vérification de l'existence de la clé avant son utilisation.

Remarquez que l'image d'une personne s'obtient grâce à son identifiant : https://graph.facebook.com/id_utilisateur/picture.



Récupération de mes informations

Obtenir la liste de ses amis

Nous allons en profiter pour faire la même chose avec les amis de l'utilisateur, nous allons afficher leurs noms et leurs dates de naissance.

C'est aussi simple que précédemment, il suffit d'invoquer la ressource située à <https://graph.facebook.com/me/friends>. Cette requête nous permet d'obtenir l'identifiant et le nom des amis de l'utilisateur, sous la forme d'un tableau JSON. Sauf que ce n'est pas suffisant, nous souhaitons obtenir leurs anniversaires. Il faut appeler la ressource suivante :

https://graph.facebook.com/id_utilisateur pour obtenir des informations complémentaires. Ce que nous allons faire de ce pas... J'en profite pour rajouter une liste d'utilisateurs sous forme d'une collection ObservableCollection dans ma classe :

Code : C#

```
public ObservableCollection<Utilisateur> UtilisateurList { get; set; }
```

Cette collection est composée d'objets représentant chaque utilisateur. Chaque objet Utilisateur est défini de la manière suivante :

Code : C#

```
public class Utilisateur
{
    public string Id { get; set; }
```

```

    public string Nom { get; set; }
    public BitmapImage Image { get; set; }
    public string DateNaissance { get; set; }
}

```

Je vais compléter les informations de chaque objet Utilisateur grâce aux informations reçues par l'API. Je démarre donc la requête permettant d'avoir la liste de mes amis. Une fois la liste reçue, j'extrais la liste des identifiants de chaque ami et je réinterroge l'API pour avoir le détail de chaque utilisateur. Une fois le détail reçu, je l'ajoute à mon ObservableCollection.

Code : C#

```

try
{
    JsonObject data = await
facebookClient.GetAsyncEx("https://graph.facebook.com/me/friends",
null);
    JSONArray friends = (JSONArray) data["data"];
    foreach (JsonObject f in friends)
    {
        string id = (string) f["id"];
        data = await
facebookClient.GetAsyncEx("https://graph.facebook.com/" + id, null);
        string name = (string) data["name"];
        Dispatcher.BeginInvoke(() => UtilisateurList.Add(new
Utilisateur { Id = id, Nom = name, Image = new BitmapImage(new
Uri("https://graph.facebook.com/" + id + "/picture")), DateNaissance
= ObtientDateDeNaissance(data) }));
    }
}
catch (Exception)
{
    MessageBox.Show("Impossible d'obtenir les informations sur les
amis");
}

```

Ouf!

Vous n'aurez bien sûr pas oublié de faire les initialisations adéquates :

Code : C#

```

public Page1()
{
    InitializeComponent();

    UtilisateurList = new ObservableCollection<Utilisateur>();
    DataContext = this;
}

```

N'oubliez pas non plus de déclarer l'affichage de notre liste dans une ListBox dans la page XAML :

Code : XML

```

<ListBox ItemsSource="{Binding UtilisateurList}" Grid.Row="1" >
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="*" />

```

```
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="auto" />
    <RowDefinition Height="auto" />
</Grid.RowDefinitions>
<Image Source="{Binding Image}" Grid.RowSpan="2" />
<TextBlock Text="{Binding Nom}" Grid.Column="1" />
<TextBlock Text="{Binding DateNaissance}" Grid.Row="1" Grid.Column="1" />
</Grid>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
```

Et voilà, nous pouvons afficher la liste de nos amis. Par respect pour les miens, je ne présenterai pas de copie d'écran avec leurs têtes et leurs anniversaires, mais vous pouvez essayer avec les vôtres, cela fonctionne très bien ! 😊

Publier un post sur son mur Facebook

Les connexions du graphe social et autres données de Facebook peuvent être exploitées commercialement. On peut aussi utiliser ces données pour publier des posts sur son mur Facebook. Imaginons par exemple que je développe un jeu pour Windows Phone, je fais un score terrible et je souhaite défier mes amis pour qu'ils tentent de me battre. Rien de tel que de poster un petit message sur mon mur Facebook pour les inciter à venir jouer et à me battre...

C'est ce que nous allons faire ici. Pour l'exemple, je vais reprendre notre jeu du plus ou du moins que nous avions fait en TP (voir Partie 1, Chapitre 8). L'intérêt ici, outre de nous amuser, sera de pouvoir poster son score automatiquement sur son mur.

Je reprends donc le XAML du TP que je mets dans ma Page1.xaml, puis je rajoute le petit bouton permettant de poster mon score sur Facebook.

Voici le XAML :

Code : XML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="TP du jeu du plus ou du moins" Style="{StaticResource PhoneTextTitle2Style}" />
    </StackPanel>

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="Veuillez saisir une valeur (entre 0 et 500)" Style="{StaticResource PhoneTextNormalStyle}" HorizontalAlignment="Center" />
            <TextBox x:Name="Valeur" InputScope="Number" />
            <Button Content="Vérifier" Tap="Button_Tap_1" />
            <TextBlock x:Name="Indications" Height="50" TextWrapping="Wrap" />
            <TextBlock x:Name="NombreDeCoups" Height="50" TextWrapping="Wrap" Style="{StaticResource PhoneTextNormalStyle}" />
            <Button x:Name="BoutonFb" Content="Publier mon score sur Facebook" Tap="Button_Tap" IsEnabled="False" />
        </StackPanel>
        <Button Content="Rejouer" Tap="Button_Tap_2" Grid.Row="2" />
    </Grid>
</Grid>
```

Passons désormais au code-behind :

Code : C#

```
public partial class Page1 : PhoneApplicationPage
{
    private FacebookClient facebookClient;
    private string token;
    private Random random;
    private int valeurSecrete;
    private int nbCoups;

    public Page1()
    {
        InitializeComponent();

        random = new Random();
        valeurSecrete = random.Next(1, 500);
        nbCoups = 0;
        Color couleur =
(Color)Application.Current.Resources["PhoneAccentColor"];
        Indications.Foreground = new SolidColorBrush(couleur);
    }

    protected async override void OnNavigatedTo(NavigationEventArgs e)
    {
        token = (string)PhoneApplicationService.Current.State["Jeton"];
        facebookClient = new FacebookClient(token);

        Dictionary<string, object> parameters = new Dictionary<string,
object>();
        parameters["client_id"] = "id application";
        parameters["client_secret"] = "id API";
        parameters["grant_type"] = "fb_exchange_token";
        parameters["fb_exchange_token"] = facebookClient.AccessToken;

        try
        {
            JsonObject data = await
facebookClient.GetAsync("https://graph.facebook.com/oauth/access_token",
parameters);
            token = (string)data["access_token"];
            facebookClient.AccessToken = token;
            PhoneApplicationService.Current.State["token"] = token;
        }
        catch (Exception)
        {
            MessageBox.Show("Impossible de renouveler le token");
        }
        base.OnNavigatedTo(e);
    }

    private void Button_Tap_1(object sender,
System.Windows.Input.GestureEventArgs e)
    {
        int num;
        if (int.TryParse(Valeur.Text, out num))
        {
            if (valeurSecrete == num)
            {
                Indications.Text = "Gagné !!!";
                BoutonFb.IsEnabled = true;
            }
            else
            {
                nbCoups++;
                if (valeurSecrete < num)
                    Indications.Text = "Trop grand ...";
                else

```

```

        Indications.Text = "Trop petit ...";
    if (nbCoups == 1)
        NombreDeCoups.Text = nbCoups + " coup";
    else
        NombreDeCoups.Text = nbCoups + " coups";
    }
}
else
    Indications.Text = "Veuillez saisir un entier ...";
}

private void Button_Tap_2(object sender,
System.Windows.Input.GestureEventArgs e)
{
    valeurSecrete = random.Next(1, 500);
    nbCoups = 0;
    Indications.Text = string.Empty;
    NombreDeCoups.Text = string.Empty;
    Valeur.Text = string.Empty;
}

private void Button_Tap(object sender,
System.Windows.Input.GestureEventArgs e)
{
    // à faire, publier sur facebook
}
}

```

Il n'a rien de transcendant, c'est comme le TP mixé à ce qu'on a vu sur Facebook juste avant. Lorsque nous trouvons la bonne valeur, on change la valeur de la propriété IsEnabled du bouton pour publier sur Facebook. Celui-ci devra construire le message et poster sur notre mur. Pour cela, on utilisera la méthode PostAsync et pourquoi pas une méthode d'extension asynchrone que nous pouvons rajouter à notre classe statique d'extensions :

Code : C#

```

public static Task<JsonObject> PostAsyncEx(this FacebookClient
facebookClient, string uri, object parameters)
{
    TaskCompletionSource<JsonObject> taskCompletionSource = new
TaskCompletionSource<JsonObject>();
    EventHandler<FacebookEventArgs> postCompletedHandler = null;
postCompletedHandler = (s, e) =>
{
    facebookClient.PostCompleted -= postCompletedHandler;
    if (e.Error != null)
        taskCompletionSource.TrySetException(e.Error);
    else

taskCompletionSource.TrySetResult((JsonObject)e.GetResultData());
};

facebookClient.PostCompleted += postCompletedHandler;
facebookClient.PostAsync(uri, parameters);

return taskCompletionSource.Task;
}

```

Nous pourrons alors poster sur le mur en passant un message dans la propriété message, puis en envoyant tout ça en REST sur l'adresse du mur. Une fois que l'envoi est terminé, on est en mesure de déterminer si l'envoi est bien passé ou pas :

Code : C#

```
private async void Button_Tap(object sender,
```

```
System.Windows.Input.GestureEventArgs e)
{
    string message = "Message correctement posté";
    try
    {
        Dictionary<string, object> parameters = new
Dictionary<string, object>();
        parameters["message"] = "Je viens de trouver le nombre
secret en " + nbCoups;
        await
facebookClient.PostAsyncEx("https://graph.facebook.com/me/feed",
parameters);
    }
    catch (Exception)
    {
        message = "Impossible de poster le message";
    }
    Dispatcher.BeginInvoke(() =>
    {
        MessageBox.Show(message);
    });
}
```

Et voilà !



Le message est posté sur Facebook depuis l'application Windows Phone

Côté Facebook, nous pouvons constater l'apparition du message sur mon mur comme je vous le montre dans la dernière figure.



Le message est affiché sur mon mur

Utiliser les tasks

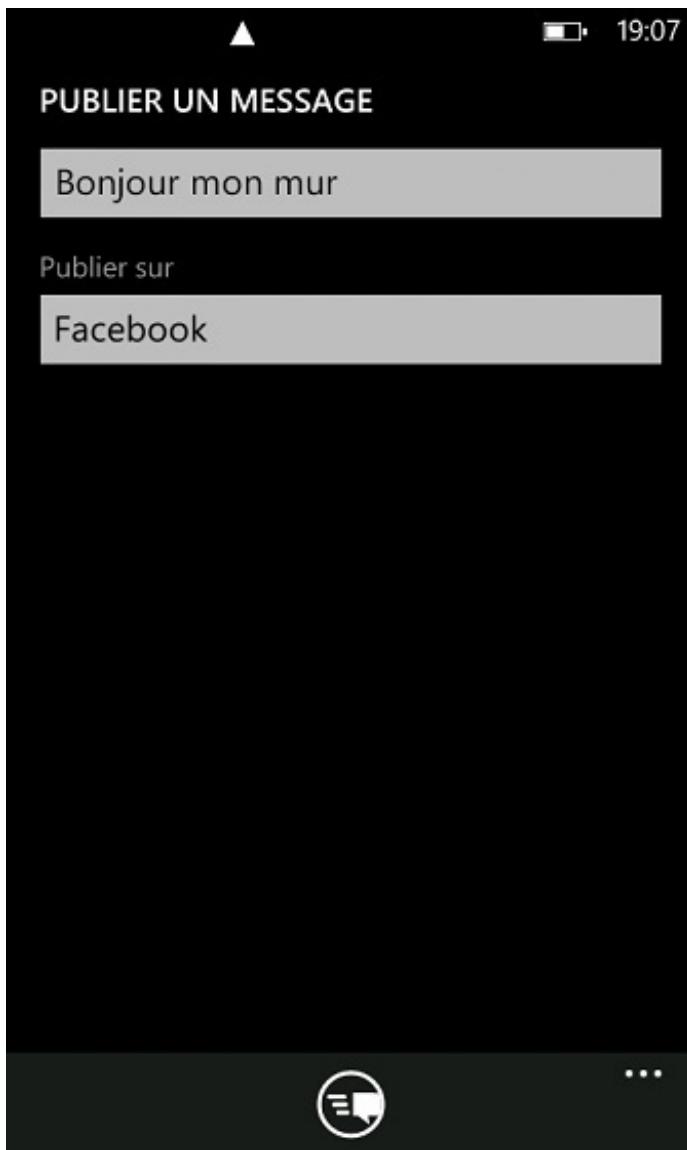
Il existe une autre solution pour poster un message sur son mur. Très simple, sans avoir besoin d'une application Facebook, à partir du moment où vous avez configuré votre téléphone avec votre compte Facebook, ce qui est souvent le cas. Il suffit ensuite d'utiliser un launcher spécifique du téléphone.

Poster un message sur son mur se fait en quelques lignes de code, par exemple pour mettre à jour son statut :

Code : C#

```
ShareStatusTask shareStatusTask = new ShareStatusTask();
shareStatusTask.Status = "Bonjour mon mur";
shareStatusTask.Show();
```

Évidemment, il demande une confirmation à l'utilisateur. Vous ne pourrez pas le voir sur l'émulateur, alors je vous montre dans la figure suivante une copie de mon téléphone.



Partage de statut via le launcher

Ce n'est cependant pas vraiment un partage Facebook, c'est un partage pour tous les réseaux sociaux configurés du téléphone, et on peut en l'occurrence choisir Facebook. C'est ce que j'ai fait ici dans la deuxième zone.
Il est également très facile de publier un lien (voir code et illustration qui suivent).

Code : C#

```
ShareLinkTask shareLinkTask = new ShareLinkTask();  
  
shareLinkTask.Title = "Mon livre pour apprendre le C#";  
shareLinkTask.LinkUri = new Uri("http://boutique.fr.openclassrooms.com/boutique-614-797-apprenez-a-developper-en-c.html", UriKind.Absolute);  
shareLinkTask.Message = "A lire absolument ...";  
shareLinkTask.Show();
```



Grâce à ces deux launchers, il devient très facile de publier un statut ou un lien, même si on est fatalement un peu plus limité car on ne peut que poster sur son mur et pas par exemple sur le mur de nos amis pour leur souhaiter un bon anniversaire.

Nous avons vu comment lire des informations dans le graphe social et comment y poster un message. Il y a beaucoup d'autres informations intéressantes dans ce graphe social. Des applications à but commercial pourraient les exploiter avec intérêt. Étant donné qu'il est très facile de récupérer ce que vous aimez (les « j'aime »), il pourrait être très facile de vous proposer des produits en adéquation avec ce que vous aimez. Les entreprises d'e-commerce ne s'y trompent pas et essayent de vous attirer sur leurs

applications Facebook. À partir du moment où vous autorisez l'accès à vos informations, vous avez pu voir comme il est simple de les récupérer. Cela ouvre beaucoup de possibilités pour rendre vos applications Windows Phone sociales et dynamiques.

- Il faut créer une application Facebook afin de pouvoir interagir avec Facebook depuis notre application Windows Phone.
- On utilise le contrôle WebBrowser pour établir l'authentification OAuth.
- Les permissions, si elles sont acceptées, nous permettent d'accéder aux informations du graphe social.
- Le SDK nous simplifie l'accès aux ressources REST du graphe social.
- On peut utiliser des launchers pour partager très facilement des informations sur les réseaux sociaux.

Publier son application

Ça y est, votre application est prête. L'émulateur vous a bien aidé dans votre phase de développement et vous vous sentez prêts pour passer à la suite. Nous allons voir dans ce chapitre comment faire pour tester son application sur un téléphone et pour soumettre son application sur le Windows Phone Store.

Mais la première chose indispensable à faire est de tester son application sur un vrai téléphone. Même si l'émulateur est capable de simuler pas mal de choses, c'est quand même le vrai téléphone qui va être capable de se rapprocher au maximum de ce que vont avoir les utilisateurs au quotidien, forcément... 😊

Ensuite, cela permet de valider plein de petites choses que l'on ne voit pas forcément dans l'émulateur (comme les images de l'application).

Enfin, et c'est presque le plus important, cela permet de valider la performance de notre application.

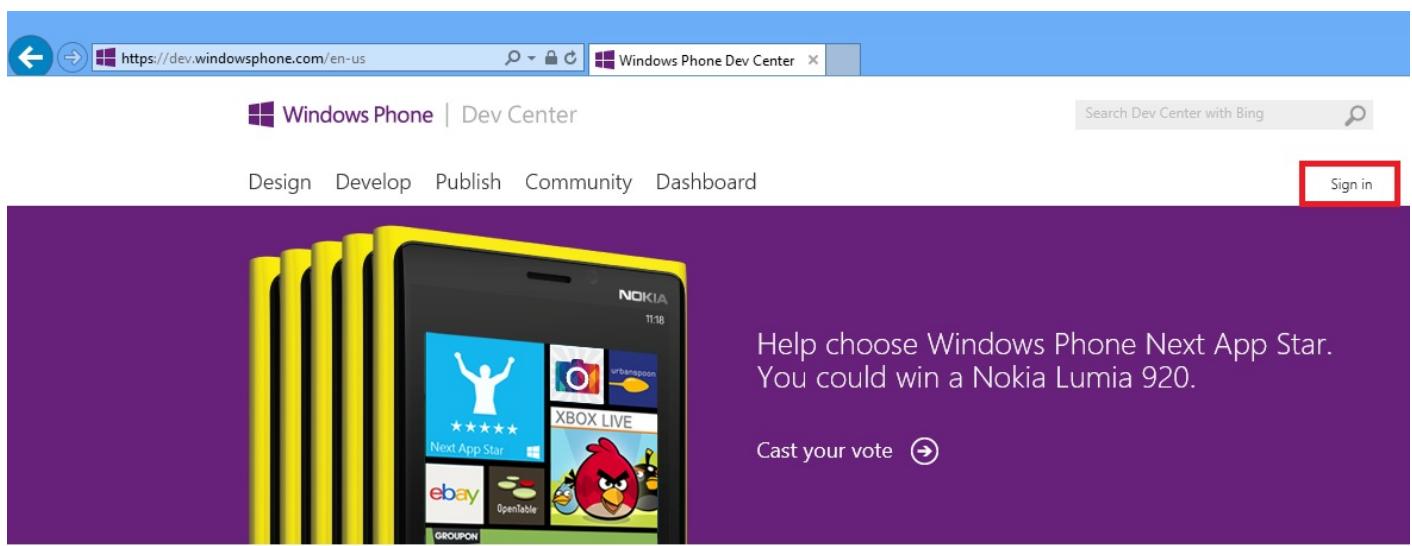
En effet, un téléphone est généralement bien moins puissant qu'un ordinateur de développeur. Il m'est souvent arrivé de constater que mon application était super fluide dans l'émulateur, mais sur mon téléphone, celle-ci était inutilisable tellement elle était lente.

Pour la terminer correctement, il faut donc déployer son application sur un téléphone.

Créer un compte développeur

La première chose à faire est de créer un compte développeur. C'est une opération qui coûte 19\$ par an (14€). Elle est indispensable pour déployer son application sur un téléphone et pour pouvoir publier sur le Windows Phone Store.

Rendez-vous donc sur <https://dev.windowsphone.com/>. Le souci, c'est que pour l'instant, c'est en anglais, donc si vous n'êtes pas trop anglophone, suivez le guide. On doit dans un premier temps créer un compte en cliquant sur « Sign in » (voir figure suivante).



Note : Il est possible que les images ne reflètent pas exactement ce que vous voyez car le site est en perpétuelle évolution. Mais ne vous inquiétez pas, l'idée est là !

Il faut ensuite s'authentifier avec son compte Windows Live (voir figure suivante).

The screenshot shows the 'Sign in to Dev Center' page. At the top, there's a navigation bar with back and forward buttons, a search icon, and a Microsoft account status. Below it, the title 'Windows Phone | Dev Center' is displayed. The main content area has a heading 'Sign in to Dev Center' and a sub-instruction: 'We're glad you're here! Sign in with your Microsoft account info to access your personalized Dashboard.' To the right, a red box highlights the sign-in form. It includes fields for 'Microsoft account' (with placeholder 'someone@example.com'), 'Password', and a 'Keep me signed in' checkbox. A 'Sign in' button is at the bottom of the form. Below the form, links for 'Can't access your account?' and 'Sign in with a single-use code' are visible.

Authentification avec compte Windows Live

Saisissez votre e-mail et votre mot de passe. Vous êtes redirigés sur la page d'accueil présentée dans la figure suivante, où vous pourrez cliquer sur n'importe quel lien. Tout d'abord, cliquez sur l'onglet « Dashboard ».

The screenshot shows the Windows Phone Dev Center dashboard. At the top, there's a navigation bar with back and forward buttons, a search icon, and a Microsoft account status. Below it, the title 'Windows Phone | Dev Center' is displayed. The dashboard menu includes 'Design', 'Develop', 'Publish', 'Community', and 'Dashboard', with 'Dashboard' highlighted by a red box. The main content area features a large image of several yellow Nokia phones standing upright, showing various app icons like Next App Star, eBay, OpenTable, and Angry Birds. To the right, there are two columns of text: 'Help choose' and 'You could', followed by 'Cast your vote'.

Création d'un compte via l'onglet Dashboard

Nous pouvons alors nous inscrire en cliquant sur le bouton « Join Now » (voir figure suivante).

The screenshot shows the Windows Phone Dev Center website at owsphone.com/en-us/join. The page title is "Join". On the left, there are three buttons: "SUBMIT APP" (with an upward arrow icon), "GET SDK" (with a downward arrow icon), and "VIEW SAMPLES" (with a folder icon). To the right of these buttons, the text reads: "The new Windows Phone Dev Center has everything you need to create great apps for the Windows Phone Store for the world to see, try, and buy. You can start creating apps to offer them in the Store, you'll need a subscription to the Dev Center where you can add them to the Store catalog. Your subscription includes some useful tools and your apps will be tracked so you can track your apps and your earnings." Below this text is a heading "How do I join?". Underneath it, the text says "You need a valid credit card, a PayPal account, or a promo code." A note below states "Your annual subscription is \$99 USD. It's free if you're a DreamSpark student." A large purple button labeled "Join Now" is centered on the page, enclosed in a red rectangular border. Below the "Join Now" button is the heading "What do I get?" followed by the text "Inscription du compte développeur".

Cela nous permet d'obtenir le formulaire d'inscription, où l'on indique notamment son pays, puis son statut (entreprise, particulier ou étudiant).



Il est à noter que l'inscription au Windows Phone Store est gratuite quand on est étudiant.

Bref, de l'administratif, détaillé dans la figure suivante.



Account type [Info](#) [Payment](#) [Thank you](#)

Get started

Welcome to the Windows Phone Dev Center. Your subscription gives you the ability to publish apps in the Windows Phone Store.

To become a member you need a valid credit card, PayPal account, or promo code. Students with a verified DreamSpark account can register for free.

You'll need to provide your contact info, and if you're registering as a company we'll also need your legal entity name.

[Learn more about registration.](#)

Country/region of residence or business*

A dropdown menu icon is visible to the right of the input field.

Account type*

- Company
Select this option if you want to sell apps as a business or government organization.
- Individual or student
Select this option if you want to sell apps as a sole proprietor or using your DreamSpark student account.

Legal Terms*

- By selecting this box, I agree to be bound by the [Windows Phone Store Application Provider Agreement](#).
Further, if accepting on behalf of a company, then I represent that I am authorized to act on my company's behalf.

[Next](#) [Cancel](#)

Les informations à remplir

Toujours dans l'administratif, on a besoin ensuite de saisir nos informations personnelles, nom, prénom, etc., ainsi que notre nom d'auteur sur le Windows Phone Store (voir figure suivante).

Account info

First name*

Last name*

Email address*

Confirm email address*

Phone number*

 ()

Ext.

Website

Address*

Address 1

Address 2

City

Postal code

Saisir ses

Country/Region

France

Publisher info

Your publisher name appears in the Windows Phone Store as the publisher for your apps. The name must be unique and you must have permissions to use the name you pick.

Publisher name*

Check availability**Previous****Next****Cancel**

informations

Passons à la suite. C'est le moment délicat, le paiement de l'abonnement annuel, présenté dans la figure suivante. Il faut choisir le type de règlement, paiement ou bon de réduction.

Country/region

Pick the country/region where you live or where your business is located.

Pick account type

If you want to link this to another Microsoft developer account so they both share the same publisher display name, you must go back and sign in with your other Microsoft account. [Learn more](#)

| Individual | Company |
|--|---|
| <ul style="list-style-type: none">• Develop apps as an individual or a small unincorporated group• Submit Windows Store apps and Windows Phone apps | <ul style="list-style-type: none">• Develop apps as a business, mobile operator, or OEM• Submit Windows Store apps, Windows Phone apps, and desktop apps• Use additional app capabilities |
| Annual price: 14.00 EUR | Annual price: 75.00 EUR |

Mode de règlement

L'abonnement est donc de 14€. Il est possible de régler avec un bon de réduction, par exemple si on a de l'argent sur son compte Xbox. Enfin, pour les étudiants, pour les détenteurs d'un code promotionnel, ou pour les membres inscrits au [programme accélérateur Windows Phone](#), c'est gratuit ! Il ne reste qu'à saisir les informations de facturation.

Et voilà, le compte est désormais créé. Nous recevons un e-mail de confirmation qui va nous permettre d'activer notre compte via un lien. Une fois ce lien cliqué, nous obtenons confirmation de la bonne création du compte.

Et c'est terminé pour cette étape. Nous avons donc associé notre compte Windows Live à un compte Windows Phone Store.

Inscrire un téléphone de développeur

Avant de pouvoir installer son application sur son téléphone, il faut l'enregistrer. Première chose à faire, relier son téléphone à son ordinateur via le câble approprié.

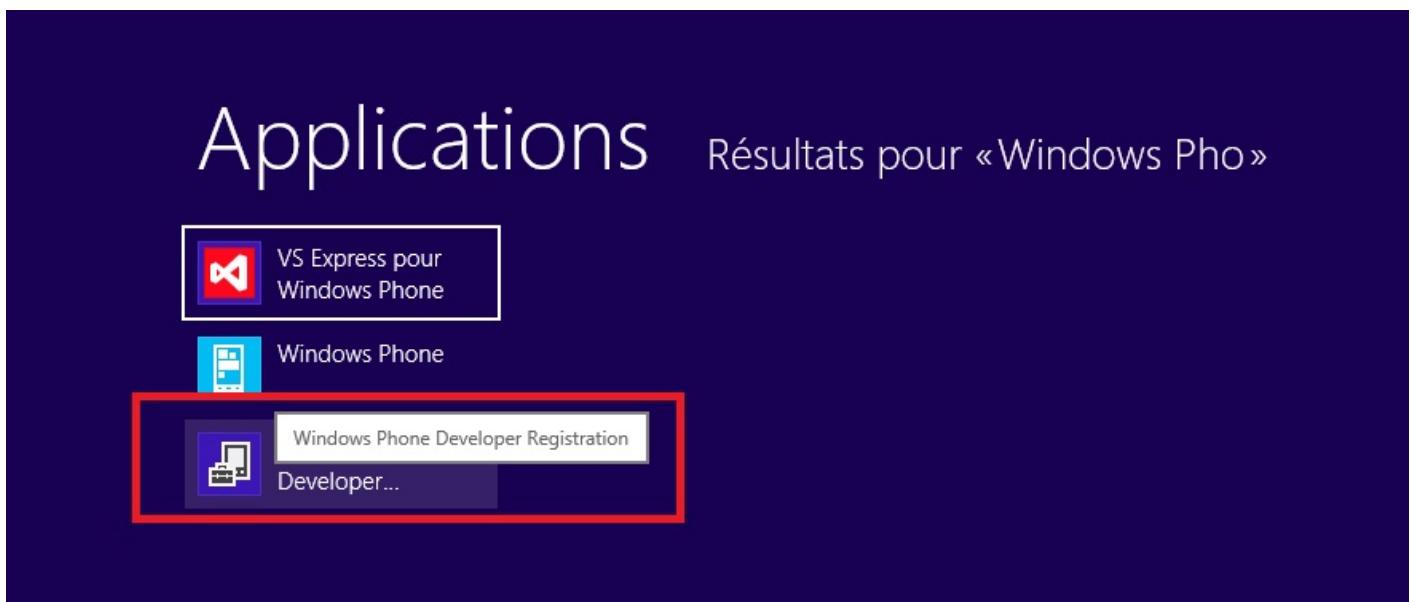


Si vous n'avez pas de téléphone, vous pouvez vous inscrire au [programme accélérateur Windows Phone](#) afin qu'on vous en prête un.

Comme vous êtes sous Windows 8, le programme permettant de synchroniser son Windows Phone est déjà présent sur votre ordinateur. Ce programme s'appelle « Windows Phone » et est nécessaire pour la suite des opérations et pour faire les mises à jour du téléphone. Il existe en version « Modern UI » et en version « Bureau Traditionnel » (vous pouvez utiliser les deux versions, elles ne diffèrent qu'au niveau de l'apparence esthétique de l'interface).

Une fois votre téléphone branché, si vous avez un code de verrouillage protégeant votre téléphone, vous devrez le saisir.

Ensuite, vous devrez enregistrer le téléphone comme étant un téléphone de développeur. Pour cela, démarrez le programme « Windows Phone Developer Registration », accessible par exemple via la barre de recherche de Windows 8 (voir figure suivante).

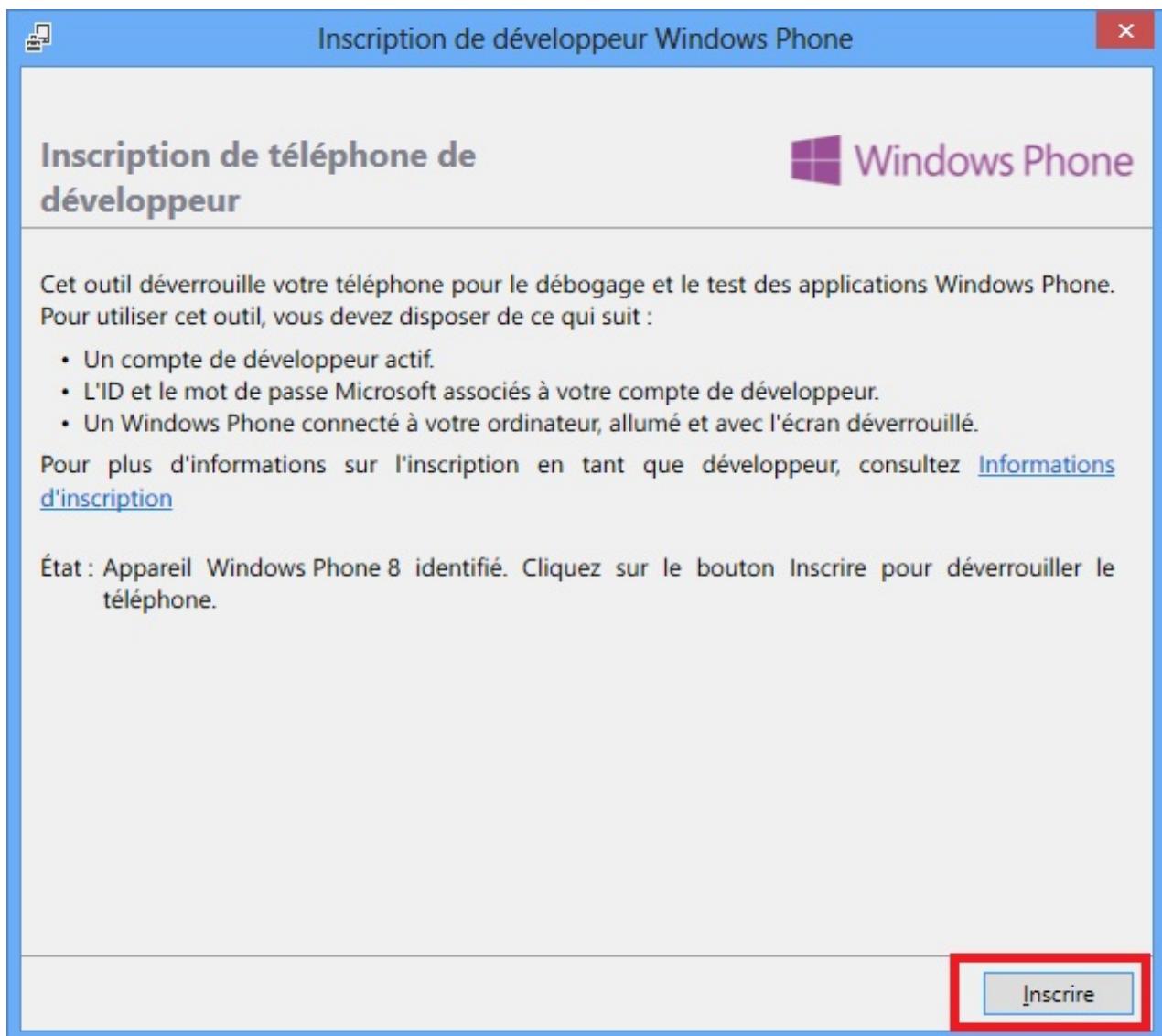


Démarrage de l'application d'enregistrement du téléphone



À noter que vous pourrez enregistrer jusqu'à 5 téléphones différents pour le même compte développeur, ce qui permet de tester son application sur plusieurs appareils.

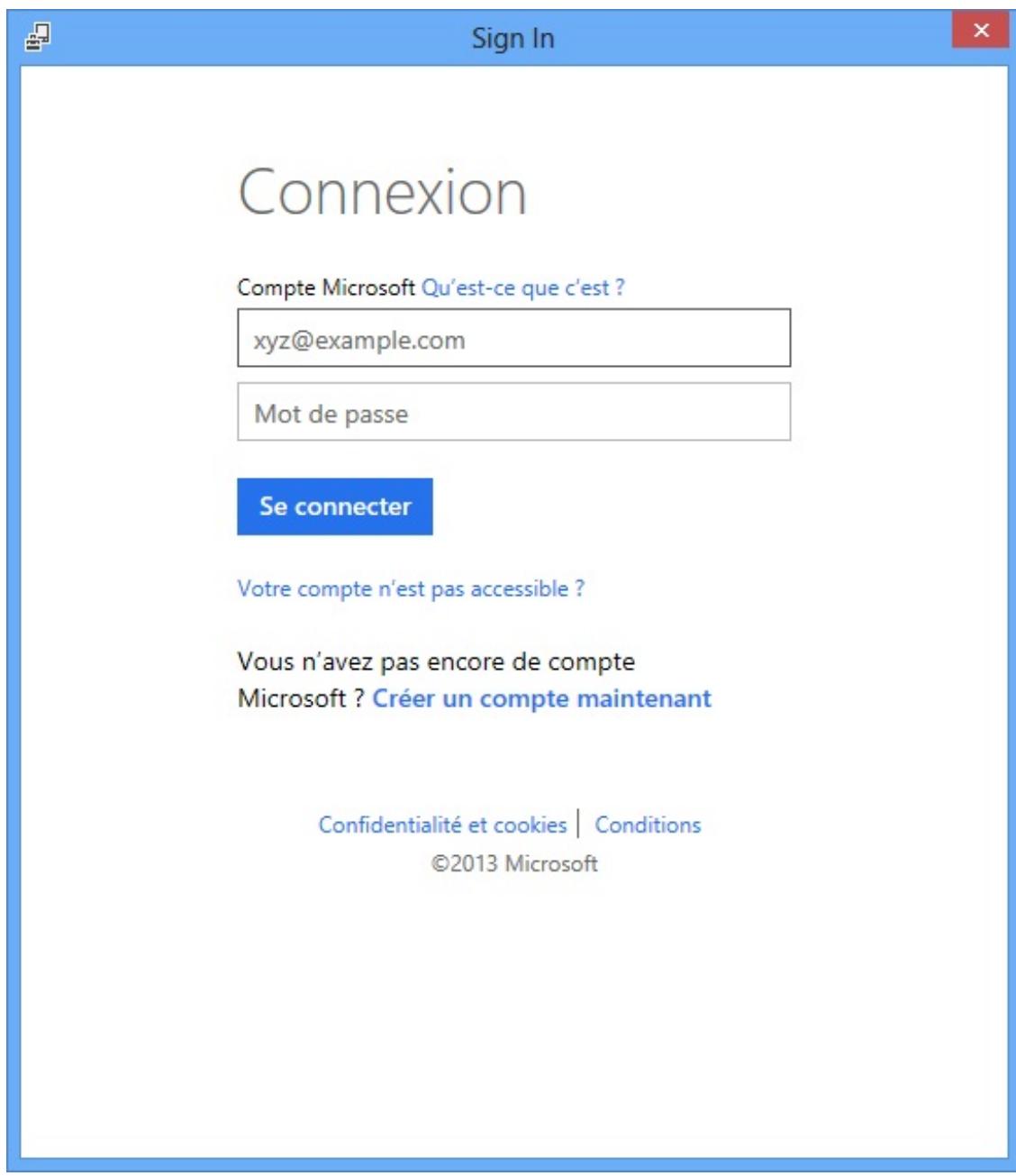
Une fois l'application démarrée, vous pourrez voir l'écran suivant qui permet d'inscrire le téléphone comme étant un téléphone de développeur.



téléphone comme étant un téléphone de développement

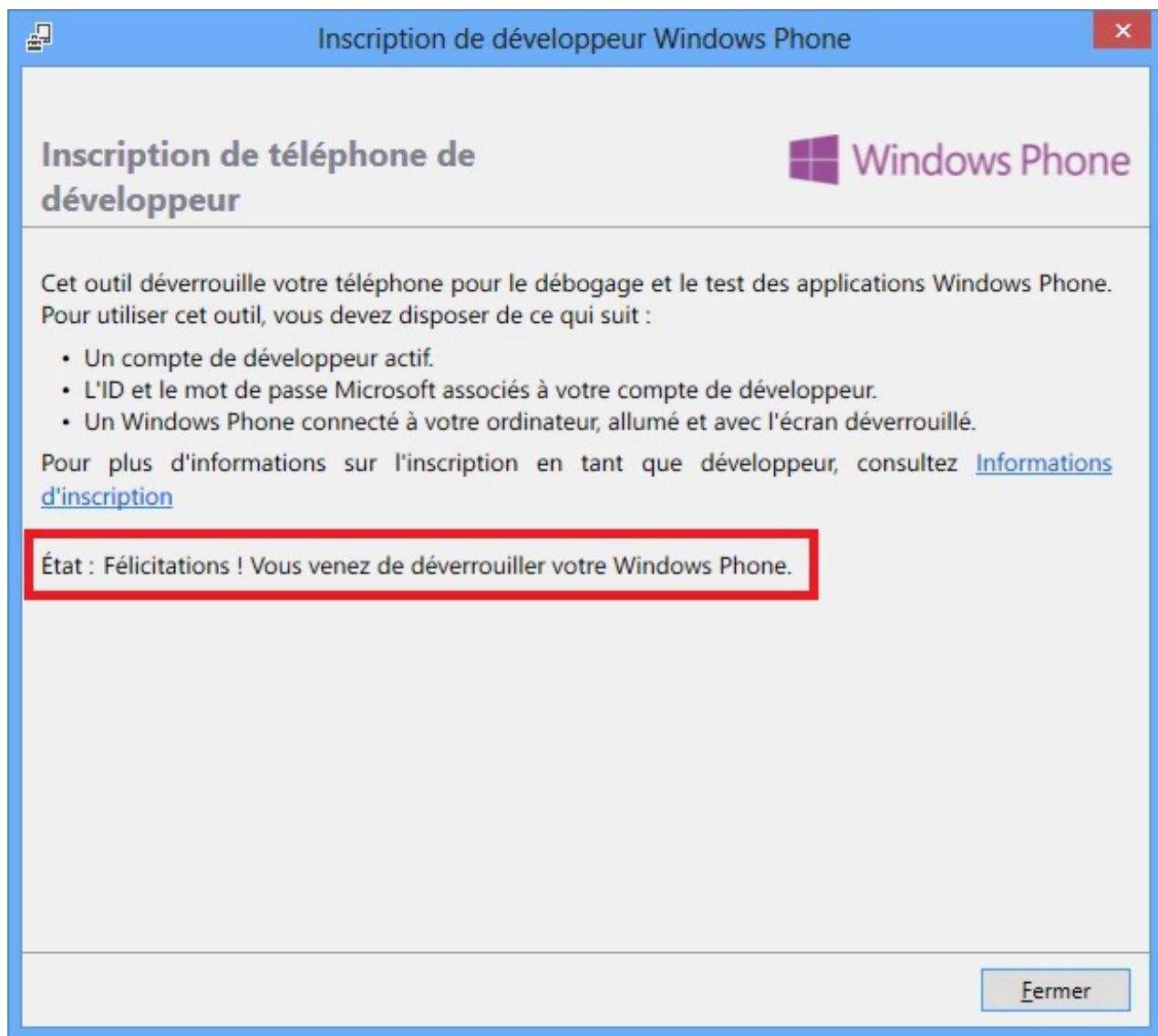
 À noter que vous devrez avoir l'écran de votre téléphone déverrouillé, et que votre PC et que votre téléphone doivent être connectés à Internet. Si toutes ces conditions ne sont pas réunies, vous aurez un bouton « Réessayer ».

 Vous validez bien sûr en cliquant sur « Incrire ». À ce moment-là, vous devez vous connecter à votre compte Windows Live avec lequel vous avez souscrit votre abonnement de développeur (voir figure suivante).



Windows Live

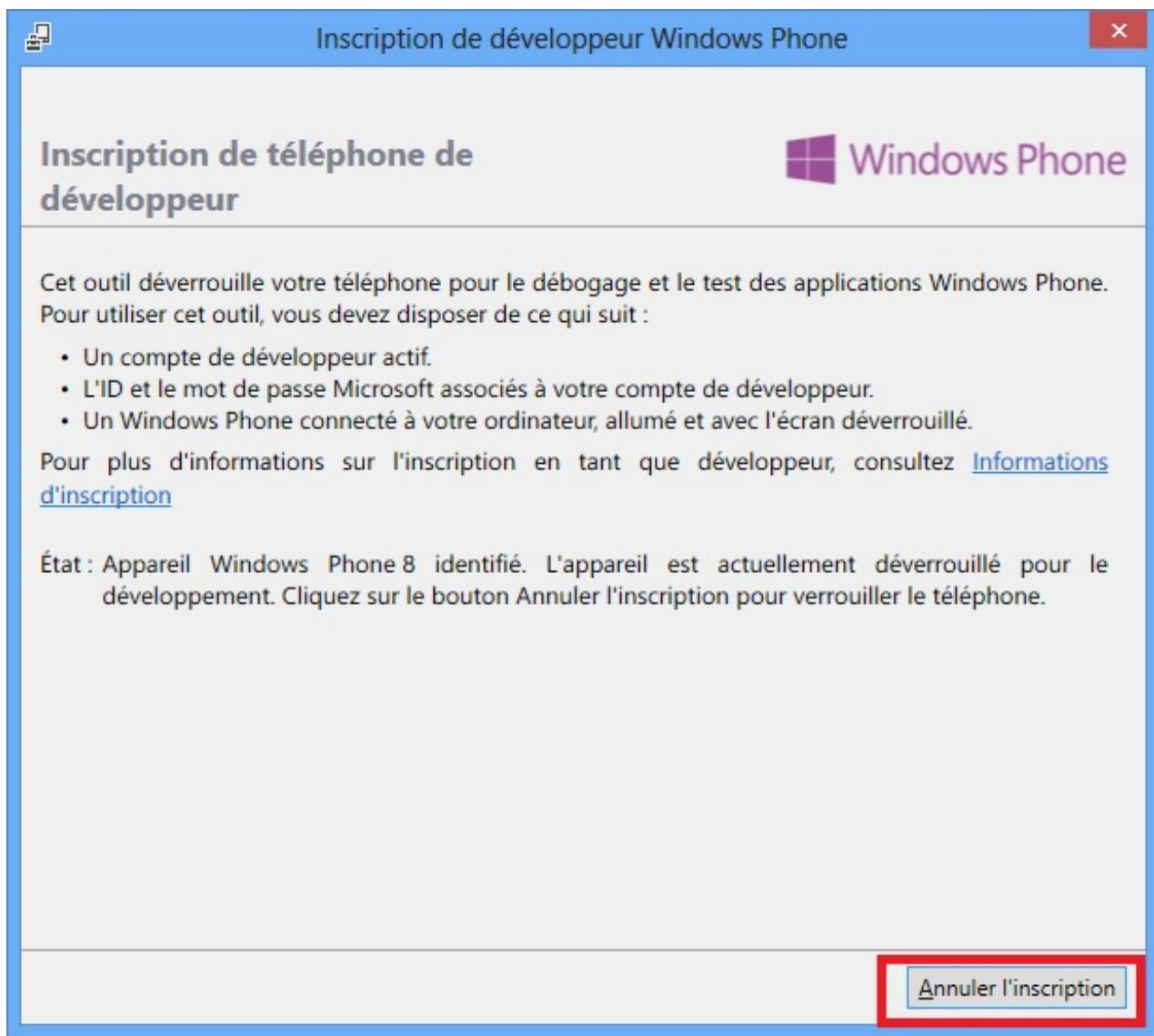
Le programme tente alors de déverrouiller le téléphone. Et voilà, comme vous pouvez le voir dans la figure suivante, c'est fait !



Téléphone

déverrouillé

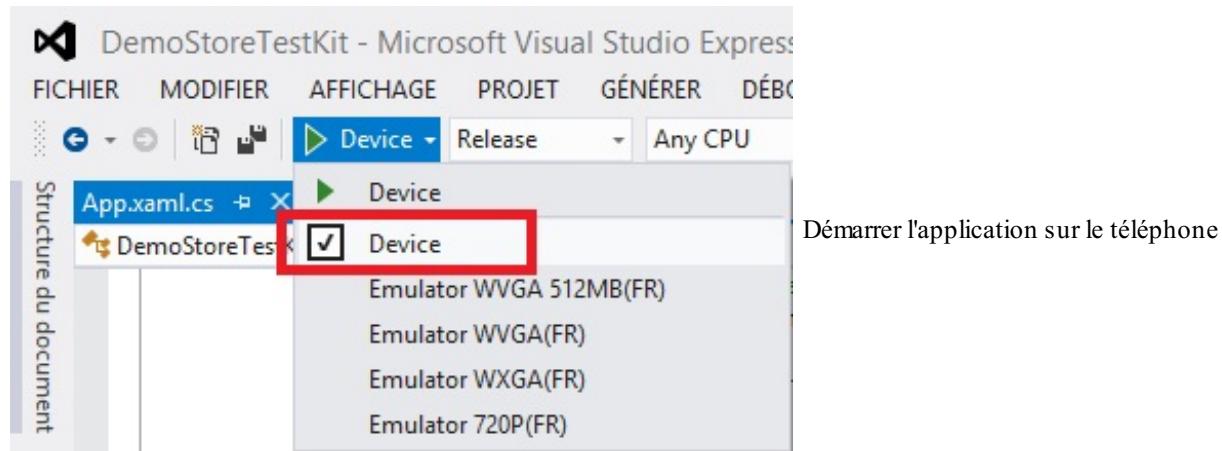
Remarquez que si le téléphone est déjà enregistré, vous pourrez le dés-enregistrer de la même façon (présentée dans la figure suivante).



enregistrement du téléphone

Il est important de le faire si jamais vous voulez vendre votre téléphone ou s'il ne doit plus servir à des fins de débogage. En effet, dans la mesure où vous êtes limités sur le nombre de téléphones utilisables via un compte développeur, il est important de dés-enregistrer un téléphone qui ne servira plus. Mais bon, ce n'est bien sûr pas le moment de la faire.

Voilà, votre téléphone est prêt à recevoir votre application. Pour cela, dans Visual Studio, il faut changer la destination du déploiement dans la liste déroulante où l'émulateur avait été auparavant sélectionné (voir figure qui suit).



Ce qui est formidable, c'est que vous allez pouvoir bénéficier de tout l'environnement de Visual Studio avec votre application directement sur votre téléphone. Cela veut notamment dire avoir accès au débogueur, ce qui vous permet de mettre des points

d'arrêts ou de voir des valeurs de variables alors que vous êtes en train de vous servir de votre téléphone. Bon, ok, il y aura forcément le câble qui relie le PC au téléphone, mais c'est quand même très pratique !
Tellelement pratique qu'une fois que vous y aurez goûté, vous ne voudrez plus revenir sur l'émulateur.
Sur le téléphone, vous vous rendrez mieux compte des problèmes de performance de vos applications. Il sera également plus facile d'utiliser le service de localisation ou de vous déconnecter d'Internet pour faire des tests hors réseau.

Proposer une version d'essai de votre application

Si vous avez un Windows Phone, vous avez pu remarquer que, dans le Windows Phone Store, certaines applications payantes sont disponibles en version d'essai. Ceci permet en général de tester un peu l'application avant de se décider si celle-ci mérite d'être achetée ou non.

Lors de la soumission de votre application, vous pourrez décider du prix de votre application et si elle dispose d'un mode d'essai ou non.

Windows Phone possède une API qui permet de savoir si l'application est en mode essai ou non. C'est la méthode IsTrial de la classe `LicenseInformation`, qui est disponible en incluant l'espace de noms suivant :

Code : C#

```
using Microsoft.Phone.Marketplace;
```

Ainsi, vous pourrez vérifier si l'application est en mode d'essai avec le code suivant :

Code : C#

```
LicenseInformation license = new LicenseInformation();
bool modeEssai = license.IsTrial();
```

Après, libre à vous de faire ce que vous voulez en fonction de la valeur du booléen. Peut-être voudrez-vous limiter le nombre de fonctionnalités ? Peut-être voudrez-vous limiter l'application dans le temps ? Dans tous les cas, une fois la période terminée, vous aurez intérêt à renvoyer l'utilisateur vers le Windows Phone Store afin qu'il achète votre application, comme nous l'avons vu c'est avec le launcher MarketplaceDetailTask.

Remarquez que, pendant la phase de développement, la variable booléenne IsTrial() vaut toujours Vrai. Vous pourrez encapsuler cet appel dans une instruction conditionnelle pour vos différents tests :

Code : C#

```
private bool EstTrial()
{
    #if DEBUG
        return true; // ou false, comme vous voulez ^^
    #else
        var license = new
Microsoft.Phone.Marketplace.LicenseInformation();
        return license.IsTrial();
    #endif
}
```



C'est une très bonne idée de proposer une version d'essai. Achèteriez-vous une voiture les yeux fermés ? À moins qu'elle n'ait une excellente réputation, achèteriez-vous une application sans savoir si elle fonctionne correctement ?

Certifier son application avec le Store Test Kit



Alors, maintenant que nous avons réalisé notre application dans le débogueur et que nous avons validé son fonctionnement sur notre téléphone, nous avons terminé ?

Eh non, pas encore ! Nous avons une superbe application, mais encore faut-il savoir si elle est autorisée à être distribuée sur le Windows Phone Store.

En effet, les applications ne peuvent être distribuées qu'après avoir passé une phase de certification, faite par les équipes

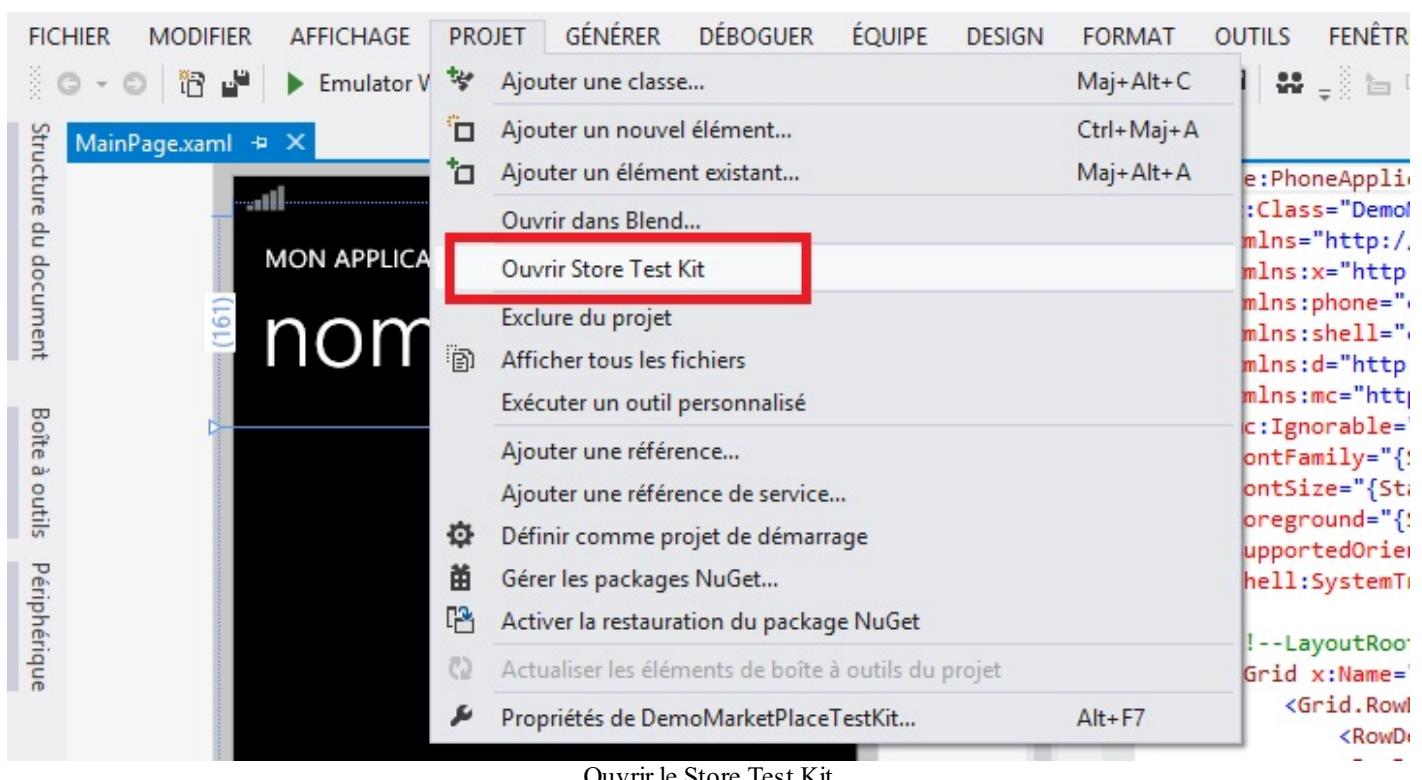
Windows Phone.

Cela permet de vérifier que l'application n'a pas de problèmes (malware, etc.), et qu'elle est suffisamment réactive et fonctionnelle pour l'utilisateur.

Pour nous aider à passer cette certification, Visual Studio dispose d'un outil, le « Store Test Kit ».

Il s'agit d'une série de tests, certains automatisés, certains manuels qui permettent de vérifier certains points en amont de la certification, ce qui va nous permettre de gagner du temps et d'éviter des allers-retours avec l'équipe de Microsoft.

Ce kit est disponible via le menu Projet > Ouvrir Store Test Kit (vous devez avoir sélectionné le projet de démarrage en cliquant dessus), ou bien en faisant un clic droit sur le Projet > Ouvrir Store Test Kit illustré dans la figure suivante.



Ouvrir le Store Test Kit

Visual Studio nous ouvre un écran contenant plusieurs options, y compris celle de faire des tests de l'application pour savoir si elle respecte les besoins pour la certification Windows Phone Store. Il y a des tests automatisés qui vont vérifier si les images sont bonnes, si l'application n'est pas trop grosse, etc. Il y a des tests de performance permettant de voir si l'application n'utilise pas trop de mémoire, si elle démarre assez vite, etc. Enfin, il y a des tests manuels à effectuer (voir liste qui suit).

Structure du document Boîte à outils Périphérique

Store Test Kit MainPage.xaml

Détails de l'application Cliquez sur le bouton Exécuter les tests ci-dessous pour exécuter les cas de test automatisés.

Tests automatisés Exécuter les tests

Tests manuels

Les cas de test n'ont pas encore été exécutés.

| Résultat | Nom du test | Description du test |
|---|--------------------------|---|
| i En attente | Exigences du package XAP | Validation des fichiers de contenu et de la taille du fichier XAP |
| i En attente | Iconographie | Validation des icônes de l'application |
| i En attente | Captures d'écran | Validation des captures d'écran |

L'application est évaluée selon les exigences de la soumission à Windows Store. Utilisez l'analyse de l'application pour déterminer les erreurs et les problèmes. Cliquez sur Démarrer l'analyse de l'application pour lancer une session d'analyse.

Démarrer l'analyse de l'application Windows Phone

Liste des tests à effectuer

Dans le premier onglet de détails de l'application, nous devrons préciser les images dont nous allons avoir besoin lors de la soumission de l'application et qui permettent d'illustrer notre application. Pour réaliser de telles images, vous pouvez utiliser l'émulateur ainsi que l'outil supplémentaire permettant de réaliser des copies d'écran de l'émulateur. Pour faire des bonnes copies d'écran, il faut que les compteurs qui apparaissent sur la droite de l'émulateur ne soient pas visibles. Pour cela, allez modifier la ligne suivante dans le App.xaml.cs :

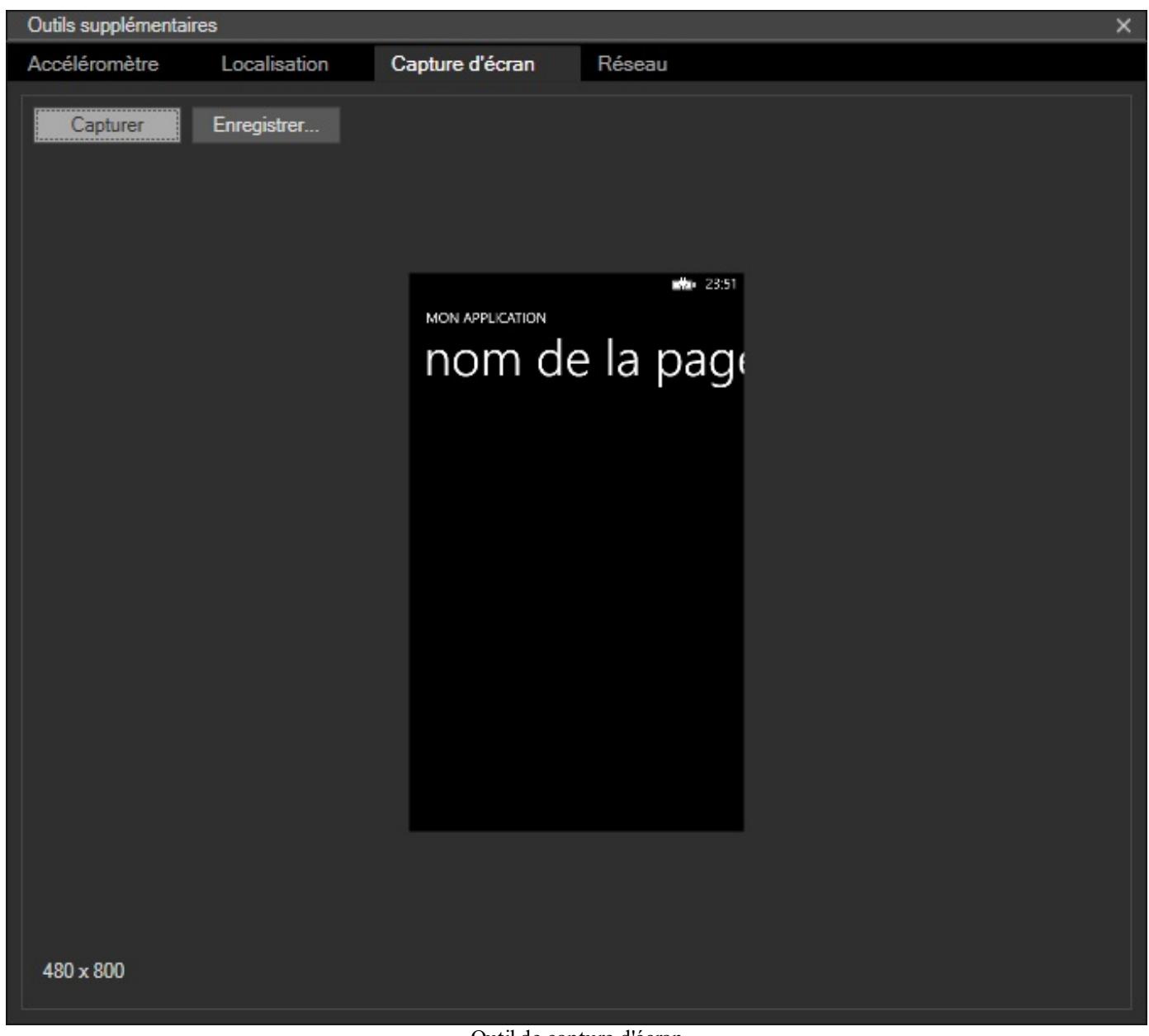
Code : C#

```
Application.Current.Host.Settings.EnableFrameRateCounter = true;
```

Changez la valeur de la propriété de true à false :

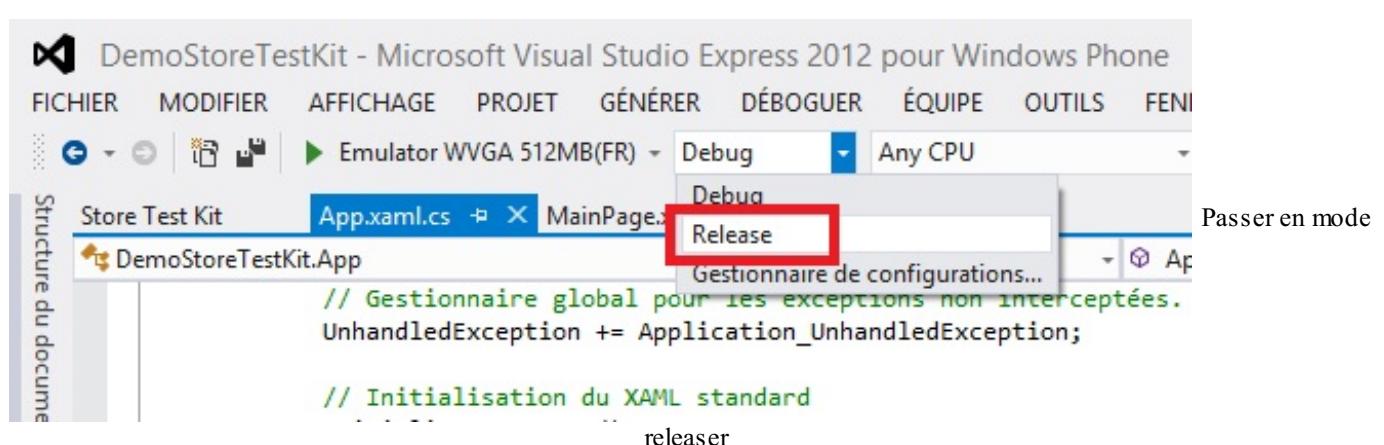
Code : C#

```
Application.Current.Host.Settings.EnableFrameRateCounter = false;
```



Pour avoir des images de bonne qualité, le zoom de l'émulateur doit être en zoom 100%.

Pour lancer les tests automatisés, le fichier .xap doit être compilé en configuration release. Pour cela, il faut modifier la liste déroulante à côté du choix de l'émulateur pour passer de debug à release.



Remarque : à partir de ce moment-là, le fichier .xap sera déployé dans le répertoire nomProjet\Bin\Release



Une fois que le fichier .xap est compilé, nous sommes autorisés à lancer les tests automatisés, ce qui nous permet de constater leurs succès ou leurs échecs affichés dans la prochaine fenêtre.

| Détails de l'application | Cliquez sur le bouton Exécuter les tests ci-dessous pour exécuter les cas de test automatisés. | | |
|--------------------------------|--|---|---|
| Tests automatisés | <button>Exécuter les tests</button> | | |
| Tests manuels | | | |
| Transmis : 1 Échoué : 2 | | | |
| Résultat | Nom du test | Description du test | Détails des résultats |
| ✓ Réussite | Exigences du package XAP | Validation des fichiers de contenu et de la taille du fichier XAP | |
| ✗ Échec | Iconographie | Validation des icônes de l'application | [ERREUR] : Vignette de l'application Windows Store n'est pas requise dans le cadre de la soumission à Windows Store. |
| ✗ Échec | Captures d'écran | Validation des captures d'écran | [ERREUR] : Aucune capture d'écran n'a été spécifiée pour une capture d'écran. [ERREUR] : Aucune capture d'écran n'a été spécifiée pour une capture d'écran. [ERREUR] : Aucune capture d'écran n'a été spécifiée pour une capture d'écran. |

Succès ou échecs des tests automatisés

Par exemple, dans la fenêtre « Succès ou échecs des tests automatisés », mes deux derniers tests ont échoué car je n'avais pas fourni les images que je dois envoyer au Windows Phone Store lors de la soumission. Je dois donc faire mes captures d'écrans dans les bonnes résolutions, puis créer une image de l'application en 300x300 et les positionner dans l'onglet « Détails de l'application » visible sur la figure suivante.

Structure du document Boîte à outils

Store Test Kit App.xaml.cs MainPage.xaml

Détails de l'application Fournissez les ressources image pour exécuter le test fourni par Store Test kit. Ces tests v
processus de soumission à Windows Store, mais la réussite de ces tests ne garantit pas que

Tests automatisés

Tests manuels

Package d'application : c:\L... s\visual studio 2012\Projects\DemoStoreTestKit\DemoStoreT

Vignette Windows Store :



Parcourir...

Captures d'écran d'application : WVGA



Facultatif

Les images de l'application sont définies dans le Store Test Kit

Lorsque ceci sera résolu, nous pourrons passer aux tests manuels.

Pour ceux-ci, nous devons démarrer l'application et l'utiliser sur notre téléphone (et non dans l'émulateur). Il y a chaque fois une description de ce qu'il faut faire (non traduite). Il s'agit en fait d'une liste d'éléments à vérifier soi-même pour éviter que l'application ne soit refusée lors de la soumission au Windows Phone Store.

Il ne faut pas oublier de tester l'application en envisageant toutes les circonstances possibles :

- Le téléphone peut se verrouiller après une période d'inactivité.
- On reçoit un coup de téléphone pendant l'utilisation de l'application.
- On passe dans un tunnel et le réseau n'est plus accessible.
- Etc.

Vous avez également à votre disposition un outil d'analyse de performance qui se trouve en bas de l'onglet des tests automatisés. Vous pouvez démarrer cette analyse automatique sur votre téléphone, elle génère un rapport avec des informations intéressantes comme par exemple le temps de démarrage de l'application, si l'application répond rapidement, etc. Cela peut vous donner des pistes d'améliorations de votre application (voir le rapport de performance présenté).

RÉSUMÉ

0 Alert(s)

RAPPORT

Différents paramètres de l'application tels qu'ils ont été mesurés pendant la session d'analyse

| | | |
|-----------------------------------|--------------|--|
| Temps de démarrage | 0,86 sec | Le temps de démarrage de l'application est conforme aux exigences. |
| Réactivité | --- | L'application est réactive. |
| Total des données téléchargées | 0,00 Ko | Le total des données chargées par l'application est de 0,000 Ko |
| Total des données téléchargées | 0,00 Ko | Le total des données téléchargées par l'application est de 0,000 Ko |
| Charge de batterie restante | 16,46 heures | La session a consommé 1,36 mAh de charge batterie en 53,90 secs. Avec ce taux, il reste environ 16,46 heures |
| Utilisation maximum de la mémoire | 7,56 Mo | L'utilisation maximale de la mémoire par l'application est de 7,56 Mo |
| Utilisation moyenne de la mémoire | 7,49 Mo | L'utilisation moyenne de la mémoire par l'application est de 7,49 Mo |

Rapport de performance

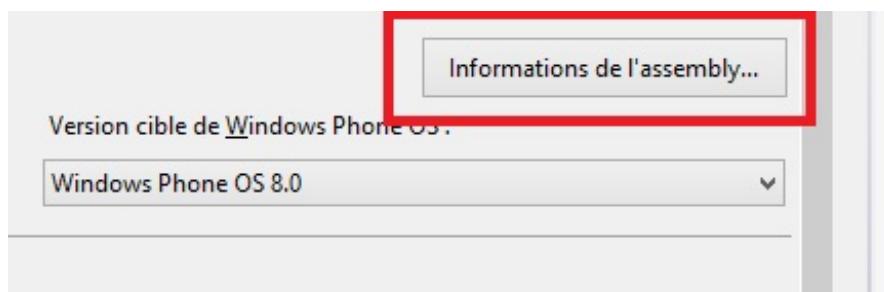
Le Store Test Kit constitue donc une espèce de grande check-list qui va nous permettre de vérifier que notre application fonctionne dans toutes les situations décrites. Cela va nous faire gagner du temps, car si nous détectons une anomalie, il sera possible de la corriger rapidement. Dans le cas contraire, c'est Microsoft qui la détectera et qui bloquera la publication de notre application tant que le problème ne sera pas corrigé.

À noter que nous serons avertis du rapport de certification par e-mail.

Publier son application sur le Windows Phone Store

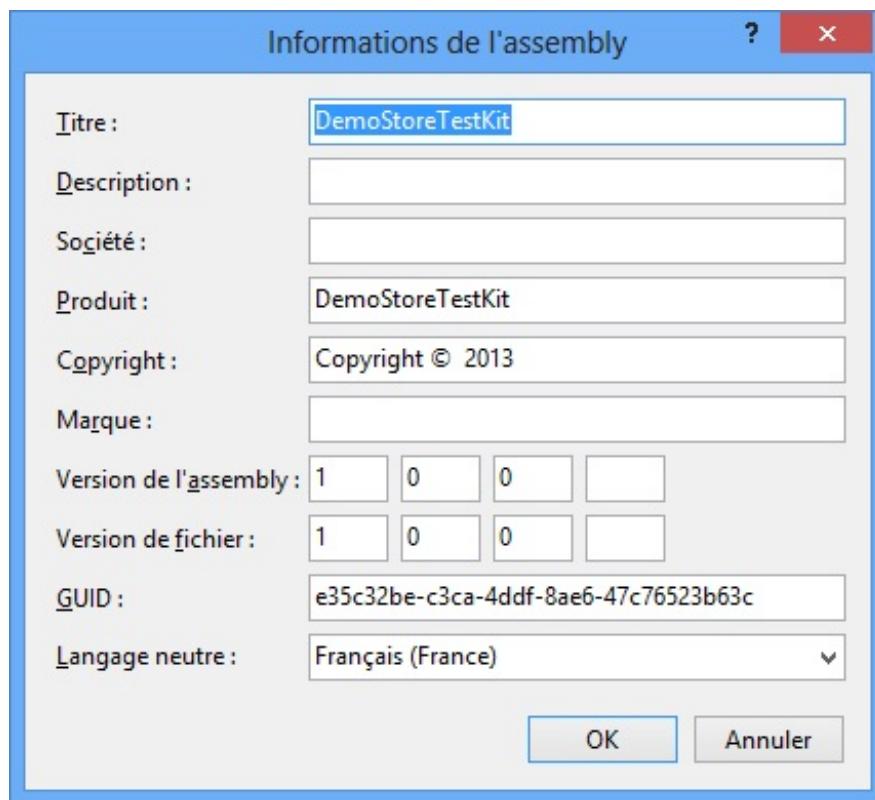
Ça y est, vous avez réussi tous les tests ! Il est enfin temps de passer à la soumission de notre application. C'est la dernière étape de ce processus. Cela consiste à envoyer notre application, à la décrire, à indiquer son prix, etc. Bref, à fournir toutes les informations nécessaires.

Mais avant ça, il y a une dernière petite chose à faire. Il s'agit de préciser la langue principale de notre application. Cela se fait dans les propriétés du projet. Cliquez sur informations de l'assembly dans l'onglet « Application » (voir figure suivante).



Accéder aux informations de l'assembly

Choisissez la langue.



Recompilez en mode Release et c'est fini pour le code !

Revenons à la soumission de notre application et rendons-nous sur <http://create.msdn.com/>. Connectons-nous avec notre compte Windows Live en cliquant sur « Sign in ». Puis cliquons sur « SUBMIT APP » (voir figure qui suit).

The screenshot shows the Windows Phone Dev Center homepage. At the top, there's a navigation bar with links for Dashboard, Develop, Community, and Help. Below the navigation is a large grid of app icons, including XBOX LIVE, CNN, ebay, NETFLIX, and various games like Angry Birds and Farm Heroes. To the right of the grid, a red box contains the text "Welcome De" and "Démarrer la". Below the grid, there are three main calls-to-action: "SUBMIT APP" (with a red border), "GET SDK", and "VIEW S". Each action has a brief description below it. Further down, a section titled "Developer registration opens in 13 new markets" includes a link to "More than 100 fixes and improvements in latest Dev Center update, includina new markets soumission de l'application". On the right side, there are sections for "Blogs" and "Using ads in yc" (partially visible).

en-us Search

Windows Phone | Dev Center

Dashboard Develop Community Help

Welcome De
Démarrer la

SUBMIT APP
Join Dev Center and publish your app in the Windows Phone Store.

GET SDK
Download the tools to build great Windows Phone apps.

VIEW S
View code samples, community topics

Developer registration opens in 13 new markets

More than 100 fixes and improvements in latest Dev Center update, includina new markets soumission de l'application

Blogs

Using ads in yc

Developer lessc increase mobile

Si vous n'êtes pas connecté, il faudra le faire. La première chose à faire est de donner quelques informations sur notre application. Cliquez sur l'icône rouge contenant le chiffre 1.

The screenshot shows the Windows Phone Dev Center interface. At the top, there's a navigation bar with a search icon, a lock icon, and a refresh icon. The title 'Windows Phone Dev Center' is displayed next to a small Windows logo. Below the title, there's a sub-navigation bar with links for 'Dashboard', 'Develop', 'Community', and 'Help'. On the right side of this bar is a 'Search Dev' input field. The main content area has a large heading 'Submit app'. Below it, a text block says: 'You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.' There are two numbered steps: '1 App info' (highlighted in red) and '2 Upload and describe your XAP(s)'. Step 1 includes a sub-instruction: 'Give your app an alias, price it, and enter other relevant info'. Step 2 includes a sub-instruction: 'For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.' The word 'Saisir' is written in French at the end of the first step's sub-instruction.

Submit app

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

Required

Saisir

1

App info

Give your app an alias, price it, and enter other relevant info

2

Upload and describe your XAP(s)

For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

Optional



Market selection and custom pricing

For apps, you have the option to define different pricing and availability for different countries/regions.

les informations de l'application

Nous pouvons saisir un nom, une catégorie et un prix pour notre application dans la page « App info » qui suit.

App info

The info on this page is used to refer to your app here in the Dev Center, and also controls how it appears in the Store.

App info

App alias*

This name is used to refer to your app here on Dev Center. The name your customer sees is read directly from your XAP file.

Category*

Subcategory

Saisie du nom, de

Pricing

Base price*

Free or paid? If paid, how much? [Learn how](#) this affects pricing in different countries/regions.

  EUR

You'll need to provide your tax or bank info (or both) if you want to submit paid apps.

- Offer free trials of this app. Before you select this option, make sure you've implemented a trial experience in your app. [Learn more.](#)

Market distribution

- Distribute to all available markets at the base price tier
- Distribute to all markets except those with stricter content rules. [Learn more.](#)
- Continue distributing to current markets

la catégorie et du prix de l'application

La catégorie doit être la plus proche possible du type de notre application. Suivant les cas, il est également possible d'indiquer une sous-catégorie. Cela va permettre d'identifier plus facilement le type de votre application dans le Windows Phone Store. Il est également important de fixer le prix de notre application. Elle peut être gratuite ou allant de 0.99€ ou 424,99€. Sur cet écran, nous pourrons également indiquer si l'application possède un mode d'évaluation. Nous en avons parlé un peu plus haut, ce mode permet de tester une application payante en limitant par exemple ses fonctionnalités ou sa durée.

Nous avons également la possibilité d'indiquer dans quels pays notre application sera téléchargeable. En général, on laissera l'exception portant sur les pays qui ont des politiques de règles plus strictes pour éviter tout problème 😊.

À noter qu'il est également possible de faire en sorte que son application soit disponible en version bêta restreinte à certains utilisateurs — ce qui est pratique si l'on souhaite la faire valider par des personnes de son entreprise par exemple.

Une fois ces informations saisies, nous pouvons passer à l'étape suivante en cliquant sur l'icône rouge contenant le chiffre 2 où nous pourrons envoyer notre fichier .xap.

Submit app

Mon nom d'application

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

Required



App info

Give your app an alias, price it, and enter other relevant info



Upload and describe your XAP(s)

For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

Passer à

Optional



Add in-app advertising

Getting paid through ads? It's all here.



Market selection and custom pricing

For apps, you have the option to define different pricing and availability for different countries/regions.

l'envoi du fichier .xap

Pour rappel, ce fichier .xap doit être compilé en mode **release**. Cliquez sur Parcourir (voir figure suivante) pour envoyer le fichier .xap.

The screenshot shows the Windows Phone Dev Center interface. At the top, there's a blue header bar with the URL 'us/AppSubmis' and icons for search, lock, and refresh. Next to it is the 'Windows Phone Dev Center' logo. Below the header, the main content area has a title 'Upload and describe your XAP'. Underneath, a sub-section titled 'Mon nom d'application' is visible. A note says: 'If your app contains more than one XAP, you can upload additional files as soon as you upload the first one here. [Learn more.](#)' To the right of this note is a button labeled 'Envoi du fichier'. Below the note, there's a section for 'Upload XAP' with a 'Parcourir...' button. At the bottom left is a 'Save' button.

Une fois le fichier envoyé, nous pouvons voir des informations extraites de notre fichier et notamment les **capacités**. Rappelez-vous, ce sont les éléments Windows Phone que va utiliser notre application et dont l'utilisateur pourra être potentiellement averti. Par exemple, si vous utilisez le GPS dans votre application, la capacité ID_CAP_LOCATION sera détectée et un message d'avertissement sera affiché à l'utilisateur lorsqu'il voudra télécharger votre application lui indiquant qu'elle utilise le GPS. Libre à lui d'accepter ou de refuser l'installation en sachant cela.

C'est le moment également de saisir une description pour votre application, description qui devra être dans la langue utilisée par l'application. C'est cette description (voir figure suivante) que l'utilisateur retrouvera lorsqu'il affichera votre application sur le Windows Phone Store.

The screenshot shows the 'Windows Phone Dev Center' interface for uploading an XAP file. At the top, there's a navigation bar with back, forward, and search icons, followed by the URL 'https://dev.windowsphone.com/en-us/AppSubmission'. Below the URL, a blue header bar says 'Upload XAP'.

The main content area starts with a note: 'This is an important page, because in addition to uploading your XAP to Dev Center, this page is also where you'll include the description and screenshots of your app that will appear in the Store. Consider this the place where you create your customer's first impression of your app.'

Below the note, there's a section for selecting an XAP file. A dropdown menu shows 'DemoXap.xap'. There are three buttons: 'Update selected' (in red), 'Delete selected' (in red), and 'Add new' (in red). To the right, a link 'More XAP options' with a dropdown arrow is visible.

The next section is 'XAP version number*', with a input field containing '1 . 0 . 0 . 0'.

Below that, a section titled 'XAP details detected from file' lists the following information:

| File name | VpPrenom.xap |
|----------------------|---|
| File size | 692 KB |
| Capabilities | ID_CAP_IDENTITY_USER ID_CAP_LOCATION ID_CAP_MEDIALIB ID_CAP_NETWORKING ID_CAP_PHONEDIALER ID_CAP_SENSORS ID_CAP_WEBBROWSERCOMPONENT ID_FNCTNL_SILVERLIGHT_FRAMEWORK ID_FNCTNL_TRIAL |
| Detected language(s) | French |
| Supported OS | 7.1 8.0 |
| Detected resolutions | WVGA |

Under 'Languages for Mon nom d'application', it says: 'A XAP file can contain multiple languages. Select each language and enter descriptions and screenshots below. By default, we'll use the description for search, but you can add keywords by clicking More options per language to the right.'

A dropdown menu for 'Language details*' shows 'French'. To the right, a link 'More options per language' with a dropdown arrow is visible.

The final section is 'Description for the Store*', which is a large text input area.

Description de l'application, y compris la liste de ses capacités

Vous aurez intérêt à bien soigner cette description pour deux raisons. Premièrement, parce que c'est la première chose que les utilisateurs liront ce qui pourra leur donner envie ou non de télécharger l'application. Décrivez les fonctionnalités de l'application, mais pas trop non plus. Il faut qu'ils puissent obtenir une information synthétique et utile pour faciliter leur choix de télécharger ou non votre application. Et deuxièmement, l'outil de recherche du Windows Phone Store va se fonder sur ce descriptif. Il faut donc que les mots-clés que les utilisateurs sont susceptibles d'utiliser pour chercher une application comme la vôtre soient présents. Pensez aux mots-clés que vous utiliseriez si vous cherchiez une telle application...

Enfin, vous devrez fournir les images qui illustrent votre application dans l'interface présentée dans la figure suivante, comme celles que vous avez données dans le Store Test Kit. Soignez-les, elles doivent donner envie à l'utilisateur de télécharger votre application et elles doivent également être représentatives de votre application.

The screenshot shows the Windows Phone Dev Center application submission interface. At the top, there are navigation icons (back, forward, search) and the URL <https://dev.windowsphone.com/en-us/AppSubmit>. To the right is a lock icon and a save button. The main area has a blue header bar with the text "Windows Phone Dev Center". Below this, a note says "Utilisez ce lien pour télécharger toutes les images en une fois, ou téléchargez-les séparément en cliquant sur chaque espace réservé ci-dessous." A red "Upload all" button is visible. A note below it specifies "App tile icon 300 x 300 px". There is a placeholder image for the background image (1000 x 800 pixels), which is a light gray rectangle with a white plus sign in the center. Below this, under the heading "WVGA", there are two rows of four placeholder images each, also labeled with a plus sign. A "Save" button is located at the bottom left.

Téléchargez les images de votre application

Une fois cette étape terminée, il vous reste deux étapes facultatives.

Submit app

Mon nom d'application

You've spent hours developing and designing your app, and now it's time for the rest of the world to experience your masterpiece. In just two steps we'll gather the information we need to successfully launch your app in the Windows Phone Store. [Learn more](#) about the steps for successfully submitting your app.

Required



App info

Give your app an alias, price it, and enter other relevant info



Upload and describe your XAP(s)

For each XAP in your app, this is where you'll enter descriptions and upload screenshots that will showcase your app in the Store.

Les

Optional



Add in-app advertising

Getting paid through ads? It's all here.



Market selection and custom pricing

For apps, you have the option to define different pricing and availability for different countries/regions.

étapes facultatives

La première permet de créer ou d'utiliser un compte de publicité. Cela va vous permettre d'afficher des bandeaux publicitaires sur votre application et potentiellement de générer des petits revenus (voir figure suivante).

The screenshot shows a web browser window with the URL <https://dev.windowsphone.com/en-us/Account/A>. The page title is "Windows Phone Dev Center". The main heading is "Sign up for ad-funded apps". Below it, a text block says: "Instead of charging for your app, you also have the option to make money for your hard work through ads. To create an ad-funded app, you'll need an account with the Microsoft Advertising platform (also known as pubCenter).". A note below states: "If you already have an account with the Microsoft Advertising platform, great! Otherwise you can create a new one for free here." A link "Learn more" is provided. Two radio button options are shown: "Create a pubCenter account" (selected) and "Use my existing pubCenter account". A dropdown menu is open, showing "J' ai27: I (nic les il e) ▾". At the bottom are "Save" and "Cancel" buttons.

Souscrire à un compte pour afficher de la publicité

L'autre étape facultative permet d'indiquer plus finement les pays où votre application sera accessible et définir potentiellement un prix différent pour chaque pays. Si l'application n'est qu'en français par exemple, cela a peu de sens de la distribuer aux Chinois !

https://dev.windowsphone.com/en-us/AppSubmit

Windows Phone Dev Center

Define market pricing

Mon nom d'application

Change where your app is available and how much it will cost in those individual markets.

These countries\regions have more restrictions for app or in-app product content. [Learn more.](#)

All

| | | | | | |
|---|------|-----|--|------|-----|
| <input type="checkbox"/> Afghanistan ⓘ | 0.00 | USD | <input checked="" type="checkbox"/> Lithuania | 0.00 | LTL |
| <input type="checkbox"/> Albania ⓘ | 0.00 | USD | <input checked="" type="checkbox"/> Luxembourg | 0.00 | EUR |
| <input type="checkbox"/> Algeria ⓘ | 0.00 | DZD | <input checked="" type="checkbox"/> Macao SAR | 0.00 | USD |
| <input checked="" type="checkbox"/> Andorra | 0.00 | EUR | <input checked="" type="checkbox"/> Macedonia FYRO | 0.00 | USD |
| <input checked="" type="checkbox"/> Angola | 0.00 | USD | <input checked="" type="checkbox"/> Madagascar | 0.00 | USD |
| <input checked="" type="checkbox"/> Antigua and Barbuda | 0.00 | USD | <input checked="" type="checkbox"/> Malawi | 0.00 | USD |
| <input checked="" type="checkbox"/> Argentina | 0.00 | ARS | <input type="checkbox"/> Malaysia ⓘ | 0.00 | MYR |
| <input checked="" type="checkbox"/> Armenia | 0.00 | USD | <input type="checkbox"/> Maldives ⓘ | 0.00 | USD |
| <input checked="" type="checkbox"/> Australia | 0.00 | AUD | <input type="checkbox"/> Mali ⓘ | 0.00 | USD |
| <input checked="" type="checkbox"/> Austria | 0.00 | EUR | <input checked="" type="checkbox"/> Malta | 0.00 | EUR |
| <input type="checkbox"/> Azerbaijan ⓘ | 0.00 | USD | <input checked="" type="checkbox"/> Marshall Islands | 0.00 | USD |
| <input type="checkbox"/> Bahrain ⓘ | 0.00 | BHD | <input type="checkbox"/> Mauritania ⓘ | 0.00 | MRO |
| <input type="checkbox"/> Bangladesh ⓘ | 0.00 | BDT | <input checked="" type="checkbox"/> Mauritius | 0.00 | USD |
| <input checked="" type="checkbox"/> Barbados | 0.00 | USD | <input checked="" type="checkbox"/> Mexico | 0.00 | MXN |
| <input checked="" type="checkbox"/> Belarus | 0.00 | USD | <input checked="" type="checkbox"/> Micronesia | 0.00 | USD |
| <input checked="" type="checkbox"/> Belgium | 0.00 | EUR | <input checked="" type="checkbox"/> Monaco | 0.00 | EUR |
| <input checked="" type="checkbox"/> Belize | 0.00 | USD | <input checked="" type="checkbox"/> Mongolia | 0.00 | USD |
| <input checked="" type="checkbox"/> Benin | 0.00 | USD | <input checked="" type="checkbox"/> Montenegro | 0.00 | EUR |
| <input checked="" type="checkbox"/> Bhutan | 0.00 | USD | <input type="checkbox"/> Morocco ⓘ | 0.00 | MAD |
| <input type="checkbox"/> Bolivia | 0.00 | USD | <input type="checkbox"/> Mozambique | 0.00 | USD |
| <input type="checkbox"/> Bulgaria | 0.00 | EUR | <input type="checkbox"/> Namibia | 0.00 | USD |
| <input type="checkbox"/> Cambodia | 0.00 | USD | <input type="checkbox"/> Niger | 0.00 | USD |
| <input type="checkbox"/> Chile | 0.00 | USD | <input type="checkbox"/> Nigeria | 0.00 | USD |
| <input type="checkbox"/> Costa Rica | 0.00 | USD | <input type="checkbox"/> Pakistan | 0.00 | USD |
| <input type="checkbox"/> Ecuador | 0.00 | USD | <input type="checkbox"/> Paraguay | 0.00 | USD |
| <input type="checkbox"/> El Salvador | 0.00 | USD | <input type="checkbox"/> Peru | 0.00 | USD |
| <input type="checkbox"/> Georgia | 0.00 | USD | <input type="checkbox"/> Philippines | 0.00 | USD |
| <input type="checkbox"/> Greece | 0.00 | EUR | <input type="checkbox"/> Portugal | 0.00 | USD |
| <input type="checkbox"/> Guatemala | 0.00 | USD | <input type="checkbox"/> Puerto Rico | 0.00 | USD |
| <input type="checkbox"/> Honduras | 0.00 | USD | <input type="checkbox"/> Qatar | 0.00 | USD |
| <input type="checkbox"/> India | 0.00 | INR | <input type="checkbox"/> Romania | 0.00 | USD |
| <input type="checkbox"/> Indonesia | 0.00 | USD | <input type="checkbox"/> Saudi Arabia | 0.00 | USD |
| <input type="checkbox"/> Israel | 0.00 | ILS | <input type="checkbox"/> Senegal | 0.00 | USD |
| <input type="checkbox"/> Italy | 0.00 | EUR | <input type="checkbox"/> Singapore | 0.00 | USD |
| <input type="checkbox"/> Jordan | 0.00 | USD | <input type="checkbox"/> South Africa | 0.00 | USD |
| <input type="checkbox"/> Kenya | 0.00 | USD | <input type="checkbox"/> Spain | 0.00 | USD |
| <input type="checkbox"/> Kuwait | 0.00 | USD | <input type="checkbox"/> Sri Lanka | 0.00 | USD |
| <input type="checkbox"/> Lebanon | 0.00 | USD | <input type="checkbox"/> Switzerland | 0.00 | USD |
| <input type="checkbox"/> Libya | 0.00 | USD | <input type="checkbox"/> Turkey | 0.00 | USD |
| <input type="checkbox"/> Malta | 0.00 | EUR | <input type="checkbox"/> United Arab Emirates | 0.00 | USD |
| <input type="checkbox"/> Morocco | 0.00 | USD | <input type="checkbox"/> United Kingdom | 0.00 | USD |
| <input type="checkbox"/> Oman | 0.00 | OMR | <input type="checkbox"/> United States | 0.00 | USD |
| <input type="checkbox"/> Pakistan | 0.00 | PKR | <input type="checkbox"/> Uruguay | 0.00 | USD |
| <input type="checkbox"/> Qatar | 0.00 | QAR | <input type="checkbox"/> Venezuela | 0.00 | USD |
| <input type="checkbox"/> Saudi Arabia | 0.00 | USD | <input type="checkbox"/> Yemen | 0.00 | USD |
| <input type="checkbox"/> Senegal | 0.00 | USD | <input type="checkbox"/> Zimbabwe | 0.00 | USD |
| <input type="checkbox"/> Singapore | 0.00 | SGD | | | |
| <input type="checkbox"/> South Africa | 0.00 | ZAR | | | |
| <input type="checkbox"/> Spain | 0.00 | EUR | | | |
| <input type="checkbox"/> Sri Lanka | 0.00 | LKR | | | |
| <input type="checkbox"/> Switzerland | 0.00 | CHF | | | |
| <input type="checkbox"/> Turkey | 0.00 | TRY | | | |
| <input type="checkbox"/> United Arab Emirates | 0.00 | USD | | | |
| <input type="checkbox"/> United Kingdom | 0.00 | GBP | | | |
| <input type="checkbox"/> United States | 0.00 | USD | | | |
| <input type="checkbox"/> Uruguay | 0.00 | UYU | | | |
| <input type="checkbox"/> Venezuela | 0.00 | VES | | | |
| <input type="checkbox"/> Yemen | 0.00 | YER | | | |
| <input type="checkbox"/> Zimbabwe | 0.00 | ZWD | | | |

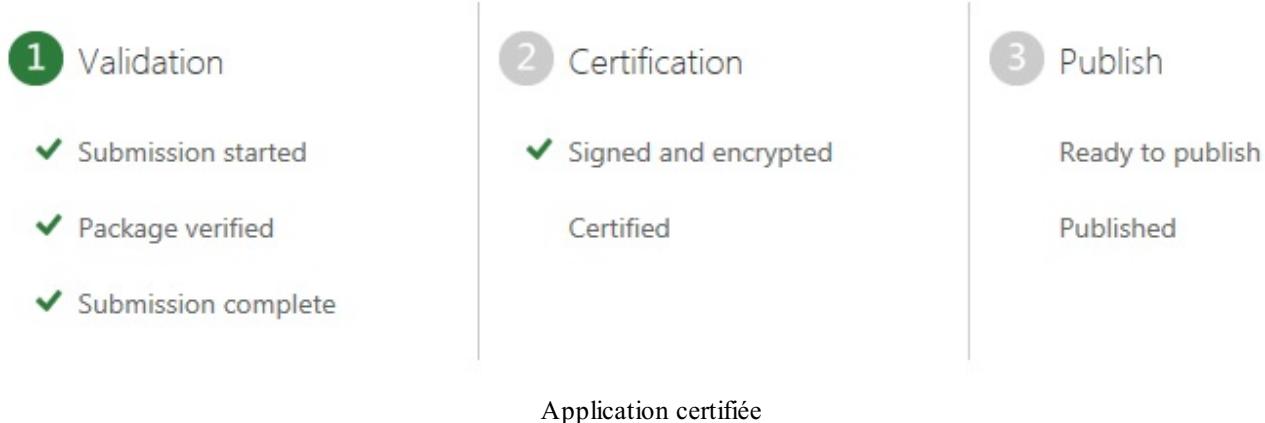
Définition des prix et des marchés à atteindre

Et voilà, c'est terminé. Il ne vous reste plus qu'à cliquer sur « Submit » en bas de la page et votre application est soumise au Windows Phone Store. Il faut compter environ une petite semaine avant d'avoir des retours de Microsoft par e-mail, vous informant de la soumission de l'application ou des éventuelles erreurs.

À tout moment, vous pouvez suivre le processus de soumission de votre application en allant sur votre compte Windows Phone Store et en allant voir dans votre tableau de bord (*dashboard*).

| Submitted on | Progress | |
|--------------|--|---|
| 3/22/2012 |  Certified | Soumission en cours dans le tableau de bord |

This page displays the certification lifecycle of your app. You can follow your app as it progresses through the various stages of certification and publication. Here you have the ability to review and take actions as needed. For more information, go [here](#).



Voilà, ce cours est maintenant terminé. Vous faites maintenant partie de ceux qui savent développer des applications pour Windows Phone ! Bravo.

Vous commencez à maîtriser correctement le XAML que nous avons appris tout au long de ce cours. De même, les contrôles Windows Phone et l'exploitation des données asynchrones n'ont plus de secrets pour vous. Que dire des différents capteurs du téléphone ... Le GPS sur le bout des doigts, l'accéléromètre haut la main ! Et j'en passe ... En plus, vous connaissez les subtilités des tuiles et des notifications, ce qui n'est pas peu dire.

C'est vrai, nous n'avons pas pu tout voir car cela demanderait le double de pages et risquerait de fortement vous lasser, mais vous avez déjà beaucoup d'éléments en main vous permettant de vous régaler et commencer à créer des applications dignes de votre imagination. Allez, avouez que vous êtes maintenant complètement fan de cette plateforme tellement la création d'applications y est un plaisir. Personnellement, je le suis et j'espère avoir pu vous transmettre un peu de cette connaissance vous dégrossissant le travail.

Car non, ce n'est pas fini ! Hier Windows Phone 7, aujourd'hui Windows Phone 8, mais tout laisse penser que Microsoft ne s'arrêtera pas là et continuera à nous offrir plein de bonnes choses dans son système d'exploitation, que nos smartphones se feront un plaisir d'exploiter à travers vos futures applications. Vous avez de bonnes bases maintenant, vous êtes prêt pour créer des applications, mais restez à l'écoute des futures nouveautés. Vos applications et vos utilisateurs vous en remercieront.