www.e-naxos.com

Formation – Audit – Conseil – Développement

XAML (Windows Store, WPF, Silverlight, Windows Phone), C#

Cross-plateforme Windows / Android / iOS / Xamarin

UX Design

ALL DOT.BLOG

TOME 6

Windows Store Apps – WinRT & Modern UI

Tout Dot.Blog par thème sous la forme de livres PDF gratuits !

Reproduction, utilisation et diffusion interdites sans l'autorisation de l'auteur

Olivier Dahan
odahan@gmail.com

Table des matières

Présentation 9

La plateforme 10

WinRT : les bases 10

WinRT 10

Architecture 12

Les Métadonnées 13

La projection des langages 14

Le runtime broker 15

Asynchronisme 15

Quel est le contenu de WinRT ? 16

Metro Style / WinRT : les API's 18

Bêta / Preview 18

Un voyage solitaire 18

Les API's de WinRT 19

Conclusion 21

Windows 8 : Le cycle de vie d'une application 22

Un mot d'ordre : la transparence pour l'utilisateur 22

Le cycle de vie d'une application Windows 8 "Modern UI" 24

Quelques conseils 31

Conclusion 33

Windows Store Apps : comprendre les templates 34

Les templates 34

La structure des templates 35

Faut-il utiliser les templates ? 37

Les autres templates 37

Conclusion 39

De Silverlight/WPF à WinRT 40

De Silverlight à WinRT (partie 1) 40

WinRT englobe la quasi-totalité de Silverlight 40

La déclaration “classique” d’un espace de noms 40
La déclaration sous WinRT 41
Utiliser le même code Xaml ? 41
Les déclarations dans le code 41
De Silverlight à WinRT (partie 2) 42
Le Thread de l’UI 42
Le nouveau Dispatcher 43
Un code portable 43
Conclusion 44
De Silverlight/WPF à WinRT : .NET pour Metro Style (partie 3) 45
.NET pour Modern UI 45
.NET APIs pour Modern UI 45
.NET APIs + WinRT 46
Un .NET allégé 46
Des substituions à connaître 47
Une simplification de l’accès aux APIs 47
Créer des librairies Metro Style portables 48
La conversion du code existant 48
Conclusion 55
De Silverlight à WinRT : Mesurez les différences grâce au « WinRT Genome Project » (partie 4) 56
WinRT Genome Project 57
Types en commun 57
Membres en commun 58
Pousser la comparaison plus loin 59
Conclusion 59
De Silverlight à WinRT (partie 5) 60
La série “De Silverlight à WinRT” – épisodes précédents 60
Ce qu’il faut savoir en 10 points 61
WinRT par l’exemple 73

Windows 8 / WinRT : Premiers pas 73
Le set de travail 73
Le fameux Hello World 74
Autres langages 81
Conclusion 82
Windows 8 : créer des tuiles vivantes (live tile) 82
La tuile 82
La librairie NotificationsExtensions 84
Les astuces à connaître 85
Les thèmes pour tuiles 86
Conclusion 94
WinRT : Prendre des photos 95
Principes 95
La version simple 95
Autoriser le crop 96
Une application de capture 96
Conclusion 106
Windows Store : créer un package et le valider avec le Windows App Certification Kit 106
La validation étape obligatoire 106
Le Windows App Certification Kit 106
Packager pour publier, et aussi pour tester 107
Construire un package 107
Conclusion 116
Cachez ces images que je ne saurais voir ! (WinRT et le cache image) 117
Cache-cache, par défaut 117
Pas toujours souhaitable 118
Accéder au cache dans l'URL 118
Accéder au cache via l'objet BitmapImage 119
Conclusion 119

Windows Store Apps : gérer la navigation 119
La navigation 119
Un exemple 120
Conclusion 123
Debug d'applications WinRT sur Surface 124
Déployer et Déboguer sur Surface RT 124
Les outils 124
Conclusion 129
Windows Phone 8 : le casse tête de la taille des icônes 129
Deux questions en une 129
Un set d'images et des tests 129
Ce qu'on peut en déduire 130
Conclusion 131
Utiliser des Behaviors dans des applications Windows 8 Store 132
Les Behaviors 132
Des propriétés … attachantes 132
Des propriétés auxquelles ont devient… dépendant 133
Les Behaviors : de l'action sans code 133
La mauvaise nouvelle 135
Acte 1 : La Hollande à notre secours 135
Acte 2 : Le Windows 8.1 Behahior SDK 136
Conclusion 138
SQLite sur Windows 8 (ARM et Intel) 138
Une base de données embarquée 138
SQLite 139
Petit mais costaud 140
Installation 142
Configuration 143
Utilisation 144
Conclusion 145

Faites frémir les sens de Windows 8 ! (ou comment taquiner les capteurs) 146
Des tas d'exemples 147
Du code, tout de suite ! 147
Conclusion 154
Windows Store Apps : Comment créer et gérer une AppBar 155
Le rôle de la AppBar 155
Comment ajouter une AppBar ? 156
Partager une AppBar au travers des pages 158
Ouverture et Fermeture 158
Figer la barre 159
Les évènements d'ouverture et fermeture 159
Les styles cachés des boutons 159
Conclusion 162
Faites griller les toasts ! … Avec Windows 8 162
Comment remplacer les fenêtres ? 162
Toasts locaux et distants 164
Un choix de toasts assez large 165
Comment faire griller les toasts ? 170
Conclusion 172
Mock-up Windows 8 apps avec PowerPoint et PowerMockup 172
Mock-up 172
PowerPoint 172
PowerMockup 173
Les bases d'un projet 175
Communiquer et améliorer 175
Simplicité 176
Payant 176
Conclusion 176
Mieux gérer les capteurs sous Windows 8 177
Les capteurs : des amis du développeur 177

Du code pour gérer plus facilement les capteurs 177
Allo ? Surface ? Me captez-vous ? 177
Surface et le monde physique 179
Des axes et des unités 179
Du Code ! 183
Organisation du code 184
L'application de test en marche 192
Conclusion 194
Mock-up gratuit d'applications Windows 8 avec PowerPoint 194
Le tout est dans le tout 194
Et le mock-up dans tout ça ? 196
Mocker n'est pas sketcher 198
De Photoshop à Visio en passant par PowerPoint 199
PowerPoint : les bons templates 200
Conclusion 201
Développement Windows 8 : Le livre 201
Windows 8 ou WinRT ou Windows Store App ou même Modern UI ? 202
Pourquoi un livre ? 202
Tous les langages à l'honneur 202
Mon Rôle dans tout cela 203
Combien ? Où ? et Quand ? 204
Conclusion 205
Utiliser PathIO sous WinRT 206
Faire simple est parfois compliqué ! 206
PathIO, une des portes vers la simplicité 206
Conclusion 207
Control Calendrier pour WinRT 207
WinRT Xaml Calendar 208
Conclusion 209
WinRT : RoamingSettings, quota et Sérialisation 210

Le stockage itinérant (Roaming) 210
Le problème de la sérialisation des données génériques 210
Sauvegarder et relire les settings 212
Les contraintes des données d'itinérance 212
Le quota 213
Quota dépassé ? 214
Prendre en compte le quota 214
Conclusion 215
WinRT : expressions régulières et panneau de recherche (Windows Store apps) 216
Le panneau de recherche 216
Suggérer 217
S'abonner, répondre, proposer 218
Conclusion 220
Transcoder de la vidéo sous WinRT 221
L'espace de nom Transcoding 221
Une API simplifiée 221
Mise en œuvre 221
Conclusion 222
WinRT : utiliser le presse-papiers 222
Les API du presse-papiers 223
Les principales utilisations 224
Conclusion 225
Windows.Storage pour Windows Phone 8 et Windows 8 226
Le stockage local de données 226
Windows Phone 8 et WinRT, un noyau commun 227
Deux approches pour les données locales 227
Conclusion 229
Windows 8 : Protocole d'activation personnalisé (CPA) 230
Devenir gestionnaire pour un nom de schéma 230
Comment faire ? 231

CPA en Action....236
Conclusion....239
Donner du “charme” à vos applications Windows 8....239
Des charmes charmants pour plus de charme....240
Rechercher dans les applications....240
La dérive… C’est selon votre éthique !....241
Intégrer son application dans le charme de recherche....242
Conclusion....247
WinRT réinvente les Ria Services (les nouveaux WCF Data Services)....248
Le tooling....248
Construire l’exemple....249
Les données....250
La partie cliente....252
Conclusion....259
Avertissements....261
E-Naxos....261

Présentation

Bien qu'issu des billets et articles écrits sur Dot.Blog au fil du temps, le contenu de ce PDF a entièrement été réactualisé lors de la création du présent livre PDF en novembre 2013. Il s'agit d'une version inédite corrigée et à jour, un énorme bonus par rapport au site Dot.Blog ! Un mois de travail a été consacré à la réactualisation du contenu. Corrections du texte mais aussi des graphiques, des pourcentages des parts de marché évoquées, contrôle des liens, et même ajouts plus ou moins longs, c'est une véritable édition spéciale différente des textes originaux toujours présents dans le Blog !

Toutefois les billets n'ont pas été réécrits totalement ils peuvent donc parfois présenter des anachronismes sans gravité. Tout ce qui est important et qui a changé de façon notable a soit été réécrit soit a fait l'objet d'une note, d'un apparté ou autre ajout.

C'est donc bien plus qu'un travail de collection déjà long des billets qui vous est proposé ici, c'est une relecture totale et une révision et une correction techniquement à jour au moins de novembre 2013. Un vrai livre. Gratuit.

Astuce : cliquez les titres pour aller lire sur Dot.Blog l'article original et ses commentaires ! Tous les liens Web de ce PDF sont fonctionnels, n'hésitez pas à les utiliser !

La plateforme

WinRT : les bases

WinRT (*Windows RunTime*) est au cœur de la nouvelle plateforme Windows 8. Beaucoup de rumeurs, d'attentes aussi ont laissé s'installer une méconnaissance donnant l'impression d'un flou alors que tout cela est naturel : WinRT est une plateforme nouvelle qui mérite un petit investiment d'apprentissage ! L'accueil un peu tiède de Windows 8.0 n'a pas non plus joué en faveur d'une diffusion immédiate et massive des connaissances techniques autour de ce nouvel OS. Dot.Blog a été présent dès le départ et même avant… Ce billet a été écrit avant la sortie officielle de Windows 8.0 ! Maintenant que Windows 8.1 est sorti, il est temps de préciser les choses pour ceux qui n'ont pas encore sauté le pas. Qu'est-ce que WinRT ?

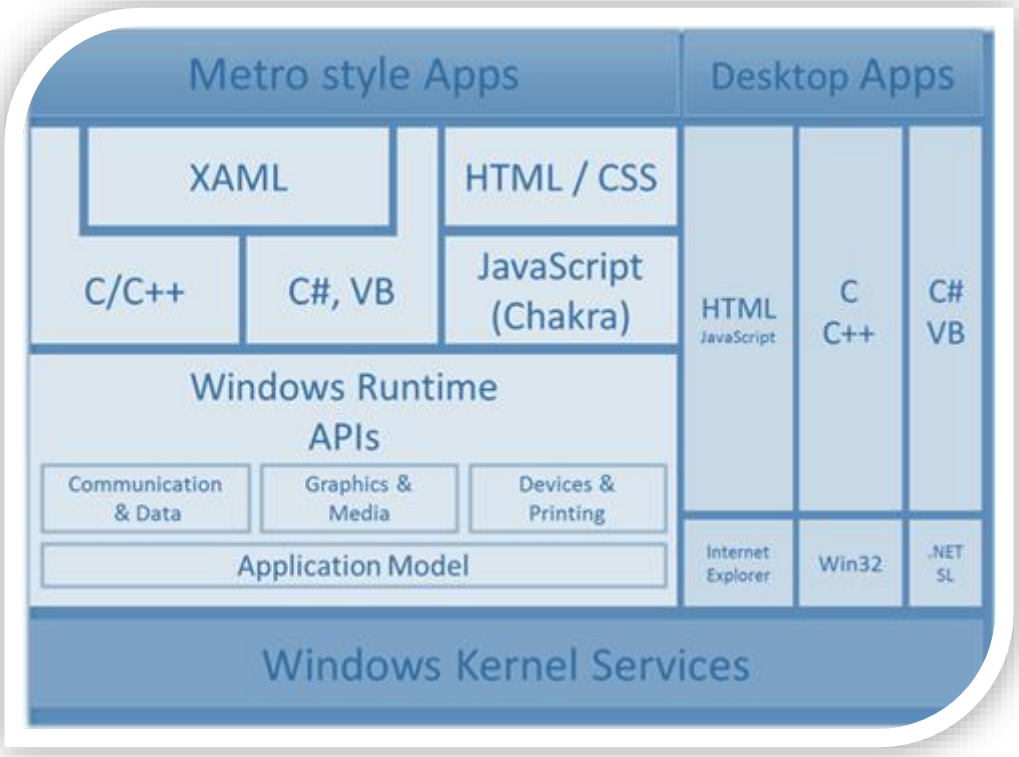
WinRT

C'est un nouveau modèle de programmation, un nouveau framework qui permet de construire des applications Metro Style (qu'on a aussi appelé Windows 8 App, puis Windows Store App mais aussi Modern UI) en utilisant le langage de votre choix, qu'il s'agisse d'un langage managé (C# ou VB.NET), C++ ou JavaScript.

> *J'utiliserai encore parfois le terme Metro ou Metro Style ici pour parler des applications Windows 8 écrites sous WinRT et respectant la charte graphique particulière appelée Metro jusqu'à lors. Vous le savez, ce nom a été abandonné faute d'un accord avec la société Metro AG en Allemagne. Modern UI n'a pas l'impact ni l'historique de Metro, ni même sa justification conceptuelle. C'est un nom de rechange sans ancrage dans le « Design Metro », sans vraies racines, sans âme. C'est dommage mais il faut parler de Modern UI et non plus de Metro !*

WinRT permet aux développeurs de construire des applications qui utilisent massivement les services et fonctionnalités exposées par Windows, ce qui était assez difficile auparavant il faut bien le dire (malgré l'avancée énorme qu'à représenté .NET sur ce point en « objectivant » la majorité des API natives de Windows).

Ci-dessous vous pouvez voir le célèbre diagramme qui a fait peur à autant de monde qu'il en a soulagé d'autres…



Ce diagramme montré au Build de septembre 2012 à la sortie de Windows 8.0 a été commenté en long et en large mais pas forcément de façon constructive et technique.

On note surtout que WinRT (Windows Runtime APIs) est une couche fine qui repose directement sur le noyau de l'OS pour en exposer les fonctions. Ces fonctions peuvent être utilisées par toutes les applications "Metro", depuis tous les langages.

Dans ce contexte Microsoft parle de "language projection", ce qui permet d'utiliser WinRT d'une façon qui semble native à tous les langages supportés.

Pour un développeur .NET, utiliser WinRT est très proche de l'écriture de code .NET. Microsoft n'a pas tout réinventé ni tout changé, et même si Sinofsky qui détestait .NET, XAML et C# a tout fait pour tenter d'étouffer les racines de WinRT, ce dernier est avant une copie de l'API .NET avec quelques réorganisations de namespaces et des classes nouvelles (notamment pour les unités mobiles). Les concepts habituels tels que les constructeurs, les propriétés, le développement asynchrone, et la grande

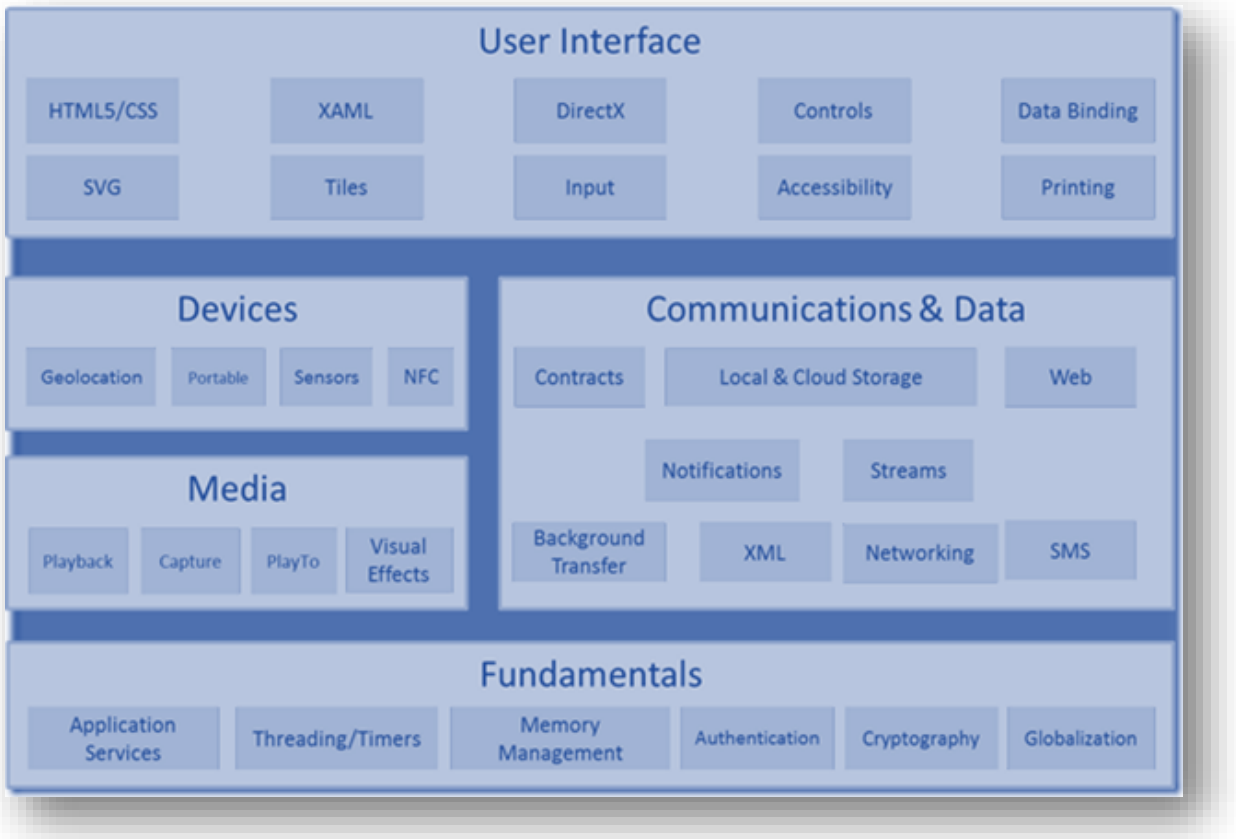
majorité des autres spécificités de .NET sont toujours les mêmes lorsqu'on écrit une application WinRT.

Le schéma ci-dessus nous montre que XAML est programmable aussi bien depuis C# et VB.NET que depuis C/C++ alors que JavaScript ne peut être utilisé que pour créer des applications utilisant Html/Css.

WinRT expose un nombre assez impressionnant de fonctions aux applications Metro. Un grand nombre de choses qui se programmaient via .NET est aujourd'hui pris en charge par WinRT.

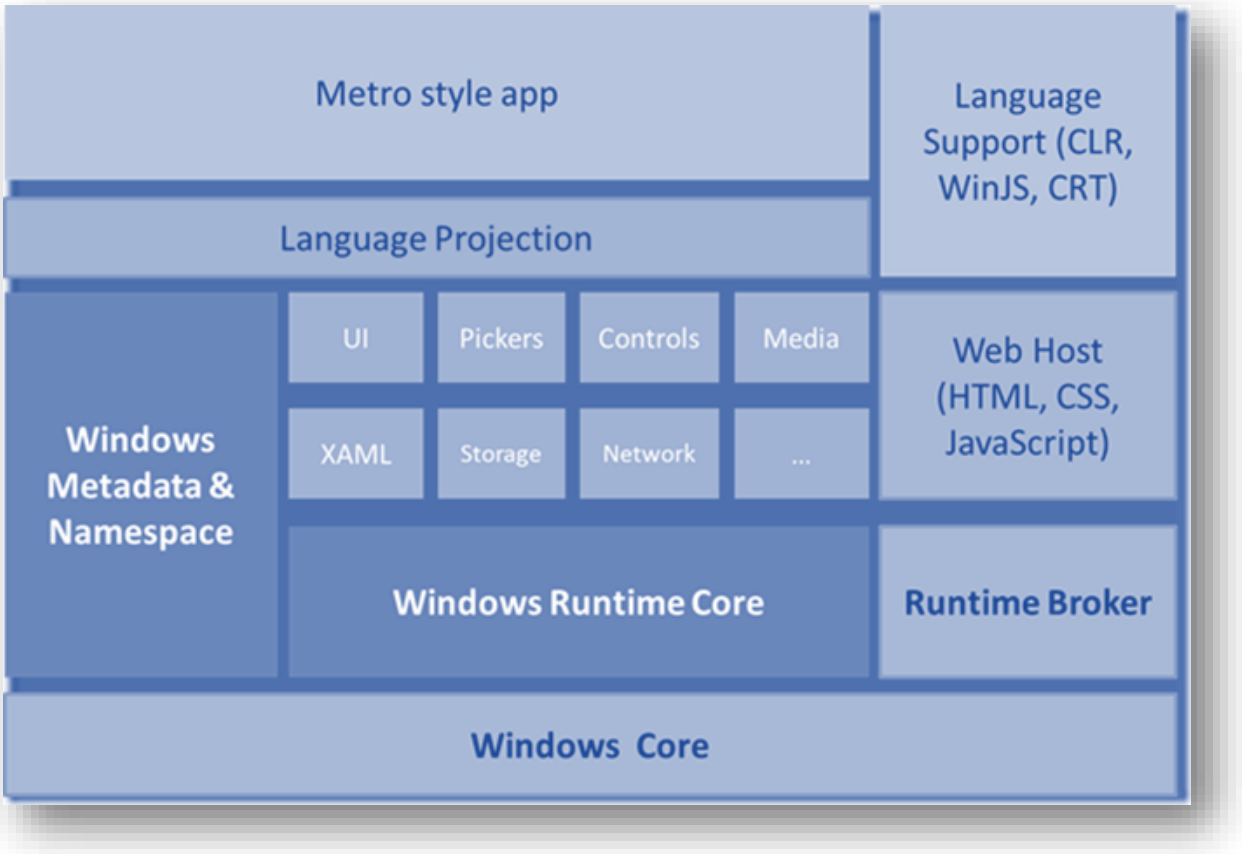
Des choses comme les I/O, l'accès aux services, et même Xaml lui-même sont devenues des parties intégrantes de WinRT donc de l'OS.

Le schéma suivant montre les fonctions couvertes par WinRT :



Architecture

Le schéma suivant montre un vue simplifiée de l'architecture de WinRT :



A la base nous trouvons Windows et son noyau. Directement au-dessus se trouvent les blocs de base de WinRT.

Il s'agit d'une plateforme vraiment énorme, plus grosse encore que le framework .NET complet. Plus complexe. Malgré tout chaque langage avec ses spécificités utilise tout ce potentiel de façon native.

Les Métadonnées

La gestion des métadonnées est visible sur la gauche du schéma précédent. Les Windows Metadata (aussi appelées WinMD) sont des fichiers assez similaires aux fichiers de métadonnées de .NET. Sous WinRT, WinMD est conceptuellement identique : ces fichiers décrivent ce qu'un composant WinRT peut faire. Cela diffère de .NET dans le sens où il s'agit bien ici d'un fichier séparé (sous .NET les métadonnées sont stockées dans l'assemblage lui-même). Les WinMD sont installés avec Windows 8 et sont utilisés par exemple par Intellisense quand on écrit des applications Metro dans Visual Studio.

Les fichiers WinMD sont stockés dans `Windows\System32\WinMetaData`.

On peut les ouvrir avec ILDASM. On découvre alors toute l'influence de .NET sur WinMD.

Toutes les classes WinRT sont dans le namespace `Windows.*`, point d'entré principal vers ce gigantesque univers d'APIs.

La projection des langages

Le terme a été évoqué plus haut. WinRT est écrit en c++ natif. Etant natives, les APIs de WinRT auraient normalement dû être appelées en utilisant du code d'interop (COM).

Or, la plupart des développeurs n'aiment pas COM. Mais heureusement pour eux Microsoft l'a compris ! Ils ne nous ont donc pas donné les APIs natives comme framework de base. A la place nous avons la projection des langages. Il faut entendre par là une technique permettant d'utiliser les composants natifs d'une façon naturelle et familière pour chaque langage. Microsoft ne force personne à apprendre ou utiliser un nouveau langage. Ils ont simplement "projeté" WinRT dans chaque langage que nous connaissons déjà.

De fait, les applications Windows 8 "Modern UI" sont écrites en utilisant soit un langage managé, soit C++ ou JavaScript. Pour les applications du bureau classique de Windows 8 rien ne change comme le montre le premier schéma, on utilise soit du managé avec .NET, soit du natif avec C/C++ soit du WPF; soit de l'Html pour les documents de type Web visibles avec un navigateur, Silverlight étant aujourd'hui une option sur le déclin, hélas.

Finalement, s'il n'y avait pas la norme de présentation très spécifique de Metro style, Windows 8 n'apparaitrait être qu'une nouvelle version de Windows 7 se programmant avec exactement les mêmes langages (managés ou non, Silverlight, WPF, Html/Js, C++ etc). La couche WinRT ainsi que l'aspect fortement normatif de l'interface Modern UI créent la nouveauté, le reste en fait ne change que peu. C'est une impression bien entendu, l'OS lui-même et WinRT sont construits sur des bases techniques assez différentes de celles de .NET.

Ainsi les changements dans l'OS sont réels et nombreux et si nous avons cette impression de continuité c'est parce que Microsoft a justement tout fait pour ne pas bouleverser les habitudes des développeurs ! Il ne faudrait pas comprendre le problème à l'envers en prenant cet immense effort de continuité comme l'expression naïve d'un non changement ou d'un heureux hasard.

Tout a changé, mais tout a été fait pour que ces changements impactent le moins possible nos habitudes. Voilà ce qu'est WinRT.

Le runtime broker

Le broker est responsable du contrôle des applications Modern UI. Par exemple il est essentiel pour la sécurité de vérifier qu'une application déclare correctement ses capacités, comme accéder au répertoire "Mes Images" ou accéder à Internet par exemple, et qu'elle les montre bien à l'utilisateur en lui permettant de les autoriser ou de les refuser. Accéder à une Webcam demandera à la fois de le déclarer convenablement et d'être accepté par l'utilisateur par exemple, c'est le broker qui s'assure de tout cela pour la sécurité de l'OS et de l'utilisateur.

Si WinRT n'est pas managé, il s'apparente par certains contrôles à un environnement managé. Le broker est l'élément clé de cette surveillance des applications.

Asynchronisme

Windows 8 est présenté comme un OS "rapide, fluide et réactif". Et de ce qu'on peut tester c'est plutôt vrai sur Surface et sur PC avec les indispensables corrections apportées à Windows 8.1. Microsoft met très souvent en avant les performances de son nouvel OS. Ils veulent que les utilisateurs aient une expérience fluide et réactive même sur des machines qui ne le sont pas forcément. Pour la première fois d'ailleurs passer à une nouvelle version de Windows n'implique pas forcément d'acheter une machine plus puissante avec plus de Ram. Windows 8 peut même redonner vie à des machines anciennes. Je l'ai testé avec succès.

Techniquement il n'y a pas des milliers de solutions à cette équation, les miracles n'existant pas... l'asynchronisme est l'unique issue.

Cela implique que les développeurs doivent faire beaucoup d'efforts pour comprendre et utiliser l'asynchronisme de WinRT. Car une limite fatidique a présidé à la conception des APIs : Tout code pouvant durer plus de 50 millisecondes fait l'objet d'une API asynchrone sans option pour une version synchrone.

C'est radical, le curseur n'a pas été réglé très haut, et le couperet tombe vite. Un nombre important d'APIs de WinRT se trouvent donc être asynchrones.

Par exemple tous les accès à des fichiers sont asynchrones car potentiellement qu'il s'agisse de la création, de l'accès ou de la modification d'un fichier sur disque dur, SSD ou autre, tout cela peut prendre plus de 50ms.

Une telle programmation avec les langages que nous possédons aujourd'hui serait un véritable enfer. Heureusement, Microsoft a anticipé cet inconvénient en introduisant `async` et `await` dans C#. Cela permet de traiter les APIs asynchrones de façon simple et performante.

Nous verrons souvent dans de prochains billets sur la programmation WinRT à quel point ces nouveaux mots clé sont indispensables.

Quel est le contenu de WinRT ?

Une fois posés les fondamentaux de WinRT, ses fonctions, son architecture, il est intéressant de se demander ce qu'il a dans la boite en tout cas pour un développeur .NET ...

J'ai posté la liste des APIs de WinRT dans un précédent billet (Metro Style / WinRT : les APIs), j'invite le lecteur à s'y référer pour regarder chaque namespace et son contenu.

Dans ma série "De Silverlight à WinRT" en quatre parties (partie 1, partie 2, partie 3) j'expose aussi de nombreuses facettes de WinRT qui permettent de mieux en comprendre les rouages.

Enfin, dans le quatrième de volet de cette série je parle du projet "WinRT Genome Project" qui permet de comparer les APIs .NET à celles de WinRT, outil bien pratique pour trouver des équivalences et mesurer les différences entre les deux frameworks.

On comprend alors que la surface de WinRT est énorme et que dans de nombreux cas on écrira du code purement WinRT et non plus du code managé. Par exemple en Xaml, un simple contrôle de type Button ne se trouve plus dans `System.Windows.Controls` mais plutôt dans `Windows.UI.Xaml`. Comme expliqué plus haut, toutes les APIs WinRT sont dans le namespace "`Windows.*`". Le button Xaml est donc un code purement WinRT et non plus .NET même lorsqu'on pense écrire du code managé C#/Xaml sous WinRT.

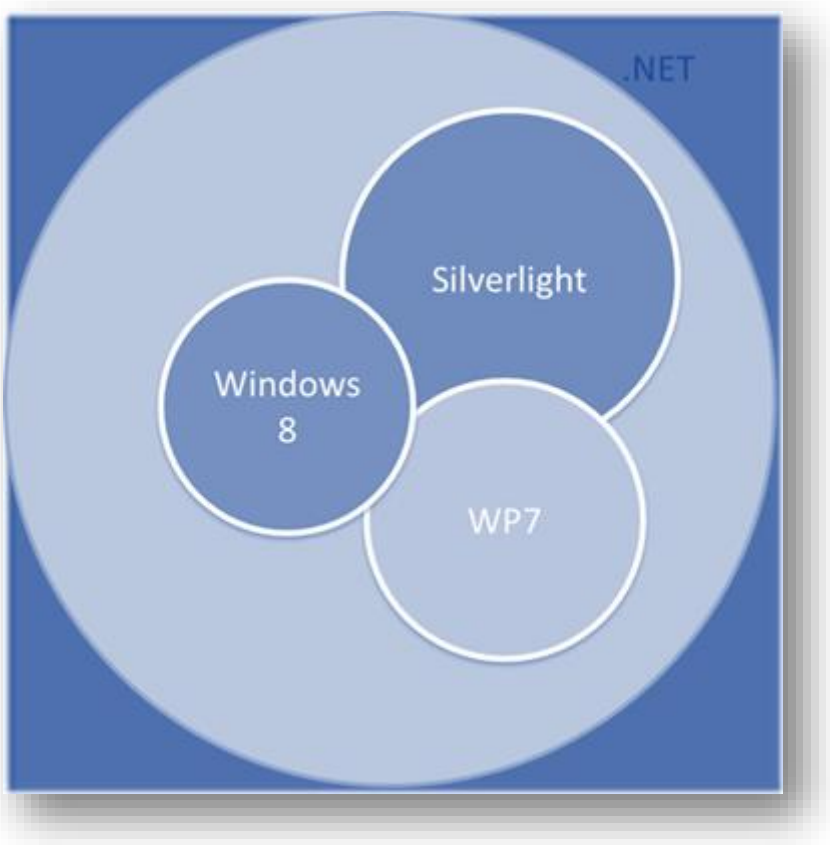
La question reste alors de savoir si on peut utiliser tout .NET dans une application WinRT C#/Xaml...

La réponse est non.

Le framework .NET a été au fil du temps "saucissonné" pour s'adapter à différents contextes et il existe en réalité déjà plusieurs frameworks .NET... Le complet, le client profile, celui de Silverlight, celui de WPF, celui de Windows Phone.

Microsoft a créé différents “profils” d’utilisation de .NET, chacun ayant une surface adaptée à son contexte et possédant moins d’APIs que le framework complet, mais souvent en contrepartie des APIs spécialisées pour le contexte en question.

Windows 8 n’échappe pas à cette règle. Le framework .NET utilisable sous WinRT (et non pas celui du bureau classique, complet) n’est qu’un profil parmi les autres. Le schéma suivant résume la situation de façon évidente :



Dans ce profil particulier les namespaces suivants sont disponibles :

- `System.Collections`
- `System.ComponentModel`
- `System.Diagnostics`
- `System.Dynamic`
- `System.Globalization`
- `System.IO`
- `System.Linq`
- `System.Net`
- `System.Numerics`
- `System.Reflection`

- `System.Resources`
- `System.Runtime`
- `System.Security`
- `System.ServiceModel`
- `System.Text`
- `System.Threading`
- `System.Xml`

Même si cette liste parait longue, le profil WinRT de .NET est assez compressé... Il est même plus petit que le profil .NET de Windows Phone ! Comme les applications Modern UI ciblent principalement le consommateur de base, tout ce qui concerne l'entreprise n'est pas couvert. Il n'y pas de support du mode Console, pas d'intégration ASP.NET, pas d'Entity Framework... Gardez ces limites à l'esprit lorsque vous allez vous lancer sous WinRT !

D'un autre côté n'allez pas conclure que la liste ci-dessus résume tout ce qui est utilisable en C#/Xaml... Bien entendu que non : comme expliqué les applications managées ont aussi accès à tout WinRT via le namespace `Window.*` !

Mais certaines choses n'auront pas d'équivalent. Il faut en avoir conscience avant de se lancer dans une application ou surtout dans un portage d'application existante.

Metro Style / WinRT : les API's

WinRT... avant que je n'aborde durant l'été toute une série d'articles sur le sujet, je vous propose un premier voyage initiatique, le meilleur, celui qui permet de savoir de quoi on dispose pour travailler : les API's.

Bêta / Preview ...

Au moment de l'écriture de ce billet Windows 8 n'était pas releasé et seule une CP (sorte de Release Candidate qui cache son nom on ne sait pas trop pourquoi) était disponible pour le public et pour les développeurs (avec les outils comme VS et Blend). La CP est forcément très proche de la version finale et on voit mal, à quelques mois de la sortie de cette dernière, Microsoft revoir de fond en comble toutes les API, ce qui ne fut pas le cas d'ailleurs.

Un voyage solitaire

Comme indiqué en introduction, toute une série d'articles est en préparation sur WinRT – vous les retrouverez dans le présent livre PDF puisqu'ils ont déjà été publiés entre temps. Il n'est donc pas question ici de parler d'un article au sens habituel mais bien d'un voyage de découverte à faire seul, pour se mettre en tête quelques points de repère qui seront utiles par la suite.

Il n'est bien entendu pas question non plus que je présente chaque API l'une après l'autre et que je traduise les centaines de pages de la documentation MSDN, cela n'aurait ni sens ni intérêt.

Alors il faudra vous promenez seul dans la liste ci-dessous, piocher de ci de là, en fonction de votre humeur, de vos centres d'intérêts.

Le but n'est pas d'apprendre par cœur les APIs mais de vous faire une idée des namespaces dont on dispose et de ce qu'ils couvrent. C'est essentiel avant de commencer à utiliser la plateforme.

Certains trouveront cela "un peu court", mais qu'importent les grincheux, ceux qui sauront commencer le voyage par son commencement en seront récompensés ! :-)

Alors bonne découverte !

Les API's de WinRT

Utilisez un clic souris pour ouvrir les liens ci-dessous dans le PDF.

- **Windows.ApplicationModel**
- **Windows.ApplicationModel.Activation**
- **Windows.ApplicationModel.Background**
- **Windows.ApplicationModel.Contacts**
- **Windows.ApplicationModel.Contacts.Provider**
- **Windows.ApplicationModel.Core**
- **Windows.ApplicationModel.DataTransfer**
- **Windows.ApplicationModel.DataTransfer.ShareTarget**
- **Windows.ApplicationModel.DataTransfer.SendTarget**
- **Windows.ApplicationModel.Infrastructure**
- **Windows.ApplicationModel.Resources**
- **Windows.ApplicationModel.Resources.Core**
- **Windows.ApplicationModel.Search**
- **Windows.ApplicationModel.Store**
- **Windows.Data.Json**
- **Windows.Data.Xml.Dom**
- **Windows.Data.Xml.Xsl**
- **Windows.Devices.Enumeration**
- **Windows.Devices.Enumeration.Pnp**
- **Windows.Devices.Geolocation**

- Windows.Devices.Input
- Windows.Devices.Portable
- Windows.Devices.Printers.Extensions
- Windows.Devices.Sensors
- Windows.Devices.Sms
- Windows.Foundation
- Windows.Foundation.Collections
- Windows.Foundation.Diagnostics
- Windows.Foundation.Metadata
- Windows.Globalization
- Windows.Globalization.Collation
- Windows.Globalization.DateTimeFormatting
- Windows.Globalization.Fonts
- Windows.Globalization.NumberFormatting
- Windows.Graphics
- Windows.Graphics.Display
- Windows.Graphics.Imaging
- Windows.Graphics.Printing
- Windows.Graphics.Printing.Advanced
- Windows.Management.Core
- Windows.Management.Deployment
- Windows.Media
- Windows.Media.Capture
- Windows.Media.Devices
- Windows.Media.Playlists
- Windows.Media.PlayTo
- Windows.Media.Protection
- Windows.Media.Transcoding
- Windows.Media.VideoEffects
- Windows.Networking
- Windows.Networking.BackgroundTransfer
- Windows.Networking.Connectivity
- Windows.Networking.NetworkOperators
- Windows.Networking.Proximity
- Windows.Networking.PushNotifications
- Windows.Networking.Sockets
- Windows.Security.Authentication.Live
- Windows.Security.Authentication.Web
- Windows.Security.Credentials
- Windows.Security.Cryptography
- Windows.Security.Cryptography.Certificates
- Windows.Security.Cryptography.Core
- Windows.Security.Cryptography.DataProtection
- Windows.Storage
- Windows.Storage.AccessCache
- Windows.Storage.BulkAccess
- Windows.Storage.Compression
- Windows.Storage.FileProperties
- Windows.Storage.Pickers
- Windows.Storage.Pickers.Provider

- **Windows.Storage.Search**
- **Windows.Storage.Streams**
- **Windows.System**
- **Windows.System.Display**
- **Windows.System.Threading**
- **Windows.System.UserProfile**
- **Windows.UI.ApplicationSettings**
- **Windows.UI.Core**
- **Windows.UI.Core.AnimationMetrics**
- **Windows.UI.Input**
- **Windows.UI.Input.Inking**
- **Windows.UI.Notifications**
- **Windows.UI.Popups**
- **Windows.UI.StartScreen**
- **Windows.UI.ViewManagement**
- **Windows.UI.WebUI**
- **Windows.UI.Xaml**
- **Windows.UI.Xaml.Automation**
- **Windows.UI.Xaml.Automation.Peers**
- **Windows.UI.Xaml.Automation.Provider**
- **Windows.UI.Xaml.Automation.Text**
- **Windows.UI.Xaml.Controls**
- **Windows.UI.Xaml.Controls.Primitives**
- **Windows.UI.Xaml.Data**
- **Windows.UI.Xaml.Documents**
- **Windows.UI.Xaml.Input**
- **Windows.UI.Xaml.Interop**
- **Windows.UI.Xaml.Markup**
- **Windows.UI.Xaml.Media**
- **Windows.UI.Xaml.Media.Animation**
- **Windows.UI.Xaml.Media.Imaging**
- **Windows.UI.Xaml.Media.Media3D**
- **Windows.UI.Xaml.Navigation**
- **Windows.UI.Xaml.Printing**
- **Windows.UI.Xaml.Resources**
- **Windows.UI.Xaml.Shapes**
- **Windows.Web.AtomPub**
- **Windows.Web.Html**
- **Windows.Web.Syndication**

Conclusion

Rien de bien palpitant il faut le dire dans une telle liste, mais ce qui se cache derrière et la connaissance qu'on peut en tirer sont loin d'être négligeables.

Et puis c'est un premier contact. Une sorte de mise en bouche.

Pour l'entrée, le plat et le dessert, il faudra attendre que le chef entre en cuisine, et ça c'est cet été !

Windows 8 : Le cycle de vie d'une application

Comme je l'ai déjà évoqué les applications WinRT sous Windows 8 possèdent un cycle de vie particulier se rapprochant de celui des applications pour smartphone, notamment le fameux "tombstoning". Comment gérer cette particularité dans vos développements ?

Un mot d'ordre : la transparence pour l'utilisateur

Le modèle de cycle de vie des applications sous Windows 8 signifie est calqué sur celui mis en place dans les OS de smartphones : l'utilisateur n'a pas à gérer la fin d'exécution d'un programme.

Sur les OS pour smartphone l'intention est manifestement technique bien plus qu'ergonomique. En effet, la durée de vie des batteries, les simplifications nécessaires pour en faire des machines "grand public", la faible mémoire installée, la puissance limitée des processeurs, etc, tous ces éléments ont forcé les concepteurs d'OS à imaginer un mécanisme permettant de donner l'illusion d'une fonctionnement fluide et multitâche là où en réalité il n'y a qu'une application d'avant-plan et nécessitée de supprimer de la mémoire certaines applications sans que cela ne gêne l'utilisateur. Ce dernier doit en effet pouvoir revenir à l'une des applications précédentes comme on passe d'une fenêtre à l'autre sur un PC : sans rupture (sans avoir l'impression que l'application vient d'être relancée en ayant perdu le contenu en cours au moment de son abandon).

Sous Windows 8 qui se voue aussi bien aux PC qu'aux unités mobiles (tablettes et smartphones) toutes ces motivations sont présentes. Toutefois sur PC Microsoft aurait pu choisir un mode de fonctionnement plus proche des habitudes "desktop". Il n'en a rien été. Il s'agit donc ici d'un choix de cohérence (Windows 8 fonctionne partout de la même façon) et d'un parti pris très fort bousculant les habitudes des utilisateurs de PC. Choix assumé, Windows 8 se veut réellement et radicalement différent.

Techniquement les développeurs peuvent aussi tirer meilleur parti des machines en concevant une expérience utilisateur fluide qui n'affecte pas la durée de vie des batteries lorsque l'application passe en arrière-plan.

En utilisant les nouveaux évènements de cycle de vie, vos applications Windows 8 donneront la sensation qu'elles sont toujours vivantes, même si elles ne fonctionnent réellement plus lorsqu'elles passent hors de l'écran.

L'utilisateur, certainement habitué au PC et à ses facilités, quitte souvent une application en cours d'exécution pour faire autre chose, momentanément ou non. Sur un PC il sait qu'il retrouvera la fenêtre minimisée ou laissée en arrière-plan dans l'état où il l'a laissée. Il s'attend à ce que cela se passe de la même façon sur son smartphone ou sa tablette.

Quant aux nouveaux utilisateurs de type "mamie du Cantal" ou Ados se fichant bien des contraintes techniques, il est évident que pour eux il est "naturel" de pouvoir revenir sur qu'on a abandonné sans avoir tout perdu du travail en cours. Ces utilisateurs-là ne se posent aucune question sur cette gestion du cycle de vie des applications car elle est tout simplement ce à quoi ils s'attendent !

C'est ainsi que Windows 8 généralise sur PC un mode de gestion du cycle de vie des applications qui nous vient des unités mobiles à la fois parce que cet OS fonctionne aussi sur les machines de ce type et que Microsoft a eu le courage d'innover en imposant une expérience utilisateur cohérente, fluide et transparente quel que ce soit le type d'ordinateur le faisant tourner.

Ainsi, le nouveau modèle de cycle de vie sous Windows 8 met l'accent sur les applications au premier plan pour assurer à l'utilisateur une expérience dynamique et immersive en exploitant au mieux la puissance de la machine qu'il utilise.

En rupture totale avec le passé, Microsoft propose pour la première fois un OS qui consomme moins que les versions précédentes et qui sait s'adapter aussi bien à un processeur ARM lent avec peu de mémoire qu'à une bombe de bureau dotée d'un processeur 8 cœurs et de plusieurs Gigas de RAM...

Le nouveau cycle de vie des applications participe pleinement à ce nouvel objectif. Il est donc essentiel de le comprendre et de le maitriser. Je serai même tenté de comparer ce changement radical à celui que nous avons vécu lorsque Windows est passé du multitâche coopératif au multitâche préemptif. Toutefois Windows 8 a été conçu avec tout ce qu'il faut pour gérer simplement ces nouvelles contraintes là où le passage du multitâche coopératif au préemptif fut plus douloureux pour certains développeurs...

Le cycle de vie d'une application Windows 8 "Modern UI"

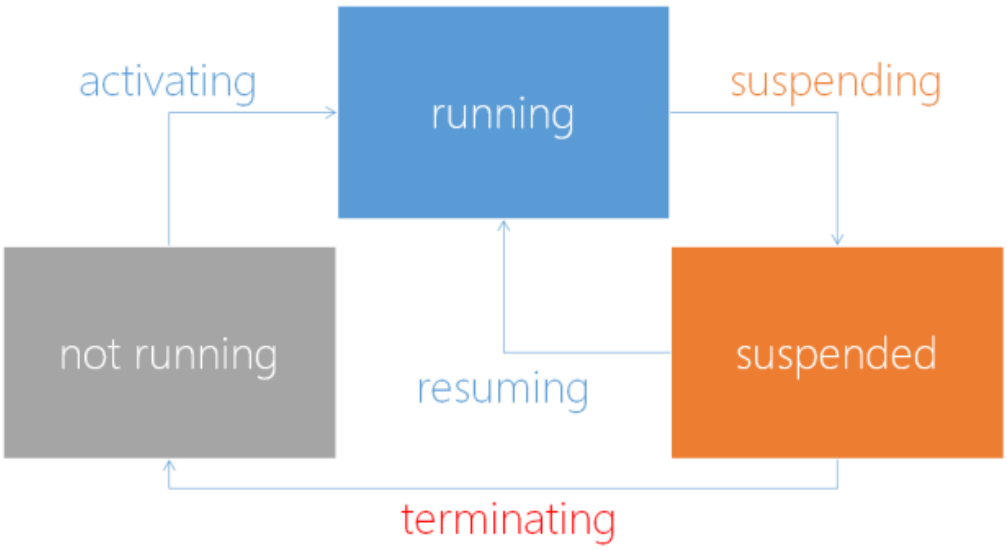
Les choses sont simples, au départ au moins : une application Windows 8 est à tout moment dans l'un des quatre états suivants :

- not running (ne tournant pas)
- running (en cours d'exécution)
- suspended (suspendue)
- terminated (fermée)

Comme durant toute sa "vie" une application ne fera que passer de l'un de ces états à un autre, il existe des transitions d'état matérialisées par des évènements qu'elle peut (et doit) gérer.

Ce que j'appelle la "vie" d'une application est l'espace de temps de son existence sur une machine donnée, c'est à dire entre le moment où elle installée et celui où elle est désinstallée. Entre ces deux bornes, elle ne pourra être que de l'un des quatre états listés ci-dessus.

Le schéma ci-dessous montre ces quatre états ainsi que les quatre évènements de transition que l'application aura à gérer :



Toute application Windows 8 oscille entre ces 4 états au gré des actions de l'utilisateur qui passe d'une application à l'autre.

Généralement une application passera son temps à basculer entre l'état "running" et "suspended". Il est donc indispensable de gérer les transitions correctement.

On peut résumer tout cela par le tableau suivant :

Evènement	Depuis	Vers
activated	not running	running
suspending	running	suspended
resuming	suspended	running
terminating	suspended	not running

La Suspension

En général une application Windows 8 s'arrête de fonctionner quand l'utilisateur passe à une nouvelle application (ou retourne au menu principal).

Windows 8 suspend une application qui n'est plus en avant-plan (évènement "suspending"). Quand l'application est suspendue son état est figé en mémoire. Elle ne peut pas tourner dans cet état qui est une sorte de "parking à application". En revanche Windows 8 peut instantanément la sortir du "parking" et la réactiver lorsque l'utilisateur la fera passer à nouveau en avant-plan (évènement "resuming").

La suspension d'une application garantit notamment qu'elle ne peut plus tirer sur la batterie ce qui est essentiel sur les unités mobiles, et que l'application d'avant-plan qui arrive à sa place bénéficiera de toutes les ressources disponibles pour en assurer la fluidité.

L'astuce est simple, mais elle donne d'excellents résultats puisqu'on retrouve le même procédé sous Android depuis quelques années déjà et avec la réussite qu'on sait pour cet OS mobile.

Le mécanisme réel est malgré tout plus subtil. En fait quand une application perd l'avant-plan Windows 8 attend quelques secondes avant de la suspendre. Ce petit temps contribue lui aussi à assurer une meilleure fluidité de l'expérience utilisateur puisque si ce dernier revient rapidement sur l'application l'OS n'a pas eu à perdre de temps à la ranger puis à la ressortir du "parking" des applications suspendues. L'application elle-même n'ayant reçu aucun évènement particulier dans ce laps de temps n'a pas perdu de temps à enregistrer puis à relire son état.

Passé ce petit délai, Windows 8 va décider de suspendre l'application si l'utilisateur ne la reprend pas tout de suite. C'est à ce moment qu'elle va recevoir l'évènement `suspending`.

C'est un moment particulier pour l'application car elle doit utiliser cette (courte) opportunité pour sauvegarder son état sur un stockage permanent (disque, ssd,

flash...). Très souvent l'application est reprise (*resume*) tel quel, c'est à dire depuis la mémoire de la machine et il n'y a rien de spécial à faire alors (son état n'a pas changé puisque son image mémoire est restée intacte). Toutefois il est nécessaire de sauvegarder l'état quand l'évènement "`suspending`" arrive car l'application ne peut pas savoir à l'avance si elle va rester en mémoire ou bien si l'OS va décider à un moment donné de la supprimer totalement de la RAM pour gagner de la place. Et là, lorsque "`resuming`" sera reçu par l'application ce sera après avoir été rechargée depuis le stockage et il lui faudra se remettre dans l'état dans lequel l'utilisateur l'a quittée sans que celui-ci ne se doute de quoi que ce soit.

C'est une des parties du mécanisme simple mais subtile qui donne l'impression à l'utilisateur que les applications sont toujours "vivantes" peu importe combien de temps il les a laissé en arrière-plan (et même si la machine a été éteinte entre temps !).

Windows 8 laisse 5 secondes à une application pour sauvegarder son état. 5 secondes après avoir notifié l'application par l'évènement "`suspending`" Windows 8 la "terminera" (suppression de la mémoire).

C'est là que les choses se compliquent un peu. Comme le temps imparti à la sauvegarde de l'état est limité, comme certains machines peuvent être lentes, comme il peut y avoir beaucoup de choses à sauvegarder, on comprend vite que si on veut que cette phase reste déterministe l'application doit se débrouiller pour sauvegarder son état au fur et à mesure de son fonctionnement, sans toutefois trop consommer en invoquant sans cesse des entrées/sorties... C'est ce qu'on appelle la sauvegarde incrémentale.

Typiquement la gestion de suspending ressemble à cela :

```
public App()
{
    InitializeComponent();
    this.Suspending += new SuspendingEventHandler(OnSuspending);
}

async protected void OnSuspending(object sender, SuspendingEventArgs args)
{
    // La suspension donne une chance à l'application de sauvegarder son état
    // Comme l'écriture disque est asynchrone nous demandons un defferal
    // qui s'assure que l'application ne sera pas terminée avant la fin
    // d'écriture sur disque.
```

```
    SuspendingDeferral deferral = args.SuspendingOperation.GetDeferral();
    // Nous utilisons SuspensionManager qui gère les données de session
    // to a Dictionary and then serializing that data to a file
    SuspensionManager.SessionState["CodeClient"] = currentCustomerID;
    SuspensionManager.SessionState["Facture"] = currentInvoice;
    SuspensionManager.SessionState["Solde"] = currentCustomer.Balance;
    await SuspensionManager.SaveAsync();
    // envoi d'une notification sur la tuile (optionnel)
    Tile.SendTileUpdate(currentCustomer.Name, currentInvoice.ID,
       currentInvoice.Total);
    deferral.Complete();
}
```

Dans cet exemple fictif nous voyons :

- La prise en charge de l'évènement `Suspending`
- Le gestionnaire d'évènement `OnSuspending`
    - L'obtention d'un "`deferral`" pour bloquer en quelque sorte le processus le temps de sauvegarder l'état de l'application
    - L'utilisation de `SuspensionManager` qui permet de gérer des données de session un peu comme on le fait sous ASP.NET
    - La sauvegarde asynchrone des données (avec un `await` C# 5)
- La mise à jour de la tuile de l'application, ce qui est optionnel
- L'indication de fin de traitement en "relâchant" le "`deferral`"

Un "deferral" est un objet particulier permettant de demander à l'OS un sursis. La traduction française est d'ailleurs explicite pour une fois, puisque "deferral" se traduit par "ajournement" ou "report".

Le Resume (reprise)

Quand une application est "résumée" (horrible anglicisme puisqu'on peut traduire "resume" par "reprise" mais qui permet d'appuyer sur le sens techniquement particulier qu'on donne ici au terme) elle reprend l'avant-plan dans l'état où elle était au moment où Windows l'a suspendue.

Techniquement : les données et l'état de l'application sont gardées en mémoire pendant la suspension (le fameux "parking"), ainsi quand Windows 8 la reprend (*resume*) elle se trouve toujours dans le même état et il n'y a pas de nécessité de restaurer les données préalablement sauvegardées.

Néanmoins, comme vous voulez que votre application donne l'impression d'être "vivante"' il sera peut-être nécessaire d'effectuer quelques rafraichissements. Par exemple si vous affichez des données provenant d'un service Web, d'une base de données distante, etc, et comme votre application peut rester longtemps en état suspendu, il sera certainement nécessaire de faire quelques requêtes pour afficher les données dans leur état actuel et non celui dans lequel elles étaient lors de la suspension...

Quand l'application sort du "parking" des suspendues, elle reçoit un évènement "`resuming`", c'est dans le gestionnaire de ce dernier que vous effectuerez les mises à jours nécessaires.

Si nous reprenons l'exemple de code précédent (une gestion commerciale fictive, l'application affichant un client et une facture de ce dernier ainsi que le solde du compte client) nous comprenons que si le client ou sa facture ne risquent pas d'avoir changés il convient malgré tout :

1. de vérifier que la facture est toujours dans le même état (par exemple toujours due ou réglée),
2. que le compte client existe toujours (il peut avoir été supprimé),
3. dans tous les cas il faudra interroger le solde du compte client qui a toutes les chances d'avoir bougé (nouvelles factures émises, nouveaux règlements reçus...).

Suivant cet exemple, le code d'une reprise sera le suivant :

```
public App()
{
    InitializeComponent();
    this.Resuming += new EventHandler<object>(App_Resuming);
}

 async private void App_Resuming(object sender, object e)
 {
    //mise à jour des données
    currentInvoice = await IsInvoiceValid(currentInvoice.ID) ?
currentInvoice : null;
    currentCustomer = await IsCustomerValid(currentCustomer.ID) ?
currentCustomer : null;
    currentCustomer.Balance = await GetCustomerBalance(currentCustomer.ID);
    if (currentCustomer==null) customerSelection.Visible = true;
    if (currentInvoice==null && currentCustomer!=null)
invoiceSelection.Visible = true;
    // notifications de la tuile
    Tile.SendTileUpdate(currentCustomer==null?"":currentCustomer.Name,

currentInvoice==null?"":currentInvoice.ID,currentCustomer==null?0d:currentC
```

```
ustomer.Balance);
}
```

Encore une fois, cet exemple est purement fictif, ce sont les mécanismes et leurs principes qui comptent, pas l'exactitude du code.

L'Activation

Au départ les choses sont simples : l'activation de l'application n'est rien d'autre que le moment où elle lancée par Windows.

Mais nous sommes dans un environnement hautement collaboratif et intégré... De fait c'est au moment de l'activation qu'une application peut prendre connaissance du contexte de cette activation et celui-ci peut dépendre des contrats pris en charge ainsi que du mode "*main view activation*" ou "*hosted view activation*". La différence est importante puisque dans un cas l'application sera appelée "pour elle-même" alors que dans l'autre elle sera invoquée à l'intérieur d'un process dont elle n'est pas l'acteur principal (par exemple si elle est choisie comme cible ou source d'un partage de données).

Le sujet méritant de plus amples explications j'y reviendrais dans un autre billet. Pour l'instant nous resterons concentrés sur le cycle de vie, considérant l'activation sous cet unique angle de vue.

Une application peut être terminée par Windows 8 à n'importe quel moment une fois qu'elle a été suspendue. De multiples raisons peuvent entrainer l'arrêt définitif d'une application : l'utilisateur la ferme volontairement, ou se délog, ou bien l'OS a besoin de ressources et il doit libérer de la mémoire.

Si l'utilisateur lance l'application après qu'elle ait été "terminée" elle reçoit l'évènement d'activation et Windows affiche son Splash Screen le temps qu'elle se charge. Le Splash Screen, comme nous l'avons vu dans un billet précédent est présent dans les assets du projet créé par défaut par Visual Studio, charge bien entendu au développeur (ou plutôt au designer) de le personnaliser. Mais c'est un mécanisme désormais intégré au fonctionnement "normal" d'un logiciel après 15 ou 20 ans d'existence sous la forme d'un ajout optionnel que chacun faisait à sa manière.

Windows 8 normalise ainsi de nombreuses choses, et ce côté normatif fort est à la fois un avantage certain et une contrainte à bien gérer tant pour le designer que pour le développeur.

> *Il faut noter que certains designers considèrent que l'affichage d'un Splah Screen est l'une des pires erreurs de design qu'on puisse commettre : l'application doit s'afficher tout de suite et proposer immédiatement quelque chose d'utile, le splash pouvant donner l'envie de zapper avant même que l'application ne soit affichée. C'est une conception des choses assez intéressante et qui mérite débat et au moins d'y réfléchir car cela implique un premier écran très rapide à charger mais fonctionnel. Sous Android une application est une série d'Activités indépendantes chargées en fonction de la page affichée. Il est donc plus facile d'avoir une première Activité très simple et rapide à charger que sous WinRT. Le problème soulevé reste toutefois pertinent !*

L'évènement d'activation est un point d'entrée qui permet à l'application de savoir si elle doit ou non restaurer un contexte d'utilisation et si elle doit exécuter le code de restauration dans l'affirmative.

Les arguments de l'évènement précisent l'état précédent de l'application (que celle-ci ne peut pas deviner) notamment par la propriété `PreviousExecutionState`.

La valeur de cette propriété est l'une de celle prévue dans l'énumération `Windows.ApplicationModel.Activation.ApplicationExecutionState`.

C'est en partie grâce à cette valeur que l'application qui reçoit le signal d'activation peut décider si elle doit se charger avec son affichage par défaut ou bien si elle doit restaurer un état enregistré.

Le tableau ci-dessous résume les cas particuliers et les actions à entreprendre :

Raison de l'arrêt	PreviousExecutionState	Action à exécuter
Terminée par l'OS	Terminated	Restaurer la session
Fermée par l'utilisateur	ClosedByUser	Démarrage normal
Fin inopinée ou autre	NotRunning	Démarrage normal

En réalité comme on le voit un seul cas réclame la restauration de la session de l'utilisateur : lorsque l'application a été terminée par Windows 8 (après une suspension).

Si l'utilisateur a fermé l'application, ou bien si l'application a connu un arrêt inopiné (crash application ou système, ou bien si l'utilisateur n'a pas relancé l'application depuis que la session utilisateur a commencé), aucune restauration n'est nécessaire et l'application doit démarrer avec son affichage par défaut.

Il existe deux autres états possibles pour `PreviousExecutionState` : `Running` et `Suspended`. Si ces derniers peuvent être pris en compte dans des cas particuliers ils n'impliquent en revanche pas la restauration de la session puisque l'application est encore "intacte" en mémoire.

Gérer l'activation se fait comme pour les exemples précédents, seul le code de restauration est intéressant et ce dernier varie en fonction des données qui sont sauvegardées. Nous nous passerons ainsi d'un exemple fictif de code ici.

Quelques conseils

Le décor a été posé et le mécanisme décrit. Mais reste à savoir deux ou trois petits choses bien utiles pour bien gérer le cycle de vie d'une application sous Windows 8.

Démarrer sans reprise de session

Parfois il est préférable de ne pas restaurer la session utilisateur. Par exemple cela fait très longtemps que l'utilisateur n'a pas lancé votre application, "longtemps" est ici relatif et dépend de l'application et de la temporalité de son contenu, à vous de voir combien de temps cela fait exactement. Mais dans un tel cas je vous conseille d'afficher l'écran de démarrage normal et non pas les vieilles données. Cela rendra votre application plus intelligente aux yeux de l'utilisateur.

Si nous imaginons une application qui affiche des news par exemple, et si l'utilisateur n'a pas utilisé l'application depuis "longtemps", restaurer des news qui datent donnera une impression négative, dans un tel cas mieux vaut ne rien afficher d'autre que l'écran de démarrage. Les exemples de ce type sont nombreux (météo, blog reader…), mais lorsqu'on écrit son application il faut y penser…

Sauvegarder les bonnes données au bon moment

Comme évoqué plus haut dans ce billet, l'évènement de suspension n'est pas un espace sans fin, Windows 8 vous donne du temps pour sauvegarder vos données mais passé un délai il "coupera le jus" à l'application et s'en sera terminé… Si

l'application gère des données volumineuses ou longues à sauvegarder, il est préférable alors d'adopter une stratégie de sauvegarde incrémentale au fur et à mesure de l'utilisation.

Il y a deux types de données qu'une application peut manipuler : les données de session et les données utilisateurs.

Les premières sont celles qu'on sauvegarde de façon temporaire (voir les exemples de code dans le billet) pour se rappeler dans quel état est l'application (le numéro de la page d'un document qui était en cours d'affichage, le nom de ce document, le code d'une action boursière affichée, le nom de la page de l'application en cours d'affichage, etc, etc...).

Les secondes sont les données produites par l'utilisateur. Un texte en cours de rédaction, une photo, une capture vocale, un dessin... Ces données-là doivent être accessible à l'utilisateur peu importe ce qu'il se passe. Elles ne sont pas sauvegardées dans les données de session mais dans l'espace dédié à l'application pour l'utilisateur en cours.

Il est donc important de bien faire la différence entre ces deux types de données et de les sauvegarder au bon moment et aux bons endroits !

Gestion des ressources externes

Si l'application a acquis des ressources externes comme un handle de fichier, une référence à un service, etc, elle doit absolument les libérer quand elle reçoit l'évènement `Suspending`. De la même façon, c'est quand elle est reprise (*resume*) qu'elle doit réclamer à nouveau ces ressources.

C'est un petit détail à ne pas oublier car il peut faire tout planter !

Ne fermez pas vos applications !

Aussi troublant que cela puisse paraitre quand on utilise Windows 8 dans les premiers temps les applications ne doivent pas donner à l'utilisateur la possibilité d'être fermées. La première fois que j'ai lancé la première bêta de Windows 8 j'ai pesté un bon moment car je ne trouvais pas la petite croix ou un procédé similaire pour fermer les applications trouvant délirant d'être obligé de faire Alt-F4 pour mettre fin à chaque appli lancée... avant de comprendre que j'avais affaire à un OS de téléphone et non plus à Windows 7 !

Il faut bien comprendre, autant côté développeur qu'utilisateur, que Windows 8 gère le cycle de vie des applications lui-même et qu'il optimise la mémoire tout seul. Ni

l'utilisateur ni le développeur ne doivent fermer une application dans l'espoir de gagner de la place. C'est à l'OS de se débrouiller.

Et dans les cas extrêmes, il reste Alt-F4 bien entendu ...

Windows 8 prévoit aussi une gestuelle (le swipping de haut en bas) qui a le même effet, mais Chut! Il ne faut pas le dire puisque l'utilisateur ne doit pas fermer les applications !

Conclusion

Avec trois évènements (`Suspending`, `Resuming` et `Activated`) une application Windows 8 dispose d'un mécanisme lui permettant de paraitre toujours vivante, toujours prête à fonctionner et sachant reprendre le fil de la dernière action en cours comme si elle avait toujours été en avant-plan... Ces évènements sont le reflet d'une gestion totalement nouvelle du cycle de vie des applications, en tout cas sur PC, et garantissent, par les mécanismes sous-jacents, une utilisation fluide, dynamique et transparente des applications installées tout en préservant la puissance processeur et les ressources de la machine.

La cohérence de l'expérience utilisateur du smartphone au PC est un des avantages de Windows 8 sur l'ensemble de sa concurrence. La cohérence de son fonctionnement pour le développeur aussi.

Après un petit temps d'hésitation, tellement la surprise est grande, on finit par comprendre que ce choix est intelligent. C'est un peu la même démarche intellectuelle que celle que nous avons connu lorsque nous sommes passés de langages et d'environnement dans lesquels les destructeurs d'objet étaient essentiels à C# et son environnement managé qui prenait en compte totalement la libération des objets, la fameuse gestion « managée ». Beaucoup ont résisté, voire pesté contre cette avancée pour finir par l'adopter et la trouver totalement naturelle et normale aujourd'hui...

Il en sera pareil demain pour la gestion du cycle de vie des applications de Windows 8. Un jour on aura même oublié qu'il y avait des versions de Windows avec des fenêtres qu'il fallait fermer pour libérer de la RAM et redonner de l'air à son CPU...

Mais je reviendrais fatalement sur tous ces aspects dans d'autres posts, alors...

Windows Store Apps : comprendre les templates

Les templates de projets Windows Store Apps offrent des structures de base pour débuter le développement d'applications. Connaître et comprendre l'architecture de ces projets est un préliminaire incontournable.

Les templates

Il existe plusieurs catégories d'applications et chacune propose un lot de templates, mais parmi ces derniers on en retrouve trois qui sont communs à toutes les catégories : "`Blank App`", "`Grid App`" et "`Split App`" (comme toujours j'utilise uniquement des versions US, le lecteur saura j'en suis sûr retrouver la traduction de ces noms dans la VF).

Tour d'horizon

Les templates sont formatés pour un look tablette mais ils sont bien entendu utilisables pour des applications destinées aux PC tant qu'on reste, cela est évident, dans le cadre d'applications Windows Store.

Le modèle "Blank"

Ce modèle créée un projet ayant une seule page. Il n'a aucun contrôle prédéfinis ni aucune mise en page. Si vous exécutez le template "Blank" vous obtiendrez une page totalement vide sans aucune fonctionnalité.

Le modèle "Grid App"

Ce modèle créée un projet ayant trois pages dotées d'une navigation basée sur des groupes d'éléments arrangés sous forme de grille.

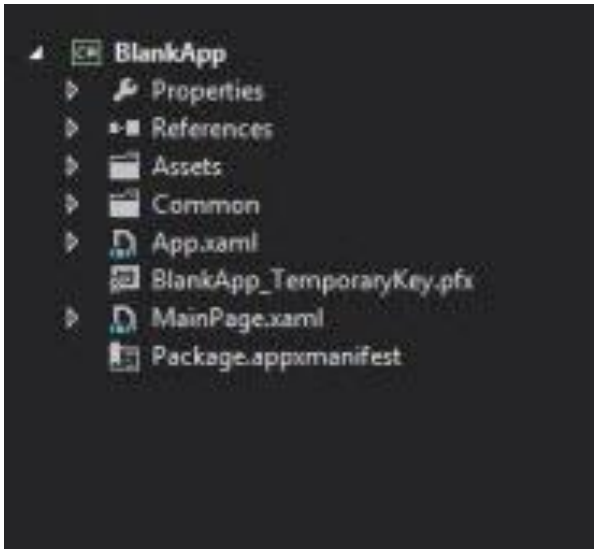
La page principale présente tous les éléments dans des groupes. Quand on clique sur un groupe l'application navigue vers une page de détail listant les éléments. Un clic sur l'un de ceux-ci navigue vers un niveau de détail encore plus fin, un seul item occupant tout l'espace. Des flèches de navigation sont affichées permettant à l'utilisateur de revenir en arrière facilement. On remarquera, non sans une certaine fascination, que la touche « arrière », absolument indispensable, ne fait partie d'aucune norme ni visuelle, ni dans l'OS lui-même et ni dans les « gestures », son apparition est laissée à la discrétion du développeur ce qui est un choix pour le moins dérangeant en terme d'UX globale. A la fois très rigide et normalisateur, Modern UI laisse perplexe sur des choix (ou non choix) de ce type qui sont pourtant essentiels dans l'interaction avec l'utilisateur. N'oubliez jamais de prévoir une place dans le visuel (et son code) pour la touche de retour arrière dans vos applications, WinRT ne gérant pas cela…

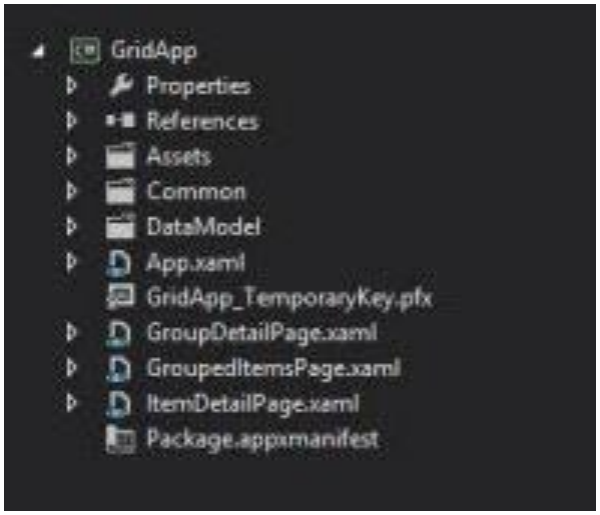
Le modèle "Split App"

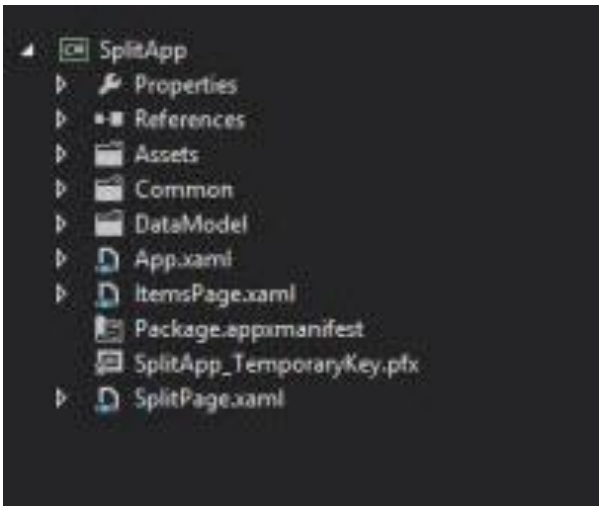
Ce template créée un projet de deux pages qui permet lui aussi de naviguer dans un ensemble d'éléments. La première page permet de sélectionner un groupe d'éléments, la seconde affichant le détail du groupe sélectionné.

La structure des templates

Chaque template créée un projet possédant une structure légèrement différente même si l'essentiel reste semblable :







Chaque projet possède deux répertoires : `Assets` et `Common`. Le second et le troisième type offrant un répertoire de plus "`DataModel`". "`Assets`" est le répertoire de stockage des ressources de l'application, images, icones, etc, "`Common`" est conçu pour recevoir les fichiers communs utilisés par toute l'application. Le répertoire "`DataModel`" doit être vu comme une bibliothèque contenant les Modèles et les données de test. Les ViewModels peuvent y être placés mais il est préférable de leur réserver un emplacement séparé.

Tous ces projets sont dotés d'un `App.xaml` comme toutes les applications C# / Xaml et jouant le même rôle (avec les spécificités de WinRT comme le *tombstoning*). De même on trouve un fichier `Package.appxmanifest` qui stocke le manifeste de l'application (ses icones, les autorisations qu'elle réclame, les déclarations qu'elle fait, comme s'intégrer dans le charme de recherche).

Enfin, le fichier `{nom appli}_TemporaryKey.pfx` stocke la clé de signature.

Les autres fichiers sont des fichiers Xaml dont le nom et la fonction dépendent du type de projet.

Faut-il utiliser les templates ?

Voici une question plus intéressante… Faut-il utiliser ces templates pour créer de nouvelles applications ? Ce qui ouvre la voie à un autre questionnement "Ces templates sont-ils bien conçus ?".

A la dernière question je peux répondre immédiatement "oui". Oui, les templates sont bien conçus. Mais uniquement si on les regarde comme des exemples de mise en page et de navigation. Cette nuance permet de répondre à la première question : Non, je ne crois pas qu'il faille utiliser ces modèles comme des "starter kits".

En dehors du modèle "`Blank`", il y a fort peu de chance qu'une application un peu sérieuse puisse se satisfaire des modèles proposés. Il faudra par exemple ajouter un framework MVVM qui imposera sa structure propre le plus souvent.

Bref, les templates fournis sont intéressants à faire tourner et à étudier, ils peuvent faire de très bons "début de commencement" pour des applications très simples répondant strictement à la mise en page proposée. On retrouve ici le côté fermé et normalisateur de Modern UI dont la première cible reste le grand public avec une volonté affichée par Microsoft d'uniformisation à des parsecs du discours sophistiqué sur le Design qui a pu être utilisé autour de Silverlight ou WPF. Il ne faut pas se laisser piéger par ses entonoirs que Microsoft a placés pour enfermer la démarche de création. Une application qui ressemble aux autres ne se vend pas, assumez sans honte de décrocher des guidelines du Design Modern UI lorsqu'elles vous paraissent contraire à vos intérêts !

Dans tous les autres cas vous aurez intérêt à partir des templates fournis avec le framework MVVM que vous choisirez (MVVM Light, Caliburn.Micro, Okra, MvvmCross, …).

L'intérêt des modèles se trouve dans ce que tout logiciel doit viser : "la satisfaction immédiate" c'est à dire que Visual Studio, Expression Blend, se doivent comme tout logiciel de permettre en quelque clics de "faire quelque chose" de gratifiant. Les modèles jouent ce rôle. Au-delà il vous faudra penser une organisation adaptée à chaque projet.

Les autres templates

A côté des trois templates décrits ci-dessus, chaque catégorie offre d'autre templates spécialisés.

Dans les catégories Visual C# et Visual Basic il existe par exemple trois templates de plus "`Class Library`", "`Windows Runtime Component`" et "`Unit Test Library`".

Le modèle "`'Class Library`" permet de créer une bibliothèque de classes managée pour les Windows Store Apps. Le modèle "`'Windows Runtime Component`" est plus générique car il permet de créer des composants utilisables par tous les langages WinRT. Vous pouvez ainsi créer en C# un composant qui sera réutilisable dans une application écrite en JavaScript/Html par exemple. Le troisième modèle permet de créer un projet de test unitaire.

Les projets C++

Hélas pour ceux qui avaient vu en C# un fantastique progrès offrant un moyen moderne de faire des applications Windows en échappant à cette horreur d'un autre âge qu'est C++, ce langage antédiluvien étant le seul que connaisse Sinosfky il redevient plus que jamais incontournable dès qu'il s'agit d'attaquer Windows à la racine… Depuis Sinofsky a été viré, mais bien entendu WinRT n'a pas été réécrit pour autant… Attentifs nous devons rester et du côté obscur toujours se méfier il faut !

Bref, pour l'instant, si vous voulez faire du Direct 2D, du Direct 3D, écrire une DLL, une bibliothèque "statique" pour Windows Store Apps, il n'y a que C++ qui le permet. D'un autre côté, en dehors des développeurs de jeux ou de drivers de périphériques C++ n'apporte pas grand chose et peu continuer à être ignoré.

Les projets JavaScript

Même si je ne conseille à personne d'écrire de tels projets, ils existent. Les templates diffèrent un peu et on ne peut, en plus des trois templates commun présentés plus haut, rien espérer de plus qu'un modèle "`Fixed layout`" et un autre appelé "`Navigation`" pour produire des applications Windows Store forcément pauvres et mal testées (pas de Unit testing, pas de librairie de composant, etc).

Ces applications JavaScript on le sait sont là uniquement pour attirer des développeurs Web vers Windows 8. Microsoft, malgré un discours de type "ce n'est pas grave d'avoir peu d'applications sur le Store" rame intensément pour tenter de remplir le fameux Store qui, forcément, fait triste mine en nombre d'applis par rapport à ses concurrents.

Ces deux problèmes n'en sont pas.

Celui de la quantité est un faux problème. La qualité prime sur la quantité, et il suffit qu'il existe quelques bonnes applications sur le Windows Store pour concurrencer les 500 versions de la lampe de poche sous Android et IOS… C'est vrai. Utilisateur d'Android (et programmant aussi pour cet OS) je le sais bien, le Play Store est un fatras d'applis qui se copient les unes les autres dont la qualité est généralement très moyenne (même s'il y a de très bonnes applis aussi). Windows Store peut relever le défi avec de bonnes applis, c'est une carte à jouer. Et puis selon une formule popularisée par Mitterrand, il faut donner du temps au temps… Laissons au Windows Store le temps de se remplir.

Celui des développeurs Web n'est pas un gros souci non plus. Je leur dis même "bienvenue !". Dot.Blog leur est grand ouvert et je serai heureux de leur faire découvrir que malgré tout, pour développer sous Windows 8, C# / Xaml, il n'y a rien de mieux !

Conclusion

Il est important de comprendre la raison d'être des templates (le principe de "gratification instantanée" que tout logiciel doit offrir pour séduire rapidement, VS ou Blend n'échappant pas à cet impératif vendeur).

Il est tout aussi important de savoir ce qu'on peut en tirer (apprentissage, base pour faire des tests, etc).

De même il faut savoir qu'elles en sont les limitations (leur simplicité et leur généricité qui ne sont pas appropriées à des applications MVVM utilisant un framework par exemple).

Il faut aussi savoir ce qu'il est possible de faire en C# / Xaml et ce qui reste interdit à ce couple, notamment les applications 2D et 3D.

Il ne faut pas négliger la catégorie JavaScript car elle permettra (peut-être) d'attirer des développeurs Web de talent qui enrichiront par leur présence la qualité des applications du Windows Store autant que nous essayerons d'enrichir leur propre horizon technique en leur apprenant à comprendre C# et Xaml ! Pour ça, vous pouvez compter sur Dot.Blog pour leur faire oublier JavaScript :-)

De Silverlight/WPF à WinRT

De Silverlight à WinRT (partie 1)

Quand on connait déjà Silverlight, la question qui se pose tout de suite est de savoir comment réutiliser au mieux son savoir-faire sous WinRT, ce qui implique de comprendre et connaître les différences principales entre les deux plateformes. Dans cette première partie j'aborderai les déclarations des espaces de noms, car sans eux, rien ne tourne.

WinRT englobe la quasi-totalité de Silverlight

C'est le premier point à savoir, et il est plutôt encourageant, WinRT englobe la quasi-totalité de Silverlight. Comme je vous en ai parlé dans mon billet "De Silverlight à WinRT : Mesurez les différences grâce au "WinRT Genome Project" sur les 2189 classes de Silverlight (de base) on en retrouve 1582 dans WinRT, avec une différence de 607, absentes, qui pour beaucoup concernent des choses inutiles sous WinRT comme l'Out Of Browser par exemple.

Je renvoie le lecteur à ce billet pour plus de précisions et notamment pour l'adresse d'un site qui compare les équivalences Silverlight / WinRT au niveau des types et des membres communs ou non. Cela peut être d'une très grande aide lors d'un portage de code, ou simplement pour trouver une équivalence quand on développe.

La déclaration "classique" d'un espace de noms

Silverlight et WPF partagent de ce point de vue une syntaxe similaire.

Pour déclarer un espace de noms "`MaLib.dll`" contenant l'espace de noms `MaLib.MesOutils` on écrit, en Xaml :

```
xmlns:local="clr-namespace:MaLib.MesOutils;assembly:MaLib"
```

On déclare de façon identique, mais simplifiée, l'accès à des espaces de noms contenus dans l'assemblage courant (celui qui contient le code Xaml dans lequel on veut ajouter la déclaration) :

```
xmlns:local="clr-namespace:MaLib.MesOutils"
```

On remarque que seule disparait la référence "assembly" qui pointe l'assemblage, la DLL ou l'exe en cours étant pris par défaut.

La déclaration sous WinRT

Même s'il semble que l'ancienne syntaxe puisse passer dans certains cas, WinRT impose une nouvelle syntaxe.

Les grandes différences sont :

- L'utilisation de "using" à la place de "clr-namespace"
- L'absence systématique du nom de l'assemblage

De fait, la déclaration précédente devient :

```
xmlns:local="using:MaLib.MesOutils"
```

Utiliser le même code Xaml ?

La proximité entre Silverlight et WinRT est telle qu'on est tenté de se demander s'il est possible d'utiliser le même code pour les deux plateformes...

C'est légitime mais hélas les constructions #IF n'existent pas en Xaml...

Les déclarations de namespaces étant un peu différentes il semble qu'il faudra créer des fichiers Xaml différents. De toute façon d'autres éléments de mise en page ou de navigation différent à un tel point qu'il sera préférable de recorder les UI sous WinRT. Cela n'interdit pas de réutiliser des DataTemplate, des dictionnaires de styles Xaml ou des constructions partielles, ce qui peut déjà faire gagner beaucoup de temps.

Les déclarations dans le code

Les espaces de noms qui concernent l'UI ont été changés sous WinRT. Globalement, "System.Windows" a été remplacé par "Windows.UI.Xaml". D'autres changements de ce type sont intervenus. Mais côté code nous disposons heureusement des constructions de type #IF et d'un marqueur spécifique à WinRT "NETFX_CORE". Il devient donc très facile de construire un code C# pouvant s'adapter aisément aux deux plateformes, au moins en ce qui concerne les espaces de noms.

Un groupe Using fonctionnant aussi bien en Silverlight qu'en WinRT ressemblera ainsi à cela :

```
 1: using System;
 2: using System.Collections.Generic;
 3: using System.Linq;
 4: using System.Text;
 5: using System.Threading.Tasks;
 6: #if !NETFX_CORE
 7: using System.Windows;
 8: using System.Windows.Controls;
 9: using System.Windows.Data;
10: using System.Windows.Documents;
11: using System.Windows.Input;
12: using System.Windows.Media;
13: using System.Windows.Media.Imaging;
14: using System.Windows.Navigation;
15: using System.Windows.Shapes;
16: #else
17: using Windows.UI.Xaml;
18: using Windows.UI.Xaml.Controls;
19: using Windows.UI.Xaml.Data;
20: using Windows.UI.Xaml.Documents;
21: using Windows.UI.Xaml.Input;
22: using Windows.UI.Xaml.Media;
23: using Windows.UI.Xaml.Media.Imaging;
24: using Windows.UI.Xaml.Navigation;
25: using Windows.UI.Xaml.Shapes;
26: #endif
```

Encore une fois il s'agit d'illustrer le propos, une approche « cross-plateforme » complète est bien préférable à de tels bricolages. Je vous laisse découvrir cet aspect des choses dans le Tome déjà publié de la série ALL DOT BLOT consacré à ce sujet bien particulier.

On pourra aussi s'inspirer d'une documentation Microsoft qui traite de ces différences "Migrating a Windows Phone 7 app to a Metro Style App using XAML".

De Silverlight à WinRT (partie 2)

Parmi les petites différences entre Silverlight et WinRT, en voici une qui concerne l'accès au thread de l'UI, ce qui est essentiel.

Le Thread de l'UI

C'est "ze" thread d'une application Silverlight, WPF et même WinRT. C'est ce thread qui est le seul à pouvoir manipuler les objets d'interface, c'est lui qui, s'il est bloqué par un traitement long, fige l'application, situation aujourd'hui interdite par toutes les guidelines.

Sous ces environnements on travaille donc systématiquement sur des threads secondaires pour décharger le thread de l'UI, ce que C#5 avec sa gestion `async`/`await` simplifie grandement, l'asynchronisme devenant sous WinRT la base même des mécanismes de programmation même les plus simples.

Décharger le thread de l'UI n'est pas compliqué en soit. Ce qui l'est c'est de mettre à jour l'interface utilisateur depuis un thread secondaire.

Dans une construction MVVM utilisant le Binding, on n'a généralement pas à s'en soucier car les mécanismes de Binding prennent en charge ce problème. Le ViewModel peut changer la valeur d'une propriété, même depuis un thread secondaire, la propagation vers l'interface se passe correctement.

Il n'en va pas de même dès qu'on manipule directement un objet d'interface.

Pour ce faire le thread secondaire doit demander d'exécuter l'action au thread principal (celui de l'UI). Ce qui oblige à passer par le `Dispatcher`. Certains toolkit MVVM (Mvvm Light, Jounce..) offrent leur propre dispatcher, mais les plateformes SL et WinRT ont aussi le leur.

Le nouveau Dispatcher

Sous WinRT le nouveau Dispatcher ressemble fortement à l'ancien. Mais les paramètres sont légèrement différents. On dispose notamment d'un paramètre permettant d'indiquer le niveau "d'urgence" de l'exécution de la tâche dispatchée. La classe mère du Dispatcher semble avoir aussi changé.

Un code portable

Pour rendre un code portable Silverlight / WinRT il est donc nécessaire d'utiliser une construction `#IF`. Ce qui, en première approximation donnera :

```
 1: #if !NETFX_CORE
 2:   Dispatcher.BeginInvoke(
 3: #else
 4:   Dispatcher.Invoke(CoreDispatcherPriority.Normal,
 5: #endif
 6:             (sender, args) => { ... }
 7: #if NETFX_CORE
 8:       , this, null);
 9: #endif
10:    );
```

Un tel saussissonage est absolument affreux et n'est donné que pour la démonstration. Je conseille aujourd'hui d'utiliser un stratégie de développement cross-plateforme plus simple et plus saine à mettre en œuvre (voir les autres Tomes de ALL DOT BLOT).

Dans la réalité on aura plutôt intérêt à écrire une méthode ou une classe d'aide qui se chargera de faire la différence une bonne fois pour toute et à l'utiliser partout dans son code au lieu, bien entendu, de farcir ce dernier de `#IF` dans tous les coins...

Conclusion

Rien de bien compliqué ici encore. La ressemblance du code Xaml et C# de Silverlight avec celui de WinRT est telle qu'on peut assez facilement envisager un code portable sur les deux plateformes même si la portabilité semble plus pertinente aujourd'hui à mettre en œuvre entre WinRT et WPF et Android, ce qui réclame d'autres techniques que je décris longuement dans un Tome consacré au cross-plateforme (sans oublier les 12 vidéos de 8h sur le sujet).

Il reste bien entendu milles autres détails qui rendent un tel portage pas aussi simple que cela. Par exemple l'asynchronisme systématique sous WinRT qui peut introduire de grosses différences dans le code.

De plus, s'il semble illusoire de vouloir créer très facilement un code complet et complexe pouvant passer sans souci entre SL et WinRT, nous avons déjà de bonnes assurances qu'un certain type de code pourra passer sans gros problème, comme par exemple des classes définissant des données ou des traitements simples. C'est déjà toute la structure de base d'une application qui est portable, et c'est déjà beaucoup.

Aujourd'hui centraliser ce code commun dans des PCL portables semble être une option bien plus « propre » que de vouloir à tout prix utiliser un code truffé de code condtionnel.

Plus loin, je ne suis pas convaincu que dans le futur nous serons amenés à écrire beaucoup d'applications se devant d'être compatibles entre Silverlight et WinRT. Je pense plutôt que le problème se posera entre des applications WPF et WinRT. Plus d'un an après avoir écrit ces lignes je maintiens cet avis que j'élargirais à la compatibilité WinRT / Windows Phone / Android.

Sur la base d'une même application on pourra vouloir une version WinRT pour attaquer le marché RT tout en continuant à produire une version WPF échappant au

Windows Store et ses taxes, ou tout simplement par souci de compatibilité pour continuer à vendre une application à tous les possesseurs d'un Windows plus ancien (XP, Vista, Seven) qui représentent en novembre 2013 75% du parc Windows malgré tout.

De Silverlight/WPF à WinRT : .NET pour Metro Style (partie 3)

Je vous ai déjà présenté différents éléments de WinRT, notamment ses différences avec les frameworks Silverlight et WPF. Soyons plus précis. Avec C#/Xaml il est possible de créer des applications Metro Style, ces applications fonctionnent sous WinRT, mais utilisent avant tout une version spéciale de .NET, un peu comme Silverlight. Et il y a beaucoup à dire sur les différences entre SL et .NET pour Metro Style (Modern UI).

.NET pour Modern UI

Avant tout, utiliser C# avec Xaml pour créer des applications Windows 8 c'est utiliser une version spéciale et allégée de .NET, exactement comme le runtime Silverlight est lui aussi une version allégée du framework .NET complet. Seul WPF permet en réalité d'utiliser le framework complet (si on reste dans les applications Xaml).

Dans les comparaisons que j'ai déjà publiées, on remarque que le framework .NET allégé de Silverlight est presque totalement contenu dans celui disponible sous WinRT.

Mais de nombreuses et parfois subtiles différences existent.

Essayons de faire le tour des principales.

.NET APIs pour Modern UI

C'est le nom complet de ce framework .NET spécialement conçu pour concevoir des applications Metro Style en C# ou en VB.

Allégé ?

Oui pour au moins une bonne raison : tout ce qui ne permet pas directement de concevoir des applications Metro Style a été supprimé.

La règle est simple, prendre les décisions classe par classe a forcément du être moins simple. C'est pourquoi certains de ces choix nous apparaitrons évidents et logiques et d'autres plus difficiles à comprendre...

.NET APIs + WinRT

Lorsqu'on travaille en C#/Xaml on utilise ainsi les APIs **.NET pour Modern UI**, APIs simplifiées et écrémées de tout ce qui ne sert pas à faire du Modern UI. Mais tout cela repose malgré tout sur **Windows Runtime** (**WinRT**), et les types de WinRT sont aussi utilisables, en plus de ceux du framework .NET spécifique Modern UI...

Cela est très important à comprendre. Les APIs .NET pour Modern UI tournent au-dessus de WinRT qui lui-même se présente d'une façon semblable à .NET même s'il a été conçu différemment et toutes les APIs WinRT sont utilisables depuis C# comme on le faisait avec les APIs .NET. Mais cela est juste une impression, WinRT n'est pas un .NET modifié, c'est autre chose, conçu autrement, mais se présentant sous une forme semblable.

Ainsi, le développeur utilisant C#/Xaml pour développer sous Windows 8 Modern UI (le bureau classique à part donc) dispose en réalité d'un ensemble d'APIs séparées en deux groupes principaux : d'une part le profil du framework .NET spécifique Modern UI, et d'autre part toutes les APIs WinRT. C'est donc **un ensemble très vaste** offrant l'accès à toutes les APIs utiles et nécessaires qui s'offrent au développeur C#/Xaml.

Ce n'est pas rien !

Un .NET allégé

Comme expliqué plus haut ce profil du framework .NET spécial pour Modern UI a été conçu avec un objectif clair : se concentrer sur tout ce qui est utile pour développer des applications Modern UI et rien que cela. Il en découle que certains types sont logiquement absent de ce framework:

- Les types et les membres qui ne peuvent pas servir à créer des applications WinRT comme la Console ou les types ASP.NET.
- Les types obsolètes conservés uniquement pour la compatibilité avec les versions antérieures de .NET.
- Les types qui doublent des types existant dans les APIs WinRT.
- Les types et les membres qui encapsulent les fonctions de l'OS comme `System.Diagnostic.EventLog` ou les compteurs de performance.
- Les membres considérés comme introduisant une certaine confusion (comme les méthodes "`Close`" sur les types gérant des entrées/sorties).

Comme je le disais un tel découpage à la hache, même fait avec grande intelligence, possède sa part d'ombre. Il sera certainement aisé de comprendre certains choix, et

plus difficile d'en accepter d'autres. Mais tout comme le framework limité de Silverlight, il faudra faire avec.

Toutefois, si le développeur Silverlight se trouvait devant un mur face à certaines omissions (comme les méthodes de coercition de données sur les propriétés de dépendance), le développeur WinRT aura souvent la possibilité de se "rabattre" sur les types de l'API WinRT qui est pléthorique (plusieurs milliers de types) alors que sous Silverlight une telle échappatoire n'existe pas...

Toute l'astuce de l'apprentissage consistera donc à bien connaître les limites du framework .NET pour Modern UI tout autant que les APIs "natives" de Windows 8 (WinRT).

Des substituions à connaître

Si j'ai déjà évoqué les changements de namespaces dans de précédents billets, j'évoque ici des substitutions plus subtiles, du type de celles qui réclament justement de bien connaître les APIs WinRT.

Par exemple le développeur sera très étonné de ne plus trouver dans le framework .NET les APIs concernant l'*Isolated storage*.

Comment s'en passer ? Pourquoi ont-ils supprimé ces types essentiels ?

On prend du recul, et on relit les règles... sont omis notamment "*les types qui doublent des types existant dans les APIs WinRT*"...

Bien entendu WinRT fourni un service totalement équivalent à tous les langages de la plateforme. Il aurait été idiot de laisser des choses spécifiques dans le .NET pour Modern UI alors même que WinRT fournit les mêmes services de façon globale à toute la plateforme;

C'est ainsi que `System.IO.IsolatedStorage.IsolatedStorageSettings` est par exemple omis de .NET pour Modern UI car on peut accéder tout aussi facilement à `Windows.Storage.ApplicationDataContainer` qui offre des services équivalents.

Une simplification de l'accès aux APIs

Lorsqu'on créée une application Metro Style avec C#, la totalité des APIs est automatiquement référencée dans le projet. On peut dès lors utiliser directement n'importe quel type supporté par .NET Modern UI sans actions supplémentaires.

Créer des librairies Metro Style portables

Vous pouvez également créer un projet de classes portables (PCL Portable Class Library) pour développer une bibliothèque .NET Framework qui pourra être utilisée à partir d'une application WinRT (peu importe son langage) ou même d'applications WPF en bureau classique (et mieux, des applications iOS ou Android en passant par Xamarin) ! Le projet doit inclure .NET pour Modern UI comme l'un des plates-formes cible (si on souhaite un partage avec cette plateforme bien entendu). La bibliothèque de classes portables est particulièrement utile lorsque vous voulez développer des classes qui peuvent être utilisés à partir d'applications pour différents types de plates-formes, comme une application Windows Phone, une application de bureau, et une application WinRT. Dans le cadre d'une stratégie cross-plateforme unifiée on utilise systématiquement les PCL pour coder le « noyau » contenant toute la logique métier. Et la PCL ainsi créée permet d'utiliser le même code sous toutes les plateformes Microsoft ainsi que sous iOS et Android (en ajoutant le plugin Xamarin).

Ces librairies "portables" sont une nouveauté et peuvent s'avérer essentielles pour qui désire développer un logiciel tournant sous WinRT et WPF ou Windows Phone 8 et WPF, etc...

La conversion du code existant

Partir de zéro sur une nouvelle plateforme est certainement le meilleur moyen de "faire propre", mais puisqu'ici nous sommes sur des plateformes "cousines" étudiées pour être compatibles entre elles (jusqu'à un certain point), il est tout à fait envisageable de vouloir porter un code existant, WPF ou Silverlight, vers WinRT.

Je parle de "cousinage" car comme un cousin germain, s'il existe une "proximité génétique" indéniable, les différences sont suffisamment importantes pour être visibles... Ainsi lors d'un portage vers WinRT il faudra veiller à de nombreuses différences qui se cachent derrière la masse des similitudes, car il y a des changements dans :

- La gestion de l'interface utilisateur;
- La gestion des Entrées/Sorties;
- La gestion du réseau;
- La manière dont le multitâche fonctionne;
- La gestion des mécanismes de réflexion;
- La gestion de la sécurité;

- La gestion des ressources;
- La façon dont WCF fonctionne et réagit;
- A l'intérieur même de certains types .NET

Les changements dans la gestion de l'interface utilisateur

La conversion d'un code d'interface depuis Silverlight n'est pas si difficile du point de vue UI car les bases restent identiques. J'ai pu dans mes expériences transférer des `UserControls` avec leurs styles, des templates, etc, sans rencontrés d'énormes problèmes et même avec parfois une facilité qui m'a étonnée.

La différence première est le stockage des types habituels qui passe de `System.Windows` à l'espace de noms `Windows.UI.Xaml`.

Les types sont similaires mais il y a parfois des différences dans les membres. Ce genre de "surprise" se découvre dans l'action et il serait fastidieux d'en dresser la liste complète...

Le premier point à se rappeler est donc de changer toutes les références à `System.Windows.*` par `Windows.UI.Xaml.*`. Ainsi la classe `Border` se trouve désormais dans `Windows.UI.Xaml.Controls` au lieu de `System.Windows.Controls`.

Pas trop compliqué surtout si, comme la plupart du temps, les espaces de noms sont indiqués une fois pour toute en entête de code par le biais d'une instruction "`using`". Seul cet endroit du code devra donc être modifié.

Les changements dans la gestion des Entrées / Sorties

Ici les changements sont radicaux car tout dans WinRT est asynchrone. C# 5 introduit des nouveaux mots clé "`async`" et "`await`" dont je reparlerai bien évidemment et qui simplifient grandement la prise en charge de l'asynchronisme. Il n'en reste pas moins vrai que toutes les opérations d'E/S sont devenus asynchrones et réclameront d'être "revisitées" parfois en profondeur.

Par exemple la lecture des flux via les méthodes `System.IO.Stream.BeginRead/EndRead` est remplacée par une unique méthode `System.IO.Stream.ReadAsync`, totalement asynchrone.

Dans le même esprit les écritures sur un flux (BeginWrite/EndWrite) sont remplacées par un unique `WriteAsync`.

La méthode "`Close()`" a souvent fait couler beaucoup d'encre. Un fichier par exemple une fois sa méthode `Close()` appelée, est-il "disposé" ou non ? Peut-on directement

appeler `Dispose()` ? Etc. L'enfer est pavé de bonnes intentions... C'est en voulant simplifier les choses du point de vue sémantique que `Close()` a été ajouté (en pur équivalent à `Dispose()`). Mais c'était une fausse bonne idée. Sous WinRT les méthodes `Close()` n'existent plus, reste toujours `Dispose()`. On peut appeler `Dispose()` directement, dans un bloc `try/finally`, ou mieux dans un bloc "`using`".

Par exemple la lecture d'un flux sous WinRT, en prenant en compte l'asynchronisme, s'écrira de la façon suivante :

```
using (StreamReader sr =
  new StreamReader(await passedFile.OpenStreamForReadAsync()))
{
    while ((nextLine = await sr.ReadLineAsync()) != null)
    {
        contents.Append(nextLine);
    }
}
```

On voit clairement l'utilisation d'un bloc '`using`' tel qu'il était de toute façon conseillé d'en utiliser partout où l'on travaillait sur des objets offrant une méthode `Dispose()` ayant un rôle réel (libérer des ressources non managées occupées par l'instance).

On remarque l'utilisation de '`await`' qui allège considérablement l'asynchronisme.

Autre problème récurent, la lecture d'un fichier texte. Lorsqu'il n'est pas trop long on utilise souvent `System.IO.File.ReadAllText`, sous WinRT on utilisera `Windows.Storage.PathIO.ReadTextAsync`.

L'exemple suivant lit deux fichiers texte, l'un se trouvant dans le package du logiciel, l'autre se trouvant dans le répertoire local de l'application :

```
public static async void ReadFileSamples()
{
  // Read a file from package
  StorageFolder packageFolder =
      ApplicationModel.Package.Current.InstalledLocation;
  StorageFile packagedFile =
      await packageFolder.GetFileAsync("FileInPackage");

  // Read a file from AppData
  StorageFolder localFolder = ApplicationData.Current.LocalFolder;
  StorageFile localFile =
    await localFolder.GetFileAsync("FileInAppData");
}
```

On remarque une fois encore l'utilisation de '`await`' et des méthodes dont le nom se termine par '`async`'. Absolument toutes les I/O sont asynchrones sous WinRT.

L'isolated Storage

Puisque nous parlons des E/S et de portage depuis Silverlight, il est important de noter la différence dans la gestion de l'Isolated Storage.

(Ce problème ne se pose pas pour un portage depuis WPF puisque ce dernier n'a pas d'équivalent à l'Isolated Storage, mais en revanche si on veut écrire un code portable SL / WinRT / WPF il faudrait forcément faire attention à "simuler" ce fonctionnement pour WPF).

Sous .NET pour Silverlight on utilise la classe `System.IO.IsolatedStorageFile`, sous WinRT on utilisera la propriété `LocalFolder de Windows.Storage.ApplicationData`.

De même `System.IO.IsolatedStorage.IsolatedStorageSettings` sera remplacé par la propriété `LocalSettings de Windows.Storage.ApplicationData`.

On noter au passage que les espaces de noms de WinRT suivent une logique différente de .NET mais qu'on gagne un peu, en tout cas à mon avis, en intelligibilité. Tout ce qui concerne les E/S se trouve dans Windows.Storage par exemple. C'est clair. Les données de l'application sont dans `Windows.Storage.ApplicationData`, c'est ultra clair. Forcément, avec le recul, Microsoft a pu retravailler certains aspects comme le nommage des espaces de noms et nous gagnons en cohérence au passage.

Les changements dans la gestion du réseau

Là aussi les changements sont radicaux et demanderont de revisiter le code. D'abord en raison de la nature asynchrone des nouvelles APIs et parce que ces dernières ont une philosophie un peu différente.

AInsi, les échanges en lecture et écriture via `Http` utilisent sous Silverlight la classe `System.Net.WebClient` alors que sous WinRT il s'agira de `System.Net.Http.HttpClient`. Un nom plus parlant quant aux techniques utilisées.

WinRT propose d'ailleurs une autre API destinée aux uploads et downloads de grandes quantités de données : `Windows.Networking.BackgroundTransfer`. Ici aussi les noms clarifient l'intention. On comprend immédiatement qu'il s'agira de transférer des données dans un thread d'arrière-plan. Ce qui, bien entendu, est mieux adapté lorsqu'il s'agit de communiquer des données de taille importante.

Autre changement, tous les types qui se trouvent dans `System.Net.Sockets` se retrouvent dans `Windows.Networking.Sockets`.

Parmi les différences qui peuvent avoir un impact non négligeable on remarquera aussi que les URI relatives ne peuvent pas être passées au classes de WinRT qui réclament des adresses absolues. Cela peut être délicat pour certains codes.

Un dernier aspect concerne la gestion des exceptions. `UriFormatException` doit être changée dans les '`catch`' par `FormatException` qui est la classe parente de `UriFormatException`. Ne cibler que cette dernière semble laisser des "trous". En utilisant la classe parente on est certain d'attraper toutes les exceptions que Silverlight gère (plus celles propres à WinRT certainement dans ce cas).

Les changements dans la gestion du multitâche

Certaines classes de gestion du multitâche du framework .NET connaissent des changements dans leurs membres, ce qui peut s'avérer délicat à gérer. Certaines ont totalement disparu... mais heureusement, c'est qu'ici aussi on retrouve un équivalent géré par WinRT qu'il est possible d'utiliser. Mais cela modifie le code ce qui a un poids non négligeable dans le cas d'un portage.

Toutefois il faut relativiser certains changements qui concernent des propriétés peu utilisées comme `MemoryBarrier` de la classe `Thread` ou `ManagedTheadId`.

Plus déroutant peut être l'absence de `Thread.CurrentCulture` ou `CurrentUICulture` qu'on retrouve dans `CultureInfo` de l'espace de noms `System.Globalization`. Mais je trouve qu'ici aussi on gagne en cohérence. Que des aspects liés à la localisation soient "cachés" dans une classe telle que Thread relève plus de l'astuce que de la bonne écriture d'un framework... Alors que retrouver ces informations dans `System.Globalization` est d'une logique imparable. Mais chacun appréciera !

La disparition de `System.Threading.Timer` ne doit pas vous affoler non plus. On retrouve un équivalent en la classe `ThreadPoolTimer` de `Windows.System.Threading`. De même que la classe `System.Threading.ThreadPool` se trouve déplacée dans `Windows.System.Threading`.

Pour placer un code dans le pool on utilisera un code structuré de cette façon :

```
Task.Run(() =>
{
  // job à faire ici
});
```

Un code qui place un job dans le pool et qui souhaite attendre sa fin :

```
await Task.Run(() =>
{
  // job à exécuter et à attendre ici
});
```

On remarque l'étonnante simplicité qu'introduisent 'async' et 'await'.

Pour un code qui créé un job pouvant être long on utilisera la construction suivante :

```
Task.Factory.StartNew(() =>
{
  // long job ici
}, TaskCreationOptions.LongRunning);
```

Les changements dans la gestion de la réflexion

Les changements ne sont pas immenses mais la plupart des membres de `System.Type` ont été déplacés dans `System.Reflection.TypeInfo` ce qui malgré tout impact de façon non négligeable un code à porter.

`Type.Assembly` se retrouve par exemple dans `type.GetTypeInfo().Assembly`, ce n'est pas compliqué mais encore faut-il retrouver cette équivalence !

De même `type.GetMethod("maMéthode", BindingFlags.DeclaredOnly)` se retrouve par `type.GetTypeInfo().GetDeclaredMethod("MaMéthode")`. Dans la même logique `type.GetNestedTypes()` s'obtient par `type.GetTypeInfo().DeclaredNestedTypes`.

Si tous ces changements obligent à revoir le code existant, on notera qu'un simple "chercher/remplacer" pourra faire le travail dans la majorité des cas et que dans les autres il s'agit de méthodes ou de membres peu utilisés (en tout cas souvent utilisés dans des parties de code très ciblées ce qui en facilite le changement).

Les changements dans la gestion de la sécurité

Ici presque tout a été supprimé car WinRT fournit bien entendu à l'ensemble de la plateforme et de façon homogène tout ce qui est nécessaire à la gestion de la sécurité, aux authentifications ou à la cryptographie.

Il sera donc nécessaire d'étudier de près les espaces de noms suivants pour retrouver les équivalents :

- System.Security
- System.Security.Principal
- Windows.Security.Authentication.OnlineId
- Windows.Security.Authentication.Web
- Windows.Security.Credentials

- Windows.Security.Credentials.UI
- Windows.Security.Cryptography
- Windows.Security.Cryptography.Certificates
- Windows.Security.Cryptography.Core
- Windows.Security.Cryptography.DataProtection
- Windows.Security.ExchangeActiveSyncProvisioning

Les changements dans la gestion des ressources

Dans les applications Modern UI on créé un fichier unique de ressources ce qui est une philosophie différente des applications desktop généralement. Je reviendrai sur tous ces aspects dans de prochains billets bien entendu, il ne s'agit, pour l'instant, que de débroussailler le chemin...

On notera que les classes de `WIndows.ApplicationModel.Resources` et `Windows.ApplicationModel.Resources.Core` sont à utiliser à la place de celles de `System.Resources`.

Les changements dans la gestion des exceptions

Dans certains cas un type managé peut lever une exception qui n'est pas incluse dans le framework .NET pour Modern UI, ce qui créé des situations assez dangereuses si des '`catch`' ont été placés sur ces exceptions particulières.

L'astuce, si on peut parler d'astuce, consiste à 'catcher' la classe parente de l'exception en question... c'est le conseil de Microsoft, je trouve cela un peu bricolage et dangereux, mais il faudra faire avec...

Un exemple a été évoqué plus haut à propos de `UriFormatException`. S'il faut utiliser la classe parente `FormatException` c'est tout simplement que `UriFormatException` est levée par une partie du code managée mais qu'elle n'existe pas dans le framework .NET épuré pour WinRT. Situation curieuse, ambigüe et pour parler franchement très "casse gueule". Méfiez-vous et vérifiez bien vos '`catch`' !

Les changements dans WCF

En fait les changements se résument à un seul changement, mais de taille : dans une application WinRT on peut utiliser tous les services de WCF pour obtenir des données mais il est impossible de créer un service WCF pour servir des données... Radical !

C'est ce genre de choses qui expliquent, imposent même, la création du premier OS à deux têtes au monde... Et c'est pourquoi Windows 8 Modern UI est fait pour cohabiter longtemps avec le bureau classique... WinRT ne permet pas d'écrire de

nombreux types d'applications, le bureau classique reste indispensable dans certains cas. WinRT a été conçu pour des applications grand public puisant leurs données dans le Cloud, absolument pas dans l'optique de créer une plateforme du futur remplaçant l'ancienne et encore moins en pensant aux entreprises... Et c'est pour moi tout le problème de Windows 8 : des idées géniales, de vraies innovations, mais poussées tellement à l'extrême qu'on se retrouve obligé d'avoir un OS bicéphale ce qui est une perte de cohérence énorme, voire une erreur de design impardonnable.

Les changements dans les types .NET

Ils sont assez nombreux malgré tout et éparpillé un peu partout. Ils seront souvent simples à repérer puisque l'avantage d'un langage comme C# sur JavaScript notamment (vous voyez à quel buzz je pense...) permet de s'assurer à la compilation que tout est ok. On imagine quel serait l'enfer de tels changements dans le futur pour ceux qui auraient choisi un langage non fortement typé et non compilé...

Visual Studio saura vous avertir avant même la compilation que `System.Xml.XmlConvert.ToDateTime` n'est plus accessible. Pas de possibilité de laisser le soft planter chez le client "un jour" quand la fonction sera appelée. En revanche il vous faudra chercher un peu pour savoir qu'il faut utiliser à la place `XmlConvert.ToDateTimeOffset` car ici le nom change aussi.

Il sera assez rapide de s'apercevoir aussi que `System.ICloneable` n'existe plus, le code sera rejeté. Ecrire une méthode *ad hoc* qui retourne le bon type sera la seule solution. Mais cela ne devrait pas trop couter puisqu'il suffira de reprendre le code créé pour supporter `ICloneable`.

Vous découvrirez d'autres changements... Le site que je vous ai proposé dans un billet précédent et qui s'efforce de lister les équivalences entre Silverlight et WinRT devrait vous aider à trouver la parade (voir le billet sur WinRT Genome Project).

Conclusion

Pareil / pas pareil ... ? That is the question !

Beaucoup de similitudes, beaucoup de petits et parfois grands changements entre Silverlight et WinRT.

Ecrire un code portable entre les deux environnements sera certainement plus facile que de porter un code existant. Quand on sait qu'il y a des différences on prévoit son code en conséquence, quand on reprend un code qui "ne savait pas" qu'il y aurait des changements c'est parfois toute sa structure qui est à revoir.

Mais malgré tout, soyons honnêtes, l'effort de compatibilité entre Silverlight et WinRT est gigantesque.

Tous ceux qui avaient choisi le couple C# / Xaml pour développer seront ravis de voir que 95% de leur savoir-faire reste totalement utile et d'actualité.

Il reste à Windows 8, l'OS à deux têtes, à séduire autant le grand public que les directions informatique ce qui après un an et demi au moins de présence semble plus difficile que prévu par Microsoft…

Personnellement je serai enclin, spontanément, à préférer encore Silverlight pour tout ce qui est de type intra et extranet en entreprise notamment. Mais finalement, WinRT n'est pas un mauvais choix…si on a décidé de passer à Windows 8 tout le parc informatique de l'entreprise !

Sinon le choix de WPF semble s'imposer partout pour tout logiciel qui ne peut supporter les limitations de WinRT, le Windows store et la sandbox. J'ai toujours aimé WPF. J'ai adoré Silverlight. Mais j'aime toujours WPF. Je renverrai le lecteur qui connaitrait mal WPF vers cet article de 2008 "10 bonnes raisons de choisir WPF" ou même "9 raisons de plus d'utiliser WPF" de 2009. Ca commence à dater, mais finalement l'histoire n'étant qu'un éternel recommencement, la chute de Silverlight ravive tout l'intérêt de WPF dans bien des cas, même sous Windows 8 !

Pour les applications grand public en revanche, WinRT est le choix le plus intelligent, si on désire suivre Microsoft dans son grand "shift"… Mais lorsqu'on voit l'indigence des outils de développement pour Java, Html 5, Android ou iOS comparativement à des monstres (gentils) comme Visual Studio et Expression Blend, franchement, à moins d'être masochiste et qu'on soit ou non d'accord avec la totalité de la nouvelle démarche de Microsoft, il faudrait être fou pour ne pas choisir la voie Modern UI et WinRT…

De Silverlight à WinRT : Mesurez les différences grâce au « WinRT Genome Project » (partie 4)

Passer de Silverlight à WinRT est encore plus simple que depuis WPF tant les deux plateformes se ressemblent. Les deux sont sandboxées, et pour développer de Windows Phone 8 à Windows 8 Pro en passant par RT, C# et Xaml sont aujourd'hui clairement poussés par Microsoft (contre tout ce qui a été dit auparavant sur Html…). Mais quelle distance sépare Silverlight de WinRT ? C'est tout l'intérêt du "WinRT Genome Project" de répondre, au moins partiellement, à cette question.

WinRT Genome Project

Il s'agit d'un projet visant à comprendre le cœur de WinRT. Et puisque Silverlight en est si proche, le mieux est de comparer ces deux plateformes.

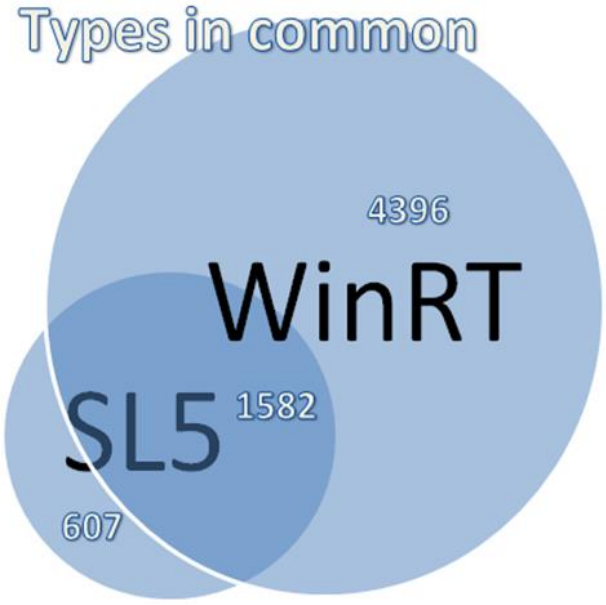
La question qui se pose d'emblée est de savoir jusqu'à quel point ces plateformes sont similaires ? Quelle est la quantité de choses nouvelles qu'il faudra apprendre pour qui connait Silverlight ou WPF ?

Tim Greenfield a tenté de répondre à ces questions. Il a commencé par écrire un programme qui balaye tous les assemblages par défaut de WinRT pour les comparer à leurs équivalents Silverlight.

Vous pouvez lire son analyse (en anglais) sur son blog "Programmer Payback".

Types en commun

La première comparaison consiste à voir combien de types sont communs. Tim résume la situation par le schéma suivant :



WinRT est farci de milliers de types différents ! 5978 comptés par Tim, ce qui est beaucoup au vu des 2189 que compte Silverlight, plus du double.

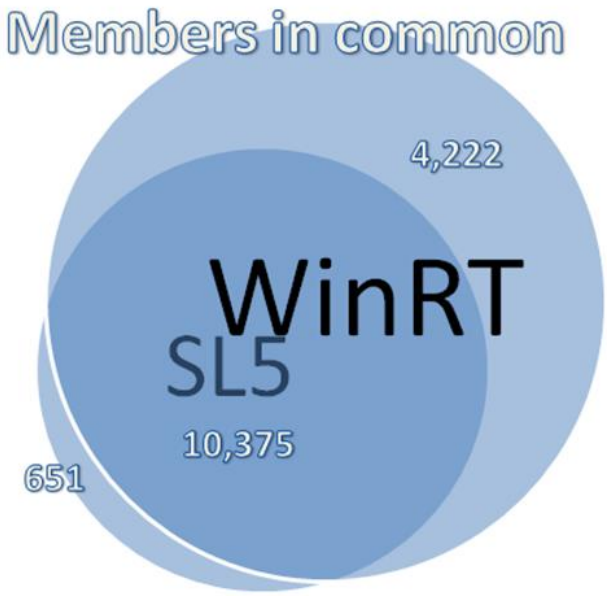
Mais cela est trompeur. Silverlight peu compter bien plus de types que cela dès qu'on ajoute toutes les librairies utiles à une vraie application : WCF Ria Services, le Toolkit, toolbox MVVM, etc.

Sur les 2189 types de base de Silverlight, 1582 sont commun avec WinRT et 607 ne se retrouvent pas dans ce dernier. Presque tout Silverlight se retrouve donc dans WinRT. Ici encore il faut nuancer les choses, 607 types non portés dans WinRT c'est par exemple toutes les classes servant à la gestion de la passerelle avec DOM ou l'Out of brower qui n'ont plus aucun sens sous WinRT. De même, les 4396 types de plus de WinRT intègrent de nombreuses nouvelles API pour gérer l'accéléromètre, les caméras, etc.

Membres en commun

Parmi les 1582 types communs, Tim s'est attaché à pousser la comparaison plus loin pour voir si la proximité de ces types est totale ou partielle, et quantifier la différence si elle existe.

Bien entendu, tout comme de nombreux types communs entre Silverlight et WPF possèdent des différences, on va retrouver des différences entre les types Silverlight et WinRT, ce que résume le graphique suivant :



Ici la nuance est plus faible, ce qui est naturel. Si un type est commun on peut supposer que le besoin l'est aussi et que les différences seront légères. S'il y a trop de différences alors le type n'est tout simplement pas commun...

Sur les 14597 membres recensés de WinRT, 10375 sont communs avec les membres équivalents de Silverlight (c'est la version 5 qui est prise comme base de calcul). Seuls 651 membres Silverlight ne sont pas dans WinRT. En revanche, les types communs

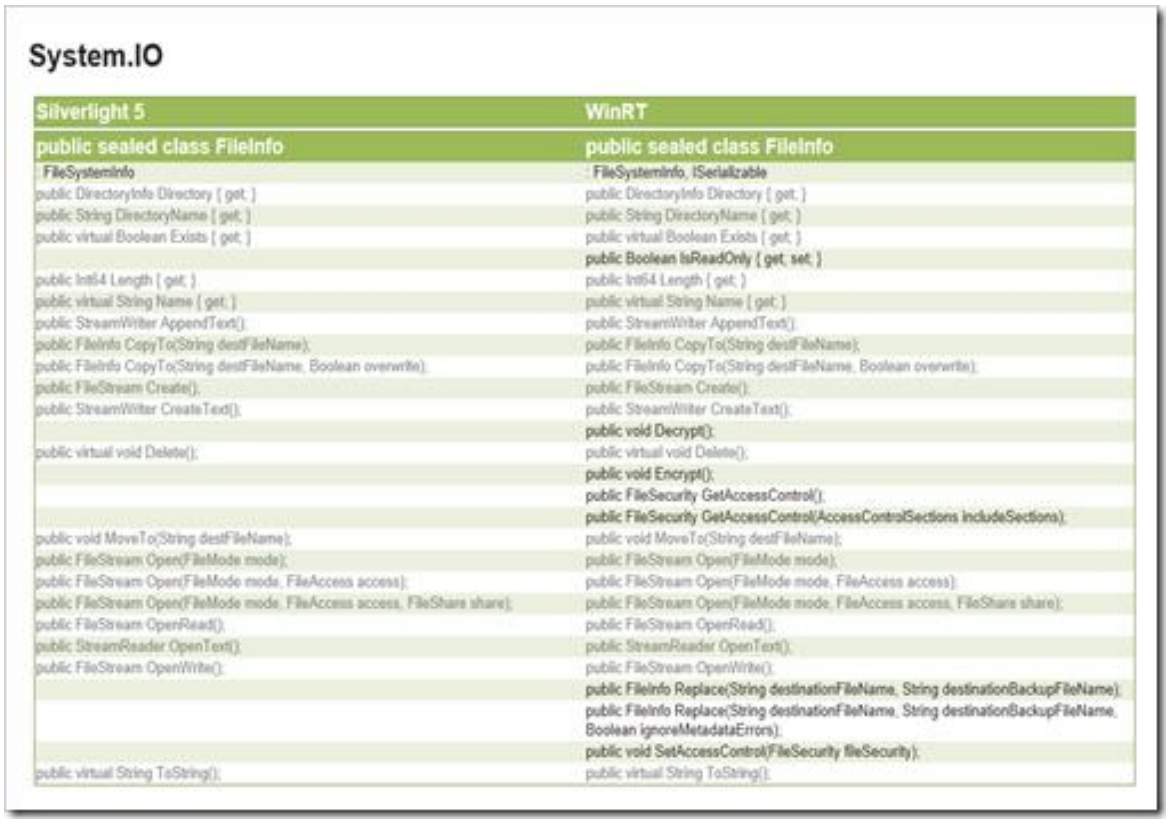
entre les deux plateformes sont souvent mieux pourvus sous WinRT. 4222 membres apparaissent en plus sous WinRT pourtant sur des types communs à la base.

Pousser la comparaison plus loin

Bien entendu tout cela reste assez général. Mais ça donne tout de même une bonne idée de l'ensemble. De la distance entre Silverlight et WinRT, ce fameux delta qu'il faudra acquérir pour connaitre le second quand on connait le premier.

Mais Tim est allé beaucoup plus loin dans les comparaisons en produisant un différentiel classé par espace de noms et groupé selon que les types appartiennent à WinRT exclusivement ou à la partie commune avec Silverlight.

Cela se présente sous la forme suivante (exemple ici de l'espace de nom `System.IO`, avec la colonne SL5 et la colonne WinRT) :



System.IO

Silverlight 5	WinRT
public sealed class FileInfo	**public sealed class FileInfo**
FileSystemInfo	FileSystemInfo, ISerializable
public DirectoryInfo Directory { get; }	public DirectoryInfo Directory { get; }
public String DirectoryName { get; }	public String DirectoryName { get; }
public virtual Boolean Exists { get; }	public virtual Boolean Exists { get; }
	public Boolean IsReadOnly { get; set; }
public Int64 Length { get; }	public Int64 Length { get; }
public virtual String Name { get; }	public virtual String Name { get; }
public StreamWriter AppendText();	public StreamWriter AppendText();
public FileInfo CopyTo(String destFileName);	public FileInfo CopyTo(String destFileName);
public FileInfo CopyTo(String destFileName, Boolean overwrite);	public FileInfo CopyTo(String destFileName, Boolean overwrite);
public FileStream Create();	public FileStream Create();
public StreamWriter CreateText();	public StreamWriter CreateText();
	public void Decrypt();
public virtual void Delete();	public virtual void Delete();
	public void Encrypt();
	public FileSecurity GetAccessControl();
	public FileSecurity GetAccessControl(AccessControlSections includeSections);
public void MoveTo(String destFileName);	public void MoveTo(String destFileName);
public FileStream Open(FileMode mode);	public FileStream Open(FileMode mode);
public FileStream Open(FileMode mode, FileAccess access);	public FileStream Open(FileMode mode, FileAccess access);
public FileStream Open(FileMode mode, FileAccess access, FileShare share);	public FileStream Open(FileMode mode, FileAccess access, FileShare share);
public FileStream OpenRead();	public FileStream OpenRead();
public StreamReader OpenText();	public StreamReader OpenText();
public FileStream OpenWrite();	public FileStream OpenWrite();
	public FileInfo Replace(String destinationFileName, String destinationBackupFileName);
	public FileInfo Replace(String destinationFileName, String destinationBackupFileName, Boolean ignoreMetadataErrors);
	public void SetAccessControl(FileSecurity fileSecurity);
public virtual String ToString();	public virtual String ToString();

Pour accéder au site de comparaison et à tous les espaces de noms, cliquez sur l'image ci-dessus... (en plus ce sera plus lisible !)

Conclusion

Je vous ai déjà proposé récemment d'approcher WinRT par ses API. C'est le meilleur moyen de prendre connaissance d'une nouvelle plateforme quand on sait programmer. On y trouve des types, des membres, dont les noms parlent immédiatement, on apprend à se situer comme avec la carte géographique d'un nouveau lieu à découvrir.

Aucun pilote de rallye, aucun cycliste, aucune troupe armée, aucun randonneur sérieux ne se lance sur le terrain sans avoir étudier des cartes au préalable.

C'est aussi l'avis de Tim visiblement...

Aborder les choses par le gros bout de la lorgnette en plongeant dans des exemples de code sans avoir déjà une bonne idée du terrain sur lequel ils se reposent est une méthode facile, accrocheuse certes, mais pédagogiquement sans valeur.

Alors avant d'aborder WinRT par l'exemple, commençons par nous imprégner de ses espaces de noms, des types que propose son API, des similitudes et des différences avec Silverlight et WPF.

C'est une bonne entrée en matière. Sans la surévaluer, et malgré son austérité, ne négliger pas cette première approche. Quand nous verrons du code dans quelques temps vous apprécierez cette sensation de déjà vu...

De Silverlight à WinRT (partie 5)

Cinquième partie de cette série intitulée "De Silverlight à WinRT" visant à aider les développeurs connaissant bien Silverlight (ou WPF) à aborder WinRT et ses différences. Cette 5ème partie synthétise l'ensemble des points à connaitre.

La série "De Silverlight à WinRT" – épisodes précédents

Le lecteur intéressé trouvera sur Dot.Blog de très nombreux billets sur WinRT, Modern UI (classés parfois sous le terme Metro). Parmi ces billets voici ceux qui s'inscrivent dans la logique du présent :

- De Silverlight à WinRT (partie 1)
- De Silverlight à WinRT (partie 2)
- De Silverlight/WPF à WinRT : .NET pour Metro Style (partie 3)
- De Silverlight à WinRT : Mesurez les différences grâce au "WinRT Genome Project" (partie 4)
- WinRT : les bases

Le dernier billet listé peut être considéré comme une introduction à la plateforme WinRT, qu'on vienne du monde Silverlight ou non.

Ce qu'il faut savoir en 10 points

Pourquoi 10 points et pas 5 ou 32… En réalité tout choix est subjectif, et celui de ces 10 points à venir l'est aussi. Mais cela permet de se faire une idée d'emblée. Dix points ça se compte sur les doigts, cela reste une quantité "raisonnable". On évite ainsi d'effrayer le visiteur en lui parlant des 642 différences, car même si le nombre de différences réelles devait être aussi grand on aurait vite fait de les regrouper en 10 catégories pour retrouver les 10 points…

Ok. C'est artificiel. Mais cela permet de fixer le décor et de proposer un découpage des différences, nombreuses, entre Silverlight et WinRT (programmé en C#/Xaml).

Les différences fondamentales

Applications Silverlight	Applications WinRT
Conçues pour le clavier et la souris	Conçues pour les écrans tactiles
Environnement VS 2008 à 2012, Windows et Mac	Uniquement VS 2012 et seulement sur un PC Windows 8, Uniquement VS 2013 et un PC en Windows 8.1 pour faire du WinRT
Accès à tous les framework .NET de 2.0 à 4.5	Uniquement le framework 4.5
Support partiel de XNA	XNA est supprimé et Direct3D est accessible uniquement en C++
La majorité des applications utilise C# et XAML, quelques unes utilisent le couple VB/Xaml	Choix plus vaste C#/VB/C++ avec XAML et HTML5/JS/CSS3
Les apps tournent sur PC, MAC et Linux (avec MoonLight)	Les apps ne tournent que sous Windows 8
Les apps fonctionnent dans le plugin au sein d'un browser ou en mode Out Of Browser sur le bureau	Les apps tournent dans un profil .NET spécial au-dessus de WinRT dans Windows 8

Ces différences sont vraiment essentielles et primordiales, elles tiennent à la nature intrinsèque des environnements. On notera que si Silverlight donne l'impression de toucher plus de cibles que Windows 8, cela se discute largement. D'abord parce que

le support des différents browsers semble se tasser depuis Silverlight 5 et que même le support Mac parait en voie de délaissement. Ensuite parce que du côté Windows 8, certes les apps ne peuvent tourner que sous cet OS, mais c'est l'OS lui-même qui est "cross-form factor", du smartphone au PC en passant par les tablettes.

Le cycle de vie d'une application

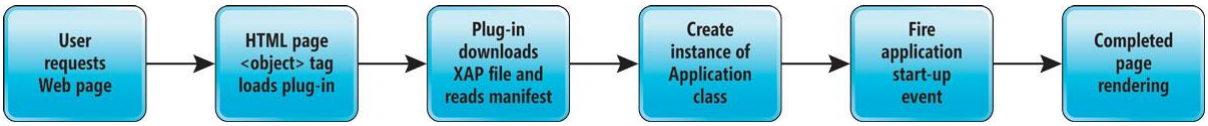
Du point de vue du cycle de vie Silverlight, tout comme WPF se différencient énormément de WinRT.

Une application Silverlight est exécutée et arrêtée, comme une application desktop WPF, qu'elle fonctionne in-browser ou Out-of-Browser. On retrouve là un modèle classique en place depuis des lustres.

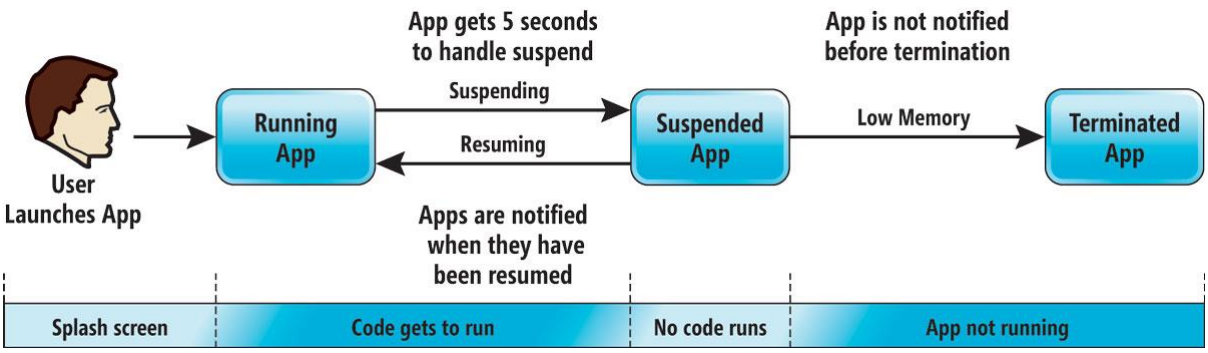
WinRT fonctionne comme un OS de smartphone, et pour cause, il est en mesure de tourner aussi sur des unités mobiles.

Les applications sont lancées par l'utilisateur mais leur fin réelle est pilotée par l'OS avec l'existence d'une sorte de "coma", le tomb-stoning (terme utilisé sur smartphone et pas sous Windows 8 par Microsoft bien qu'il s'agisse de la même chose).

Ci-dessous le modèle classique tel qu'utilisé par Silverlight et son plugin :



Ce modèle devient le suivant sous WinRT :



Les conventions sont plus nombreuses et plus contraignantes pour l'application, notamment à cause de l'existence de timing précis à respecter et de celle de l'état "suspendu", le fameux "coma" évoqué plus haut.

C'est l'OS qui décide de mettre fin à une application en fonction de la mémoire libre et de ses besoins. Mais pour l'utilisateur une application Windows 8 ne s'arrête jamais : quand il la reprend, qu'elle vienne de la mémoire ou qu'elle soit rechargée puis réhydratée, elle apparait telle qu'elle était lors de sa dernière utilisation.

Cela créée des contraintes supplémentaires pour le développeur bien entendu.

Les espaces de noms

Il y a eu une réorganisation des espaces de noms, WinRT étant en lui-même très différent de .NET, la couche .NET qui est placée au-dessus a du se plier à certaines changements. Ce qui brise la compatibilité directe de certains codes XAML ou C#, même si les adaptations sont rarement difficiles à effectuer (sauf pour des fonctions ayant disparu comme les Behaviors par exemple).

Par exemple un UserControl Silverlight sera défini de la façon suivante :

```
<UserControl x:Class="MyTestApp.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:MyTest="clr-namespace:MyTestApp" >
</UserControl >
```

Sous WinRT le même contrôle sera défini comme suit :

```
<UserControl x:Class="MyTestApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mytest="using:MyTestApp" >
</UserControl >
```

On voit que "`clr-namespace`" n'est plus utilisé et que "`using`" le remplace, de même les assemblages ne sont plus directement nommés. Il s'agit d'une évolution accessoire mais intéressante, toutefois elle brise le code existant.

Beaucoup d'espaces de noms ont été modifiés ou renommés, beaucoup d'autres apparaissent aussi, mais c'est dans la sphère de l'UI que ces changements sont les plus importants et parfois les plus gênants.

En général un code Silverlight standard déclare les usings suivants :

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

Alors que le même type de code, sous WinRT déclarera plutôt cet ensemble-là :

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
```

Pour prendre connaissance plus en profondeur des changements dans les espaces de noms le lecteur peut se référer aux billets cités en début d'article (notamment celui présentant le WinRT Genome Project).

Les requêtes Web

La plupart des applications modernes font des appels au Web. Sous Silverlight certaines restrictions existaient, WinRT en impose d'autres :

- La classe `WebClient` n'existe plus. On accède au Web via `HttpClient` sous WinRT (lorsqu'il s'agit de requêtes HTTP bien entendu)
- La classe `WebRequest` dans WinRT fonctionne comme tout le reste sous cet environnement : en mode asynchrone, et sans aucune choix possible alternatif d'une version synchrone. Les parties de code utilisant de telles requêtes sont à revoir et nécessite de comprendre les nouveaux mots clés `await`, `async` et les `Task<T>`.
- Tous les callbacks, comme `IAsyncResult`, qui peuvent être utilisés en Silverlight doivent être totalement réécrits.

Les différences sont marquantes à ce niveau et tout code utilisant intensivement des appels Web devra être revu sous WinRT.

Le listing suivant montre un appel utilisant `WebClient` sous Silverlight :

```
public void btnSearchClick(object sender, RoutedEventArgs e)
  {
    string topic = txtSearchTopic.Text;
    string diggUrl = String.Format(

"http://services.digg.com/stories/topic/{0}?count=20&appkey=http%3A%2F%2Fsc
ottgu.com",
      topic);

    // Initiate Async Network
    WebClient diggService = new WebClient();

    diggService.DownloadStringCompleted += new
      DownloadStringCompletedEventHandler(
      DiggService_DownloadStoriesCompleted);
    diggService.DownloadStringAsync(new Uri(diggUrl));
  }

private void DiggService_DownloadStoriesCompleted(object sender,
    DownloadStringCompletedEventArgs e)
  {
    if (e.Error == null)
    {
      // Call a Method that parses the XML data.
      DisplayStories(e.Result);
    }
  }
```

Voici le même code utilisant les appels Web de WinRT :

```
public async void btnSearchClick(object sender, RoutedEventArgs e)
  {
    // Retrieve Topic to Search for from WaterMarkTextBox
    string topic = txtSearchTopic.Text;
    // Construct Digg REST URL
    string diggUrl = String.Format(

"http://services.digg.com/search/stories?query={0}&appkey=http://www.scottg
u.com",
      topic);

    // Initiate Async Network call
    var client = new HttpClient();
    var response = new HttpResponseMessage();

    //Get response
    response = await client.GetAsync(new Uri(diggUrl));
    Task <string > responseString = response.Content.ReadAsStringAsync();
    string result = await responseString;

    // Call a Method that parses the XML data
    DisplayStories(result);
  }
```

Grâce à `await` et `async` le code est finalement plus fluide est plus clair, presque plus séquentiel bien que totalement asynchrone.

Si les bénéfices sont évidents en termes de clarté et de maintenabilité du code, là encore il y a totale incompatibilité avec l'existant qu'il faudra réécrire.

Fichiers et Isolated Storage

Les applications Silverlight et WinRT tournant dans une sandbox on retrouve à peu près les mêmes contraintes par exemple lors de l'accès au système de fichiers ou à l'Isolated Storage.

Toutefois, ici aussi, les nuances de concept et surtout de programmation sont nombreuses.

Malgré tout, en ce qui concerne l'Isolated Storage concept et techniques d'accès sont parfaitement équivalents. On peut créer, lire, mettre à jour ou supprimer n'importe quel fichier à l'intérieur de l'IS privé de l'application.

Pour les folders spéciaux, dont "`Mes Documents`" ou "`Mes Images`" il existe la possibilité sous WinRT d'utiliser un `File Picker`. Si on veut éviter cette intervention de l'utilisateur il faut réclamer les droits via le Manifeste de l'application. Une facilité non offerte aux applications Silverlight (en mode Web sans élévation de droit).

Pour tous les autres emplacements sur les disques locaux, WinRT offre le même mécanisme, c'est à dire un accès en lecture et écriture via le `File Picker`. Mais il reste impossible d'écrire dans un répertoire (comme le `C:\temp` par exemple) sans avoir réclamé l'autorisation à l'utilisateur. Sous Silverlight 5 il est possible d'éviter cela en utilisant les élévations de droit.

Comme on le voit, si les fondamentaux d'un runtime sanboxé se retrouvent dans les deux environnements, il y a autant de différences que de similitudes ce qui rend le "jonglage" un peu délicat lorsqu'on porte un code de Silverlight vers WinRT.

Navigation

Silverlight propose un service de navigation basé sur des URI. On l'utilise assez facilement avec du code de ce type pour changer de page :

```
this.NavigationService.Navigate(new Uri("/MaPage2.xaml",
UriKind.Relative));
```

Personnellement c'est un procédé que j'utilise rarement, utilisant Silverlight pour créer des applications LOB et non des applications mimant le Web. De fait mes applications utilisent plutôt un Shell avec une plusieurs régions et les pages sont généralement des `UserControl` disposant d'un ViewModel. De plus, ce qui est propre au Web dans sa navigation (pouvoir aller en arrière et en avant) s'accorde assez mal avec des applications LOB (en tout cas cela demande un effort de penser ces applications avec l'esprit Web et non pas bureau).

Il n'en reste pas moins vrai que le système de navigation de Silverlight, et notamment le deep linking, peuvent rendre de grands services ponctuellement.

Sous WinRT on retrouve aussi un service de navigation fonctionnant sur des principes assez similaire (type Web donc) mais avec une implémentation radicalement différente. Là encore la compatibilité est brisée.

Sous WinRT ce service utilise un composant `Frame` qui, en généralement, utilise un `UriMapper` qui joue l'aiguilleur. C'est un peu plus sophistiqué que sous Silverlight, peut-être un peu plus souple, mais cela réclame aussi plus de travail que la simple ligne donnée en exemple plus haut. Voici à titre d'illustration une `Frame` WinRT utilisant un `UriMapper` pour gérer la navigation dans l'application :

```
<navigation:Frame x:Name="ContentFrame"
  Style="{StaticResource ContentFrameStyle}"
  Source="/Home" Navigated="ContentFrame_Navigated"
NavigationFailed="ContentFrame NavigationFailed" >
  <navigation:Frame.UriMapper >
    <uriMapper:UriMapper >
      <uriMapper:UriMapping Uri="" MappedUri="/Views/Home.xaml"/ >
      <uriMapper:UriMapping Uri="/{pageName}"
         MappedUri="/Views/{pageName}.xaml"/ >
    </uriMapper:UriMapper >
  </navigation:Frame.UriMapper >
</navigation:Frame >
```

La navigation dans le code s'écrira alors :

```
this.Frame.Navigate(typeof(NewPage));
```

Il n'y a plus d'URI mais un type. WinRT gère aussi le passage de paramètres qui pourront être récupérés par la page appelée dans les arguments de l'évènement OnNavigatedTo :

```
this.Frame.Navigate(typeof(NewPage),new CustomerInfo{Id=5236});
```

En dehors des nuances d'implémentation on retrouve malgré tout le même esprit avec les classes `Frame` et `Page`. On retrouve aussi tout ce qu'il y a de frustrant, à mon avis, dans ces procédés de navigation de type Web peu assortis aux exigences d'applications LOB, mais c'est un avis personnel qui peut évoluer, juste une question d'habitude certainement.

Les Contrôles

De très nombreux contrôles de Silverlight sont absents de WinRT. On peut supposer que comme certaines features de XAML (tels les `Behaviors`) il s'agit vraisemblablement d'un manque de temps pour tout faire dans la première release de Windows 8. Mais quel que soient les raisons de ces absences, il faut convenir qu'elles sont à la base d'une cassure de compatibilité importante puisque les UI sont impactées. Heureusement la version 8.1 ajoute le support au Behaviors Blend. On retrouve ici la souplesse perdue pendant un an avec Windows 8.0. Microsoft s'étonne parfois (certainement faussement, ils ne sont pas bêtes) que les clients attendent la

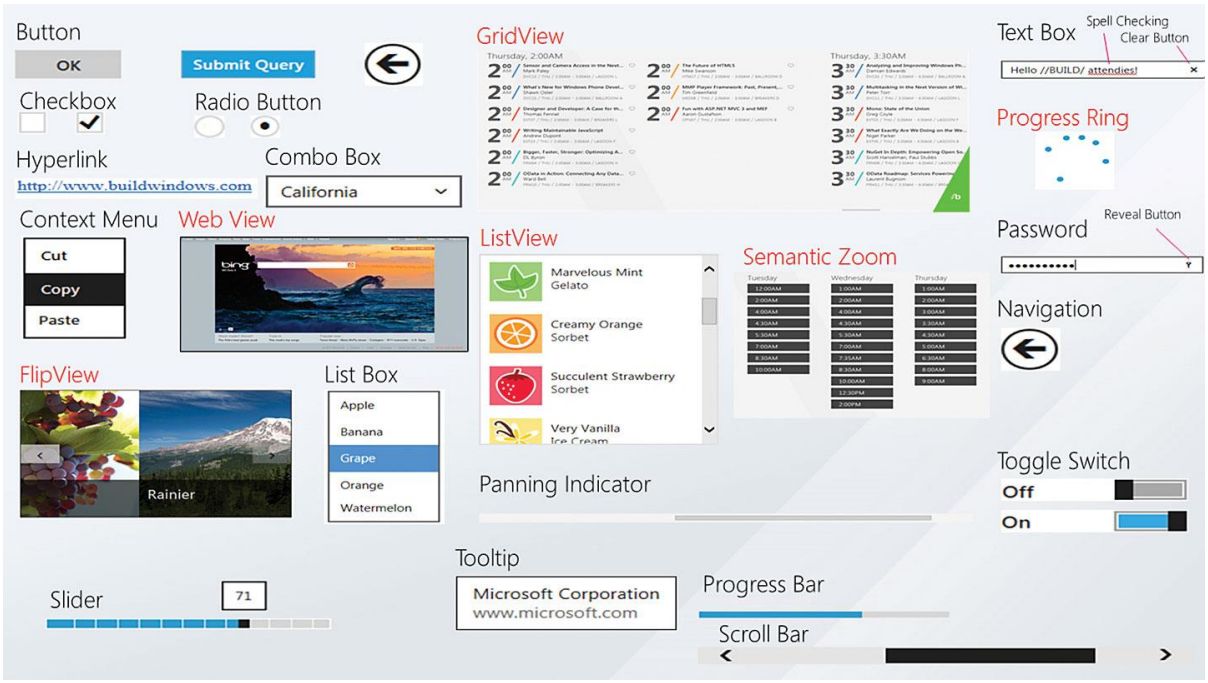
SP1 ou la SP2 avant de regarder leurs produits. La règle se vérifie ici aussi. Ceux qui se sont enquiquiné à se passer des Behaviors pendant un an ont perdu leur temps…

Par exemple, il n'y a plus (ou pas encore) de Calendar, de ChildWindow, de DataGrid, de Treeview, etc. Autant de contrôles utilisés assez largement sous Silverlight.

Toutefois il ne faut pas non plus trop penser "transposition". Je ne crois pas au portage massif d'applications Silverlight vers WinRT. En revanche je crois en la portabilité, avérée, du savoir-faire Silverlight qui permet d'aborder facilement le développement WinRT.

De fait, certains contrôles Silverlight absents de WinRT sont remplacés par des versions plus adaptées au tactile où à l'UX et à l'UI Metro.

Par exemple le `ListView` affiche une liste d'Items qui, de prime abord, le fait ressembler beaucoup à une `ListBox`. Le `GridView` offre un layout de type `Grid` mais avec le groupage des items. Le Zoom sémantique rend certains affichages Silverlight caduques car ici c'est l'UX que l'on modifie totalement. Le Media player est très proche du `MediaElement` mais possède ses propres boutons (au lieu d'avoir à les ajouter sous Silverlight). La `Progress Ring` remplace le `progress Indicator` de Silverlight avec une animation en cercle typique de Windows 8. `L'application Bar` remplace les menus, le `CarouselPanel` ou la `WrapGrid` sont de nouveaux contrôles intéressants. L'intégration dans les Charmes de recherche ou de gestion des paramètres modifie totalement la logique de ces fonctions et donc leur implémentation autant que leur aspect visuel, etc…



En réalité, une interface Modern UI est très différente de celle d'une application Silverlight, pas tellement du point de vue technique mais de celui de l'UX. Et ces changements de l'UX, la nécessite aussi de s'intégrer dans un tout, là où un site Web peut être totalement différent des autres, obligent à repenser l'approche de l'UI. La simple prise en compte du tactile (et de la taille minimale des éléments pour qu'un doigt puisse "'cliquer" sans "baver" sur les contrôles adjacents) implique de repenser les mises en page.

Une application bien faite, respectant MVVM, ne sera pas trop difficile à porter, mais bien que Silverlight et WinRT utilisent Xaml il faut les considérer comme des plateformes différentes.

Néanmoins, comme je le disais un peu plus haut, il faut se réjouir des ressemblances entre Silverlight et WinRT (en C#/Xaml) du point de vue de la "portabilité de notre savoir-faire" bien plus que de celle des applications.

Les animations

Animer une UI semblait un peu gadget à la sortie de Silverlight, tout le monde il est vrai pensait à l'utilisation souvent cheap qu'il était fait de cette fonctionnalité avec les bandeaux de pub en Flash…

Faire entrer les animations comme des éléments à part entière de l'UX et de l'UI et au sein d'applications "sérieuses" a été certainement la mission la moins discourue mais la plus moderne de Silverlight et WPF.

Dans un billet “fondateur” de ma vision de l’avenir (La cassure conceptuelle) j’invitais déjà en 2009 le lecteur à s’interroger sur la conception de nouvelles UI et d’une nouvelle UX totalement détachées du clavier, de la souris et des représentations habituelles des données. Les animations comptent pour beaucoup dans cette “nouvelle vision” (entre guillemets vu de quand date ce billet, même s’il reste très en avance encore aujourd’hui).

Windows 8 fait un pas de plus vers les concepts que j’évoquais alors. Et notamment du côté des animations puisque tout sous Windows 8 est animé. Ces animations sont souvent discrètes (et heureusement !), elle servent à la fois la **compréhension de l’action** et l’esthétique, c’est à dire le besoin d’un peu de beauté dans notre métier habitué aux grilles de chiffres et aux UI tristes à mourir de type XP.

Il y a donc beaucoup de choses sous WinRT côté animation, notamment beaucoup de comportements animés fournis de base en respect avec le look and feel global de Windows 8. Sous Silverlight il faut tout inventer à chaque fois pour chaque application (même si des toolkits nombreux sont ensuite apparus pour combler certains manques).

En dehors de cette avancée conceptuelle, les animations de WinRT (en Xaml) fonctionnent comme celles de Silverlight.

Les Charmes

Je les ai évoqués déjà dans ce billet. Ils jouent un rôle essentiel dans la nouvelle interface de Windows 8. Recherches, paramètres, partage de données, ils sont la clé d’une nouvelle façon d’architecturer une application comme appartenant à un tout.

Bien entendu cela implique moins de “folie” et plus de “standardisation”. Mais Microsoft veut que l’utilisateur connaisse une UX homogène à l’intérieur de Windows 8. Et c’est une bonne chose. Il est en revanche plus difficile pour les éditeurs de se démarquer de la concurrence, partager une photo avec l’application X via le Charme de partage ressemble beaucoup au même partage de la même photo via l’application Y… Même si le look & feel ne fait pas tout on sait que l’acte d’achat se décide souvent sur des captures écran, il va falloir être inventif !

Les Charmes rendent des services qui n’existent pas en Silverlight puisqu’une application Silverlight n’est qu’une page Web un peu spéciale dans un browser au look particulier changeant par nature entre éditeur. Sous Windows 8 l’harmonisation est un élément essentiel de l’UX. Pour le meilleur, l’UX constante pour l’utilisateur, et pour le pire, une uniformatisation peu exitante pour déclencer l’acte d’achat. De fait

Windows 8 comme Windows Phone 8 sont des OS qui plaisent à ceux qui les achètent mais qui n'ont pas totalement réussi à favoriser l'acte d'achat d'où leur position sur le marché en dessous de leur valeur réelle.

La monétisation

Le meilleur et le pire dans Windows 8 c'est le market place.

Le meilleur parce que cela permet aux développeurs de diffuser largement leurs logiciels et d'espérer en tirer un profit financier, le tout régit par Microsoft avec toute la confiance qu'on peut avoir en cette société.

Le market place créé un lien de confiance entre le client et le développeur, l'assurance d'une certaine qualité (purement technique) pour le premier, notamment l'absence de virus, et l'assurance d'être payé pour le second.

Je disais le "pire" aussi… Car hélas nous ne sommes pas dans le monde des Bisounours.

Microsoft emboite le pas à Apple et Google pour créer un marché totalement verrouillé dans lequel le développeur et le client sont captifs, enfermés dans un système monopolistique qui n'est pas sans poser de problème sous Windows. A la différence d'iOS ou Android il existe déjà un long historique avec des dizaines de milliers d'éditeurs sur PC, dont certains, puissants, ne veulent pas devenir captifs du market place ni payer une taxe à Microsoft ce qui entraînerait soit une hausse des prix rendant les produits non concurrenciels soit une baisse de C.A. inacceptable pour les dirigeants et les actionnaires… Le vide du market place Windows 8 en grosses applications « sérieuses » trouve certainement là une partie de son explication puisque les éditeurs peuvent toujours viser le bureau classique, mettant en exergue l'incohérence malgré tout d'un OS à deux têtes incompatibles entre elles, et, on le voit bien, qui se concurrencent l'une l'autre.

Faudra-t-il maintenir *at vitam aeternam* un "bureau classique" pour les logiciels haut de gamme comme Illustrator et un market place pour le "second choix" ? Cela a-t-il même un sens ? Qui vivra verra ! Mais c'est en tout cas le problème majeur de Windows 8 et Modern UI dont personne ne parle étrangement.

Conclusion

Windows 8, avec C# et XAML, est un environnement plaisant dans lequel un développeur Silverlight ou WPF n'a finalement que peu à apprendre pour devenir productif.

C'est un avantage certain, d'autant que cet effort d'apprentissage très réduit ouvre la porte aux tablettes Surface et aux smartphones sous Windows Phone.

D'un point de vue technique WinRT est une excellente plateforme, mais l'UX qu'impose Modern UI peut être un frein pour certaines applications. Conçu avec le grand public en tête, Modern UI pose des problèmes d'uniformisation et de taxe du market place qui freine l'arrivée des ténors qui pourraient en faire une plateforme gagnante. C'est donc finalement à l'inverse dans les entreprises, qui ne l'adoptent pas pourtant, que Windows 8 sera le meilleur choix en permettant aux équipes .NET maîtrisant Xaml et C# de maintenir des applications pour tous les form factors…

Tout cela est bien paradoxal et on comprend mieux que Windows 8 comme Windows Phone se cherchent une place sur le marché. Ce sont des erreurs d'approche et de positionnement. Techniquement on a du solide avec un tooling de grande qualité.

WinRT par l'exemple

Windows 8 / WinRT : Premiers pas

Dans la continuité de cette série sur Windows 8 et WinRT voici venu le moment de passer aux actes et de voir comment se programme cette plateforme concrètement !

Le set de travail

Sous Windows 8 il suffit d'un Visual Studio 2012 pour créer une application WinRT. Sous Windows 8.1 il faut absolument Visual Studio 2013, et réciproquement. C'est-à-dire que pour débloquer la fonction de création d'applications WinRT sous VS 2013 il faut absolument être sur Windows 8.1.

Ce billet de présentation d'un premier exemple WinRT a été écrit avant la sortie officielle de Windows 8 et bien que disposant déjà de certains outils je ne pouvais en parler. Donc, pour rester loyal à la NDA qui me lie à Microsoft, j'ai utilisé pour ce billet un set de travail parfaitement public lors de l'écriture :

Windows 8 RP Build 8400

Visual Studio Ultimate 2012 RC Version 11.0.50522.1 RCREL

Framework .NET 4.5.50501

Le tout dans une machine virtuelle Virtual Box Oracle

Cet ensemble était disponible à tous ceux qui veulaient tester Windows 8. Bien que nous disposions de façon publique aujourd'hui de Windows 8.1 et VS 2013 avec Blend, ce set de test n'a aucune influence sur l'exemple. Il faut juste se rappeler que le projet a été écrit dans ces conditions particulières.

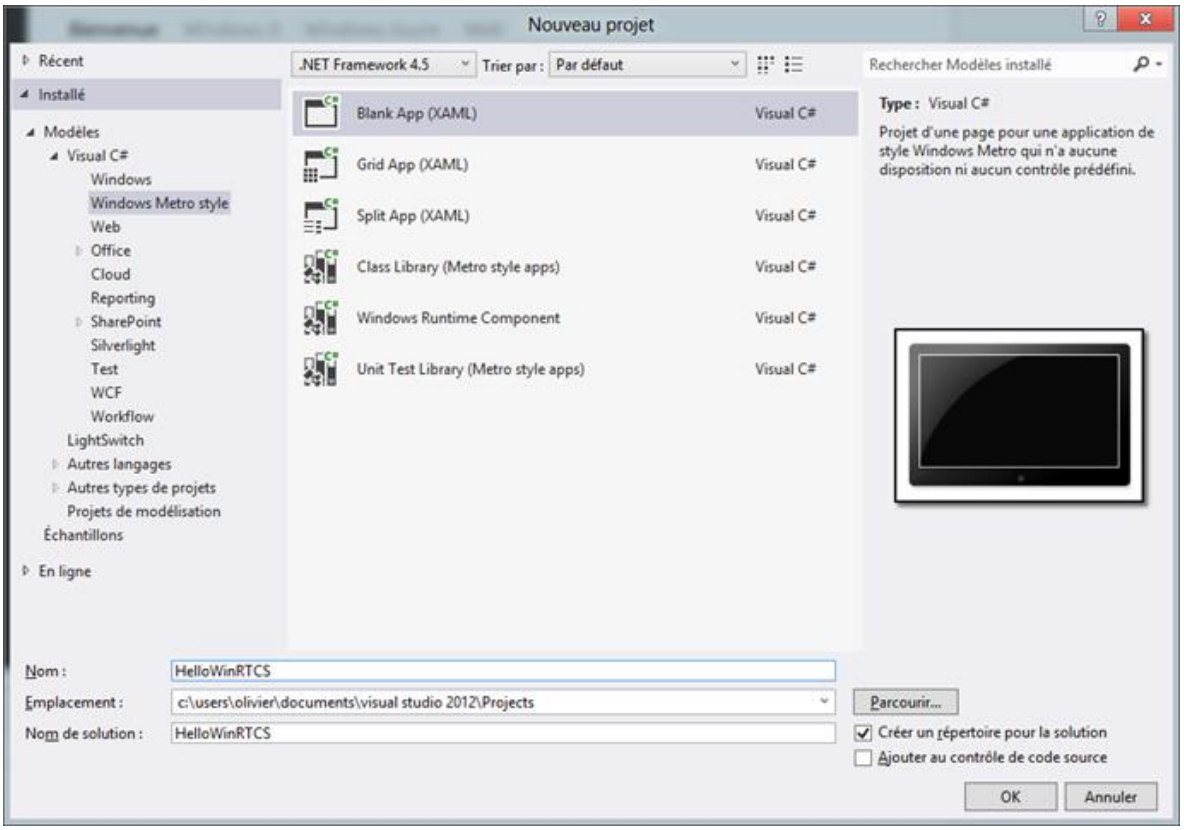
Le fameux Hello World

Même dans une discipline aussi moderne et aussi jeune qu'est l'informatique nous avons nos rites, nos traditions... Toute prise en main d'un environnement débute par l'écriture du célèbre "Hello World". Je resterai fidèle à ce passage obligé.

Dans un billet précédent sur WinRT ("WinRT, les bases"), le décor a été posé. Cependant, nous n'avions pas vraiment écrit un code. Alors faisons-le maintenant !

Création du projet

Je passerai rapidement sur le lancement de Windows 8 et sur celui de Visual Studio depuis le menu à tuile principal pour aborder directement la mise en œuvre de la première application (la création d'un nouveau projet n'a pas vraiment changée dans son fonctionnement).

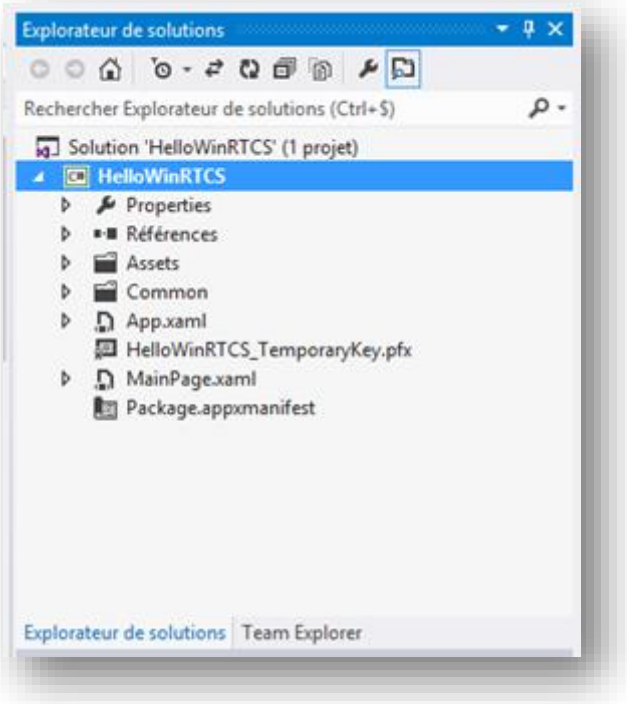


Dans le billet évoqué plus haut, j'ai parlé des projections des languages sous WinRT. Une projection de langage rend possible l'utilisation de WinRT, qui est écrit en code natif C/C++, par un autre langage de façon transparente. Pour cette raison, les développeurs peuvent continuer à utiliser le langage de leur choix pour écrire des applications de style Modern UI. La projection du langage s'assure que les constructions de langue restent intactes. Des choses comme un constructeur, le mot clé `var`, le mot-clé `await` demeurent les mêmes que vous choisissiez d'utiliser WinRT avec c# ou c++. Les projections de langage disponibles actuellement sont : JavaScript, c#, VB et C++. Tous les langages disposent des mêmes modèles de projet. Aucun n'est « plus natif » que l'autre puisque les projections ont été conçues non pas pour "simuler" le fait d'être natif ou non, mais bien pour "rendre natif" tout langage projeté.

Mes préférences sont toutefois connues, et c'est par C# que je commencerai. Créons un nouveau projet (juste en sélectionnant le modèle d'application standard pour l'instant) et nommons-le `HelloWinRTCS`. Visual Studio ouvre un concepteur visuel qui est familier pour le développeur Silverlight ou WPF. Nous disposons d'un environnement en tout point similaire à ces derniers.

Différences avec Silverlight / WPF

Les premières nuances se font voir dans l'organisation du projet, l'arbre de ce dernier (créé par défaut) est le suivant :

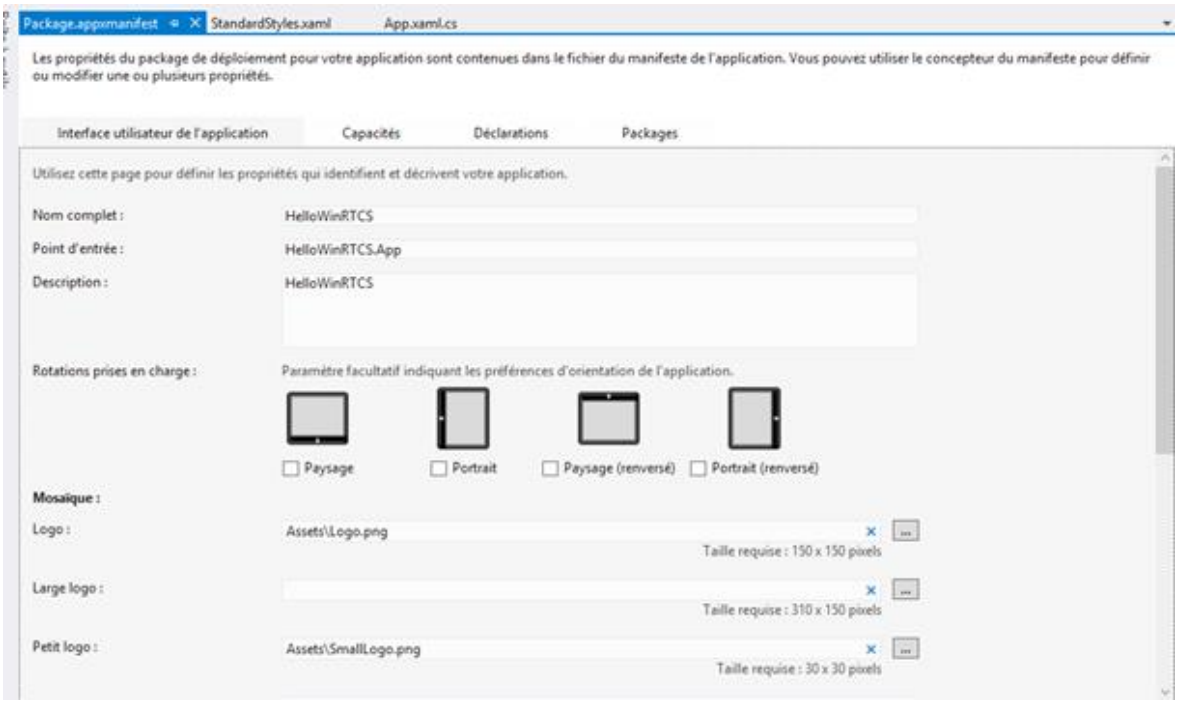


Deux sous-répertoires nouveaux apparaissent : `Assets` et `Common`. Le premier contient pour le moment le Logo de l'application décliné dans toutes les tailles gérées par WinRT ainsi que le Splash Screen et le Logo spécial pour le market place. Bien entendu ces assets devront être personnalisées ce que nous ne ferons pas ici.

`Common` contient pour l'instant `StandardStyles.xaml`, un dictionnaire de ressources Xaml qui, s'il n'est pas obligatoire techniquement, contient de nombreuses définitions généralement utilisées par les applications "Windows 8 App".

On trouve aussi un fichier de signature temporaire, les applications WinRT devant être signées pour passer sur le market place.

Enfin, apparait un fichier `Package.appxmanifest` qui s'édite avec un éditeur spécialisé comme les propriétés du projet. Il s'agit en réalité d'une extension de ce dernier. Les "propriétés" du projet étant spécifique à notre projet en C# managé, alors que ce "manifeste" est spécifique à WinRT.



On découvre dans ces réglages tout ce qui concerne le fonctionnement de l'application sous Windows 8 comme son nom complet, sa description, les rotations d'écran prises en charge, le nom et l'emplacement de tous les logos, les notifications, le splashScreen, mais aussi les capacités requises (autorisations de type accès à Internet ou non par exemple) qui sont autant de verrous sécurisant l'application vis à vis de l'utilisateur qui peut prendre connaissance des capacités requises et accepter ou refuser l'installation de l'application selon ce qu'il pense être "safe" ou non.

On notera ici qu'une application bien faite doit tout faire pour limiter les autorisations demandées. L'expérience sur smartphone montre que soit on a affaire à du grand public qui n'y connait rien et plus il y a de questions embarrassantes plus il y a de chance que la peur l'emporte sur la raison (et le logiciel n'est pas installé) soit on a affaire à un public averti et ce dernier est ultra méfiant... Une application de traitement de texte purement local qui me réclame des accès à Internet et à ma Webcam ne sera jamais installée...

Autre différence de taille, mais j'y reviendrai bien entendu plus en détail, c'est le fichier `App.xaml.cs` qui contient la gestion des évènements propres à Windows 8 ressemblant comme deux gouttes d'eau à ceux qu'on trouve sous Windows Phone et qu'on appelle le "*tombstoning*". Nous ignorerons cet aspect des choses dans le Hello World.

On retrouve bien en revanche la fameuse "`MainPage.xaml`" page principale et unique par défaut du projet.

Lorsqu'on l'ouvre le concepteur visuel est légèrement différent de l'habitude qu'on a sous Silverlight ou WPF puisque l'espace simulé ressemble à une ... tablette. On sent ici si ce n'est la vocation (bien plus large) mais en tout cas l'ambition de Windows 8 : conquérir le marché des tablettes... Il est étonnant qu'il n'existe pas un gabari de type PC sans le dessin d'une tablette.

Le code

Voici le code Xaml de la page créée dans le projet par Visual Studio :

```
<Page
    x:Class="HelloWinRTCS.MainPage"
    IsTabStop="false"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HelloWinRTCS"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">

    </Grid>
</Page>
```

Les habitués de Xaml ne sont pas perdus. Les namespaces sont bien entendu légèrement différents mais l'ensemble ressemble à une application Silverlight ou WPF, notamment avec sa grille par défaut. Elle ne porte plus le nom de `LayoutRoot` mais elle joue le même rôle. Quant à sa couleur de fond elle est déjà fixée via un Binding.

Il ne reste plus qu'à ajouter le nécessaire pour notre exemple bien modeste : un `TextBlock` qui affichera le fameux message, et un `Button` pour déclencher cet affichage afin de mettre un peu d'interactivité dans cet *Hello World* :

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
  <TextBlock x:Name="txtMessage" HorizontalAlignment="Left"
Margin="95,62,0,0"
             TextWrapping="Wrap" Text="TextBlock" VerticalAlignment="Top"
FontSize="18"/>
  <Button x:Name="btnHello" Content="Cliquez Moi !"
HorizontalAlignment="Left"
          Margin="89,165,0,0" VerticalAlignment="Top"
```

```
Click="btnHello_Click" />
</Grid>
```

Nous retrouvons naturellement dans la page de code-behind le gestionnaire de l'évènement programmé en Xaml :

```
private void btnHello_Click(object sender, RoutedEventArgs e)
{
    txtMessage.Text = "Bonjour tout le monde !";
}
```

Rien que de très naturel pour qui programme déjà en Xaml donc.

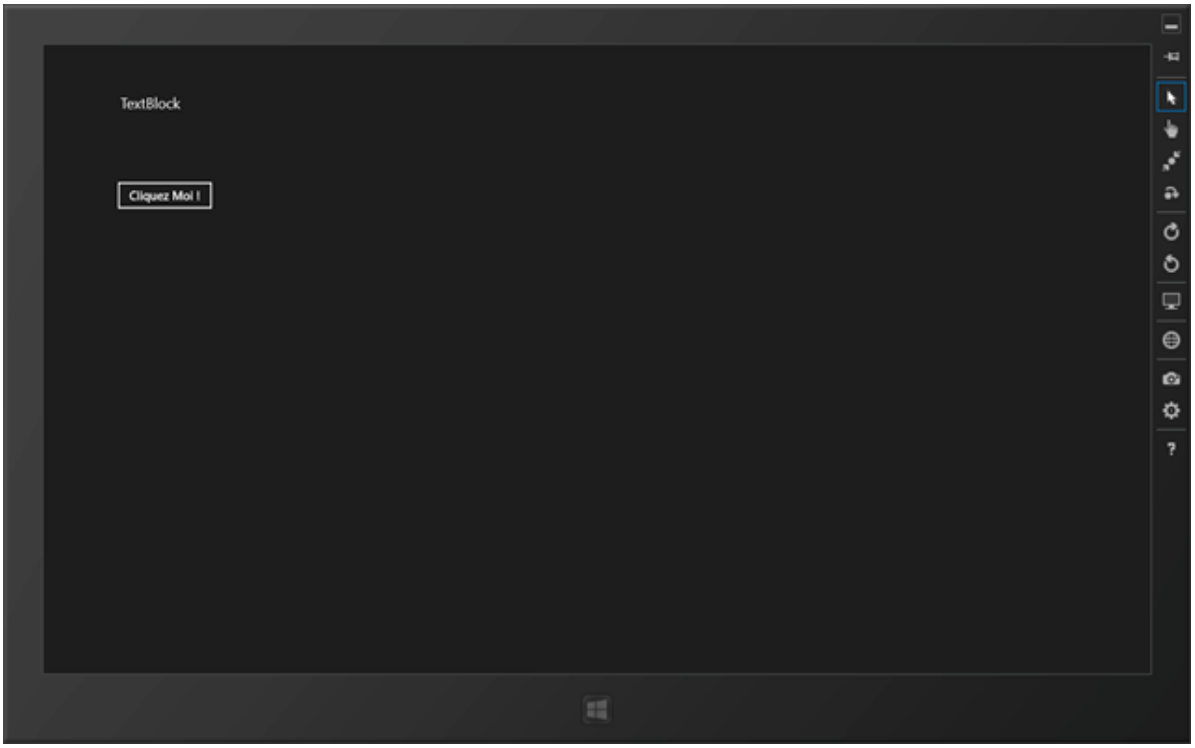
Exécution

Par défaut l'ordinateur local est utilisé. C'est à dire que bien que le concepteur visuel ressemble à une tablette, c'est bien une application Windows 8 qui peut marcher aussi en Desktop que nous venons de créer.

C'est d'ailleurs la force de WinRT : l'unification. Notre application fonctionne désormais aussi bien sur un PC de bureau que sur une tablette ARM ou un smartphone, et sans modification (l'unification avec Windows Phone n'était pas encore totale) !

Certains éditeurs comme Google ou Apple ont pris sur le terrain une avance considérable sur Microsoft, c'est un fait, mais aucun n'a l'expérience et le savoir-faire de Microsoft en matière d'OS et d'environnement de développement… Et ce que propose Microsoft aujourd'hui vient certes avec retard sur le marché, mais c'est un projet abouti et cohérent, des années en avance sur tout ce qui existe aujourd'hui, laissant loin derrière iOS et Android et même Mac OS…

Bref, exécutons le projet en choisissant le simulateur de tablette, c'est tellement fun de se dire qu'on a écrit un logiciel pour unité mobile sans s'en rendre compte…



Oh la belle tablette Surface !

Cliquons sur le bouton...

Ca marche... (le suspens était toutefois limité j'en conviens)

On notera au passage que le chargement du simulateur est presque instantané (même avec une machine virtuelle). C'est impressionnant quand on connait par exemple le simulateur Android qui même sans machine virtuelle prend des plombes à se charger (l'accélérateur Intel à installer à part améliorant un peu les choses il faut l'admettre). Le fameux savoir-faire d'un spécialiste des OS dont je parlais plus haut à propos de Microsoft, ça se voit quand même, il n'y a pas photo...

Pour la défense d'Android on remarquera que son émulateur fonctionne sous Windows 8 alors que ce n'est pas le cas d'iOS. Ce qui est bien embêtant quand on utilise Xamarin pour faire du cross-plateforme ! Mais revenons à nos moutons électriques.

Aller.. Pour le plaisir, puisque c'est une application WinRT elle marche aussi directement sur le PC. Revenons à Visual Studio, changeons le mode d'exécution et sélectionnons "machine locale".

Pour qu'il y ait une différence visuelle entre les exécutions, j'ai lancé l'application des News qui a pris le contrôle en plein écran (Windows 8 oblige...) puis j'ai fait revenir notre Hello Word mais en le dockant sur la gauche. J'ai cliqué sur le bouton, et bien que l'image soit petite vous reconnaitrez le message "Bonjour tout le monde !" sur fond gris foncé ainsi que le button "Cliquez Moi !". C'est à la page des news qu'on se rend compte que le billet originel date un peu… Une leçon à retenir, n'utilisez jamais un contenu volatile pour illustrer un article, il prendra un coup de vieux alors que tout ce qu'il montre est parfaitement … d'actualité !

Autres langages

Je pourrais vous refaire le même exemple en VB managé, en C++ ou en JavaScript/Html. Ca marcherait pareil au final. Mais je suis avant tout un amoureux de

C# et je laisse les autres langages à leurs aficionados. Ces langages ne m'intéressent pas plus que ça mais j'apprécie qu'ils soient présents et que chacun puisse choisir celui qui lui convient le mieux. La liberté ne me dérange pas. J'ai mes préférences, et même mes raisons de déconseiller certains langages, mais tant qu'on ne m'oblige à rien, laisser les autres décider pour eux-mêmes me convient parfaitement.

Conclusion

Windows 8 et WinRT c'est cela : un super .NET augmenté de nouveaux langages, de nouvelles possibilités, et multiplateforme du smartphone (Windows Phone) au PC (Windows 8) en passant par les tablettes ARM ou non (Surface et Surface Pro, versions 1 et 2).

Un vrai bonheur pour le développeur (un seul savoir à maitriser pour tout faire), un paradis pour le DSI (pas de formations multiples ni de profils différents à financer et à gérer), une aubaine pour le PDG (réduction des couts, réactivité, présence sur tous les form factors).

Comme vous l'avez vu, ce n'est pas sorcier à faire marcher.

J'aborderai bientôt d'autres aspects du développement Windows 8, plus spécifiques à cet environnement.

Windows 8 : créer des tuiles vivantes (live tile)

Les tuiles, éléments emblématique du look Modern UI (ex Metro Style) et donc de Windows 8 sous toutes ses déclinaisons (PC, tablettes, et Smartphones). Plus que de simples icônes, les tuiles sont facétieuses, changeantes, en un mot vivantes. Comment en tirer partie dans vos apps ?

La tuile

La tuile est la base du menu tactile de Windows 8. Comme on le voit sur la capture ci-dessus elle peut revêtir différentes formes. Il faut déjà différencier deux types de tuiles :

- La tuile-icône. Elle ne fait que reprendre l'icône d'une application Windows classique et l'affiche dans un carré. C'est la moins belle des tuiles, la plus passive, mais elle n'est là que pour assurer la compatibilité avec les anciennes applications. Dans la capture on reconnaît la tuile-icône de la console de développement Visual Studio (sur la droite).

- La tuile-vraie. C'est déjà une vraie tuile Windows 8. Son iconographie est adaptée à l'OS. Elle est statique mais mieux intégrée à l'ensemble. Sur la capture on notera dans ce style la tuile "Caméra" qui donne accès à l'appareil photo de la machine (la webcam sur ce PC, ou la caméra embarquée pour une tablette, un Smartphone).

Les tuiles peuvent adopter plusieurs formes :

- Les tuiles carrées. La capture en donne plusieurs exemples. Elles prennent moins de place et sont donc bien adaptées aux petits écrans (Smartphones notamment), Windows Phone 7 utilisait déjà à l'époque cette forme pour

densifier l'affichage. Elles peuvent contenir des informations changeantes mais la place est limitée.

- Les tuiles rectangulaires (dites "wide", large). Ici aussi la capture en donne quelques exemples. Plus larges, elles permettent d'afficher plus d'information et se prêtent mieux que les tuiles carrées aux contenus riches et / ou changeant.

- Windows 8.1 ajoute quelques niveaux de personnalisation supplémentaires aux tuiles.

Enfin, il faut différencier :

- Les tuiles passives. Comme celle de la caméra sur la capture écran ci-dessus elles ne font qu'afficher une icône fixe. Rien ne sert de les agrandir, elles ne feront que prendre plus de place sans rien apporter de plus.

- Les tuiles vivantes. Ce sont elles qui nous intéressent ici. Sur la capture on en découvre quelques unes mais faute d'avoir insérer une séquence vidéo ou un gif animé il est difficile de voir lesquelles "bougent"… Par exemple celle affichant un bout de clavier de PC avec un clavier Midi est une tuile vivante de la galerie photos. L'image présentée change régulièrement. La tuile des informations du jour est de même nature et présente en alternance les gros titres de l'instant.

Les tuiles qui nous intéressent aujourd'hui sont les tuiles "vraies", carrées ou rectangulaires, mais "vivantes".

La librairie NotificationsExtensions

Même si l'API WinRT est très bien faite, toute simplification est la bienvenue. Concernant la manipulation des notifications Microsoft à mis à disposition dans le SDK la Dll "`NotificationsExtensions`" qui simplifie grandement la manipulation de toutes les notifications : tuiles, badges et toasts.

Je ne parlerai que des tuiles vivantes ici, mais cette extension est utilisable dans d'autres contextes donc (`badges` et `toasts`).

La Dll se trouve dans le SDK quelque part sur votre disque et dans de nombreux exemples (notamment ceux portant sur les tuiles – voir mon précédent billet http://www.e-naxos.com/Blog/post/Telechargez-tous-les-exemples-de-code-officiels-Windows-8.aspx).

Un mode d'emploi se trouve aussi sur les sites Microsoft à cette adresse : http://msdn.microsoft.com/en-us/library/windows/apps/Hh969156.aspx.

Nous utiliserons cette extension pour simplifier le code de création d'une tuile vivante.

A noter : cette Dll se présente sous la forme d'un projet Visual Studio de composant Windows 8 à ajouter à votre propre solution. Dans les exemples de code du SDK vous en trouverez un exemplaire par exemple à cet endroit : "`Toast notifications sample\C#\NotificationsExtensions`". Pour des applications hors test je vous conseille de copier ce projet dans la solution de votre app plutôt que de pointer celui se trouvant dans le SDK.

Les astuces à connaître

La toute première astuce à connaître est plutôt un bug (ou un mauvais réglage ?) du simulateur. Les notifications ne marchent pas dans le simulateur, pour le programme exemple montré ici autant que pour toutes les autres tuiles animées. Je n'ai pas trouvé d'information sur ce point, peut-être n'est-ce qu'un paramètre à changer dans l'émulateur, en tout cas, sauf à trouver cet éventuel paramètre, ne vous fatiguez pas à tester les notifications sous simulateur… Il faut utiliser la machine locale (si vous refaites le test avec Windows 8.1 cela a normalement changé).

La seconde astuce est une vraie astuce : dans le manifeste de l'application veillez à ce que le support du mode paysage soit bien coché. Visiblement quand rien n'est coché ça marche moins bien (encore beaucoup de mystères autour de WinRT ! ça se décantera avec le temps).

La troisième astuce est vicieuse : il faut fournir un logo 150x150 mais aussi un logo en 310x150 pour que le mode tuile large soit accepté. Le logo 150x150 est fourni par défaut (un carré avec ses diagonales dessinées, pour les tests cela peut suffire), en

revanche le logo “wide” n’est pas fourni. Il faut donc en ajouter un. Lorsque cela est fait et que le programme est installé dans le menu Windows 8, on peut faire un clic droit dessus pour choisir le mode large ou carré.

On remarquera que les notifications utilisent un fond décidé par le thème en cours, du coup les logos choisis sont masqués dès qu’une notification arrive. Dans ce test ils ne servent donc à rien d’autre qu’à permettre le fonctionnement en mode large ou carré ! (les logos sont aussi visibles au lancement de l’application quand aucune notification n’a été lancée ou quand on désactive les notifications pour la tuile dans le menu contextuel du menu Windows 8). Certains de ces points peuvent avoir connu des modifications sous Windows 8.1, référez-vous à la documentation officielle à jour.

Les notifications sont des objets complexes et ils peuvent supporter des images, même provenant du Web (avec une URL donc). Ici je ne montre que le mode texte le plus simple. Il serait possible de réutiliser les logos pour écrire par dessus dans une version un peu plus sophistiquée.

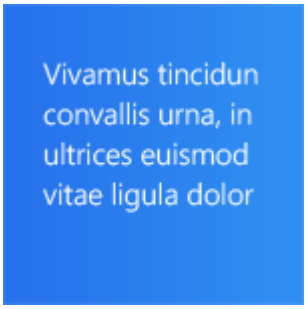
Les thèmes pour tuiles

Windows 8 définit un certain nombre de thèmes pour les tuiles. Lorsqu’on utilise des notifications on fait à appels à ces styles qui conditionnent la présentation. Cela peut être fait par code (il y a des classes et des méthodes qui permettent de créer une notification avec un style donné) ou bien en XML. Dans ce dernier mode on définit la notification à l’aide de balises.

Les tuiles peuvent donc être carrées ou large (le rectangle n’étant finalement qu’un “carré long”…).

Selon cette forme de base certains styles sont disponibles.

`TileSquareImage` `TileSquare150x150Image`	Une image qui remplit tout l’espace du carré. Aucun texte.

`TileSquareBlock`	Un gros bloc texte accompagné d'une ligne en fonte grasse mais nettement plus petite	
`TileSquareText01`	Un texte assez large d'entête et trois lignes simples de texte. Pas de wrap.	
`TileSquareText02`	Un texte assez large d'entête et une ligne simple pouvant se wrapper sur 3 lignes.	
`TileSquareText03`	4 lignes de texte simple sans wrap.	
`TileSquareText04` `TileSquare150x150Text04`	Une ligne de texte simple pouvant wrapper sur 4 lignes.	

`TileSquarePeekImageAndText01` `TileSquare150x150PeekImageAndText01`	Une image puis un texte large avec 3 lignes sans wrap.	
`TileSquarePeekImageAndText02` `TileSquare150x150PeekImageAndText02`	Une image puis un texte avec entête une ligne pouvant se wrapper sur 3 lignes.	

`TileSquarePeekImageAndText03` `TileSquare150x150PeekImageAndText03`	Une image puis 4 lignes de texte simple sans wrap.	
`TileSquarePeekImageAndText04` `TileSquare150x150PeekImageAndText04`	Une image puis une grande ligne de texte simple pouvant se wrapper sur 4 lignes.	

Bien entendu ces styles se déclinent aussi pour les tuiles larges (rectangulaires) avec quelques nuances, ce qui donne environ une quarantaine de styles supplémentaires !

L'idée étant ici de vous donner un aperçu des styles et non de recopier la documentation officielle, je renvoie le lecteur à cette adresse http://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.notifications.tiletemplatetype pour prendre connaissance de toutes les variantes possibles.

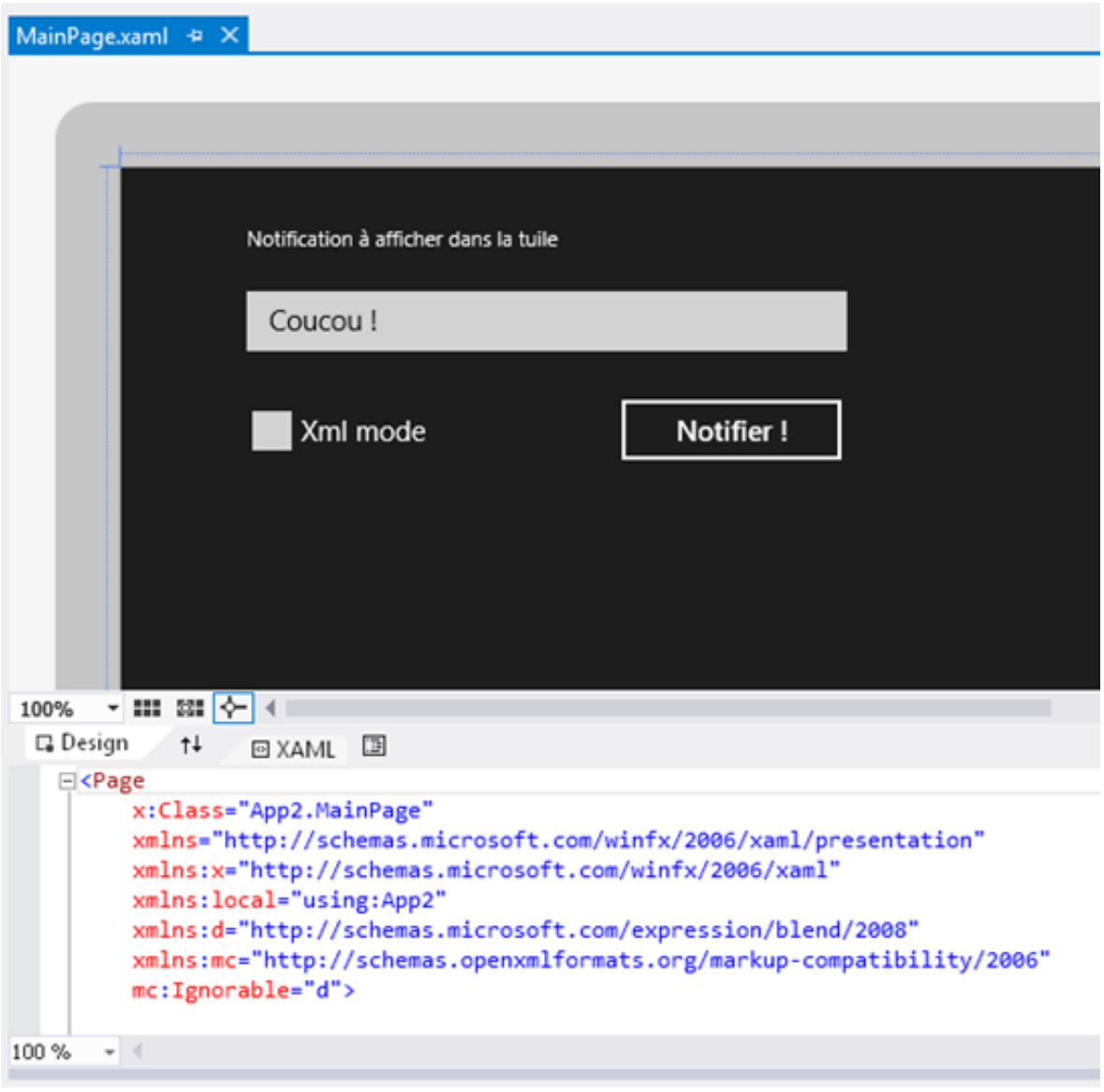
Un exemple

Le plus simple maintenant est de voir comment se servir de toutes ces connaissances sur les tuiles !

Pour commencer je vais créer une application Windows Store simple (le modèle “`blank`”). J’ajoute à la solution une copie du projet `NotificationsExtensions` évoqué en début de billet, puis je créée une interface minimale permettant de saisir un texte qui sera utilisé pour la notification accompagné d’une `CheckBox` sélectionnant soit le mode de création simple soit le mode utilisant une définition XML pour le formatage de la tuile.

Un bouton termine l’UI, à chaque clic une nouvelle notification est envoyée à la tuile de l’application (basée sur le texte saisi par l’utilisateur donc).

Ce qui donne quelque chose de ce genre :



Je vais avoir du mal à vous montrer l'effet final, en effet tout se joue dans le menu Windows 8 alors que le développement s'effectue en bureau classique d'une part et que, d'autre part, Windows 8 est "full screen" et pour voir l'effet du clic sur le bouton il faudra retourner sur le menu Windows. Enfin l'effet est une animation qui réclamerait une capture écran en vidéo.

Le mieux sera que vous jouiez avec l'application en créant des notifications tout en switchant sur le menu Windows 8 pour retrouver l'application et regarder sa tuile s'animer.

Notification simple par code

La première option prise en charge par la démonstration est la création d'une notification textuelle simple.

```
private void simpleNotification()
        {
            // création d'une tuile large + tuile carrée
            var contenu = TileContentFactory.CreateTileWideText03();
            contenu.TextHeadingWrap.Text = txtNotification.Text;
            var carré = TileContentFactory.CreateTileSquareText04();
            carré.TextBodyWrap.Text = txtNotification.Text;
            // le design carré est attaché à la tuile large
            contenu.SquareContent = carré;
            // création de l'objet de notification
            var notification = contenu.CreateNotification();
            // création de l'objet updater
            var updater = TileUpdateManager.CreateTileUpdaterForApplication();
            updater.Update(notification);
        }
```

Le mécanisme peut sembler compliqué, en fait il l'est :-) Pour une fonction si rudimentaire de l'OS je m'attendais à quelque chose de plus direct. Non seulement il faut décomposer toute l'action mais pour la rendre "humainement supportable" il faut intégrer le fameux projet d'extensions.

Mais c'est comme tout, une fois qu'on connait le code à appliquer c'est très simple, évidemment…

L'action est commentée dans le source ci-dessus, je ne me répèterai donc pas.

A noter qu'on créée une tuile (large par défaut) et qu'on créée une seconde tuile carrée qui est raccrochée à la première. On notera aussi que l'on formate les tuiles en utilisant les styles évoqués plus haut, ici sous la forme d'appels de méthodes. Enfin,

les plus perspicaces auront compris qu'il faut faire le formatage deux fois de suite, pour chaque forme de la tuile (large ou carrée), ce qui est encore un peu plus lourd.

Une API efficace aurait été une classe avec une méthode statique à laquelle on aurait passé le texte à afficher et le nom du style et qui se serait débrouillé du reste. Rien n'interdit d'écrire votre propre extension au-dessus des extensions !

Bien entendu la décomposition possède aussi son avantage : celui de pouvoir intervenir différemment à chaque étape. Par exemple nous créons ici un objet updater à chaque clic, il serait certainement plus subtil de conserver l'adresse du premier et de s'en resservir. Tout comme l'objet de notification. Ce qui prend son sens lorsque l'application doit effectuer des notifications régulièrement.

Le code exemple ci-dessus montre le mécanisme, pas forcément la façon la plus optimisée d'utiliser l'API dans un contexte de production. Démo pour comprendre et production optimisée sont deux objectifs distincts, je ne vise le plus souvent dans Dot.Blog que le premier.

Notification par XML

L'API nous offre deux méthodes pour créer des tuiles vivantes. Chacun utilisera l'approche qui lui convient le mieux.

La première est entièrement par code comme nous venons de le voir.

La seconde déplace tout le formatage de la tuile vers une description XML, et c'est ce "bout" de XML qu'on passe à l'API.

Il y a des avantages aux deux approches, l'un de ceux du mécanisme par XML est qu'il s'agit d'une chaîne de caractères qui peut être construite à la volée dans l'application, relue depuis un fichier de configuration, personnalisée par l'utilisateur ou un outil de configuration par exemple, etc.

Le désavantage principal est qu'on passe un objet string non validé à la compilation. Et tout ce qui nous écarte d'un langage compilé fortement typé est risqué.

Bref, code ou XML, chacun choisira en fonction du contexte et de ce que cela peut lui apporter mais surtout en connaissance de cause et en assumant les risques et désavantages de chaque méthode. Mon rôle n'est pas de censurer mais de vous prévenir. A vous de voir !

Voici exactement le même code mais utilisant la définition de la tuile en XML :

```
private void xmlNotification()
        {

            string xml = "<tile>"
                            + "<visual>"
                            + "<binding template='TileWideText03'>"
                            + "<text id='1'>"+txtNotification.Text+"</text>"
                            + "</binding>"
                            + "<binding template='TileSquareText04'>"
                            + "<text id='1'>" + txtNotification.Text + "</text>"
                            + "</binding>"
                            + "</visual>"
                            + "</tile>";

            var tuileDom = new XmlDocument();
            tuileDom.LoadXml(xml);
            var notification = new TileNotification(tuileDom);
            TileUpdateManager.CreateTileUpdaterForApplication().Update(notification);
        }
```

Côté mécanisme de notification c'est beaucoup plus simple (les deux dernières lignes). Côté XML ce n'est pas trop lourd non plus (deux lignes qui pourraient être écrites en une seule).

C'est donc plus léger, sans aucun doute.

Reste à définir le XML. Dans un cas comme notre exemple on s'aperçoit que cela revient à écrire autant de lignes. Dans le premier cas (par code) tout sera contrôlé à la compilation, dans le second cas ça plantera à l'exécution. Et croyez-moi, quand j'ai écrit cet exemple il a fallu que je m'y prenne à trois ou quatre fois avec plantage et exception incompréhensible (un gros code hexadécimal…) pour corriger le XML dans lequel j'avais introduit quelques petites coquilles… CQFD.

Plus prosaïquement le lecteur attentif reconnaîtra dans le code XML l'appel aux styles discutés plus haut. Par exemple `TileSquareText04` utilisé ici comme le nom d'un template auquel il faut se binder en place et lieu d'une méthode de même nom dans la version par code.

Binder ? c'est du Xml ou du Xaml ? … non c'est bien du Xml qui emprunte sa syntaxe à Xaml qui lui même est une extension construite sur Xml. Un peut tordu mais c'est ici l'effet de la lutte intestine qui s'est jouée chez MS entre les équipes Xaml et Sinofsky. Ce dernier a gagné le contrôle des opérations, il a pillé les bonnes idées de ces concurrents déchus car, au final, il n'y a pas mieux que Xaml :-)

Le code du projet

Amusez-vous avez le code, testez d'autre styles de tuile, c'est amusant (un peu fastidieux à tester malgré tout avec les allers-retours bureau classique / menu Windows 8 et retrouver l'application de test qui est tout au bout de ce long menu horizontal…).

Et pour éviter la fatigue d'une frappe sans grand intérêt, voici le projet Visual Studio 2012. Il faudra bien entendu utiliser VS 2012 sous Windows 8 et non sous Windows 7…

Solution exemple VS 2012 TuilesVivantes

Attention : ce projet est directement testable sous Wndows 8 / VS 2013, si vous l'utilisez sous Windows 8.1 / VS 2013 une phase de mise à niveau sera nécessaire (généralement automatique et sans intervention).

Conclusion

La critique est facile, l'art est plus difficile, c'est bien connu.

Sinosfky n'a jamais aimé Silverlight ni WPF, et même viré de MS il nous le fait bien sentir depuis des années, il est légitime de lui rendre la monnaie de sa pièce par quelques remarques acerbes… Cela étant dit, Windows 8 est une belle mécanique, puissante, agréable, rapide, et jouer avec les tuiles est divertissant. De ce côté là Steven il a fait son "job" correctement (!).

Reste à savoir si les tuiles sauront convaincre le monde entier d'abandonner la métaphore du bureau. Plus d'un an après avoir ce texte et avec le succès mitigé de Windows 8 derrière nous et l'adoption très calme de 8.1, on peut d'ores et déjà répondre que sur le fond mon questionnement était plus que justifié… Ah si Microsoft écoutait plus ses MVP…

Notre rôle à nous n'est pas de juger, mais d'être prêts dans tous les cas de figure puisque nous n'avons que peu pas du tout la main sur les décisions stratégiques. Refaire le match, même au foot ça m'énerve (le foot m'énerve à la base ce n'est pas un bon exemple !). Donc notre job d'ingénieur c'est de prendre les choses avec philosophie et de faire avec ce qu'on a en en tirant le meilleur parti. Ça n'empêche pas d'avoir son avis !

Dot.Blog participe à cette veille technologique et à cette diffusion indispensable de l'information technique. Succès ou pas, l'avenir du produit n'est pas entre nos mains

mais dans celles des utilisateurs. Soyons prêts à leur donner ce qu'ils aiment, WPF, Silverlight ou WinRT ou même MonoDroid, grâce à Dot.Blog vous serez prêt !

WinRT : Prendre des photos

L'avantage d'un OS PC conçu pour marcher aussi sur tablettes et smartphones est qu'il est obligé de fournir les fonctions qu'on attend de ces systèmes, comme prendre des vidéos ou des photos. Windows 8 se plie donc à l'exercice là où d'excellents OS comme Windows 7 ou même XP ne proposaient que du bas niveau. Voyons comment en tirer partie…

Principes

WinRT propose une API riche et complexe, bien plus grosse que .NET complet qui était déjà plutôt "joufflu" ! Bien entendu cette prise de poids de l'API n'est pas un hasard ni un manque de maîtrise de la part de Microsoft. Au contraire, et on le voit bien à la vitesse Windows 8, cet OS est conçu pour aller vite, même sur des machines modestes.

L'une des raisons de ce gonflement de l'API est plutôt lié à toutes les choses que Windows ignorait superbement et qui, aujourd'hui, font partie du minimum syndical même sur un smartphone… géolocalisation, partage, vidéo, photo, et bien d'autres choses doivent être gérées proprement par l'OS et non plus laissées au bon vouloir de chaque logiciel et des implémentations de niveau divers que cela suppose (incompatibles entre elles de plus).

Windows 8, le fédérateur, se devait dans sa partie WinRT, celle qui est justement portable sur tous les form factors, d'intégrer l'ensemble des API permettant à toute application d'offrir des services riches, identiques et compatibles.

De fait, l'API dédiée à la gestion des médias dans WinRT est très riche et prend en considération de nombreux paramètres.

La version simple

Dans sa version la plus simple l'API photo de WinRT se résume à quelques appels rudimentaires. Voici un squelette de code n'utilisant que le strict nécessaire :

```
private async void CapturePicture()
{
    var dialog = new CameraCaptureUI();
    var file = await dialog.CaptureFileAsync(CameraCaptureUIMode.Photo);
    if (file != null)
    {
    // utiliser le fichier …
    }
}
```

Autoriser le crop

Bien que simple, cette approche peut être facilement agrémentée d'options, comme autoriser l'utilisateur à cropper l'image capturée :

```
private async void CapturePicture()
{
    var dialog = new CameraCaptureUI()
                    { PhotoSettings.AllowCropping = true };
    var file = await dialog.CaptureFileAsync(CameraCaptureUIMode.Photo);
    if (file != null)
    {
    // utiliser le fichier...
    }
}
```

Comme on le voit l'accès aux paramètres est assez direct notamment par le biais de la propriété `PhotoSettings` dont le nom est explicite. De même la nature de la capture se fixe lors de l'appel à `CaptureFileAsync`. Ci-dessus nous indiquons un mode de capture Photo. Le même mécanisme permet de demander la capture d'un flux audio ou d'une vidéo.

Bien que sophistiqué WinRT reste compréhensible.

Une application de capture

Pour illustrer les mécanismes expliqués plus haut et quelques autres, le mieux est de créer une application Windows Store.

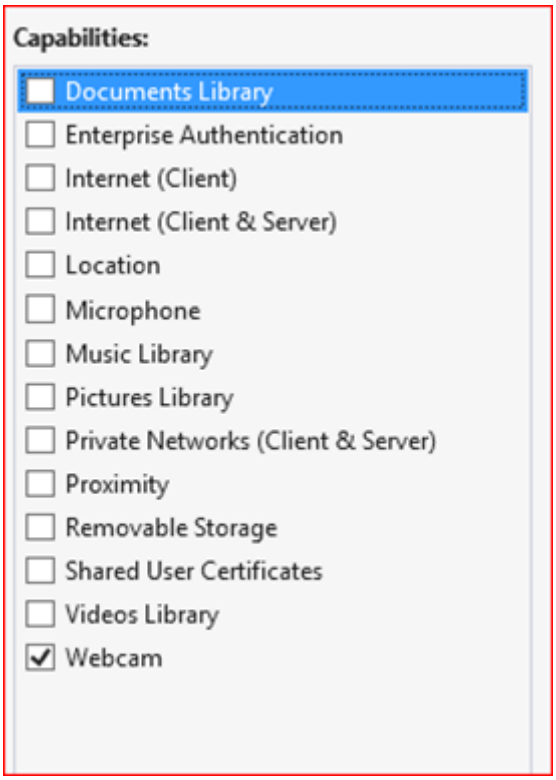
Je vais vous montrer comment énumérer toutes les devices gérant des médias, comment utiliser le File Picker pour sélectionner des photos et comment, bien entendu, prendre une photo.

L'application part d'un template "`Blank`" pour Windows Store. On peut diviser visuellement l'application en trois parties verticales, à droite la liste des devices, au

centre le système de capture, à droite l'utilisation du `File Picker`. Je dis ça pour ceux qui joueront avec le code (disponible en fin de billet).

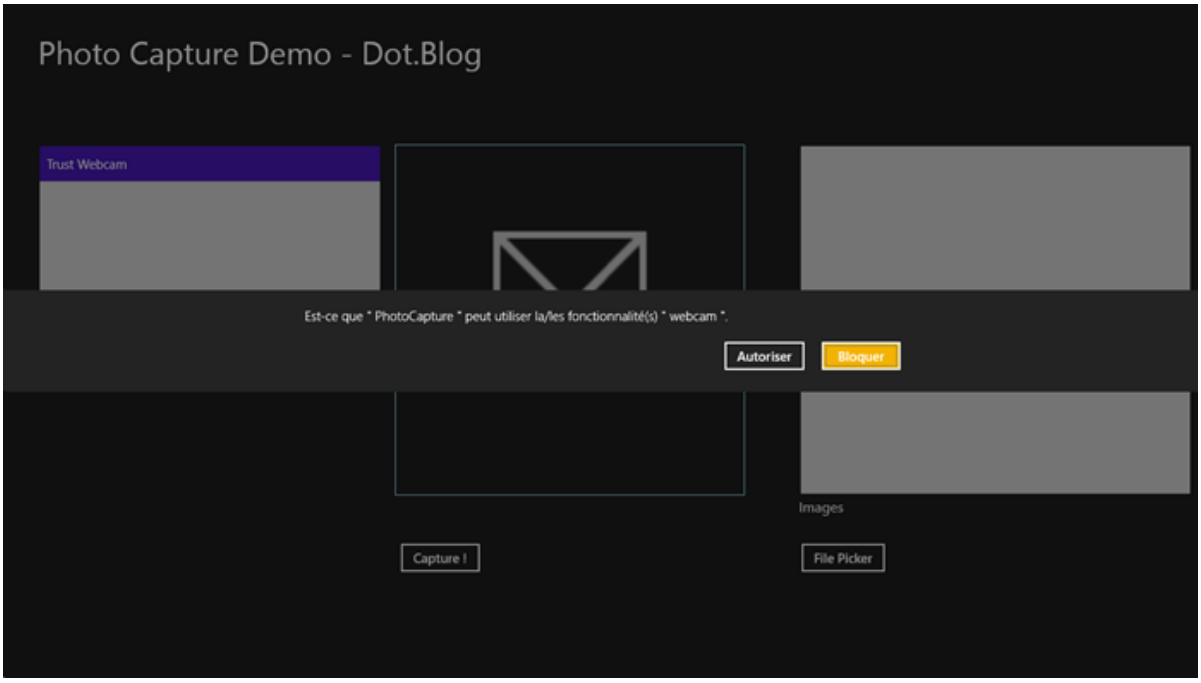
Autoriser l'accès à la webcam

Sécurité oblige, si une application veut utiliser la webcam elle doit le déclarer dans son manifeste. Et l'utilisateur sera tenu au courant et devra valider cette autorisation.



N'oubliez pas ce petit détail si vous ne voulez pas chercher inutilement un bug qui n'existe pas…

Au premier lancement de l'application, l'utilisateur verra apparaître un message de confirmation pour l'accès à la webcam :



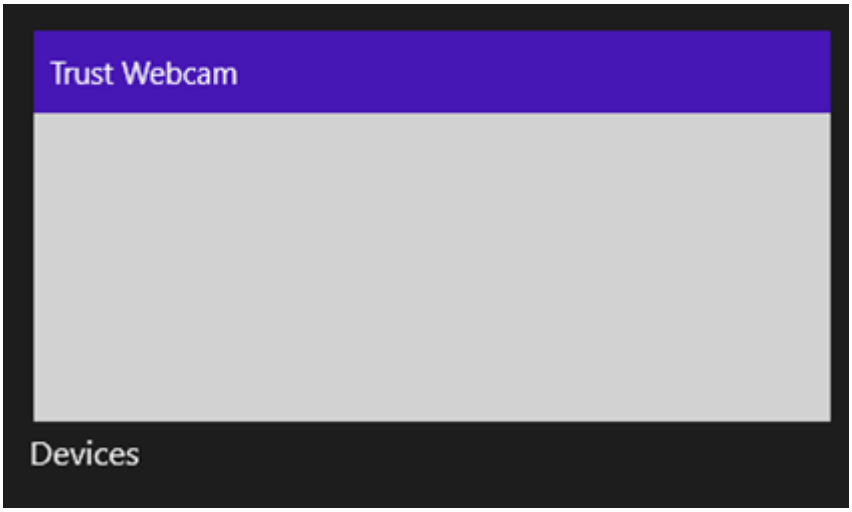
Si l'utilisateur refuse l'autorisation il faudra veiller à le tester pour éviter un plantage (certains se diront peut-être qu'à ce stade, vu le peu de confiance et d'intérêt que marque l'utilisateur, autant laisser l'application se planter, hein… ce qui n'est pas jamais un bon calcul. Même éconduit, restez polis, parfois cela ne sert à rien, mais parfois cela fait revenir les gens sur leur décision !).

Lister les devices

L'affichage est effectué par une simple `ListBox` dont le contenu est créé au lancement de l'application. La méthode "`getAllDevices()`" est appelée dans le constructeur de la page :

```
private async void getAllMedia()
        {
            try
            {
                devices = await
                 DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);
                if (devices == null || devices.Count == 0)
                {
                    await new
                      Windows.UI.Popups.MessageDialog(
                         "Aucune Webcam trouvée. Désolé.").ShowAsync();
                }
                else
                {
                    foreach (var device in devices)
                    {
                        lstDevices.Items.Add(device.Name);
                    }
                    lstDevices.SelectedIndex = 0;
                }
            } catch(Exception e)
            {
                new Windows.UI.Popups.MessageDialog(e.Message).ShowAsync();
            }
        }
```

Dans ce code totalement asynchrone (mais qui grâce à `async`/`await` s'écrit presque comme un code synchrone) j'énumère l'ensemble des périphériques ayant la possibilité de prendre des photos ou de capturer des flux vidéo. Il n'y a en fait que l'appel à `DeviceInformation.FindAllAsync` qui est intéressant. Le reste est de l'habillage pour gérer les éventuelles exceptions et pour remplir la `ListBox` avec les noms des périphériques trouvés.



Sur cette machine j'ai une WebCam Trust, on la voit apparaître dans la liste…

L'intérêt d'énumérer les périphériques est essentiel dans des conditions où plusieurs de ceux-ci sont présents pour une même fonction. Supposons une tablette Surface possédant deux capteurs vidéos (frontal et arrière), il est important de laisser le choix du capteur à utiliser à l'utilisateur. Mais sur un PC il peut se faire qu'il n'y ait aucune webcam, il faut le gérer aussi.

De même un microphone de bonne qualité peut être connecté à la machine et l'utilisateur préfèrera peut-être utiliser ce dernier pour enregistrer un flux audio que le microphone intégré à sa webcam.

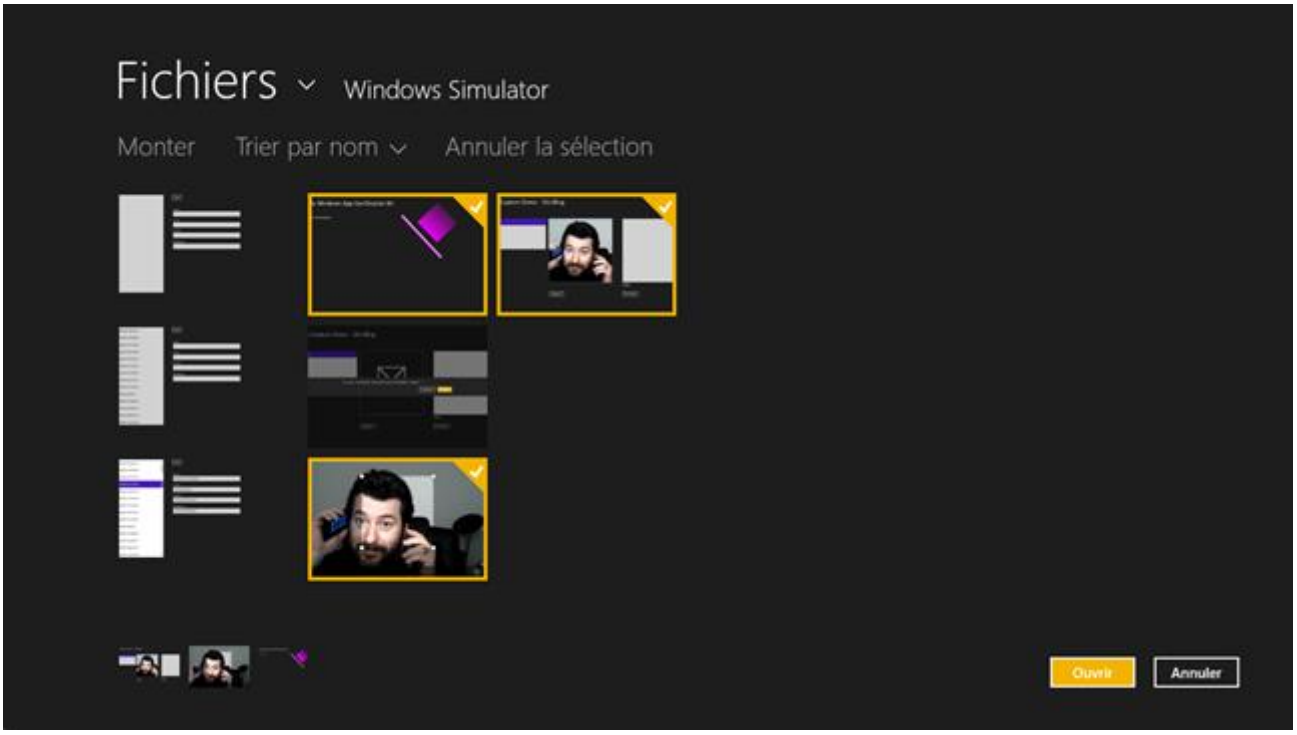
Dans toutes ces circonstances détecter les périphériques assurant certaines fonctions (comme ici la vidéo) peut faire la différence entre un bon logiciel bien fini et un soft bâclé…

Dans l'application exemple, cette liste ne sert à rien d'autre qu'à la démonstration. Dans une application réelle elle permettrait donc de choisir le périphérique le mieux adapté ou à laisser l'utilisateur faire ce choix.

Utiliser le File Picker

Le `File Picker` est un outil puissant, bien plus complet et versatile que les dialogues d'ouverture de fichier des versions précédentes de Windows. De plus son UI a été entièrement pensée pour s'adapter au mode Full Screen de Metro et au tactile.

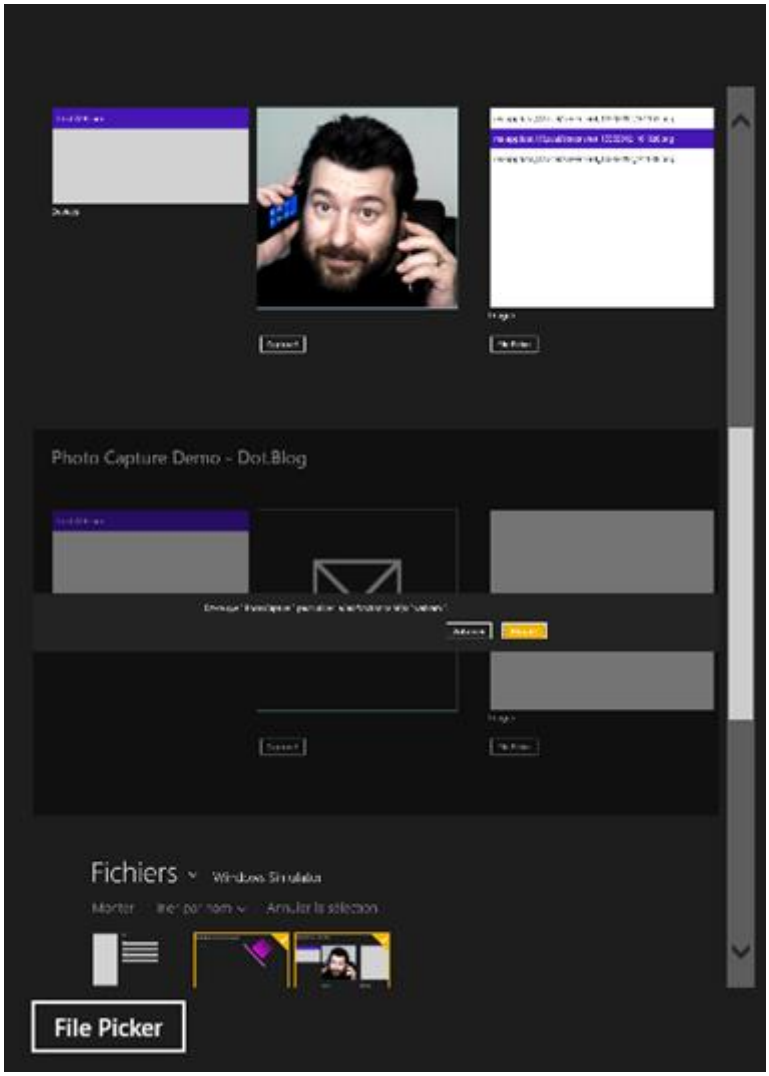
Je suis navré pour le choix des images, je n'avais que ma trombine à prendre en photo, je vous aurais bien mis des clichés de jeunes et jolies demoiselles dénudées pour maintenir en éveil votre intérêt, mais je n'ai pas de nymphettes qui dansent nues dans mon bureau prêtes à être photographiées. Sachez que je suis le premier à le regretter sincèrement, mais il faudra vous satisfaire de ma barbe de trois semaines !



Ci-dessus une capture dans le simulateur du `File Picker` en action. De nombreuses options sont disponibles à l'utilisateur, la sélection est simple et matérialisée de façon homogène avec l'ensemble de l'OS (les photos sélectionnées sont entourées d'un rectangle de couleur et le coin supérieur droit est marqué).

L'invocation du File Picker est largement paramétrable. Ici j'ai utilisé le `ViewMode` pour forcer un affichage de type `Thumbnail`, on aurait pu choisir une simple liste. De même j'ai initialisé `SuggestedStartLocation` pour que le dialogue s'ouvre sur un répertoire particulier (ici la librairie d'images de l'utilisateur en cours). Enfin, en précisant des `FileTypeFilter` j'ai limité l'affichage aux fichiers "`jpeg`" et "`png`".

Une fois la sélection effectuée par l'utilisateur je remplie une liste qui contient des URL que je fabrique au passage. Le tout est affiché par une `ListView` possédant un petit `ItemTemplate` qui utilise un composant `Image` et un peu de binding pour afficher l'image :



On voit ci-dessus les images sélectionnées apparaître dans la ListView templatée (ainsi que le bouton “File Picker” permettant de tester la fonction).

Vous noterez que si vous n’avez pas besoin d’une sélection multiple il est préférable d’utiliser `PickSingleFileAsync` au lieu de `PickMultipleFileAsync` comme je le fais dans cette démo. C’est une évidence mais c’est mieux de le dire !

Le File Picker est simple à utiliser, le code ci-dessous le montre. Simple mais pas forcément évident. En effet, lorsque l’utilisateur valide sa sélection on ne reçoit bien entendu pas une liste d’objets bitmap… Mais à la place notre code reçoit une `IReadOnlyList` de `StorageFile`.

Pour afficher (ou utiliser autrement) la ou les images retournées on peut obtenir un `Stream` sur chaque fichier et l’exploiter à sa guise (pour le transférer dans un bitmap qu’on affichera, pour le copier ou n’importe quoi d’autre).

Ici j'en profite pour vous faire voir une autre possibilité : copier dans l'espace local de l'application les images ce qui nous permet ensuite d'utiliser des URL directes pour les afficher. Ces URL sont construites de la façon suivante "`ms-appdata:///Local/filename`" où "*filename*" est le nom du fichier.

L'intérêt est d'en profiter pour parler de cette URL et de la façon de la construire et aussi de montrer la fonction de copie de fichiers.

Le code du clic sur le bouton File Picker est le suivant :

```
private async void btnFilePicker Click(object sender, RoutedEventArgs e)
{
     var openPicker = new FileOpenPicker
                    {
                        ViewMode = PickerViewMode.Thumbnail,
                         SuggestedStartLocation =
                               PickerLocationId.PicturesLibrary
                     };
     openPicker.FileTypeFilter.Add(".jpg");
     openPicker.FileTypeFilter.Add(".jpeg");
     openPicker.FileTypeFilter.Add(".png");
     var files = await openPicker.PickMultipleFilesAsync();
      if (files.Count == 0)
            {
                await new Windows.UI.Popups.MessageDialog(
                              "Aucun fichier sélectionné !").ShowAsync();
                return;
            }

     var fileStorage = Windows.Storage.ApplicationData.Current.LocalFolder;
      foreach (var file in files)
            {
                var copiedFile = await
                            file.CopyAsync(fileStorage, file.Name,

NameCollisionOption.GenerateUniqueName);
                lstImages.Items.Add(string.Format(
                      "ms-appdata:///Local/{0}", copiedFile.Name));
            }
 }
```

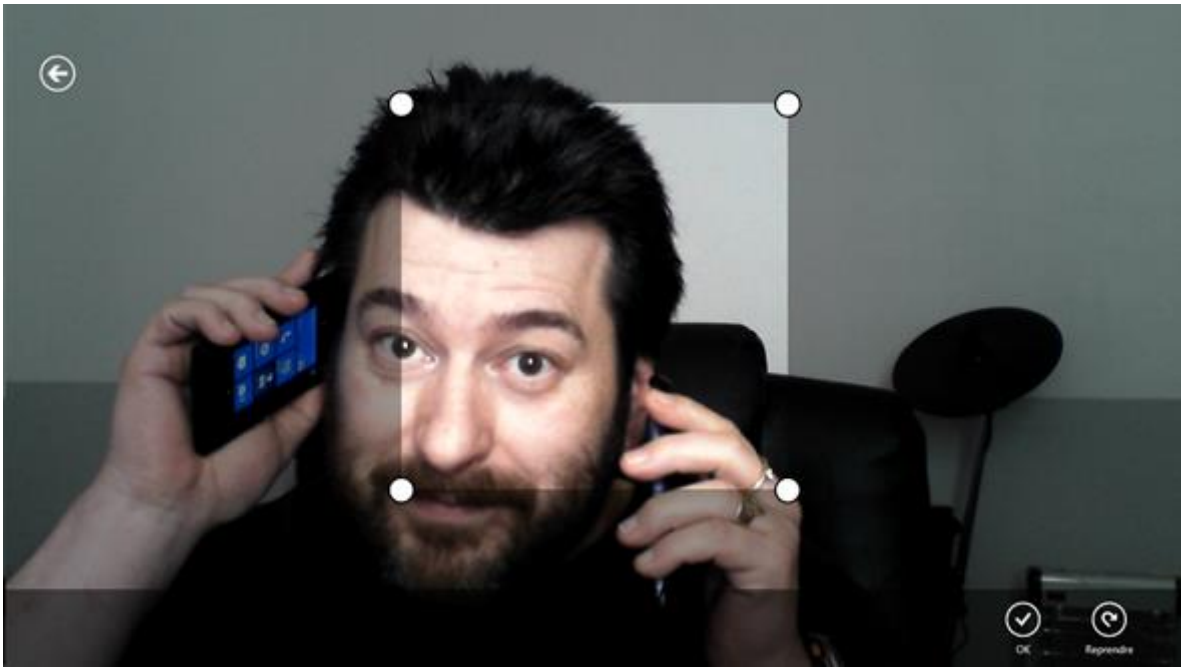
Rien de bien sorcier donc.

La ListView est templatée pour afficher l'image à partir de l'URL fabriquée par le code ci-dessus, le template est minimaliste :

```
<ListView x:Name="lstImages">
   <ListView.ItemTemplate>
       <DataTemplate>
           <Image Source="{Binding}" Stretch="UniformToFill"/>
       </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Prendre une photo

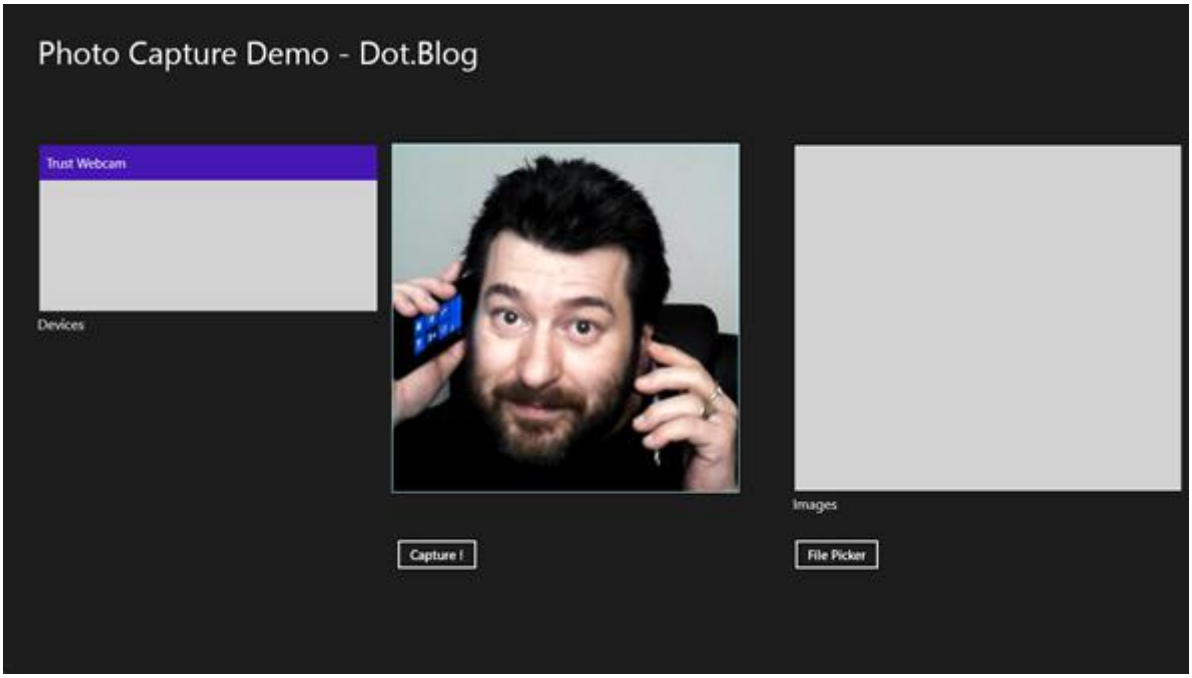
Il faut bien y venir, ce coup-ci ma barbe est incontournable et plein cadre ! J'ai agrémenté les clichés d'un Nokia sous Windows Phone pour que ça fasse plus gai… (Dans l'autre main c'est un Samsung SIII, et pour les curieux le truc rond derrière ce n'est pas une assiette de jongleur sur un bâton mais l'une des cymbales de ma batterie électronique, à cette place c'est la Ride pour les connaisseurs…).



Une fois abstraction faite du modèle qui a posé pour le cliché, on remarque que WinRT propose de nombreuses choses à l'utilisateur comme de revenir en arrière (annulation de la prise de vue), un rectangle de crop pour recadrer le cliché, et deux boutons, l'un pour valider, l'autre pour refaire le cliché.

C'est très complet. Mais certaines fonctions sont paramétrables. Le crop par exemple n'est là que parce que le code l'a demandé. D'autres comme le timer sont proposées par défaut, ce qui est bien pratique lorsqu'on doit par exemple faire l'imbécile avec un téléphone dans chaque main pour un cliché d'illustration d'une démo sur son blog (comment j'aurai pu déclencher la prise de vue sinon ?…).

Une fois validé, le cliché apparait dans l'application (avec le crop appliqué) :



Le code qui a servi à immortaliser cette scène est le suivant :

```
private async void btnCapture_Click(object sender, RoutedEventArgs e)
{
    try
    {
       var dialog = new CameraCaptureUI();
       dialog.PhotoSettings.CroppedAspectRatio = new Size(1, 1);
       dialog.PhotoSettings.Format = CameraCaptureUIPhotoFormat.Png;

       var file = await dialog.CaptureFileAsync(CameraCaptureUIMode.Photo);
       if (file != null)
          {
              var stream = await
                  file.OpenAsync(Windows.Storage.FileAccessMode.Read);
              var bmp = new BitmapImage();
              bmp.SetSource(stream);
              imgViewer.Source = bmp;
              stream.Dispose();
          }
     } catch(Exception ee)
          {
            new Windows.UI.Popups.MessageDialog(ee.Message).ShowAsync();
          }
}
```

Je n'insisterai pas sur la nature asynchrone de tout ce code, sur le rôle de `async` et `await`, mais cela ne veut pas dire qu'il n'y a rien à voir !

Le mécanisme de la prise de vue est assez simple à comprendre : `CaptureFileAsync` effectue la capture et retourne le fichier qui a été créé. Il suffit de créer un `Stream`

pour alimenter un `Bitmap` avec le contenu du fichier puis d'utiliser ce dernier comme source de l'objet Image de l'UI pour faire apparaître la photo.

Conclusion

La prise de vue sur un PC est un gadget parfois utile, mais sur une tablette ou un smartphone cela devient une fonction incontournable riche en possibilités ('interprétation de code barre, de QR code, détection de mouvement, etc).

Les API mises à disposition par WinRT sont assez simples mais très complètes. Il faut toutefois se méfier de l'asynchronisme et écrire son code en en tenant compte…

Ce billet ne montre qu'une partie des possibilités, prendre des videos, enregistrer uniquement du son peuvent s'avérer tout aussi utile. Néanmoins les mécanismes restent les mêmes et le lecteur intéressés possèdera les clés pour aller fouiller la documentation Microsoft !

Windows Store : créer un package et le valider avec le Windows App Certification Kit

Dans cette série dédiée à WinRT j'ai déjà eu l'occasion de vous présenter de nombreux aspects de la programmation sous ce nouvel environnement. La création d'un package de déploiement et sa validation avec le Certification Kit sont deux étapes cruciales avant l'accès au saint Graal : le Windows Store…

La validation étape obligatoire

Ce qui nous intéresse ici se résume à une chose : pouvoir publier une application sur le Windows Store. Pour y parvenir il y a toute une série d'étapes à franchir. Celle de l'idée, de sa mise en forme, celle de la réalisation, des tests, etc. Mais en bout de course il reste un obstacle : que l'application soit acceptée par le Windows Store qui, vous le savez certainement, utilise un processus de validation assez complexe et très strict. Louper cette étape c'est être obligé de repasser le cycle de validation ce qui a un coût au moins en tests et en temps perdu de présence sur le market place.

Le Windows App Certification Kit

Heureusement, Microsoft a eu l'excellente idée de nous fournir le **WACK**, le Certification Kit du Windows Store, un automate de même nature que ceux utilisés

par Microsoft pour valider automatiquement les applications soumises au Windows Store. Le WACK ne teste pas tout, et certains aspects de la validation définitive restent dans les mains de Microsoft qui prend en compte des critères supplémentaires.

Toutefois ce qu'il faut en retenir est assez simple : si votre application est validée par le WACK elle peut néanmoins être rejetée par le Windows Store mais en revanche publier directement une application sans la passer sous l'œil inquisiteur du WACK est une erreur et une grosse prise de risque.

Packager pour publier, et aussi pour tester

Pour publier une application il faut la packager par un processus particulier disponible dans VS 2013. Cela est indispensable pour la soumettre au Windows Store ou même la déployer en entreprise.

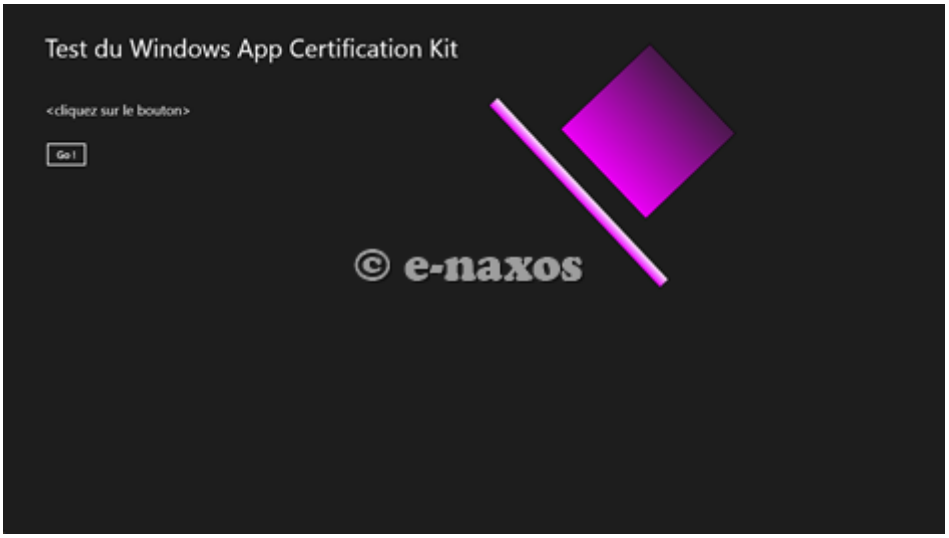
Mais cette étape s'avère tout aussi indispensable pour tester l'application avec le WACK qui tente de simuler au plus près la validation réelle effectuée par Microsoft.

Construire un package

L'application

Je ne vais pas me lancer dans un grand développement ici, je vais créer une application de base en partant du modèle "`blank`" et lui ajouter un bouton, un évènement qui changera le contenu d'un `TextBlock` ainsi qu'une ou deux figures pour donner une illusion de contenu. On pourrait fort bien partir de l'application "`blank`" sans rien ajouter ou bien d'une application complète ou en cours de finition. Dans ce dernier cas cela permet de détecter au plus vite les éventuels problèmes que posera la certification et d'y remédier rapidement.

Donc ici nous supposons que nous disposons d'une application Windows Store, je ne détaillerai pas cette étape.



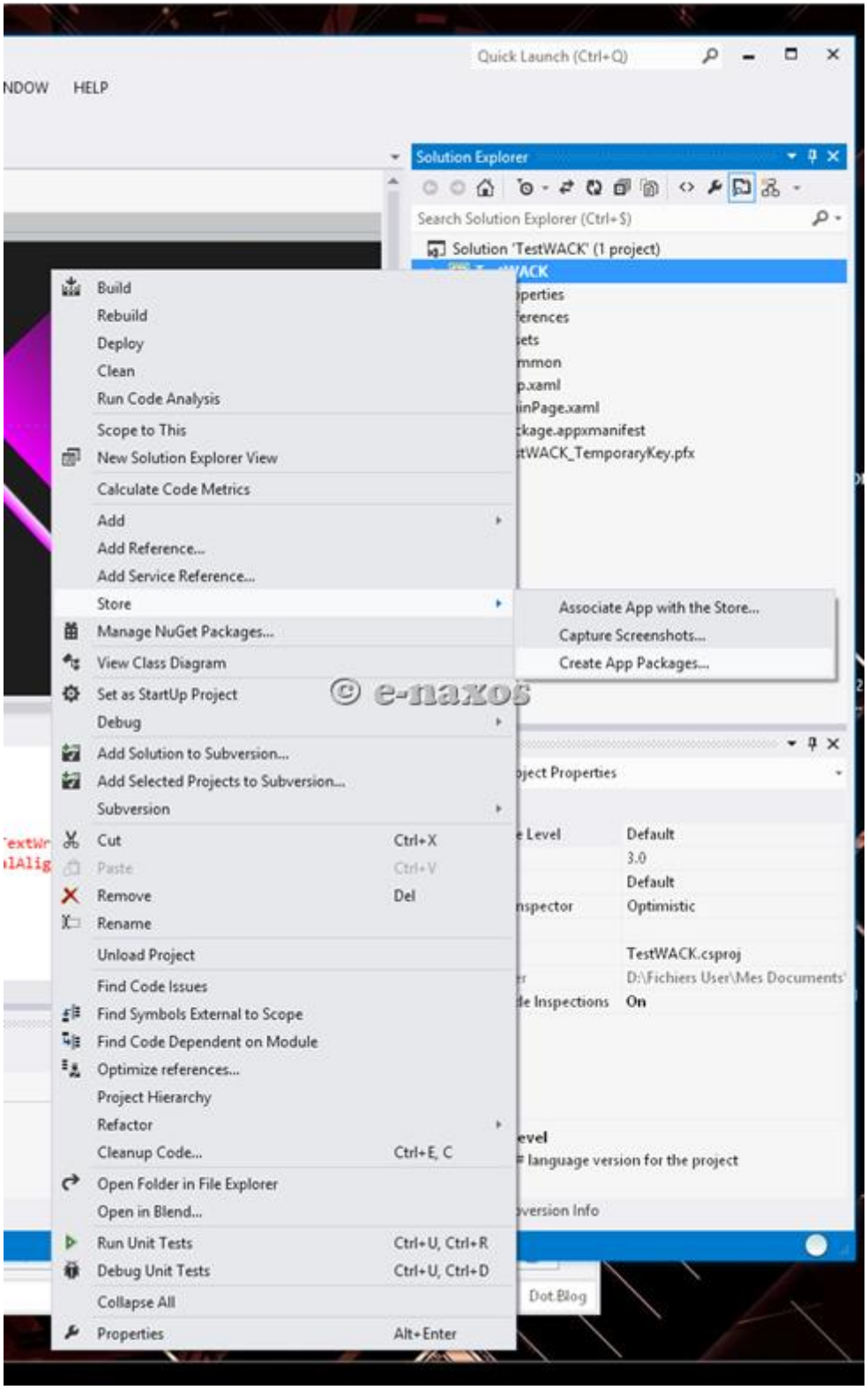
Comme on le voit sur la capture ci-dessus l'application ne fait pas grand chose, un clic sur le bouton "Go !" affiche la date et l'heure. Deux rectangles et un titre sont là pour donner une vague impression de contenu…

Le package

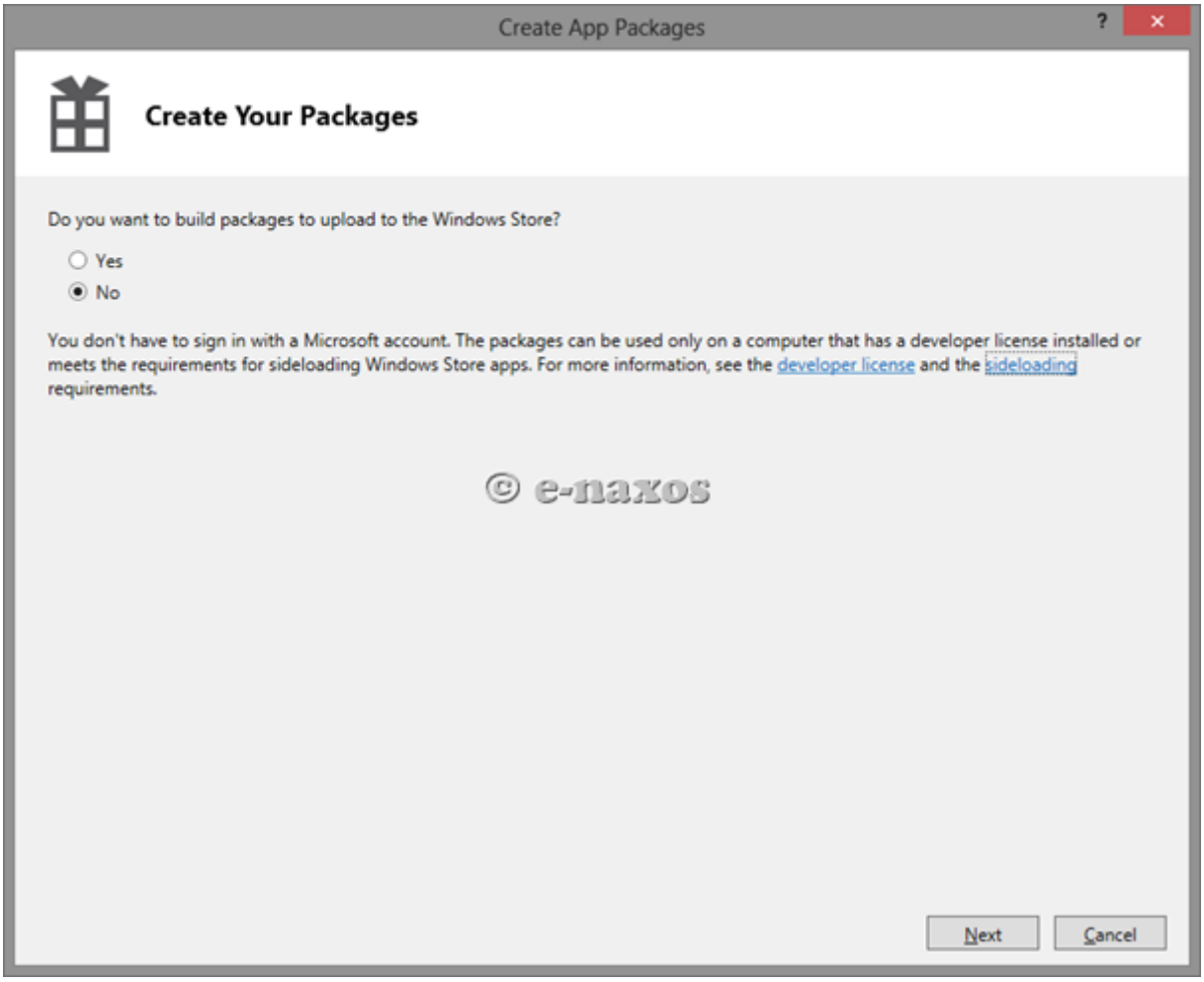
Ce qui nous intéresse vraiment ici n'est pas l'application elle-même mais de savoir comment créer un package de déploiement.

Et cela est très simple :

La première étape consiste à faire un clic droit sur le projet dans l'explorateur de solution et de sélectionner "`Create App Package`" dans l'option "`Store`" :



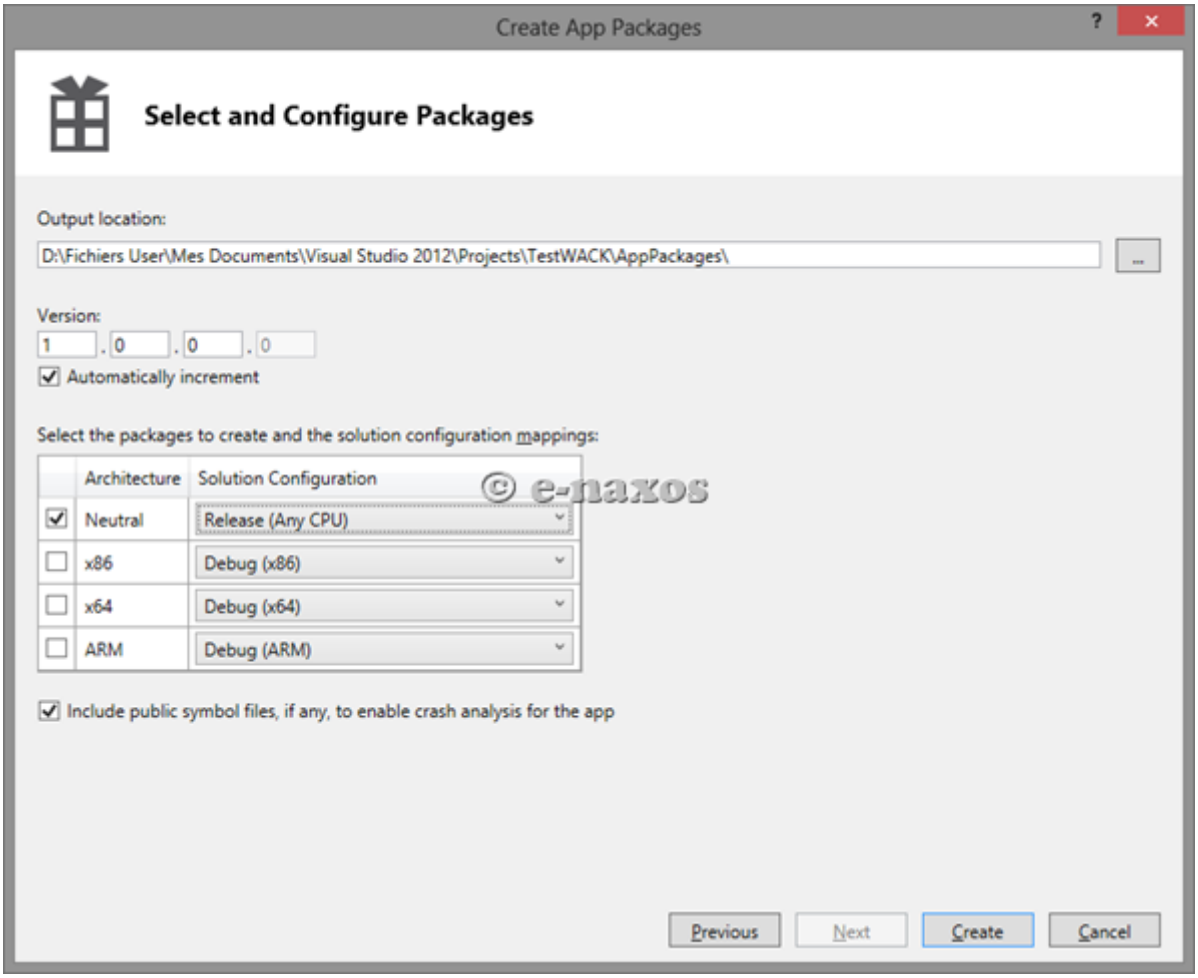
Ce qui va ouvrir un dialogue :



Ce dialogue vous demande si vous souhaitez diffusez l'application sur le Windows Store. La réponse est non ici, nous utiliserons le package en local uniquement pour les tests avec le WACK.

Certains liens proposés par cet écran sont de la plus grande importance, vous noterez notamment celui qui pointe vers le "**sideloading**" qui explique le mécanisme de déploiement d'une application WinRT en entreprise sans passer par le Windows Store. Une option dont la présence ou l'absence avait fait couler beaucoup d'encre au lancement de Wnidows 8. Windows 8.1 permet aussi désormais de créer son propre « store » dans l'entreprise pour diffuser les applications « maison ».

Une fois le choix '"no" effectué, il faut cliquer sur "next", ce qui amène un nouveau dialogue :



Les informations ne sont pas nombreuses mais essentielles :

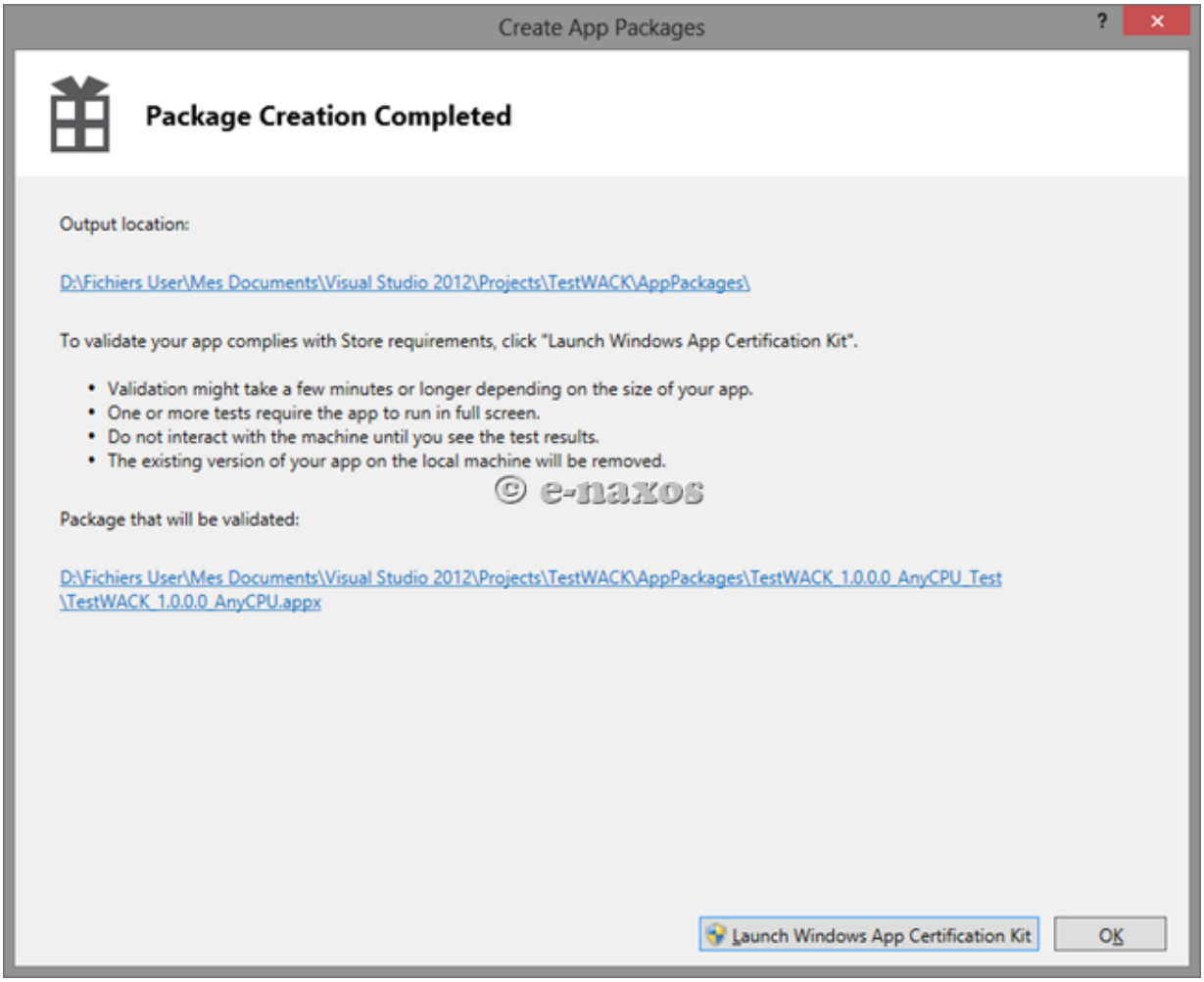
- Le choix de l'emplacement de sortie du package
- Le choix de la version de l'application avec auto incrémentation ou non
- Le choix du type de cible à tester (Neutre, x86, x64 ou ARM). Pour chaque cible il faut choisir une configuration de Build. Il doit s'agir de construction de Release et non de debug.

Enfin, le choix par une case à cocher d'inclure ou non les fichiers de symboles.

Je vous conseille de conserver certains de réglages par défaut comme l'auto incrémentation des versions et l'inclusion des fichiers de symboles.

Ne reste plus qu'à cliquer sur "Create"…

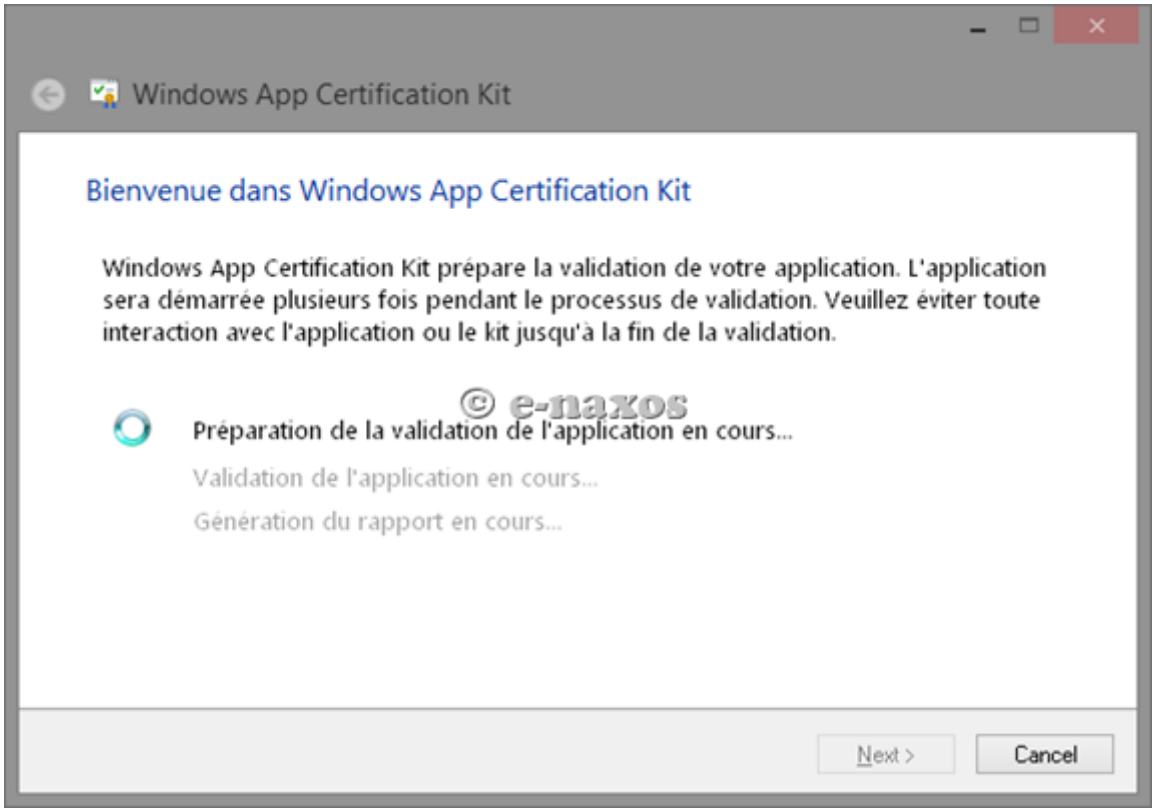
Ce qui se conclura, après une compilation réussie, par le dialogue suivant :



Il nous est rappelé, avec un lien cliquable ce qui est pratique, où se trouve le répertoire de stockage du package. Dans le même esprit le package qui va être testé est lui aussi indiqué par un lien qu'il suffit de suivre pour y accéder physiquement sur le disque.

Le dialogue permet surtout de lancer le WACK…

La validation par le WACK



Une fois le lancement du WACK validé dans le dialogue précédent le processus commence. Il se déroule en trois étapes :

La préparation

Le WACK commence par vérifier la présence de mises à jour sur le Web afin d'être le plus fidèle possible aux tests effectués sur le Windows Store. Il prépare les fichiers et passe à l'étape suivante. Cette première phase n'est pas instantanée même en l'absence de mise à jour. Il faut être patient et ne pas jouer avec l'ordinateur durant toute la phase de validation.

La validation de l'application

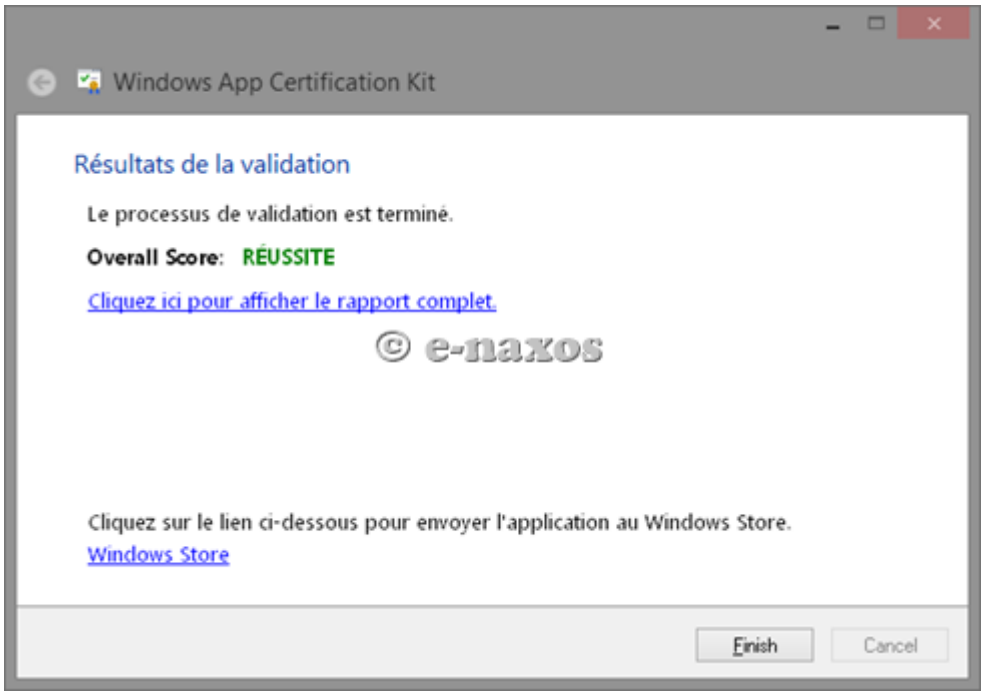
C'est la phase cruciale. Le WACK va lancer plusieurs fois votre application en plein écran, l'arrêter, réfléchir, etc. J'ai compté pour ce test au moins 6 lancements et arrêts de l'application. Sur ma machine de compétition cela a duré plusieurs minutes. Si vous disposez d'une machine plus standard vous aurez le temps d'aller faire un tour à la machine à café, surtout si l'application est plus grosse que l'exemple minimaliste que j'utilise ici.

Surtout ne rien toucher, ne pas tenter d'interagir avec l'application ni même avec le PC durant la validation. La présence d'autres applications exécutées en même temps ne gêne pas le processus mais pourrait le perturber et au moins le ralentir. Si votre

machine n'est pas du genre monstre, je vous conseille de fermer toutes les applis et de décharger le PC des tâches non essentielles durant la validation.

Le résultat

En fin de validation le couperet tombe ! ça passe ou non…



Ici - et bien heureusement vu qu'il s'agit à 99% du template "blank" ! - nous voyons que notre application WinRT passe les tests avec succès. Il est possible d'accéder au rapport qui a été établi. Dans le cas présent cela ne présente guère d'intérêt mais en cas de souci ce rapport devient un outil de travail essentiel.

Le rapport généré est assez fin et balaye de nombreux aspects. Il reste consultable même après avoir fermé la fenêtre du dernier dialogue puisqu'il est généré sous la forme d'une page Html placée dans `Bin\Debug` (alors que cela devrait plus logiquement se trouver dans Bin\Release – petit bug ? peut-être, dans des versions précédentes de test c'était bien dans Release que le rapport était créé).

Exemple de rapport du WACK

Voici le rapport généré pour l'application de test utilisée, cela vous donnera une idée plus précise des points testés par le WACK (les petites croix représente des passages que j'ai "censurés") :

```
Windows App Certification Kit - Test Results
Nom de l'application : 6946a716-6739-4020-xxxx-xxxxxxxxxxxx
Version de l'application : 1.0.0.0
Éditeur de l'application : xxxxxxxxxxxxxxxx
Version du système d'exploitation : Microsoft Windows 8 Entreprise
(x.x.xxxx.x)
Heure du rapport : xx/xx/2012 xx:xx:xx

Score général : RÉUSSITE

Test des blocages et pannes
RÉUSSITE
Tests de lancement de l'application

RÉUSSITE
Blocages et pannes

Test de conformité du manifeste de l'application
RÉUSSITE
Manifeste de l'application

Test des fonctionnalités de sécurité Windows
RÉUSSITE
Analyseur binaire

Test des API prises en charge
RÉUSSITE
API prises en charge
Test de performance
RÉUSSITE
Génération du code d'octet

RÉUSSITE
Performances du démarrage

RÉUSSITE
Performances de l'interruption

Test des ressources du manifeste de l'application
RÉUSSITE
Validation des ressources de l'application

Test de configuration de débogage
RÉUSSITE
Configuration de débogage

Codage de fichier
RÉUSSITE
Codage de fichier UTF-8

Prise en charge du niveau de fonctionnalité Direct3D
RÉUSSITE
Prise en charge du niveau de fonctionnalité Direct3D

Test des capacités des applications
RÉUSSITE
Capacités d'utilisation spéciales

Validation des métadonnées Windows Runtime
RÉUSSITE
```

```
Test de l'attribut ExclusiveTo

RÉUSSITE
Test d'emplacement du type

RÉUSSITE
Test de respect de la casse du nom du type

RÉUSSITE
Test d'exactitude du nom du type

RÉUSSITE
Test d'exactitude des métadonnées générales

RÉUSSITE
Test des propriétés
```

Conclusion

Malgré la complexité du processus de validation et de la soumission des applications Windows Store Microsoft a su proposer des outils parfaitement au point automatisant à la fois les tests et le déploiement.

Très franchement, WinRT est très agréable à programmer, comme du Silverlight.

Windows 8 est un excellent OS, WinRT est une plateforme de grande qualité servie par de grands outils. Le tout dans une cohérence unique sur le marché que la sortie de Windows Phone 8 a reforcé.

Enfin, et pour la première fois depuis des années, on arrête de nous proposer des copies de l'IPhone ou du Mac et de leurs interfaces totalement dépassées… Mieux, Microsoft nous offre l'opportunité rare d'un marché vierge où tout reste à faire. Que peut-il y avoir de plus passionnant que ça ? !

I'm a PC ? Depuis longtemps.

Mais aujourd'hui, I'm a Surface !

Pro MS ? C'est possible, mais lucide. Mes nombreux billets sont autant de preuves techniques et factuelles de la puissance de Windows 8, de C# et de Xaml, ai-je finalement besoin d'en dire plus ou de me sentir coupable, voire de me justifier de dire que j'aime quelque chose que je connais bien et dont j'apprécie la qualité ?

Pour moi l'avenir s'écrit en C# et en XAML, et ce depuis des années… Et ce que je vois de l'évolution de ce couple fantastique ne me donne pas envie de changer pour des ersatz de Java qui n'ont même pas le droit d'utiliser ce nom (Cf procès

Oracle/Google) pas plus que des cocoa vieillots (qui datent de NeXSTEP) et mal ficelés (regardez comme un NSObject implémente un compteur cracra à côté de la gestion mémoire managée de .NET et on en reparlera…) !

Certes, je reste critique sur certains points. L'OS à deux têtes est pour moi une erreur grave. Le mode full-screen sur PC je n'y croyais pas, je n'y crois pas, et plus le temps passe plus je vois que ça n'emballe pas les foules non plus. Modern UI devait remplacer totalement Classic desktop ou ne sortir que lorsque cela sera possible. La confusion de deux OS en un seul, incompatibles entre eux, la sortie trop rapide de Surface Pro dont le bureau classique a tué les ventes de surface RT 1 (900 millions de dollars de pertes provisionnées dans les comptes rien que ça !), le sacrifice idiot de Silverlight qui pouvait et devait rester maintenu comme outil d'Intranet en entreprise, etc… J'ai milles raisons de m'opposer à la Direction actuelle de Microsoft et de dénoncer ses bévues énormes. Microsoft reste un fleuron de l'industrie du logiciel sur le plan technique, mais avec le couple Sinofsky/Ballmer cette société est devenue l'ombre d'elle-même et a créé beaucoup de mécontentement, de frustration et d'incohérence. L'incompétence totale de cette Direction je la dénonce et en tant que MVP j'assume cette position. Sinofksy a été viré, Ballmer va ejecter, soyons patients et ne jetons pas les adorables bébés technologiques avec l'affreuse eau du bain ! La société Microsoft possède des produits très en avance sur ses concurrents, qu'ils soient mal vendus, que les priorités soient mal choisies, que les axes soient mal tracés, c'est vrai. Il n'y a pas de pilote aux commandes. Mais qu'importe, l'incompétence d'une direction à l'avantage de s'effacer en la changeant, reste les produits, et eux valent toute notre attention…

Cachez ces images que je ne saurais voir ! (WinRT et le cache image)

WinRT utilise un cache pour les images lues depuis une URL. Cela améliore l'expérience utilisateur et limite la bande passante consommée mais ce n'est pas toujours un comportement souhaitable…

Cache-cache, par défaut

Windows 8 est un peu blagueur et ne cesse de jouer à "cache-cache" avec les images téléchargées… Bien entendu c'est "pour le bien" de l'utilisateur, quand une image est relue plusieurs fois seule la première est réellement chargée, les autres accès sont bloqués et servis par le cache ce qui accélère grandement l'expérience utilisateur.

Pas toujours souhaitable

Cacher n'est pas jouer pourrait-on dire. En effet, si ce comportement par défaut est parfaitement raisonnable pour bon nombre d'applications, cela devient une gêne importante dans d'autres cas. Pensons à une simple image météo rafraichie régulièrement ou un graphique de statistiques mis à jour à intervalles réguliers. Toutes ces images provenant de la même URL, elles seront retournées telles qu'elles sont dans le cache, c'est à dire sans les modifications intervenues depuis la première lecture.

Fâcheux donc dans certains cas puisque c'est le fonctionnement même d'un logiciel qui peut apparaitre bogué alors que tout semble programmé correctement…

Accéder au cache dans l'URL

Il existe une première feinte publiée par Ian Walkers sur son blog et qui est assez intéressante : selon lui visiblement WinRT interprète les URL avant de leur donner accès au Web. Mieux, il sait comprendre un paramètre de cache qui permet de contrôler finement la durée de mise en cache. Je disais "visiblement", c'est à dire "de ce qu'on peut voir", parfois ce qu'on voit est trompeur d'où mes précautions. Ici WinRT ne fait rien du tout. Mais cela fonctionne tout de même car tout simplement en ajoutant un texte changeant à chaque fois à l'URL de l'image on trompe la gestion de chache…

Il suffit donc pour cela d'ajouter à l'URL de l'image les informations suivantes :

```
"?cache=" +
DateTime.Now.DayOfYear.ToString()+DateTime.Now.Hour.ToString();
```

Dans cet exemple cela indique un cache d'une durée d'une heure… Est-ce que cette durée est interprétée par « quelqu'un » dans la chaîne entre l'application et le serveur ? C'est un test à faire… Mais il est possible que rien ne soit interprété.

En utilisant une valeur en `ticks` égale à l'heure courante on devrait s'assurer que rien n'est caché. Cela me semble simpliste dans le sens où si c'est le serveur qui interprète la zone, alors il faudrait certainement utiliser une date UTC, l'heure locale n'a pas de valeur. Mais bref, ce qui compte c'est de feinter le cache des images.

Cette feinte est intéressante mais elle mériterait d'être précisée par un accès à la documentation qui la décrit. Je n'ai pas trouvé une telle documentation pour l'instant. J'y reviendrai si je le trouve, ou si un lecteur la trouve entre-temps !

Accéder au cache via l'objet BitmapImage

La classe `BitmapImage` offre d'autres moyens, mieux documentés, de gérer le cache. Notamment via les propriétés `CacheOptions` et `CreateOptions`.

On peut dès lors paramétrer plus finement les choses avec des niveaux comme "`OnDemand`, `OnLoad` ou `None`" (en plus de "`Default`").

Conclusion

C'est un petit pas pour le blogger mais c'est un pas géant pour les applis qui se croyaient boguées alors qu'elles ne l'étaient pas…

Windows Store Apps : gérer la navigation

Nous avons vu dans les précédents billets de nombreuses choses spécifiques à WinRT, les templates de projet, les protocoles d'activation personnalisés, les Charmes, etc. Un autre aspect essentiel des Apps Windows Store est la navigation. Voyons comment cela fonctionne…

La navigation

C'est un aspect essentiel de la programmation sous WinRT pour deux raisons majeures : d'une part les mécanismes de navigation sont intégrés au framework, si on peut être inventif et s'écarter de la norme il faut savoir que celle-ci existe et est conçue pour assurer la cohérence de l'expérience utilisateur; d'autre part, la navigation est un point crucial d'une application qu'on sous-estime toujours. Une mauvaise expérience de navigation peut ruiner un bon logiciel et une expérience fluide, simple, peut faire d'un logiciel banal un best-seller…

Dans le principe tout est simple puisque géré par le framework. Mais si vous venez du monde Silverlight ou Windows Phone vous risquez d'être un peu perturbé par la façon de faire sous WinRT. Sous Windows Phone par exemple la navigation se fait en utilisant `NavigationService.Navigate()` auquel on passe une URI. Sous WinRT les choses se ressemblent mais il y a des nuances…

Dans une application Windows Store chaque page possède une "`Frame`", un objet particulier qui implémente les méthodes et propriétés suivantes pour simplifier les opérations de navigation :

`GoBack()`	Navigation arrière vers la page la plus récente page de l'historique
`GoForward()`	Navigation avant vers la prochaine page de l'historique
`Navigate(type,args)`	Navigation vers la page spécifiée
`BackStackDepth`	Nombre d'entrées dans l'historique de navigation
`CanGoBack`	Indique si la navigation arrière est possible
`CanGoForward`	Indique si la navigation avant est possible

Les deux premières méthodes impliquent que l'historique de navigation est géré par l'objet `Frame` bien entendu.

Tout cela semble finalement très familier… Mais avez-vous remarqué les paramètres de la méthode `Navigate()` ?

Il s'agit d'un type et d'un argument et non plus d'une URI ! On ne passe donc plus une adresse absolue ou relative pour naviguer mais un type. Quel type ? Celui de la page à atteindre.

```
// navigation simple
Frame.Navigate(typeof(SecondaryPage));
// navigation avec passage de paramètre
Frame.Navigate(typeof(SecondaryPage), obj);
```

Les arguments (le paramètre de type `Object`) sont très utiles pour passer un contexte ou des informations de page en page et ainsi de permettre à l'application de "garder le fil" malgré la grande liberté de navigation offerte à l'utilisateur. C'est un aspect que j'aurai certainement l'occasion de développer un peu plus dans un prochain billet.

Un exemple

Le projet

Prenons une application suivant le modèle "`Blank`".

La page principale

Une fois le projet créé modifions le code de `MainPage.xaml` pour ajouter un bouton et un `TextBlock`, ce que le code Xaml suivant montre facilement lorsqu'on est habitué à ce langage :

```
<Page
    x:Class="NavigationDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
        <TextBlock Text="Main Page" Style="{StaticResource
HeaderTextStyle}"
                   Margin="100,50"/>
        <Button Style="{StaticResource BackButtonStyle}"
Margin="403,0,0,668"
                RenderTransformOrigin="0.5, 0.5"
Tapped="OnNextButtonTapped">
            <Button.RenderTransform>
                <RotateTransform Angle="180"/>
            </Button.RenderTransform>
        </Button>
    </Grid>
</Page>
```

Ce qui va donner un affichage de ce type :

La page secondaire

Maintenant, créons une seconde page que nous allons appeler `SecondaryPage.xaml`. Ce fichier sera laissé là où VS le propose, dans la racine du projet (au même niveau que MainPage). Cette page aura elle aussi un bouton (pour revenir en arrière) et un `TextBlock` pour la différencier de la page principale :

```
<Page
    x:Class="NavigationDemo.SecondaryPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid Background="{StaticResource
ApplicationPageBackgroundThemeBrush}">
        <TextBlock Text="Secondary Page" Style="{StaticResource
HeaderTextStyle}"
                   Margin="100,50"/>
        <Button Style="{StaticResource BackButtonStyle}"
Margin="29,0,0,674"
                Tapped="OnBackButtonTapped"/>
    </Grid>
</Page>
```

Ce qui donne cet affichage là :

Nous en avons terminé avec la conception visuelle de cette application extraordinaire !

Le code de navigation

Pour cet exemple je n'ajouterai pas les difficultés propres à MVVM. Nous utiliserons le code-behind pour traiter la navigation mais cela n'est pas à faire en production. Les boutons devraient être liés à des `ICommand` implémentées par les ViewModels des deux pages, commandes qui déclencheraient la navigation par l'accès à l'objet `Frame` (pas une bonne idée…) ou mieux par l'envoi d'un message de navigation que les vues transformeraient en ordre de navigation… Certains toolkits MVVM propose des approches rationnelles à ce problème.

Mais je fais abstraction ici de ces subtilités.

Dans la page principale codons le clic sur le bouton symbolisant la page suivante :

```
private void OnNextButtonTapped(object sender, TappedRoutedEventArgs e)
{
    Frame.Navigate(typeof (SecondaryPage));
}
```

Dans la page secondaire faisons de même pour le bouton de retour en arrière :

```
private void OnBackButtonTapped(object sender, TappedRoutedEventArgs e)
{
    if (Frame.CanGoBack)
    {
        Frame.GoBack();
    }
}
```

Dans le premier cas nous utilisons une navigation impérative, `Navigate()` auquel le type de la page secondaire est passé. Cela signifie que quel que soit l'état de l'historique de navigation l'utilisateur accèdera à la page secondaire.

Dans le second cas nous utilisons une autre stratégie : nous commençons par tester si le retour en arrière est possible et seulement dans l'affirmative nous invoquons la méthode `GoBack()` de l'objet `Frame`.

Bien entendu, puisque le seul moyen d'accéder à la page deux dans notre application est de la créer depuis la page principale, le test sera toujours vrai. Mais dans une application à la navigation plus complexe ce test pourrait s'avérer essentiel. D'un autre côté, dans un tel cas, je vous conseille plutôt de cacher / montrer le bouton de navigation… Afficher un bouton qui ne fonctionne pas est l'une des pires choses pour l'expérience utilisateur !

Conclusion

La navigation WinRT est finalement assez simple. Elle est basée sur une stratégie "View First" puisque c'est la vue qui navigue vers une autre vue et non pas un ViewModel vers un autre ViewModel. Comme l'exemple n'implémente pas le pattern MVVM cela n'a guère d'importance. Mais tentez de réfléchir à ce que cela signifie avec un framework MVVM qui serait basé sur une approche ViewModel First…

Naviguer, comme tout marin le sait, ça semble facile mais cela réserve des surprises à ceux qui ne savent pas être assez prudents !

Ce billet vous a montré où se trouvait le quai et les bateaux, ne croyez pas qu'il vous a donné le droit de vous prendre pour un capitaine de haute mer, il vous reste encore des choses à savoir…

Debug d'applications WinRT sur Surface

Si je reste circonspect sur l'intérêt de Surface Pro vu le prix annoncé et la faible autonomie (qui était prévisible avec de l'Intel) et malgré les efforts indéniables sur la version 2, si je reste dubitatif sur Modern UI sur PC (un PC n'est pas une grosse tablette) magré les quelques changements de 8.1, je continue à être un fan de Windows 8 sur Surface RT car seule Surface RT propose une vision neuve et cohérente qui met en valeur Modern UI (si on fait abstraction de Office en bureau classique). Deboguer des applications pour Surface sur Surface est impératif pour se rendre compte des temps de réponse par exemple. Mais comment cela se passe ?

Déployer et Déboguer sur Surface RT

Déployer ses propres petites applications personnelles sur "sa" Surface est un must pour tout développeur. Pas besoin de market place pour se faire plaisir… Et si une idée devient géniale au fil des amélirations il est toujours temps de la publier. De même, publier une application Surface qui n'aurait pas été testée sur Surface est une hérésie, le simulateur ne peut pas tout dire. Notamment les temps de réaction et de traitement, le bien fondé de telle ou telle mise en page en mode tactile etc…

Pour toutes ces situations il faut pouvoir déboguer (et déployer) directement des applications sur Surface.

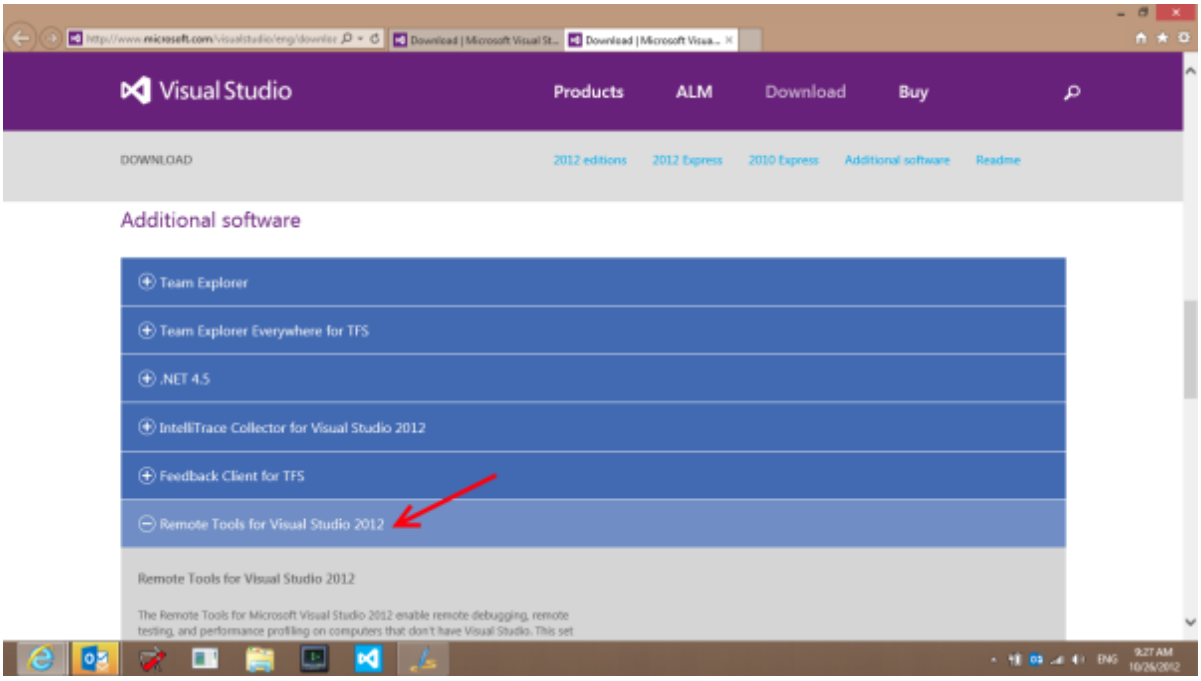
J'ai testé moi-même en m'inspirant très directement d'un post de Tim Heuer, étant sûr que la source était bonne… Ce que je vous livre ici est donc issu à la fois de ce billet et de la validation que j'ai pu en faire au travers de mes propres expériences.

Les outils

Surface RT tourne sur ARM. Pas question d'installer Visual Studio dessus pour faire joujou… Ce dernier ne marche que sur architecture Intel.

Toutefois, côté PC, il vous faudra un environnement complet. Et parmi les outils indispensable vous aurez besoin du "remote developer tools for Visual Studio" mais adapté à votre _cible_. Donc pour Surface, pour ARM.

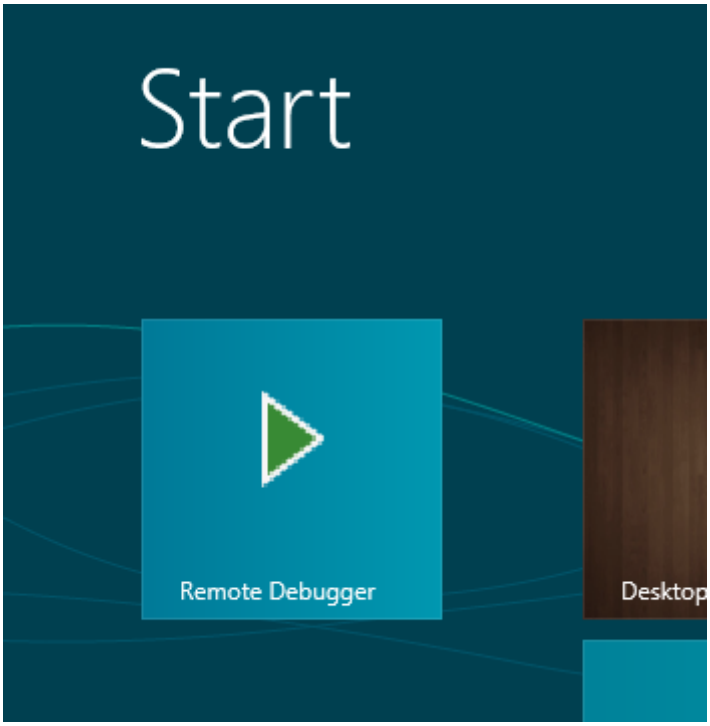
Vous pouvez télécharger cela ici : Microsoft Visual Studio Downloads. En scrollant un peu et en regardant dans la rubrique "Additional Software" vous trouverez "Remote tools for Visual Studio 2013".



La capture montre la version pour VS2012, aujourd'hui c'est celle pour 2013 qu'il faut bien entendu télécharger.

Vous pouvez effectuer l'opération directement sur Surface depuis le browser, ou bien depuis votre PC (il faudra alors transférer les fichiers sur Surface via une clé USB par exemple).

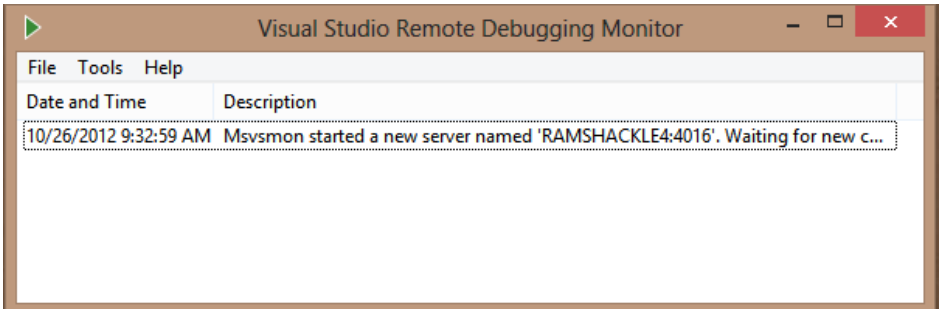
Sur Surface, installez les remote tools pour ARM. Aucune installation supplémentaire n'est requise. Une fois l'installation terminée vous disposerez d'une nouvelle tuile dans le menu Modern UI :



Il est maintenant possible de débuter une session de debug !

Configuration du Remote Debugger

Cliquez sur la tuile du Remote Debugger sur Surface. le débogueur sera lancé (après d'éventuelles confirmations de sécurité). Il sera exécute dans son mode par défaut :

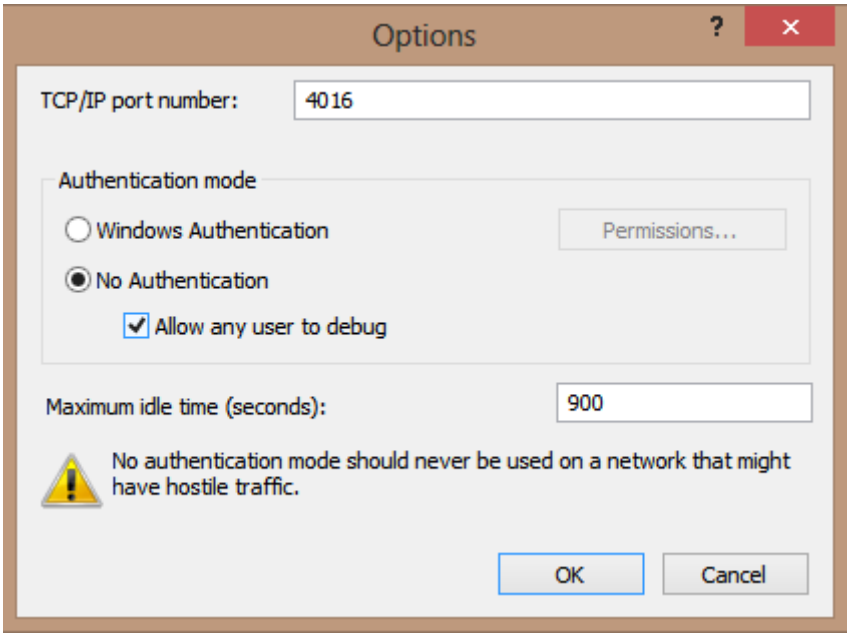


Par défaut il s'agit d'un setup sécurisé. Cela signifie que pour attacher une session distante vous aurez besoin de vous assurer que les autorisations sont correctes et d'autres détails de ce type. Mais comme votre Surface n'est probablement pas sur le même domaine / workgroup que votre machine de développement, cela peut s'avérer un peu "sportif".

Personnellement j'ai suivi les conseils de Tim qui, pour faire des essais sur sa Surface a préféré déconnecté toutes ces options de sécurités. Cela rend le travail de debug bien moins "lourdingue". Bien entendu la tablette ne doit pas être laissée

indéfiniment dans un tel état. Mais la fermeture du Remote Debugger stoppe toute activité dangereuse.

La configuration proposée par Tim et qui fonctionne très bien est la suivante :



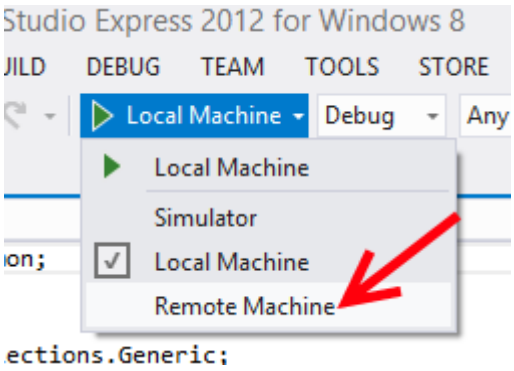
Avec cette configuration on peut exécuter une application sur la machine cible (Surface) sans avoir besoin d'une authentification.

Une fois ces options de debug réglés, Surface est prête à recevoir vos créations et vous aider à la déboguer (avec Visual Studio côté PC malgré tout…).

Lancer une application dans le débogueur de Surface

Votre surface est correctement équipée (avec le Remote Debugger), elle est configurée pour accepter vos applis, vous avez lancé le remote debugger qui est donc en écoute. Ne reste plus qu'a exécuter l'application à tester.

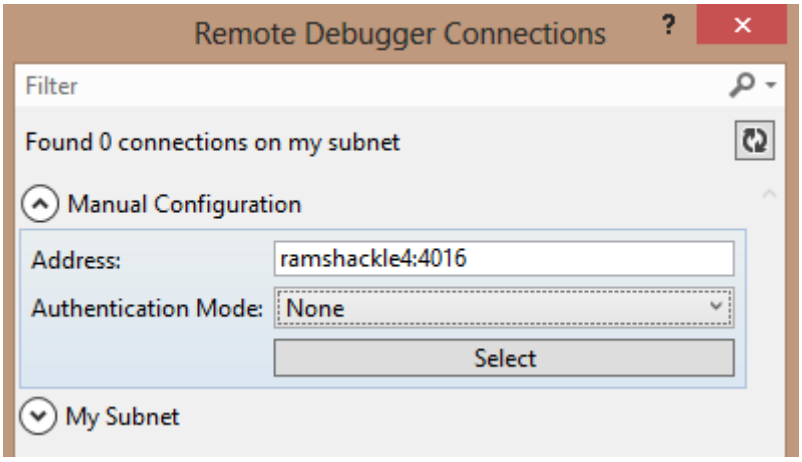
Cela se fait bien entendu depuis le PC et Visual Studio. Pour une application C# il faut cliquer sur le bouton Start et choisir "Remote Machine".



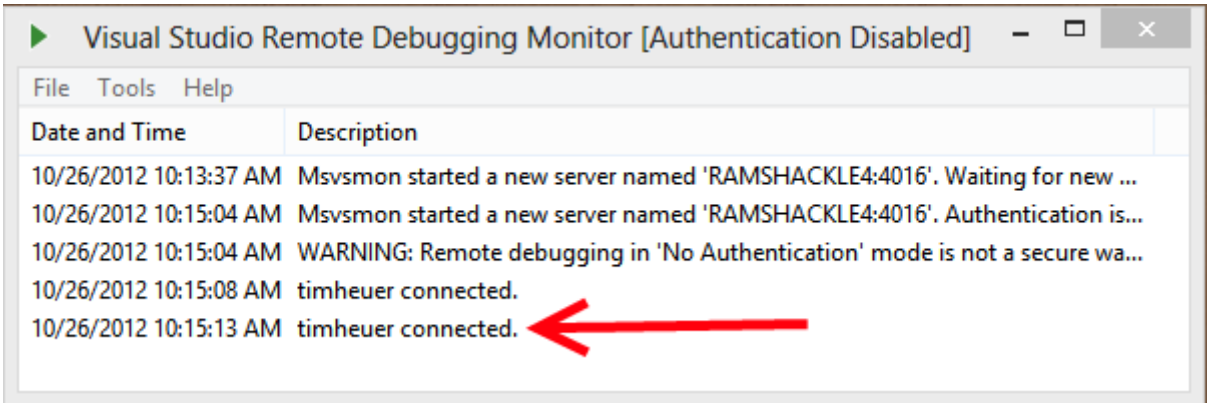
Cela fonctionne depuis Visual Studio Express aussi bien que depuis les versions plus lourde de l'environnement de développement. Une bonne chose pour tous les développeurs qui ne doivent pas forcément disposer d'une version Ultimate.

Une fois le lancement demandé, vous devrez choisir l'unité mobile à tester. Si vous travailler chez vous sur votre propre réseau, sans domaine, vous pourrez très facilement découvrir directement votre Surface dans les connexions proposées. C'est vraiment facile.

Soyez certains de bien vérifier les mécanismes d'authentification et tout le reste. Si vous suivez les conseils donnés ici, il n'y a a rien à vérifier, il suffit d'indiquer "none" dans la méthode d'authentification.



Maintenant que la configuration est prête et sélectionnée, lorsque vous tapez sur F5 Visual Studio va préparer tout le nécessaire pour déployer l'application sur Surface. Vous pouvez d'ailleurs suivre les messages aussi sur surface dans l'interface du debugger.



Tim ayant déjà fait les captures écran et ma version étant en US de toute façon j'ai repris les écrans de son billet. C'est bien entendu votre nom d'utilisateur qui apparait et non celui de Tim !

Bien entendu, dans votre code côté PC il est possible de placer des points d'arrêt, d'inspecter des valeurs, etc, bien que l'application soit en train de tourner sur Surface… Génial non ?

Conclusion

Déployer et tester des applications WinRT sur Surface est d'une grande facilité. C'est aussi très gratifiant de voir ces petits programmes d'essai tourner "comme des grands"' sur la machine cible. Une fois déployée par le Remote Debugger l'application reste installée dans les menus de Surface, on peut donc s'en servir à titre personnel… Pas besoin non plus de la désinstaller pour une nouvelle session de debug, elle sera remplacée automatiquement.

Et pour cela, au minimum, vous pouvez utiliser Visual Studio Express (gratuit) et du remote debugger tout aussi gratuit.

Windows Phone 8 : le casse tête de la taille des icônes

Si Windows Phone 8 reprend le concept de menu à tuile de WP7 déjà bien connu, c'est tout de même avec des variantes qui posent si ce n'est problème au moins question. Par exemple, quelle taille d'icône est utilisée selon le type de tuile et quel format donne le meilleur résultat ?

Deux questions en une

Nous voulons savoir deux choses :

1. Quelle doit être la taille des icônes avec le `IconicTileTemplate` ?
2. Quelle taille image est utilisée quand la tuile est en mode large ?

Le fait que la réponse n'est pas si évidente vient d'une discussion sur StackOverflow concernant la taille des tuiles et des marges que vous pouvez suivre ici (en anglais) : http://stackoverflow.com/questions/12837720/windows-phone-8-startscreen-tile-sizes-and-margins/13153522#13153522

Un set d'images et des tests

Lorsque la documentation n'est pas suffisante, reste les tests… et ici préparer un set d'images dans plusieurs dimensions.

Ainsi, un jeu d'icônes avec les tailles suivantes va être utilisé :

- 110x110 px
- 202x202 px
- 72x110 px
- 72x72 px

L'émulateur étant utilisé en mode 720p pour faire tourner les tests.

Je vous fais grâce des différents essais pour passer directement aux conclusions…

Ce qu'on peut en déduire

La première question était de savoir quelles images le `IconicTileTamplate` utilise. La réponse est simple :

Les images utilisées par ce template doivent avoir les dimensions suivantes :

- 72x110 px en mode small
- 132x202 en mode medium

Si on compare les tests avec une image carrée (202x202 px) qui semblerait, a priori, un meilleur choix, cela donne :

La taille 202x202 n'est pas celle que l'on pense… Bien que plus grandes elle apparait plus petite (seconde tuile) que l'image 132x202 qui donne en réalité le meilleur rendu (icône plus grande et plus lisible).

Lorsque la tuile contient du texte les choses changent car l'icône est alors réduite en bas à droite. C'est l'image small qui est utilisée par le template.

Là encore, on pourrait s'attendre à ce que le mode carré donne un meilleur rendu (le format se prêtant mieux à des scaling homothétiques en plus). Mais la vérité est ailleurs…

L'image carrée 110x110 px (à gauche) est certes plus grande mais elle ne donne par un rendu très agréable (bien qu'il s'agisse ici d'une question de gout il faut l'avouer).

L'image carrée en 72x72 qu'on penserait bien taillée pour ce cas (à droite) montre un décalage d'alignement pour le moins déroutant.

Finalement l'image donnant le meilleur rendu (taille et positionnement) reste celle du milieu, en 72x110 px…

La seconde question était de savoir quelle taille image est utilisée par le template lorsque la tuile est en mode large. Et la réponse est : ça dépend !

Si la tuile ne contient rien d'autre que l'icône, le template utilise l'image medium, si la tuile possède un contenu en plus de l'icône, c'est l'image au format small qui est utilisée.

Conclusion

Les tests prouvent que la taille des icônes doit être bien choisie et que cette taille n'est pas celle qu'on penserait intuitivement… Les formats non carrés donnent les meilleurs résultats.

Les petites modifications apportées par Windows 8.1 méritent aussi que vous y portiez attention, les tuiles sont plus complexes qu'il n'y parait !

Utiliser des Behaviors dans des applications Windows 8 Store

Les Behaviors ne sont pas récents, ils existent depuis Silverlight 3, et leur succès n'a d'égal que leur valeur : notamment sous MVVM (mais pas seulement) ils permettent souvent de gérer de nombreuses situations de façon élégantes côté Xaml. Hélas .NET sous WinRT ne gère pas les Behaviors. Mais il existe une solution…

Les Behaviors

En bon français nous parlerions de "comportements". En effet quand on ajoute un Behavior à un contrôle il s'agit bien de lui ajouter un "comportement" particulier.

Les Behaviors ont été introduits, comme de nombreuses autres bonnes idées de Xaml, d'abord dans Silverlight. La version 3 pour être précis. Et bien entendu dans Expression Blend, l'outil de développement le plus excitant et le mieux fait que je n'ai jamais vu. Je sais que nombreux sont les développeurs à "ne rien comprendre" à Blend, c'est pourquoi j'insiste. Blend est juste fantastique et reste le seul outil sérieux pour travailler en Xaml dès qu'on dépasse l'alignement de deux `TextBox` dans une `Grid`…

Mais revenons aux Behaviors. Après cette première percée dans Silverlight 3, et comme une si bonne idée ne pouvait rester qu'une simple curiosité d'un seul des "enfants" de Xaml, on a vite retrouvé leur implémentation dans les versions de Xaml pour Windows Phone 7 et même WPF (tout comme le Visual State Manager).

Des propriétés … attachantes

Techniquement les Behaviors ne sont rien d'autre qu'une implémentation particulière des *Propriétés Attachées*. Ces dernières permettent de définir des propriétés au niveau d'un contrôle parent qui se propagent vers les contrôles enfants. C'est par exemple le cas de `Top` et `Left` qui sont des propriétés de `Canvas` mais qui sont manipulables dans tout contrôle qu'on pose sur un `Canvas` pour en fixer la position. La propriété est bien définie au niveau de `Canvas`, et c'est elle qui gère les différentes valeurs données à chaque contrôle enfant. Mais côté développeur, c'est bien dans chaque enfant qu'on saisit le `Top` et le `Left`… Idée brillante s'il en est, comme beaucoup d'autres dans Xaml comme le Binding qu'aucun ersatz actuel n'a pu égaler (qu'il s'agisse de HTML 5, de Android ou iOS pour ne parler que des principaux environnements concurrents).

D'autres exemples simples de propriétés attachées viennent immédiatement à l'esprit : `Row` et `Column` pour les enfants d'une `Grid` notamment. Le principe est simple, l'implémentation aisée, il est dommage de constater que les développeurs ne s'en servent pas plus souvent dans leurs propres contrôles (ceux qui produisent des contrôles sont d'ailleurs assez rares eux-mêmes !).

Des propriétés auxquelles ont devient… dépendant

Tout cela n'est au final qu'une variante spécialisée des *Propriétés de Dépendances*.

Si vous ne connaissez pas bien ces propriétés particulières je vous conseille mon billet "Les propriétés de dépendances et les propriétés jointes sous WPF" accompagné d'un article PDF de 25 pages avec un projet exemple (vous noterez l'hésitation de traduction entre "jointes" et "attachées", l'article date de 2009 et la communauté française n'était pas encore très fixée sur ce point de langage…).

Comme je le disais, les Behaviors ne sont que des propriétés attachées. Dans un Behavior les valeurs des propriétés attachées sont utilisées pour configurer un ou plusieurs gestionnaires d'évènement. Un Behavior c'est à la fois un *trigger* (un évènement – ou plusieurs – dont le déclenchement est écouté) et une ou plusieurs *actions* paramétrables exécutées lorsque le ou les triggers se déclenchent.

Les Behaviors : de l'action sans code

L'intérêt des Behaviors est de pouvoir encapsuler les conditions d'un déclenchement et les actions qui doivent en découler sous la forme d'un composant réutilisable qu'on peut déposer sur n'importe quel contrôle (sauf si des limitations propres au Behavior considéré s'appliquent). L'un des gros avantages des Behaviors est d'apparaitre sous la forme de composants "visuels" se plaçant dans l'arbre graphique d'un code Xaml. Les Triggers de WPF peuvent sembler assez similaires dans leur description mais ils sont très différents dans ce qu'on peut en faire, il ne faut donc pas confondre ces deux façons d'ajouter des actions à un contrôle.

On notera qu'un Behavior est techniquement une classe générique, ce qui permet aisément de fixer en paramètre le nom de la classe cible. Celle-ci peut être très précise - `Behavior<TextBlock>`ne fonctionnera que sur des `TextBlock` – ou bien plus générale (en utilisant un ancêtre de toute une branche de Xaml). Cette souplesse offre la possibilité de pouvoir personnaliser à l'extrême un Behavior qui ne peut être

utilisé qu'avec une classe précise ou bien de créer des comportements plus généraux (par exemple qui ne vont que changer la `Visibilty`, présente dans tout contrôle).

Il est donc possible d'ajouter des comportements, éventuellement complexes, qui se posent sur un contrôle visuel avec la même simplicité qu'on placerait un effet visuel de flou par exemple (les pixel shaders). C'est une avancée énorme permettant à des développeurs de fournir aux designers tout ce qu'il faut pour travailler sans que ces derniers n'aient à savoir coder. On retrouve ici tout l'esprit de la séparation code / UI qui se prolonge dans l'application d'un pattern comme MVVM.

Le lecteur intéressé trouvera ici un billet et son code exemple sur les pixel sharders.

Dot.Blog a déjà parlé de presque tout en plus de 650 billets depuis sa création, n'hésitez pas à utiliser la fonction de recherche en haut de l'écran !

Les exemples de Behaviors ne manquent pas, j'ai publié il y a déjà un moment toute une librairie qui en regroupe des dizaines (My Behavior Collection). Par exemple on peut supposer un Behavior qui, lorsqu'il s'attache à un `TextBox` lui confère le comportement "*sélectionner tout le texte sur la prise de focus*", rendant ainsi le changement de la valeur beaucoup plus rapide pour l'utilisateur. Et tout cela sans avoir besoin de créer un contrôle `TextBox` spécialisé, ni d'écrire de code-behind et sans jamais toucher le ViewModel qui est derrière la vue concernée !

Plus fort, les Behaviors peuvent être combinés. Il est donc possible d'ajouter plusieurs Behaviors à un même contrôle pour lui conférer immédiatement tout un tas de comportements différents sans jamais écrire de code (en tout cas répétitif). Une fois un Behavior écrit il est utilisable partout, même dans d'autres projets. C'est un gain de productivité important car les Behaviors amènent la réutilisation du code propre à la programmation objet au niveau du visuel sans demander des compétences de programmation à celui ou celle qui fabrique ce visuel.

Puisque les Behaviors peuvent être combinés, il est bien entendu plus intelligent de créer des behaviors simples, s'occupant d'un seul comportement, que de vouloir créer des monstres automatisant plusieurs comportements à la fois et qui seront donc plus difficiles à réutiliser ailleurs dans d'autres projets…

Bref, les Behaviors c'est extraordinaire, il faut s'en servir le plus souvent possible.

Pour aller plus loin avec les Behaviors je vous suggère ces billets :

- Ecrire des Behaviors
- Un Behavior ajoutant un effet de réflexion visuelle

- Utiliser des éditeurs de valeur dans les Behaviors

La mauvaise nouvelle

Hélas, le profile .NET de WinRT ne contient pas les Behaviors… Faute de temps pour les implémenter dès la sortie de Windows 8 certainement.

On sait que Sinofsky voulait la peau de Roger Silverlight, celle de son complice C# ainsi que le cousin WPF et leur mère à tous .NET.

Tous les dictateurs et les psychopathes ont, je ne sais pas pourquoi, chacun leurs petites lubies vis à vis de tel ou tel groupe, de pogrom en épuration ethnique, tous ont en marotte un peuple à massacrer, même quand cela ne gêne en rien leur pouvoir absolu. Chez Sinofsky c'était .NET, C# et Xaml. Allez comprendre…

En prenant en compte cet élément essentiel à la compréhension des choses on peut imaginer que l'équipe Xaml n'a pas du se voir réserver les meilleures conditions de travail ni le plus gros budget…

Des âneries comme Html5/WinJS ou le retour de l'ancêtre barbu champion des memory leaks C++ était plus son truc à Sinofksy. Un vrai ringard dont le départ précipité de chez Microsoft aurait été encore plus efficace s'il avait eu lieu encore plus tôt.

Mais comme dans les contes de Noël l'histoire se finit bien et le méchant est puni.

La tradition chère à cette période si particulière de l'année est donc sauve et nous sommes débarrassés de l'odieux tyran qui a fait commettre à Microsoft beaucoup d'erreurs dont la mise à l'écart de C# / Xaml n'est peut-être pas la plus grave d'ailleurs.

Il reste qu'il n'y a pas de support des Behaviors dans le Xaml du profile .NET sous WinRT et personne ne peut prédire ce que sera l'action ni l'influence de Julie Larson-Green, même si nous appelons d'ores et déjà de nos vœux un changement en profondeur dans les méthodes de travail que nous espérons tous plus harmonieuses et dénuées de la haine sinofskienne irrationnelle à l'égard de C#, Xaml, WPF et Silverlight.

Acte 1 : La Hollande à notre secours

Non, je ne parle pas de la compagne de notre Président. D'ailleurs il serait difficile de l'appeler ainsi puisqu'elle ne porte pas le même patronyme, et même dans ce cas un gentleman ne saurait l'apostropher de la sorte. Même si elle semble avoir des dons certains pour buzzer sur Twitter je ne pense pas qu'elle soit assez geek pour nous sortir de l'affaire de toute façon…

Je ne parle pas non plus de "l'autre pays du fromage", avec un seul de connu cela fait maigre pour se comparer à notre patrie fière de ses centaines de merveilles dures ou coulantes, pures ou marbrées, voire veinées de moisissures amicales, aux senteurs variées comme un champ de blé ou un automne pluvieux et aux saveurs milles fois renouvelées que la planète entière nous envie (ou pas, mais ça c'est la jalousie certainement).

Je veux juste parler du pays de Joost van Schaik.

Un hollandais. De Hollande donc.

Ce gentil développeur a en effet eu l'idée de partir des propriétés attachées pour recréer le code des Behaviors, les rendant ainsi disponibles dans Xaml sous WinRT.

C'est par ce poste que les choses ont commencé : http://dotnetbyexample.blogspot.be/2012/03/attached-behaviors-for-windows-8-metro.html

Toutefois les choses se sont ensuite un peu plus structurées et cette idée est devenue un projet CodePlex, les WinRtBehaviors.

Mais ce package est *deprecated* depuis la sortie de Windows 8.1 car, comme je le présumais, les behaviors n'avaient pas été introduits par manque de temps. Plus d'un an après la sortie de 8.0 l'équipe Xaml de Windows 8 a eu un peu de temps pour combler certains manques.

On gardera de la librairie de Joost van Schaik le souvenir d'un effort louable de la communauté XAM/C# toujours très active et prompte à fournir ce qui manque dans la boîte. C'est rassurant.

On gardera aussi de cette librairie un bel exemple de code à étudier car il y a toujours beaucoup à apprendre du code des autres…

Acte 2 : Le Windows 8.1 Behahior SDK

Le 17 ocotobre 2013, donc il y a quelques jours au moment où je révise ce texte pour le livre PDF consacré à WinRT, Microsoft a releasé le Windows Behavior SDK.

Ce SDK est un complément, une librairie qu'il faut activer volontairement et non pas l'ajout direct des behaviors dans XAML. Le choix d'un SDK séparé est certainement lié au fait que cette feature a été livrée après la sortie de Windows 8 et qu'il était plus facile d'en faire un SDK à part que de modifier le cœur du moteur Xaml intégré à l'OS.

Mais en gros nous retrouvons les behaviors XAML tels qu'on les connaît sous Silverlight ou WPF, dans un SDK officiel et non une librairie tierce.

Ce SDK se compose de deux parties. La première - `Microsoft.Xaml.Interactivity` - contient les outils pour créer vos propres behaviors et actions. La seconde – `Microsoft.Xaml.Interactions` – contient quelques behaviors et actions déjà écrits par Microsoft. Ils sont construits en s'appuyant sur la première partie.

Cette première mouture du SDK ne livre que le stricte nécessaire, on est encore loin de la richesse des behaviors fournis avec SL5 par exemple. Mais ce n'est que le début, c'est officiel, et on dispose de tout ce qu'il faut pour créer ses propres classes de behavior.

Parmi les Actions on note :

- `CallMethodAction`
- `ChangePropertyAction`
- `GotoStateAction`
- `InvokeCommandAction`
- `NavigateToPageAction`
- `ControlStoryBoardAction`
- `PlaySoundAction`

Côté behavior on note la présence de :

- `DataTriggerBehavior`
- `EventTriggerBehavior`
- `IncrementalUpdateBehavior`

Tous sont déjà bien connus, sauf le dernier. Son but est de permettre la mise à jour incrémentale des `ListView` et `GridView` afin de garantir une meilleure réactivité. On fixe ce behavior aux `ItemTemplates` et leur rendu est différé jusqu'au moment où ils doivent devenir réellement visibles.

La librairie expose `IAction` et `IBehavior` pour construire ses propres classes.

Je reviendrais prochainement sur Dot.Blog sur ce SDK très récent.

Conclusion

Les Behaviors sont des outils très importants sous Xaml, souvent méconnus ou peu utilisés, leur absence sous WinRT n'aide pas à les faire mieux connaitre… Heureusement Windows 8.1 Behavior SDK vient officialiser le retour des behaviors dans le XAML de WinRT !

Je présenterais sur Dot.Blog ce nouveau SDK plus en profondeur alors Stay Tuned !

SQLite sur Windows 8 (ARM et Intel)

Ecrire des applications intelligentes qui dépassent le niveau de la lampe de poche ou de la liste "todo" est un impératif pour qu'enfin le marché Windows 8, tant sur Smartphone que tablettes et PC puisse prendre son essor. Des softs sérieux, voire LOB, voilà ce dont à besoin Windows 8. Tout comme le PC doit en partie son succès à MultiPlan, l'ancêtre de Excel. Or il n'existe pas de machine ou d'architecture qui a pu survivre et s'imposer sans une gamme de logiciels intelligents, pros. Les jeux sont une chose mais ne permettent pas d'assoir le sérieux d'un OS. Et pour toutes ces applications il faut bien souvent une base de données… SQLite est une possibilité à étudier de près…

Une base de données embarquée

Une liste de tâches à faire genre "Any.Do", cela se programme en deux secondes soit avec un petit fichier XML soit même un fichier texte. Une gestion de mots de passe ça se programme en utilisant par exemple la sérialisation d'une liste (cryptée…).

Pour des applications plus intelligentes, plus sophistiquées, plus souples, utiliser des moyens aussi rustiques déporte beaucoup trop de charge côté code, ce qui allonge le temps de développement, complique la maintenance et n'offre pas la souplesse requise.

Passé un certain point seule une vraie base de données permet au code applicatif de se concentrer sur son métier et de se libérer de la gestion des données.

Seulement voilà… SQL Server est une base d'une grande efficacité, mais elle ne peut convenir qu'à des applications lourdes ou en Bureau Classique (WPF et EF peuvent

attaquer directement le serveur, alors que sous WinRT il faudra mettre en place un service séparé ce qui est effroyablement compliqué pour si peu).

Microsoft a conçu SQL Server Compact. Une base compatible SQL Server qui ne nécessite pas une installation lourde et qui peut même être déployée sur un serveur Web hébergé en mutualisé.

J'aime beaucoup SLQ Server Compact car, pour un site Web notamment, cela permet de mettre en place une véritable base de données très rapidement qui s'hébergera n'importe où. Et si le site connait une montée en charge importante, il est très facile de basculer sur une vraie base SQL Server puissante en changeant juste la chaîne de connexion puisque la compatibilité est absolue.

C'est un point fort de SQL Server Compact qu'il ne faut jamais négliger : la scalability.

Toutefois une application Smartphone ou tablette n'évoluera jamais dans un tel sens. Elle doit embarquer tous les moyens servant à son fonctionnement ou doit, d'emblée, fonctionner avec un ou plusieurs services distants de type Cloud, Web Service ou autre. Et le choix de passer de l'un à l'autre des modèles n'apporte pas un gain en rapidité (au contraire) mais d'autres bénéfices (ceux des données dans les nuages).

Du coup, sur les petites architectures, et par cohérence, sur toutes les applications WinRT même sur PC, il faut choisir dès le départ les moyens justes et suffisants. Choisir une base de données capable de fonctionner sur toutes les cibles, dont ARM, n'est pas si simple.

C'est là qu'intervient SQLite.

SQLite

Sa présentation est claire : "SQLite est une bibliothèque de code qui implémente un moteur de base de données transactionnel, auto-contenu, sans serveur et sans configuration" (voir http://www.sqlite.org/).

Il s'agit même du moteur SGBD le plus utilisé au monde (par exemple il y a au moins 300 millions de copies avec Firefox, 20 millions de sites PHP avec un SQLite embarqué, 20 millions de Symbian, des millions de copies de l'antivirus Avast qui utilise SQLite, des millions d'IPhone avec SQLite, etc.).

En tant que que base de données pure, je n'ai jamais été un fan de SQLite. Je lui préfère SQL Server, ou même à une époque plus lointaine Interbase (et ses variantes). Toutefois, avec le temps, SQLite s'est bonifié, sa stabilité est excellente, et, dans

certains cas, dont ceux qui nous intéressent ici, cette base de données devient un choix tout à fait pertinent.

Petit mais costaud

SQLite est … Light. Il ne faut pas en attendre toutes les possibilités de SQL Server, mais même SQL Server Compact ne fournit pas tous les services de sa grande sœur.

Mais l'essentiel est là : notamment le support total du transactionnel et d'ACID (en parlant des transactions : Atomique, Cohérente, Isolée et Durable).

J'ai pu me rendre compte avec un grand étonnement dans ces vingt dernières années à quel point les développeurs sont insensibles (ou si peu sensibles) à la notion de Base de Données Relationnelle et certainement encore mois à la notion (et à l'importance cruciale) de la bonne utilisation des transactions. J'ai arrêté depuis longtemps de faire des conférences sur les SGBD-R, les lois de Codd et le transactionnel car hélas ce ne sont plus des sujets qu'un éditeur aujourd'hui souhaite placer en tête dans ses manifestations. Hélas car je mesure encore presque chaque jour l'ignorance absolue des développeurs en ce domaine pourtant si sensible et il me semble qu'encore plus qu'hier il serait indispensable de former les gens à ces sujets.

Je republierai certainement d'anciens articles qui bien que poussiéreux à cause du temps sont toujours aussi brulant d'actualité dans leur contenu.

Mais revenons à SQLite. Elle a tout d'une grande, c'est sûr, au moins sur les fondamentaux. La gestion des transactions, vue comme un "plus" est en réalité un pré-requis in-négociable pour tout SGBD-R qui se respecte. Mais SQLite garantit que les transactions resteront ACID même en cas de crash ou de coupure de courant. Cela est en revanche un plus indéniable (même si de très vieilles bases comme Btrieve lancée en 1982 (!) firent leur publicité sur ce point il y a donc plus de vingt ans !).

Zéro configuration, c'est bien. Mais ce n'est pas une nouveauté non plus, c'était l'un des arguments massue de Borland Interbase face à SQL Server ou Oracle dès les années 80 (le produit connaitra un long parcours depuis sa création jusqu'à Ashton-Tate, les concepteurs de dBASE, jusqu'à Borland dans les années 90, puis jusqu'à Firebird un fork open source toujours utilisé et maintenu offrant de nombreux avantages proches de SQLite).

Implémentation de SQL 92. C'est le minimum en 2012 non ? Mais ne riez pas car cette norme ne reste que partiellement supportée par SQLite, comme ce fut le cas de Interbase ou même de Oracle qui n'est toujours full compliant à cette norme. Ce qui est gênant pour de grosses bases de données reste anecdotique pour SQLite censée être embarquée pour gérer de petits volumes de données.

La base de données est stockée dans un fichier unique (autre avantage déjà promu par Interbase et SQL Server par exemple). C'est aussi une obligation minimale de nos jours. Seule l'ancêtre dBASE gérait de multiples fichiers en réalité. Mais là aussi SQLite ajoute un petit plus… Le fichier de base de données unique a une structure disque cross-plateforme. C'est indéniablement un avantage pour des applications qui devraient tourner sur Windows Phone 8 et Android (avec MonoDroid par exemple).

SQLite support des bases de données de plusieurs Tera et des BLOB de plusieurs Giga (octets). Ce sont des limites théoriques mais elles montrent qu'une application taillée pour les unités mobiles n'aura aucun mal à supporter un passage sur PC avec des volumes traités plus gros (application Windows 8 Store notamment).

La rapidité de SQLite est aussi mise en avant. Il est vrai que dans le cadre de ses limitations le moteur ne se fatigue pas trop là où des moteurs plus riches perdent par force plus de temps à gérer les données. Mais pour le type d'application visé, la richesse fonctionnelle est un avantage qui s'efface largement devant celui de la célérité des opérations, ce qui place SQLite en bonne position.

Les API sont assez rustiques, elles ont l'avantage de tourner sur plusieurs plateformes ce qui oblige à un certain minimalisme. Toutefois vous n'aurez pas à travailler directement avec ces API, en tout cas passous .NET puisqu'il existe des packages Nuget offrant tout le nécessaire, ce que nous verrons plus loin.

Les sources sont disponibles. C'est bien. Mais franchement déjà se plonger dans un composant Infragistics ou autre réclame un sacré investissement alors se plonger dans le code ANSI-C de SQLite, à moins de vouloir créer un fork et y passer ses jours et ses nuits, cela n'a guère n'importance.

Bref, voilà dans les grandes lignes les points forts présentés par le site de SQLite lui-même (sans mes commentaires !).

Une base de données correcte pour un site Web, parfaite pour de petites applications qui doivent stocker des données, bien adaptée à la création de démos de grosses applications ou de maquettes. Mais c'est tout.

Le passage de SQLite vers SQL Server ne dispose d'aucune aide ni ne repose sur aucune compatibilité. Si votre application doit un jour voguer sur des océans de données partagées, il vous faudra changer SQLite pour autre chose de plus adapté et cela se fera dans la douleur, là où SQL Server Compact rend le passage pour ainsi dire invisible.

Mais SQL Server Compact n'est pas à mon goût encore assez simple à installer, utiliser et configurer. Ca marche, ça supporte même Entity Framework, mais SQLite est bien plus simple encore. Et pour une application Smartphone ou tablette cela me semble un argument très favorable à SQLite (sans oublier son avantage cross-plateforme x86, x64 et ARM).

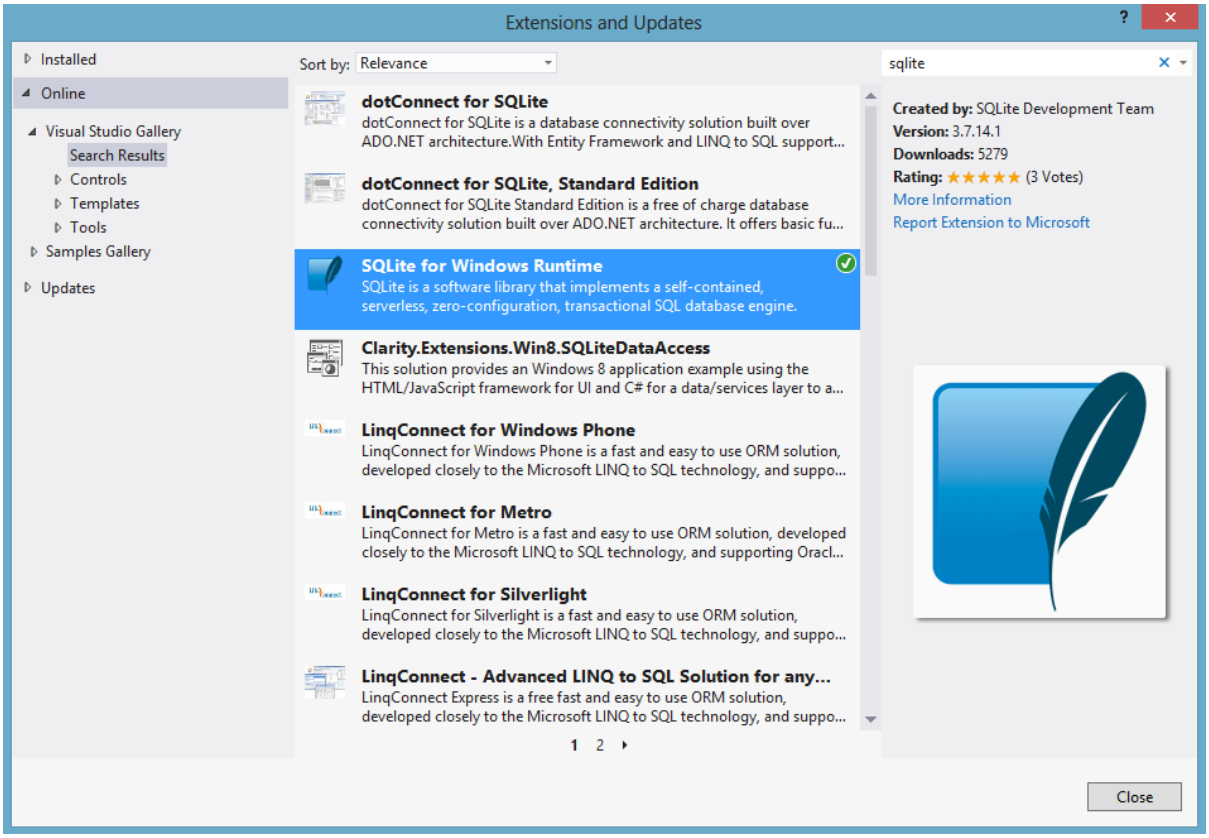
Enfin, SQLite tient dans environ 350 Ko, ce qui est négligeable de nos jours, même sur un Smartphone.

Après, c'est à chacun de juger, moi je ne fais que défricher le terrain. A vous de choisir votre route…

Installation

Il est possible de télécharger un "vsix" pour Visual Studio depuis le site de SQLite. Mais il existe bien plus simple puisque des packages Nuget existent et permettent d'installer le moteur de base de données dans l'application en ayant besoin. J'adore le concept des packages Nuget. Chaque application dispose ainsi de ses extensions conservées et mises à jour dans son propre espace disque. On peut versionner tout cela et reprendre plus tard une application pour la maintenir sans avoir à gérer l'état de l'installation des extensions sur la machine de développement. C'est vraiment parfait.

Si vous vous rendez sur la gestion des extensions Nuget depuis Visual Studio et que vous cherchez le terme SQLite pour serez surpris de voir de très nombreux packages.

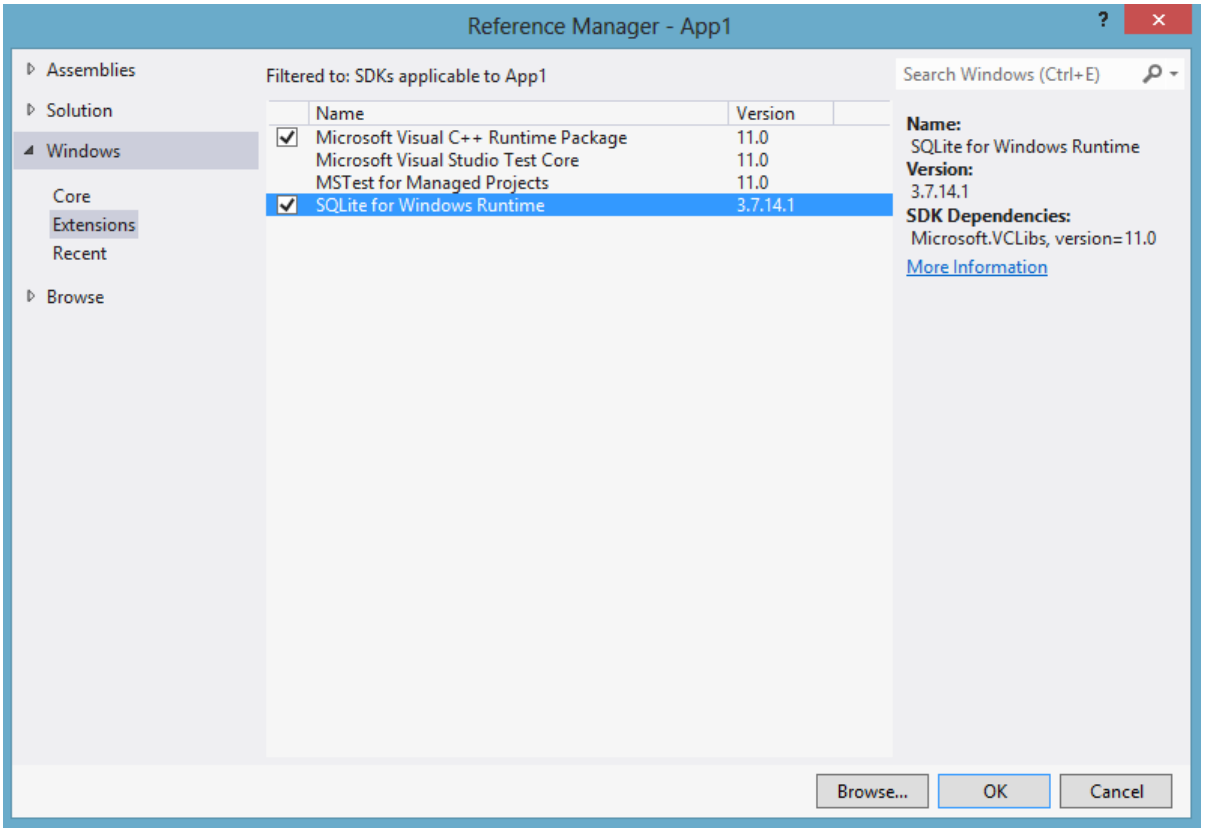


Il existe en effet plusieurs packages différents de différents auteurs permettant d'intégrer SQLite à une application. Par exemple dotConnect for SQLite vous apportera un support ADO.NET ainsi que celui de Entity Framework. Cela est un avantage certain pour designer un site Web ou une application Windows Store un peu "joufflue".

Ce qui nous intéresse pour le moment est la version des drivers pour WinRT. Si vous sélectionnez SQLite for Windows Runtime vous disposerez alors d'un bon pilote qui utilise uniquement les opérations P/Invoke autorisées par le market place. Car il y a forcément du P/Invoke entre le code Windows 8 et la DLL ANSI-C de SQLite… Et il faut être certain que ces opérations ne viendront pas créer un barrage infranchissable lors de la validation de l'application par Microsoft…

Configuration

Une fois le package Nuget installé, il ne reste plus qu'à ajouter la référence…



On notera qu'il faut ajouter aussi le Microsoft Visual C++ Runtime Package pour que tout fonctionne correctement.

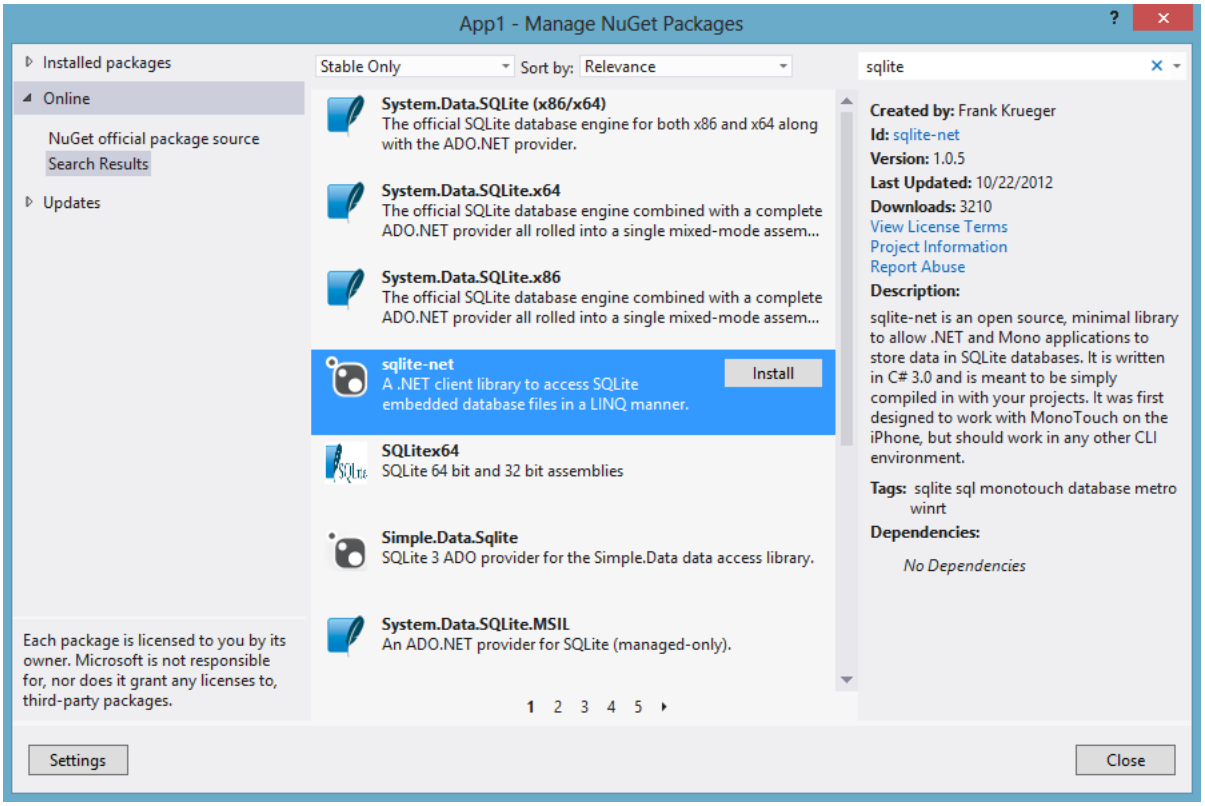
Petite astuce : n'essayez de construire une application en mode "Any CPU", cela ne marchera pas. Vous devez choisir un build x86, x64 ou ARM. Pour cela modifier les configurations aussi bien Debug que Release.

Autre point essentiel pour le market place : placez-vous toujours en mode Release avant de déployer vers le Windows Store !

Utilisation

SQLite est maintenant disponible dans votre application, il ne vous reste plus qu'à vous plonger dans sa documentation pour en tirer le meilleur parti…

Toutefois je vous conseille d'ajouter une petite couche qui rendra l'accès aux API plus agréable. Retournez dans les packages Nuget et sélectionnez sqlite-net créé par Frank Krueger.



Ce petit ajout en C# a été conçu au départ pour MonoTouch (donc pour fonctionner sur IPhone et Ipad), mais il est compilable aussi sous profile .NET en WinRT.

S'installant au-dessus du pilote SQLite, sqlite-net simplifie grandement l'accès à la base de données en offrant une API C# simple et pratique.

Conclusion

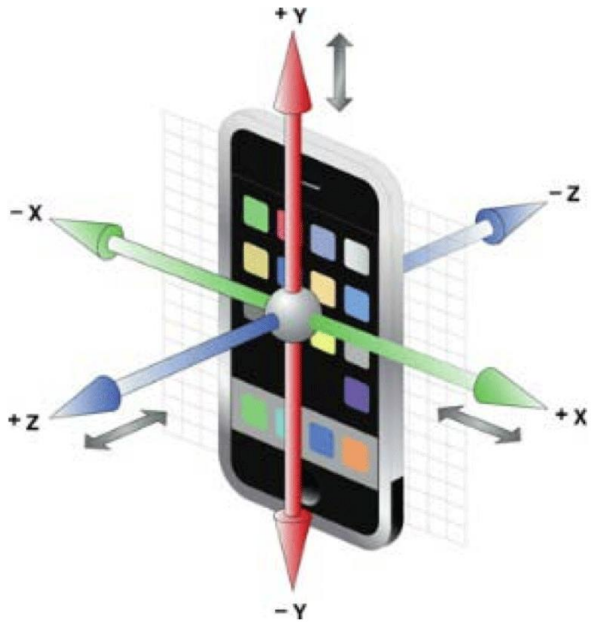
Sans être une panacée, SQLite est une excellente petite base de données relationnelle transactionnelle ne pesant pas très lourd, sans configuration, rapide et robuste. Elle est parfaitement taillée pour les applications Smartphone ou tablette et peut parfaitement être utilisée dans des applications Windows Store pour PC.

Portable, cross-plateforme, le même fichier disque peut être utilisé pour une application Windows 8, IPhone ou Android… Un gros plus si vous décidez de concevoir une application cross-plateforme fournie avec une base de données préchargée (type recettes de cuisine ou autre) puisque vous pourrez fournir le même fichier base de données dans toutes vos versions… Pour une stratégie de développement totalement cross-plateforme n'utilisant que C# et .NET, je renvoie le lecteur à mon billet en trois parties "Stratégie de développement cross-plateforme" (partie 1) ou au Tome de ALL DOT BLOG entièrement consacré à ce sujet.

Faites frémir les sens de Windows 8 ! (ou comment taquiner les capteurs)

Windows 8… Sur PC ? Non, plutôt sur Windows Phone 8 ou Surface, car question “sensibilité” les PC sont légèrement “autistes” comparés à leurs petits frères et sœurs. Compas, accéléromètre, GPS et j’en passe, les unités mobiles sont bourrées de capteurs en tout genre. Savoir chatouiller les sens de ces petites machines ouvre les voies d’un plaisir certain (de programmation) et surtout celles de nouvelles applications !

Toucher n’est pas jouer…



On les touche, on les caresses, certes, mais les tablettes n’ont pas que leur écran de tactile ! Ce sont de petits être très sensibles à tout plein de stimulations … Les Smartphones aussi, s’ils aiment jouer à guiliguili avec vos doigts ne sont pas en reste niveau sensoriel…

Tout cela existe plus ou moins de longue date, de façon éparse d’ailleurs, mais depuis l’avènement des Smartphones, puis des tablettes, on n’avait jamais vu autant de capteurs dans un si petit volume sauf dans James Bond ou Mission Impossible.

Pire, tous ces capteurs qui existaient séparément dans des machines diverses n’étaient programmables qu’en “bas niveau”, soit au travers de drivers exotiques, quant il ne fallait pas mettre soi-même les mains dans le cambouis pour les piloter via une connexion RS232 par exemple.

Mais ça, c’était “avant”.

Avant Windows 8. Et ses API fédératrices qui permettent de prendre en charge de façon cohérence la programmation haut niveau de tout un tas de capteurs de nature différente. Du capteur photo à l'accéléromètre en passant par le compas ou le GPS.

(pour être juste, cette fédération des API des capteurs existe depuis iOS et Android au moins, mais la fédération Windows 8 est cross-form-factor ce qui est une première).

Des tas d'exemples

Le Centre de Développement Windows 8 reprend l'excellente idée de Windows 7 avec une partie du site entièrement dédiée à des dizaines (centaines certainement) d'exemples portant sur tous les aspects de la programmation sous Windows.

Vous trouverez à l'adresse suivante une foule de choses passionnantes car chaque exemple est fourni avec son code et c'est là la clé d'une bonne compréhension : pouvoir tester, modifier, sans pour autant tout coder depuis le départ : http://code.msdn.microsoft.com/windowsapps/Windows-8-Modern-Style-App-Samples

Mais vous dire cela est un peu court, vous renvoyer vers des exemples du site Microsoft n'est pas la vocation de Dot.Blog qui aime vous donner directement matière à réfléchir.

Du code, tout de suite !

C'est pourquoi sans entrer dans les détails, puisque cela serait vraiment une redite de la documentation, je vous propose un tour d'horizon des capteurs et de leur programmation au travers d'exemples simples.

L'accéléromètre

Conçu de façon très astucieuse ce composant "électro-mécanique" (puisqu'il associe un peu de mécanique et de l'électronique) permet de savoir avec beaucoup de précision en général dans quelle direction l'appareil se déplace. Comme son nom l'indique il ne détecte pas une position, mais une accélération sur l'un des trois axes X, Y, Z.

Comme la majorité des capteurs, l'accéléromètre est un "périphérique" à lecture seule. On ne fait donc que lire son état sans pouvoir le modifier (pouvoir déplacer une unité mobiles dans les 3 axes juste par programmation serait très fort !).

Pour lire l'accélération sur les 3 axes :

```
async private void ReadingChanged(object sender,
AccelerometerReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        AccelerometerReading reading = e.Reading;
        var X  =  reading.AccelerationX;
        var Y  =  reading.AccelerationY;
        var Z =   reading.AccelerationZ;
    });
}
```

Rien de bien savant donc (juste se rappeler que WinRT est essentiellement asynchrone ce qui change un peu la façon d'appeler les API par rapport à .NET ou Win32).

Le compas

Le compas permet de ne pas perdre le Nord… Oui mais lequel ?

En effet il faut savoir que les compas peuvent pointer le Nord magnétique ou le Nord "vrai". Ce dernier est la position du pôle Nord, le premier est, comme son nom l'indique, la position du Nord magnétique de notre planète, point qui n'est pas réellement au pôle et qui se déplace dans le temps, au gré des fluctuations des mouvements du noyau au centre de la Terre. La plupart des compas sont magnétiques d'ailleurs. Le Nord "vrai" s'applique simplement en prenant en compte un facteur de correction.

Cela peut être trompeur puisque la présence d'un aimant un peu fort à côté de la machine (ou une grosse masse métallique) pourra fausser la lecture…

Les plus angoissés pourront craindre les inversions de pôles … De nombreuses sont intervenues plusieurs fois d'ailleurs dans l'histoire de notre boule de billard cosmique, cela se reproduira certainement. Toutefois ces changements, s'ils peuvent être brutaux à l'échelle géologique peuvent être considérés comme négligeables à l'échelle des étincelles que sont nos vies. Donc peu de chance que votre compas magnétique pointe le pôle Sud d'un seul coup pendant que vous êtes en train de vous promener en forêt vous égarant à l'inverse de la route que vous poursuiviez !

Lire le compas est tout aussi simple :

```
async private void ReadingChanged(object sender,
CompassReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        CompassReading reading = e.Reading;
        MagneticNorth.Text = String.Format("{0,5:0.00}",
reading.HeadingMagneticNorth);
        if (reading.HeadingTrueNorth != null)
        {
            TrueNorth.Text = String.Format("{0,5:0.00}",
reading.HeadingTrueNorth);
        }
        else
        {
            TrueNorth.Text = "Données absentes.";
        }
    });
}
```

Une fois encore la documentation Microsoft sera votre amie pour en savoir plus : http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.sensors.compass.aspx

Géolocalisation (GPS)

Peut-être la plus belle invention de geek avec le smartphone : le GPS met en œuvre à la fois la pointe de la technologie (celle de l'espace, des satellites) et la pointe de physique (car pour fonctionner les GPS doivent prendre en compte le ralentissement des horloges en mouvement propre à la théorie de la Relativité).

Côté récepteur je parle de géolocalisation et non de GPS car ce dernier n'est qu'un moyen parmi d'autres de trouver la position du demandeur. Windows peut utiliser une triangulation d'après les bornes Wifi ou même par l'IP. Bien entendu ce n'est pas Windows lui-même qui effectue ce genre de choses, mais les services assurés par des serveurs Microsoft sur le Web. Android de Google fonctionne sur le même principe (c'est pourquoi les Google cars s'intéressent tant aux bornes Wifi et qu'elles en ont collecté bien plus que leur simple localisation, pas forcément dans des buts inavouables d'ailleurs, au moins au départ).

L'accès au service de géolocalisation n'est donc pas à proprement parlé un accès à un capteur. C'est bien un service qui se cache derrière, et c'est dans la réponse de ce dernier qu'on peut connaître la précision de la mesure (forcément moins fiable par l'IP ou par triangulation Wifi que par le GPS embarqué).

A ce titre, et pour digresser comme j'en ai l'habitude, certains ont ronchonné sur le fait que Surface ne possède pas de GPS. Or on sait très bien qu'un GPS même dans une voiture a du mal à passer pour peu que le pare-brise soit traité "athermique". Et en intérieur c'est carrément impossible. Et si un smartphone peut être (et est) utilisé en extérieur, une tablette de type Surface est un instrument de loisir et de travail qui s'utilise dans son salon, au bureau, chez mamy ou l'oncle Georges pour lui faire voir les photos de vacances : que des lieux clos où le GPS ne passerait pas.

Dès lors cela ne veut pas dire qu'il n'y a pas pour autant de géolocalisation ! Le service fonctionne mais se base sur les bornes Wifi, sur l'adresse IP, etc, et peut retourner une position fiable même en campagne et surtout même à l'intérieur.

Lorsqu'on a compris que géolocalisation n'était pas synonyme de GPS, on peut mieux appréhender le choix très judicieux de Microsoft dans Surface (car cela baisse le prix de revient de la machine pour un capteur qui serait rarement utilisable).

Dans le code d'une application WinRT, l'accès au service de géolocalisation est tout aussi simple que pour les autres capteurs :

```
Geoposition pos = await _geolocator.GetGeopositionAsync().AsTask(token);
 Latitude.Text = pos.Coordinate.Latitude.ToString();
Longitude.Text = pos.Coordinate.Longitude.ToString();
Accuracy.Text = pos.Coordinate.Accuracy.ToString();
```

Le Gyromètre

Confondu à tort avec le Gyroscope, le Gyromètre a une finalité toute différente…

Le Gyromètre set à mesurer une vitesse angulaire, c'est à dire l'accélération d'un mouvement circulaire, souvent une rotation sur un axe donc. Le Gyroscope, de base, permet de maintenir une position de référence pour maintenir une assiette de vol (avion, fusée) ou la stabilité d'un navire de guerre de type porte-avion par exemple (qui équilibre son assiette pour éviter le crash des avions à l'atterrissage). Un Gyroscope permet certes de connaître les mêmes informations qu'un Gyromètre mais c'est un instrument couteux, lourd et encombrant car il réclame de faire tourner une masse équilibrée sur un système à trois degrés de liberté. La pièce tournante reste dans sa position de départ quoi qu'il arrive… C'est une référence "absolue" par rapport au moment de son démarrage. Le Gyromètre fournit la même information d'accélération sur les vitesses angulaires, de façon plus directe d'ailleurs et plus simple, mais en revanche il ne saurait jouer le rôle d'un référentiel fiable et fixe.

Bref, les unités mobiles embarquent un Gyromètre et non un Gyroscope, ce qui serait génial mais qui demanderait une brouette pour porter son smartphone et une charrette pour porter les batteries…

Les besoins ne sont pas non plus les mêmes, pour un smartphone ou une tablette il s'agit de reconnaitre une accélération non pas linéaire (comme l'accéléromètre peut le faire) mais une accélération "rotative" : faire pencher son téléphone de droite à gauche, le renverser, etc. On se fiche de la position qu'il occupait exactement au moment de son allumage, on travaille plutôt sur un mouvement relatif. Et ça tombe bien des puces Gyromètres peuvent tenir sans presque rien consommer ni prendre trop de place dans un smartphone ! Alors qu'un porte-avion dispose de toute la place pour embarquer un Gyroscope qui lui permettra de régler son assiette et facilité l'envol et l'atterrissage des avions même par grosse mer…

La société Safran a investi dernièrement 50 millions d'Euros dans une unité de production de Gyroscopes qui notamment équipent le Charles De Gaulle ou nos sous-marins de dissuasion nucléaire… Gyromètre de smartphone et Gyroscope de porte-avion ne concourent définitivement pas dans la même catégorie :-)

Après cet éclairage de pure "culture G" nous voici au code :

```
async private void ReadingChanged(object sender,
GyrometerReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
        () =>
    {
        GyrometerReading reading = e.Reading;
        var X = reading.AngularVelocityX;
        var Y = reading.AngularVelocityY;
        var Z = reading.AngularVelocityZ;
    });
}
```

Je doute que le porte-avion C.d.Gaulle ou que nos sous-marins nucléaires soient programmés de façon aussi triviale !

(plus d'info : http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.sensors.gyrometer.aspx)

Inclinomètre

Du mouvement, toujours du mouvement…

L'inclinomètre fourni des informations de mouvement assez semblable aux autres capteurs de ce type que nous venons de voir, mais c'est la nature de l'information retournée qui est différente et mieux adaptée à certaines utilisations.

L'inclinomètre retourne les informations de tangage, roulis et de lacet de l'appareil. Cela permet de déterminer l'orientation de l'appareil par rapport au sol (ou plus exactement par rapport à la direction dans laquelle la gravité agit).

C'est une information importante pour savoir si l'unité mobile est inclinée ou tordu par rapport à la verticale du lieu.

La NASA fournit des explications plus précises sur ces notions très utilisées en aviation : http://www.grc.nasa.gov/WWW/K-12/airplane/rotations.html

Vous commencez à vous en douter, lire les informations de l'inclinomètre s'effectue aussi simplement que pour les autres capteurs :

```
async private void ReadingChanged(object sender,
InclinometerReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
        () =>
    {
        InclinometerReading reading = e.Reading;
        var X = reading.PitchDegrees;
        var Y = reading.RollDegrees;
        var Z = reading.YawDegrees;
    });
}
```

Moins spatiale que la NASA, la doc de Microsoft vous sera peut-être utile aussi : http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.sensors.inclinometer.aspx

Détecteur de luminosité

C'est un capteur très important pour le confort de l'utilisateur … et la consommation de l'unité mobile. En effet ce capteur mesure la quantité et la qualité de la lumière ambiante. Ce qui permet en retour de régler automatiquement l'éclairage de l'écran qui, on le sait, est un ogre pour la batterie de ces machines.

La lecture est tout aussi directe :

```
async private void ReadingChanged(object sender,
LightSensorReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
        () =>
    {
        LightSensorReading reading = e.Reading;
        var Lux =  reading.IlluminanceInLux;
    });
}
```

La encore la doc Microsoft sera un allié pour entrer dans les détails : http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.sensors.lightsensor.aspx

Le capteur d'orientation

Encore un capteur de mouvement ? Oui et non…

Le capteur d'orientation retourne une matrice qui représente la rotation et un *quaternion* qui peut être utilisé directement pour ajuster la perspective de l'utilisateur dans l'application… A la différence des autres capteurs, plus simples, le capteur d'orientation est principalement utilisé dans les jeux en 3D pour ajuster le rendu graphique selon l'angle sous lequel le joueur voit l'écran pour un réalisme et une réactivité accrus.

Le quaternion est une notation particulière permettant de retourner à la fois les orientations et les rotations.

Lire ce capteur est guère plus difficile que les autres, interpréter ses valeurs et s'en servir réclame juste un goût plus prononcé pour la géométrie dans l'espace !

```
async private void ReadingChanged(object sender,
OrientationSensorReadingChangedEventArgs e)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
        () =>
    {
        OrientationSensorReading reading = e.Reading;
        // Quaternion values
        SensorQuaternion quaternion = reading.Quaternion;
        ScenarioOutput_X.Text = String.Format("{0,8:0.00000}",
            quaternion.X);
        ScenarioOutput_Y.Text = String.Format("{0,8:0.00000}",
            quaternion.Y);
        ScenarioOutput_Z.Text = String.Format("{0,8:0.00000}",
            quaternion.Z);
        ScenarioOutput_W.Text = String.Format("{0,8:0.00000}",
            quaternion.W);
        // Rotation Matrix values
        SensorRotationMatrix rotationMatrix = reading.RotationMatrix;
        ScenarioOutput_M11.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M11);
        ScenarioOutput_M12.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M12);
        ScenarioOutput_M13.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M13);
        ScenarioOutput_M21.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M21);
        ScenarioOutput_M22.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M22);
        ScenarioOutput_M23.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M23);
        ScenarioOutput_M31.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M31);
        ScenarioOutput_M32.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M32);
        ScenarioOutput_M33.Text = String.Format("{0,8:0.00000}",
            rotationMatrix.M33);
    });
}
```

La doc Microsoft discute plus en profondeur de la classe OrientationSensor : http://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.sensors.orientationsensor.aspx

Conclusion

Nos petites machines sont bourrées de technologies à faire pâlir d'envie M. Spock… Pourtant tout cela est "in the box" et peut être utilisé de façon simple et transparente par tous les développeurs grâce à Windows 8.

Si tourner les pages d'un livre sur un mouvement rapide de rotation, si inverser l'image présentée lorsque le téléphone est placé bas en haut, ou toutes ces autres

utilisations des capteurs sont désormais des classiques. Je suis convaincu que les développeurs les plus rusés sauront trouver des applications utiles ou divertissantes bien loin de ce que pensaient les ingénieurs ayant conçu les puces !

Sky is the limit… N'est ce pas.

Réalité augmentée, retracer un parcours exact sur une carte précise, détecter le vol de votre mobile et pister le voleur dans les rues; déduire des accélérations s'il est à pied, en mobylette ou un Ferrari, créer un code de déblocage du mobile totalement gestuel, se servir d'un smartphone pour créer sa propre fusée et satelliser à moindre frais les cendres de son chien ou un message d'amour à sa chérie, vérifier la qualité et les progrès d'un entrainement physique sans présence d'un coach, toutes les applications des plus sérieuses aux plus loufoques peuvent être envisagées.

Ce ne sont pas les plus sérieuses qui se vendent le mieux nous ont appris les différents market places… Alors… Faites marcher vos neurones !

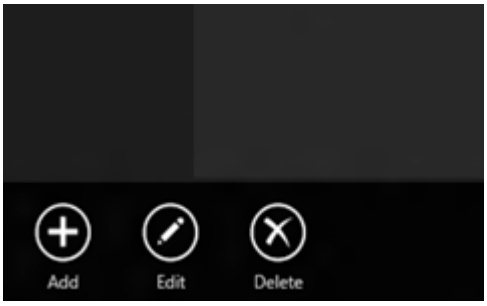
Windows Store Apps : Comment créer et gérer une AppBar

Dans une application Windows Store l'`AppBar` est cette barre de boutons qui apparaît en bas de l'écran et qui remplace dans l'esprit les menus contextuels (popup) de Win32. Il existe une variante en haut de l'écran dédiée plutôt à la navigation. C'est un élément essentiel de l'UI Windows Store sous Windows 8.

Le rôle de la AppBar

Dans une application "Metro" l'AppBar est une zone en bas de l'écran qui reste cachée et qui ne s'affiche que lorsque cela est nécessaire pour présenter des boutons : navigation, commandes, outils, il n'y a pas forcément de limite même si quelques guidelines doivent être respectées.

Bien qu'assez différente techniquement des popups Win32 déclenchés sur le clic-droit l'AppBar joue un rôle assez similaire. Les `Flyout` sont plus proches visuellement des popups, mais faute de temps les équipes de Microsoft n'ont pas encore ajouté ce composant Html/WinJS au profil C#/Xaml (il en existe néanmoins une version non officielle C# sur CodePlex, le CharmFlyout, disponible aussi en package nuget). Windows 8.1 introduit un Flyout mais cette librairie offre d'autres avantages, à chacun de choisir.



L'AppBar se situe généralement en bas de l'écran et c'est là que je vous conseille de la laisser par souci d'homogénéité avec les règles de Modern UI (comme sous Windows Phone donc). Mais l'AppBar peut aussi apparaître en haut de l'écran, voire les deux à la fois !

Dans ces cas particuliers on voit généralement l'AppBar du bas rester dédiée aux commandes alors que celle du haut sera plutôt utiliser pour naviguer par exemple (ce qui fait partie des guidelines). Mais là encore la liberté est assez grande, l'essentiel est que cela ait un sens pour l'utilisateur et que la cohérence et la qualité de l'UX soient préservées.

Comment ajouter une AppBar ?

C'est en réalité très facile puisqu'autant la barre du haut que celle du bas sont des propriétés de l'objet `Page`.

On trouve ainsi les propriétés `TopAppBar` et `BottomAppBar` dans l'objet `Page` et il est facile de les renseigner par un contenu quelconque, les deux propriétés étant de type `AppBar`, type dérivant lui-même de `ContentControl` (et si on remonte le courant tel un saumon : `Control`, `FrameworkElement`, `UIElement`, `DependencyObjet`, et `Object` !).

Une `AppBar` peut être créée par code C# ou Xaml, cela ne fait aucune différence. Le code peut changer une AppBar à sa guise (ce qui en fait un remplaçant du popup contextuel Win32) pour adapter le contenu au contexte.

Le contenu de l'AppBar est totalement libre même si, selon les guidelines, il s'agira principalement de boutons.

N'étant qu'un `ContentControl`, l'AppBar peut se définir par tout code Xaml valide possédant un objet racine qui contient tout le reste. Généralement il s'agit d'une `Grid` sans aucune propriété (donc qui prendra tout l'espace disponible par défaut) dans laquelle on retrouve des `StackPanel` contenant les boutons.

Je dis “des” `StackPanel` car il est très fréquent de voir deux groupes de boutons dans une AppBar : une série sur la gauche, l’autre sur la droite. Les `StackPanel` sont tout à fait indiqués pour mettre en page des séries d’objets surtout de même taille comme ici. Un `StackPanel` est ferré à gauche, l’autre à droite.

A cela deux raisons : la séparation est (doit être) sémantique (les deux groupes de boutons doivent impliquer deux groupes logiques de commandes), et s’agissant d’applications tournées vers la mobilité il est plus aisé de fournir un groupe pour le pouce gauche et un autre pour le droit plutôt qu’une longue barre ayant des boutons au centre qui seront difficilement accessibles (le Design c’est aussi de l’ergonomie ne l’oublions pas !).

Mais là encore la liberté est assez grande tant que les choix font sens avec le contexte.

Le code Xaml suivant définit une AppBar (dans un objet Page) :

```
<Page.BottomAppBar>
    <AppBar x:Name="bottomAppBar" Padding="10,0,10,0">
        <Grid>
            <StackPanel Orientation="Horizontal"
HorizontalAlignment="Left">
                <Button Style="{StaticResource EditAppBarButtonStyle}"
                        Click="Button_Click"/>
                <Button Style="{StaticResource RemoveAppBarButtonStyle}"
                        Click="Button_Click"/>
                <Button Style="{StaticResource AddAppBarButtonStyle}"
                        Click="Button_Click"/>
            </StackPanel>
            <StackPanel Orientation="Horizontal"
                        HorizontalAlignment="Right">
                <Button Style="{StaticResource RefreshAppBarButtonStyle}"
                        Click="Button_Click"/>
                <Button Style="{StaticResource HelpAppBarButtonStyle}"
                        Click="Button_Click"/>
            </StackPanel>
        </Grid>
    </AppBar>
</Page.BottomAppBar>
```

Ce qui, visuellement, se traduira par :

Partager une AppBar au travers des pages

Bien que l'AppBar soit souvent contextuelle, cela n'est pas une obligation. Il peut être parfois plus pertinent d'offrir la même `AppBar` à toutes les pages d'une application.

Je ne détaillerai pas la technique mais juste l'esprit : Dans un tel cas on créée une page racine possédant une Frame dans laquelle les autres pages de l'application navigueront. C'est cette page racine qui possède la AppBar. Elle override aussi le `OnNavigateTo` pour le "rerouter" les URI vers la Frame intérieure.

L'illusion est alors parfaite. Mais attention, cela se pense "avant". L'ajouter "après" peut remettre en cause certains choix. Une application Modern UI doit toujours être designée avant de commencer à code !

Ce n'est pas parce que j'en parle moins ces temps-ci, actualité oblige, que le Design n'a plus d'importance… Vous pouvez facilement retrouver tous les billets sur ce sujet en interrogeant le tag "design" de Dot.Blog : http://www.e-naxos.com/Blog/?tag=/design, de même qu'un Tome de ALL DOT BLOG est consacré à se sujet (ainsi qu'une vidéo sur la chaîne YouTube TheBotBlog reprenant ma conférence aux TechDays sur le sujet).

Ouverture et Fermeture

L'AppBar possède une propriété `IsOpen` dont le nom est clair… Lorsque la propriété est mise à `True` la barre s'ouvre, à `False`, elle se referme.

Les AppBar sont des objets à "*light dismiss*", c'est à dire qu'ils disparaissent facilement dès qu'on clic (ou tape avec le doigt) en dehors de leur zone. Il est possible de forcer l'ouverture d'une AppBar au premier affichage d'une page en mettant `IsOpen` à `True`. Dès que l'utilisateur tapera ailleurs ou qu'il aura cliqué sur l'un des boutons la AppBar se cachera.

Cela peut être un moyen simple de montrer les possibilités d'une page sans aucune autre explication. L'utilisateur voit ce qui est possible de faire, nul besoin de 15 pages de docs pour cela… Certaine pages peuvent aussi s'ouvrir en ayant immédiatement besoin d'une commande utilisateur (par exemple afficher une image prise à la webcam et afficher deux boutons "ok" / "recommencer"). L'ouverture immédiate de la AppBar trouve ainsi sa place dans de nombreux scénarios (mais cela ne s'invente pas pendant qu'on code, il faut y avoir pensé avant sinon gare non plus au "code spaghetti" mais au "design spaghetti" ce qui est un crime tout aussi condamnable !).

Figer la barre

Une AppBar disparait dès que l'utilisateur interagit avec des éléments de l'application en dehors de l'espace de la barre elle-même. Mais il est possible de figer la barre pour qu'elle ne disparaisse pas.

Pour cela une seule propriété : `IsSticky` qu'on peut placer à `True` aussi bien en Xaml qu'en C# (à condition d'avoir ajouté un "`x:Name`" pour pouvoir référencer la barre dans le code dans ce dernier cas).

Les évènements d'ouverture et fermeture

Il est très simple de réagir aux évènements d'ouverture et de fermeture d'une AppBar car elle expose deux events : `Opened` et `Closed`. Comme les noms le laissent supposer, ces évènements sont des "constats" et non des "avertissements". J'entends par là que l'action est déjà consommée, le code ne peut que constater que cette action s'est déroulée. Un event d'avertissement serait du type OnClose, permettant par exemple d'agir avant que l'ouverture ne soit définitivement opérée, voire de l'annuler. Ce n'est pas le cas ici.

L'argument de ces events est d'ailleurs un "object" de base, donc sans information (ni en entrée, ni en sortie).

Les styles cachés des boutons

Les boutons d'une AppBar sont très simples, ils sont généralement rond avec une icône de couleur unie. Vous pouvez utiliser des banques d'icônes, ou des outils comme Metro Studio dont la version gratuite est très sympa (elle permet aussi de partir d'une fonte de symboles quelconque pour fabriquer une icône ce qui est très utile).

Mais il existe aussi quelques boutons "tout fait" cachés dans la feuille de style Xaml accrochée par défaut à un projet Windows Store.

Ce dictionnaire de styles se trouve dans le répertoire `Common` du projet (`StandardStyle.xaml`). En navigant vers les lignes 403 et suivantes vous découvrirez des définitions de styles pour des boutons. Mais ce code est commenté donc non opérant.

Pour vous en servir supprimez simplement les balises de commentaire !

On trouve ainsi des boutons "`Skip back`", "`Skip ahead`", "`Play`", "`Pause`", etc.

Les propriétés définies dans ces styles peuvent être changées et vous pouvez aussi ajouter de la même façon vos propres styles de boutons. La création d'un bouton devient alors très simple dans le code même de la AppBar. Par exemple :

```
<Page.BottomAppBar>
   <AppBar x:Name="bottomAppBar" Padding="10,0,10,0">
      <Grid>
         <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
            <Button Style="{StaticResource OpenFileAppBarButtonStyle}"
                    Click="Open_Click"/>
         </StackPanel>
         <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
            <Button Style="{StaticResource PlayAppBarButtonStyle}"
                    Click="Play Click"/>
            <Button Style="{StaticResource PauseAppBarButtonStyle}"
                    Click="Pause_Click"/>
         </StackPanel>
      </Grid>
   </AppBar>
</Page.BottomAppBar>
```

On voit dans ce code les styles directement appliqués à des instances de Button sans aucun besoin de spécifier d'autres propriétés (en dehors de la gestion du clic qui sera avantageusement remplacé par une commande dans le cadre du pattern MVVM).

Le dictionnaire de styles par défaut définit les styles de button de cette façon (exemple du style pour créer un bouton "play") :

```
<Style x:Key="PlayAppBarButtonStyle" TargetType="ButtonBase"
BasedOn="{StaticResource AppBarButtonStyle}">
        <Setter Property="AutomationProperties.AutomationId"
                Value="PlayAppBarButton"/>
        <Setter Property="AutomationProperties.Name" Value="Play"/>
        <Setter Property="Content" Value="&#xE102;"/>
</Style>
```

Les principales propriétés touchées sont :

`AutomationProperties.AutomationId`

Cela permet tout bêtement de donner un ID ré-exploitable pour la gestion de commandes.

`AutomationProperties.Name`

On indique ici le nom qui pourra s'afficher sous le bouton. C'est une indication très utile pour l'utilisateur car parfois l'icône, même bien choisi, n'est pas forcément

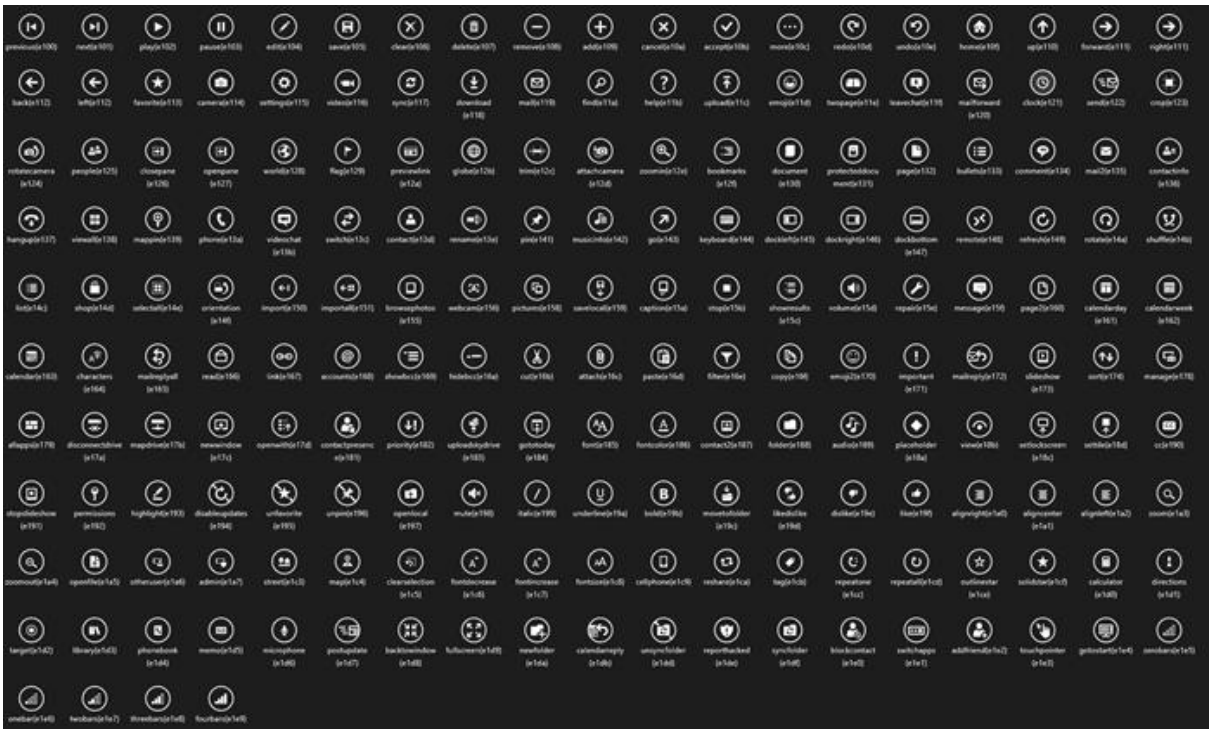
totalement évident. Une fois encore, une interface Modern UI est auto-explicative. On doit pouvoir utiliser le soft sans aucune aide autre que ce qu'on voit.

J'insiste là dessus car je sais à quel point pour nombre de développeurs c'est un saut énorme à franchir que de penser de cette façon… Le réflexe "*l'utilisateur n'a qu'à lire la doc*" est un réflexe de dinosaure qui ne vit pas avec son temps. Je le rencontre encore assez souvent hélas, ne serait-ce qu'indirectement quand je vois le "design" que certains développeurs prévoient pour leurs applications et où il est clair '(la seule chose qui le soit d'ailleurs!) qu'il faudra une longue formation des utilisateurs finaux pour s'en sortir ! Les lecteurs de Dot.Blog, grâce à mes divers billets de sensibilisation sur le Design ne sont pas comme ça, je le sais bien, mais les vieux réflexes reviennent parfois sournoisement, alors une piqûre de rappel ne coûte rien :-)

Content

C'est bien entendu l'icône centrale du bouton qu'on fournit au bouton à cet endroit.

Tout contenu Xaml est valide (un Path, un PNG…). Toutefois le système tel qu'il est fait utilise habilement la fonte "Segoe UI Symbol", autant que faire se peut, utilisez la même astuce, la fonte est déjà chargée et ses symboles sont reconnus facilement. Sinon utilisez des images ou du Xaml (Metro Studio ou le Noun Project qui est orienté SVG mais très utile tout de même, ou faites appel à un infographiste si le projet le nécessite !).



Bien entendu il reste possible de définir vos boutons d'une autre façon, tant que cela reste esthétique et cohérent avec le look Modern UI et ses guidelines, vous avez le champ libre.

Conclusion

La gestion des AppBar dans les applications est d'une redoutable simplicité mais joue un rôle central dans la qualité de l'UX.

Ne négligez pas ni la conception de styles propres et cohérent, ni la logique et l'ordre d'affichage des boutons, et encore moins le choix des icônes,

Finalement, une bonne AppBar est une AppBar bien conçue (Lapalissade), toute la difficulté est dans le "bien conçue" !

Metro pour Windows 8 est riche de bien d'autres surprises, cela permet déjà de prévoir d'autres billes à venir !

Faites griller les toasts ! … Avec Windows 8

Ah l'odeur du toast chaud le matin… Mais que les gourmands gardent leur salive, ceux dont je vais vous parler ne nourrissent pas le corps mais interpellent l'utilisateur, comme des toasts bondissant du toaster. Si les toasts Windows 8 ne brulent pas et sautent pas en dehors de l'écran, ils jaillissent de la droite pour afficher un message. Rangez la poêle pour les œufs brouillés et lancez Visual Studio !

Comment remplacer les fenêtres ?

Sous WinRT, Windows a perdu son "S" et beaucoup d'ailleurs demandent son retour, sans "S" Windows n'est pas utilisable sur les grands écrans de PC parfois multi-monitor en plus. Du full-screen sur tablette ou smartphone c'est parfait (et encore, Samsung ou LG n'ajoutent-ils pas le multitâche avec séparation de l'écran depuis que les smartphones ont des diagonales importantes ?), mais sur des écrans larges de PC c'est bien entendu un contresens que Microsoft devra résoudre assez vite car il freine à mon sens l'adoption de WinRT. De multiples occasions dans le cadre du travail journalier réclament l'affichage de plusieurs fenêtres qu'on superpose, exactement la métaphore du « bureau » sur lequel on place les documents qu'on utilise… Vouloir tuer cette métaphore sans proposer rien de mieux que le retour au full-screen de MS-DOS est forcément voué à l'échec.

> *Je ne m'étendrais pas sur les millions d'exemples qui attestent de ce besoin, chacun a les siens j'en suis convaincu, mais par exemple en ce moment même j'ai un écran qui montre un groupe de caméras de surveillance et un module de surveillance de mes serveurs, un autre qui affiche un player avec sa playlist (et je veux l'avoir sous les yeux car j'écoute des radios internet et parfois lire le titre de ce qui passe m'intéresse), écran qui affiche aussi un browser pour mes différentes recherches lorsque je tape le présent texte qui se trouve sur l'écran central (un troisième donc) dans Word avec dessous un explorateur ouvert sur un répertoire dont j'ai besoin et sur les côtés des icones pour certains softs que j'utilise souvent. J'ai mille autres exemples d'utilisation de ce genre et si j'utilise Windows 8.1 c'est avec Classic Shell et en boot sur bureau classique… Je ne comprends pas que personne chez Microsoft ne comprenne ce problème…*

Certes il ne faudrait toutefois pas réduire la vision du futur aux seules habitudes du passé… Evoluer c'est savoir abandonner ses habitudes. Si cela concerne la façon dont on fait les choses cela ne doit pas modifier la possibilité de faire ce qu'on veut.

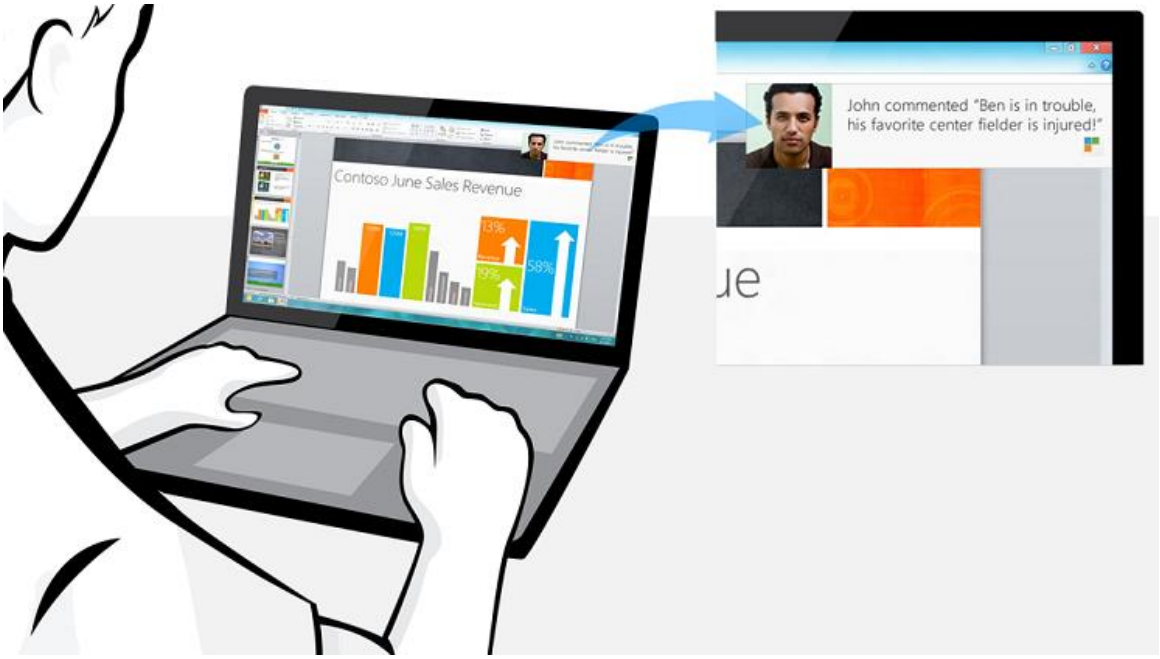
La modernité consiste bien entendu aussi à savoir repenser le passé autrement plutôt que de réitérer les mêmes schémas. Et dans certaines conditions, WinRT propose des stratégies nouvelles qui rendent caduques les anciennes façons de faire. Il faut juste savoir se projeter dans une nouvelle façon de penser les choses.

Glissements de page, Charmes, AppBar, Flyouts, et Toasts sont autant de nouveaux éléments qui doivent entrer dans la composition de vos applications et de leur Design pour trouver des solutions nouvelles à des problèmes anciens.

> *Ce qui gêne c'est d'avoir justement à faire l'effort d'essayer, d'inventer de nouvelles choses pour faire quelque chose qui existe déjà et dont tout le monde est content sans apporter réellement rien de nouveau en dehors de la nouveauté elle-même… Le nouveau pour le nouveau c'est la définition de la mode, passagère, changeante et sans grand intérêt. Le nouveau pour satisfaire de nouveaux besoins c'est la définition du progrès. Windows 8 Modern UI se situe-t-il dans le clan des modes ou du progrès ? L'avenir et les utilisateurs seront les juges de paix qui trancheront !*

Toasts

Les toasts WinRT ne croustillent pas, mais ils jaillissent depuis le bord droit de l'écran sous la forme d'un bandeau affichant généralement un message d'alerte (au sens large). L'application peut vouloir prévenir l'utilisateur qu'une erreur s'est produite tout comme elle peut avoir besoin d'indiquer qu'une tâche de fond est terminée, que de nouvelles données sont disponibles, …



Ces toasts ne sont vraiment pas de ceux qu'on aimerait avoir au petit-déjeuner : ils s'effacent tous seuls au bout d'un moment s'il n'y a aucune action utilisateur !

De même, s'ils sont généralement servis chauds, c'est à dire dès leur création, ils peuvent aussi être servis froids, c'est à dire après un délai précisé lors de leur création.

Des toasts froids qui s'évanouissent tous seuls, les gourmets seront certes déçus, mais pour le Designer c'est une aubaine car ils remplacent avantageusement et bien plus élégamment les tristes fenêtres des boites de dialogue (obligeant l'utilisateur à cliquer pour le "*dismiss*").

Toasts locaux et distants

Les toasts WinRT sont décidemment étranges. Ils peuvent être locaux "*in-app*" ou venir de l'extérieur. Un peu comme si votre voisin vous balançait deux ou trois tartines par la fenêtre…

Les toasts locaux, "in-app" sont ceux produits par l'application en cours.

Les toasts distants sont "*cloud based*" généralement déclenchés par une notification extérieure (un push ou quelque chose de ce type).

Ici je ne m'intéresserai qu'aux premiers, beaucoup plus simples à mettre en place, il sera toujours temps d'aborder à nouveau le sujet pour les seconds dans un prochain billet !

Un choix de toasts assez large

Enfin une bonne nouvelle pour les gourmands : les toasts sont fournis en plusieurs saveurs !

Il existe en effet toute une gamme de toasts, des plus légers aux plus lourds bien chargés d'information, un peu comme les tuiles (qu'on peut garder pour le dessert).

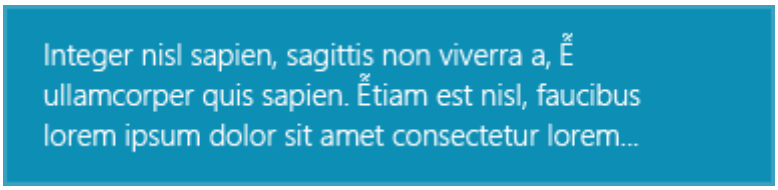
D'ailleurs comme les tuiles les toasts se définissent par un bout de XML.

Toutes ces saveurs font l'objet de plusieurs templates fournit par WinRT.

Les toasts à texte seul

`ToastText01`

Une chaine simple qui peut s'étendre sur 3 lignes maximum.

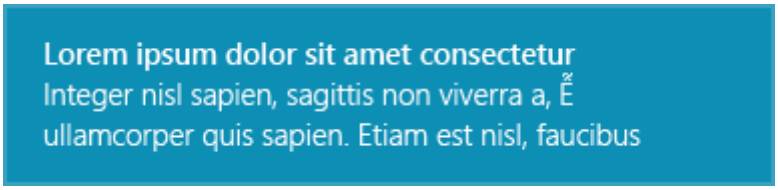


Le code XML étant :

```
<toast>
    <visual>
        <binding template="ToastText01">
            <text id="1">bodyText</text>
        </binding>
    </visual>
</toast>
```

`ToastText02`

Une chaine simple en gras, puis une chaine simple pouvant s'étendre sur deux lignes.

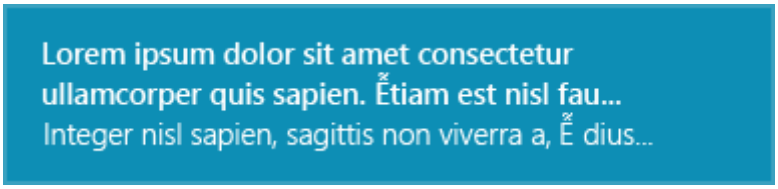


Le code XML :

```
<toast>
    <visual>
        <binding template="ToastText02">
            <text id="1">headlineText</text>
            <text id="2">bodyText</text>
        </binding>
    </visual>
</toast>
```

ToastText03

Une chaine simple de texte en gras qui s'étend sur les deux premières lignes et une chaine simple sur la troisième ligne.

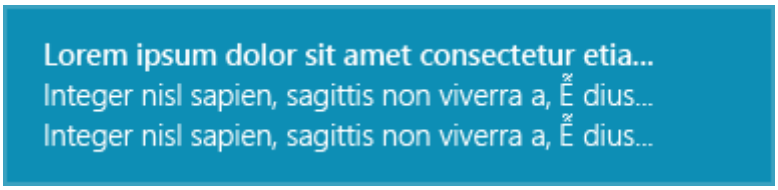


Le Code XML :

```
<toast>
    <visual>
        <binding template="ToastText03">
            <text id="1">headlineText</text>
            <text id="2">bodyText</text>
        </binding>
    </visual>
</toast>
```

ToastText04

Une ligne de texte en gras sur la première ligne, pouvant être tronqué, idem pour les deux lignes suivantes (deux chaines distinctes éventuellement tronquées, sans gras).



Le code XML :

```
<toast>
    <visual>
        <binding template="ToastText04">
            <text id="1">headlineText</text>
            <text id="2">bodyText1</text>
            <text id="3">bodyText2</text>
        </binding>
    </visual>
</toast>
```

Les autres templates

Comme on vient de le voir, il existe quatre templates aux nuances subtiles entre lesquels bien choisir n'est pas toujours évident.

On remarque aussi que ces toasts ont un visuel qui est défini en XML et non pas en XAML… WinRT lui-même n'est pas basé sur XAML, ce qui lui permet d'ailleurs de faire fonctionner des applications Html/Js sans état d'âme.

On regrettera forcément la grande cohérence qu'offrait .NET, puisqu'ici du XML jouant le rôle de XAML vient se mélanger à des langages graphiques comme XAML ou HTML, et des langages de code comme Js ou C++.

Le choix laissé au développeur est un aspect essentiel de WinRT, comme il l'était sous .NET, mais on sent bien que le clan Sinofksy a imposé sa vision C++/HTML/JS/XML en opposition franche à ce qui existait et qui marchait très bien (C#/VB/XAML) sans qu'aucun supplément de service réel ne découle de cette mise en opposition. Il s'agit d'une guerre fratricide jetant le trouble plus que servant les intérêts des développeurs et même ceux de Microsoft embarqué dans la maintenance de langages dont .NET se passait allègrement, ce qui aurait pu rester longtemps ainsi pour le plus grand bien d'une situation stable, seule apte à favoriser le business…

Mais, grâce à quelques intelligences dans la place, C#, Xaml et .NET restent des citoyens de premières classe dans cet ensemble un peu bancale. Ne nous plaignons pas ! … Même si pour préparer nos toasts il faudra en passer par du XML qui ressemble à du XAML et non du XAML…

Les templates présentés plus haut peuvent paraitre "tristounet". Il est vrai qu'ils sont en mode texte pur, sans beaucoup d'ornementation… WinRT base sont UI sur Modern UI dont le principe est le dénuement le plus total, cela n'est donc pas paradoxal.

Toutefois il peut s'avérer qu'au moins une icône ou une photo amène un peu de couleurs et de précision visuelle au message du Toast.

Microsoft y a pensé, bien entendu.

Les templates mixtes image / texte

Il existe donc pour ces situations des templates mixtes permettant d'afficher une image et du texte.

Dans ces templates les images sont définies en utilisant quatre syntaxes différentes selon d'où provient la source :

`http://` ou `https://` pour une image Web

`ms-appx://` pour une image inclus dans le package de l'application

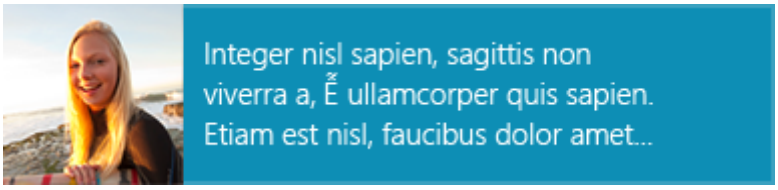
`ms-appdata:///local/` pour une image sauvegardée sur le stockage local

`file:///` pour une image locale uniquement pour les applications desktop

Une fois cette convention connue les quatre templates ressemblent beaucoup aux quatres précédents, sauf la présence d'une image en plus.

`ToastImageAndText01`

Une image et une ligne simple de texte qui peut s'étendre sur 3 lignes :

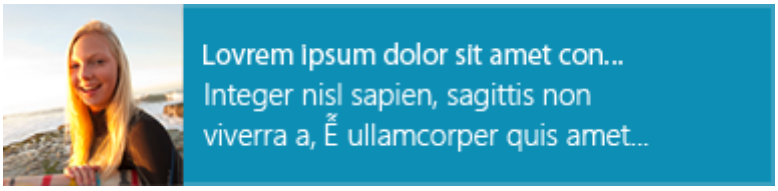


Le code XML :

```
<toast>
    <visual>
        <binding template="ToastImageAndText01">
            <image id="1" src="image1" alt="image1"/>
            <text id="1">bodyText</text>
        </binding>
    </visual>
</toast>
```

`ToastImageAndText02`

Une image, une chaine en gras, une chaine simple qui peut s'étendre sur 2 lignes :

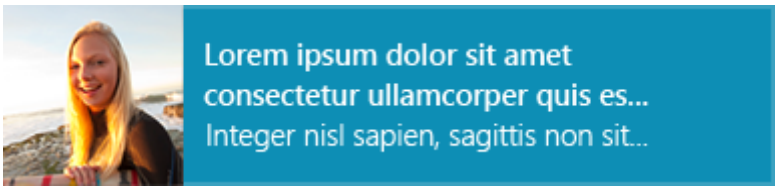


Le code XML :

```
<toast>
    <visual>
        <binding template="ToastImageAndText02">
            <image id="1" src="image1" alt="image1"/>
            <text id="1">headlineText</text>
            <text id="2">bodyText</text>
        </binding>
    </visual>
</toast>
```

ToastImageAndText03

Une image, une chaine en gras qui peut occuper 2 lignes et une ligne simple de teste sur la 3eme ligne :

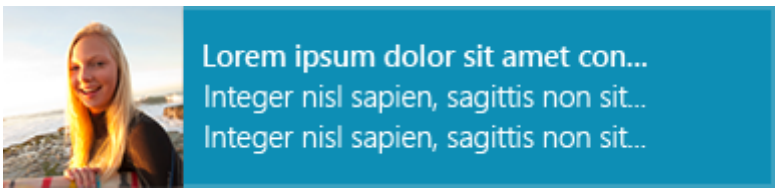


Le code XML :

```
<toast>
    <visual>
        <binding template="ToastImageAndText03">
            <image id="1" src="image1" alt="image1"/>
            <text id="1">headlineText</text>
            <text id="2">bodyText</text>
        </binding>
    </visual>
</toast>
```

ToastImageAndText04

Une image, une chaine en gras tronquée si nécessaire, deux chaines simples sur les 2 autres lignes, tronquées elles aussi si nécessaire :



Le code XML :

```
<toast>
    <visual>
        <binding template="ToastImageAndText04">
            <image id="1" src="image1" alt="image1"/>
            <text id="1">headlineText</text>
            <text id="2">bodyText1</text>
            <text id="3">bodyText2</text>
        </binding>
    </visual>
</toast>
```

Bref, ce n'est pas bien compliqué même si, en bon "Xamliste" j'aurai bien plus apprécié un simple `ContentControl` dans lequel on aurait pu placer le code Xaml qu'on voulait… Plus simple, plus puissant, moins restrictif, moins bizarre à se souvenir que ces 8 templates aux noms peu évocateurs. Que des avantages. Sauf que Sinofsky n'aimait pas XAML. A quoi peut tenir parfois la cohérence et la puissance d'un système arrivé à un tel niveau de sophistication me surprendra toujours…

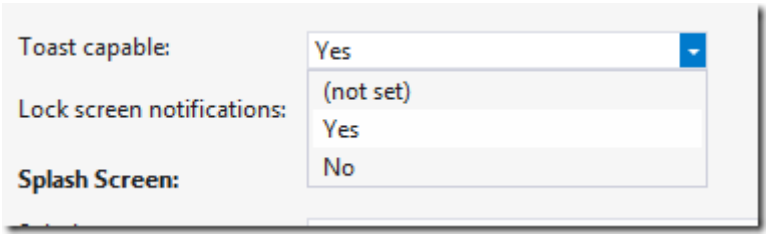
Comment faire griller les toasts ?

Nous savons ce que sont les toasts de Windows 8, nous connaissons les saveurs disponibles, reste à savoir comment les faire griller…

Rangez le toaster, appelez VS 2013 et créez un nouveau projet Windows Store App (le modèle "blank" est parfait).

Paramétrage

La première chose à faire est de déclaré qu'on veut faire des toasts. C'est pas bien méchant un toast, mais Microsoft a prévu une autorisation rien que pour ça dans le manifeste :



Si vous oubliez ce paramétrage il n'y aura aucune erreur, aucune exception. Rien ne se passera, c'est tout…

Vérification

Votre application peut ensuite vérifier que les toasts sont bien activés avant de s'en servir (sait-on jamais…) :

```
var notifier = ToastNotificationManager.CreateToastNotifier();
if (notifier.Setting != NotificationSetting.Enabled)
{
    // se débrouiller sans les toasts...
}
```

Faites chauffer !

Ne reste plus qu'à faire chauffer le toaster pour griller notre premier toast :

```
var notifier = ToastNotificationManager.CreateToastNotifier();
var template =
ToastNotificationManager.GetTemplateContent(ToastTemplateType.ToastText02);

var element = template.GetElementsByTagName("text")[0];
element.AppendChild(template.CreateTextNode("Première ligne"));

var toast = new ToastNotification(template);
notifier.Show(toast);
```

On remarque que nous commençons par obtenir un "*notificateur*" puis un *Template* (ici le modèle `ToastText02`, voir plus haut pour le détail de ce dernier).

Ensuite il nous faut récupérer les éléments par leur ID, un peu comme de la programme… html (beurk).

Une fois l'élément obtenu on lui ajoute un enfant, ici un noeud texte (`CreateTextNode`).

Enfin le Toast est créé, puis il est affiché (immédiatement).

Un Toast différé

On peut différer dans le temps l'apparition du Toast, pour cela il suffit de remplacer les deux dernières lignes de code de l'exemple ci-dessus par celles-ci :

```
var date = DateTimeOffset.Now.AddSeconds(10);
var scheduledToast = new ScheduledToastNotification(template, date);
notifier.AddToSchedule(scheduledToast);
```

Le toast est créé en utilisant une autre classe, un `ScheduleToastNotification` et au lieu d'appeler la méthode `Show()` du notificateur on ajoute le Toast à la file d'attente de ce dernier. C'est assez logique.

Conclusion

Afficher des toasts avec Windows 8 est d'une grande simplicité, c'est très utile et permet d'avertir l'utilisateur immédiatement ou bien avec délai sans pour autant bloquer l'interface avec une boite de dialogue.

A utiliser quand même avec retenue… un logiciel affichant sans cesse des toasts deviendrait vite énervant…

Mock-up Windows 8 apps avec PowerPoint et PowerMockup

Je vous ai parlé il y a quelques jours d'une solution gratuite pour créer des Mock-up à l'aide de PowerPoint et de templates gratuits. Mais on peut aller plus loin avec PowerMockup, un addon pour PowerPoint qui ajoute des formes qu'on peut personnaliser.

Mock-up

Dans le billet "'Mock-up gratuit d'applications Windows 8 avec PowerPoint" j'ai présenté avec insistance et moult détails et comparaisons l'importance des Mock-up pour concevoir des applications visuellement attractives et des UX de bonne qualité. Je renvoie le lecteur à ce billet récent pour éviter les redites sur ce sujet, mais que cette absence de commentaire sur le Mock-up ne laisse pas penser à ceux qui n'auraient pas lu ce billet qu'il n'y a rien à dire sur le sujet !

PowerPoint

On ne présente plus ce logiciel de la suite Office que tout le monde, ou presque, à déjà manipulé au moins une fois dans sa vie pour créer une présentation.

Simple, intuitif, mais puissant, sachant gérer des animations, supportant un mode d'affichage spécifiquement calibré pour les présentations avec écran externe, sachant exporter sous divers format les fameux "slides", PowerPoint est peut-être devenu au fil du temps, le notre, celui de la communication, plus essentiel que Word lui-même.

En partant de ce logiciel on peut très rapidement arriver à une solution de Mock-up d'applications Windows 8 ou Windows Phone 8.

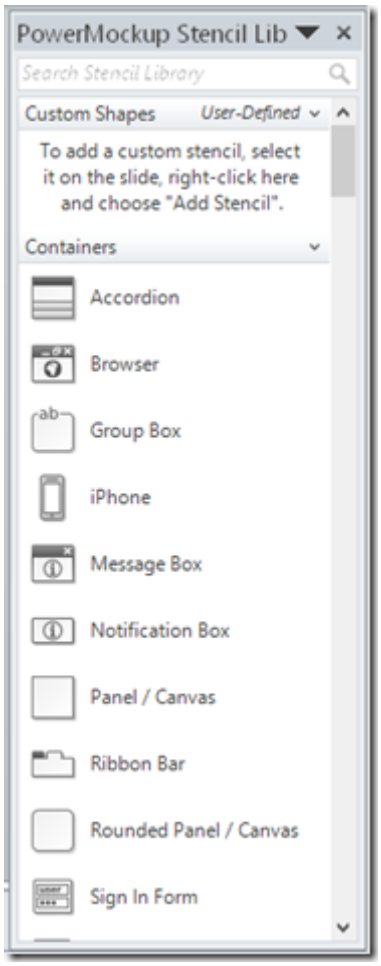
La première solution consiste à utiliser des templates de formes. C'est ce que je vous ai présenté dans le billet cité plus haut.

La seconde solution, qui n'écarte pas la possibilité d'utiliser des templates externes, consiste à adopter un petit addon spécialisé fournissant des formes déjà dessinées.

PowerMockup

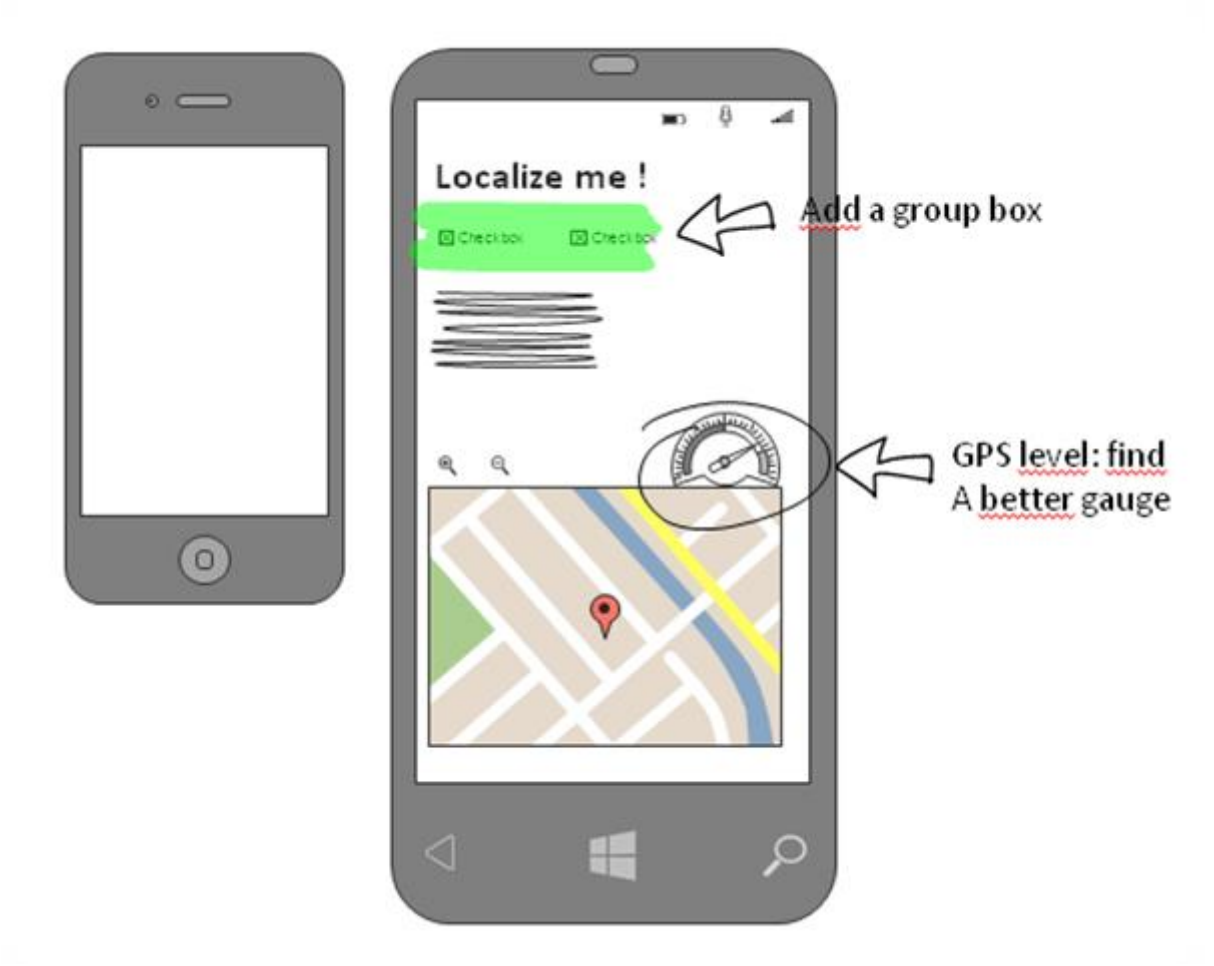
Cet addon c'est PowerMockup.

Une fois installé, il suffit de lancer PowerPoint comme d'habitude. La différence est visible d'emblée puisqu'apparait sur le côté droit une fenêtre "PowerMockup Stencil Library"



Le principe est très simple : il suffit de faire un drag'n drop depuis la librairie de formes vers la page Powerpoint en court de conception. C'est tout ! Pour faciliter les choses on peut aussi chercher les formes par leur nom dans le bloc de recherche (tout en haut) ou les parcourir par catégories (modifiables).

De là on peut personnaliser la forme, la redimensionner, lui adjoindre d'autres formes. Voici un exemple de Mock-up (totalement fictif) rapide :



La forme originale du téléphone se trouve à gauche pour comparaison. En quelques secondes j'ai transformé cet affreux Iphone en un fier Windows Phone 8 !, La forme peut maintenant être ajoutée à la librairie personnelle, plus besoin de se fatiguer…

Ensuite j'ai ajouté des éléments comme le titre, des checkbox, une zone "gribouillage" qui indique la place d'un texte non défini à ce stade du projet, un afficheur de type Bing Map, deux boutons zoom et une jauge de niveau de réception GPS. On suppose une application de localisation de la position de l'utilisateur. Tout cela est fait avec des éléments fournis par PowerMockup.

Le document PowerPoint est terminé, je le transmets à un collègue. Ce dernier me fait quelques remarques sur le design : il ajoute un coup de stabylo vert sur checkbox et indique qu'il faudra ajouter une GroupBox, et ensuite il entoure la jauge en indiquant qu'il faudrait trouver quelque chose de mieux…

En quelques secondes j'ai conçu une UI d'app Smartphone, je l'ai transmise à une autre personne qui a pu annoter (voire modifier) mon Mock-up.

Le projet avance, il est dynamique car le cycle est très rapide et peu contraignant. Tout le monde sait manipuler PowerPoint, pas de softs complexes de sketching à installer ni à expliquer !

Bien entendu PowerMockup offre aussi un gabarit Tablette et plein d'autres conteneurs, formes, Contrôles et icones. On peut personnaliser les gabarits, en ajouter de nouveaux, etc.

Les bases d'un projet

Avec un tel outil rapide de Mock-up les bases d'un projet peuvent être lancées sans cérémonial, on peut y faire participer de très nombreuses personnes car l'outil principal c'est PowerPoint que tout le monde connait, de la secrétaire à la Direction en passant par les vendeurs et les informaticiens.

Un bon projet commence par ratisser les bonnes idées le plus largement possible en demandant le minimum de savoir faire technique à chaque intervenant. Un vendeur pourra avoir des idées géniales sans être capable de manipuler Visio par exemple. Avec PowerPoint il ne se sentira pas diminué ou bloquer et pourra libérer sa créativité.

Communiquer des documents PPTX est aussi un avantage, puisqu'en entreprise presque tous les postes équipés de Office en seront dotés sans installation complexe supplémentaire. Les admins eux-mêmes auront du mal à ronchonner (ce sont des ronchonneurs!) si on leur demande d'installer PowerPoint dans le cas où il serait absent d'un poste.

Et dans les cas les plus difficiles à négocier, PowerPoint saura transférer son contenu en PDF par exemple, totalement portable en pièce jointe dans un Mail d'un bout à l'autre de l'entreprise et même à l'extérieur.

Communiquer et améliorer

Les Mock-up sont des outils fantastiques en phase de lancement d'une idée. En s'appuyant sur un logiciel commun souvent déjà présent en entreprise, PowerMockup simplifie et fluidifie la communication qui est essentielle dans ces phases précoces. C'est là qu'on entasse les bonnes idées de chacun, qu'on motive une équipe en faisant participer tout le monde, c'est là aussi qu'on détecte les premières difficultés.

Pourquoi se priver d'un outil simple qui améliorera tout le projet à lancer ?

Simplicité

PowerMockup est simple. Rien à voir avec Visio ou d'autres logiciels de diagrammes sophistiqués. Mais c'est tout son intérêt justement !

Payant

Ce petit addon est payant, à tous les sens : hélas il n'est pas gratuit (mais pas bien cher), mais s'en servir est payant aussi !

Conclusion

PowerMockup est une excellente idée, j'adore la simplicité du concept car c'est ce qui rend la chose utilisable partout en entreprise.

En se reposant sur PowerPoint largement connu et qui possède lui-même une large palette d'outils (alignement, couleurs, texte, formes…) qu'on peut utiliser pour compléter ses mock-up, on dispose au final de quelque chose de puissant et de pas si simplet.

A $60 la licence 1 utilisateur, $210 pour une petite équipe jusqu'à 5 personnes, c'est un produit qui ne nécessite pas un gros investissement, d'autant que fonctionnant sur le standard PowerPoint, chaque poste n'est pas obligé d'avoir le plugin installé pour réviser les mock-up… Une licence simple peut donc suffire s'il y a un responsable unique du design qui diffuse les documents et qui collecte les remarques pour améliorer l'ensemble.

Franchement j'ai trouvé le produit vraiment bien conçu. Cela permet d'ajouter un peu de visuel de façon très précoce et aussi de communiquer avec le client… En général une telle démarche qui mobilise toute une équipe et qui permet d'engager rapidement un dialogue constructif avec le client est toujours payante et se solde par un produit mieux adapté.

Testez gratuitement PowerMockup en téléchargeant la démonstration : http://www.powermockup.com/download/powermockup-setup.exe

Mieux gérer les capteurs sous Windows 8

Dans un récent billet (Faites frémir les sens de Windows 8 ou comment taquiner les capteurs) je vous présentais les capteurs disponibles sous Windows 8 et la façon de les interroger, dans ce billet je vous propose de créer des classes `helper` pour gérer encore plus facilement les données issues de ces capteurs.

Les capteurs : des amis du développeur

Le meilleur moyen de conquérir le marché c'est de créer des apps qui sortent de l'ordinaire, et quel beau moyen d'innover que d'aller chercher des utilisations originales du côté des capteurs sensoriels !

Les capteurs sont les amis des développeurs astucieux et imaginatifs, ne l'oubliez pas !

Dans le dernier billet sur ce sujet un lecteur proposait d'utiliser un boitier ODB-II pour gérer les paramètres de sa voiture… Des softs existent pour cela, mais pas sur Surface. Encore un marché à prendre ! Il existe de nombreuses sondes externes fonctionnant en Bluetooth, des thermomètres, des détecteurs de rayons Gamma, des sondes de température pour la cuisine, des sondes médicales…

Mais si on peut penser à toutes les applications utilisant des capteurs externes que tout bon geek bricoleur adore en général, il y a déjà fort à faire en exploitant correctement la brochette de ceux présents "in the box" !

Du code pour gérer plus facilement les capteurs

Dans mon billet de présentation des capteurs je vous ai donné à chaque fois un exemple de code qui permet de lire les valeurs.

Toutefois on peut faire mieux que ces exemples bruts. On peut construire des classes spécialisées qui faciliteront grandement l'accès aux données des capteurs.

C'est ce code que je vais vous présenterai plus loin.

Sans répéter ce que j'ai déjà expliqué dans le billet précédent sur le sujet, avant de passer au code regardons rapidement le type d'information retourné par les API.

Allo ? Surface ? Me captez-vous ?

Notre amie Surface est dotée de certains capteurs et d'un OS exposant des API pour communiquer avec eux. Il est intéressant de savoir faire la nuance entre les capteurs

réels, ceux qui sont virtuels et ceux que Surface RT ne supportent pas pour ne pas se fatiguer à inventer des idées qui ne pourront pas marcher sur ce matériel…

Surface propose les capteurs physiques suivants :

- Capteur de lumière ambiante
- Accéléromètre
- Gyromètre
- Magnétomètre

Il y a déjà de quoi s'amuser un peu, même si certains regrettent l'absence d'un GPS (que je ne juge pas très utile d'ailleurs pour un matériel étudié pour une utilisation principalement en intérieur, mais sa présence aurait été un plus). On pourrait ajouter à cette liste le Bluetooth et la Wifi car d'une certaine façon ce sont aussi des capteurs qui peuvent être utilisés pour obtenir des informations (après tout la géolocalisation des API WinRT se sert aussi de la Wifi pour connaitre la position dans certains cas). De même on pourrait ajouter les caméras (arrière et frontale) qui peuvent être détournées de leur but premier pour servir de capteurs sophistiqués. Toutefois je me limiterai ici aux capteurs "standard".

Grâce à cette série de capteurs physiques, Surface offre via WinRT plusieurs API qui interprètent les données brutes pour fournir des informations exploitables :

- L'accéléromètre
- Le gyromètre
- Le compas (grâce au magnétomètre)
- Le capteur de lumière
- Le capteur d'orientation
- Un capteur d'orientation simplifié
- Un inclinomètre

C'est tout de suite plus riche vu comme ça…

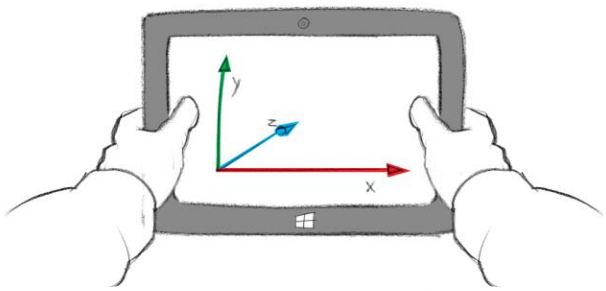
Le lecteur pourra trouver sur le site MSDN des informations techniques précises notamment sur l'exploitation des capteurs de mouvement et de de position (Integrating Motion and Orientation Sensors). Le site contient bien entendu bien d'autres pages dédiées à la programmation des capteurs qui sont au moins aussi intéressantes.

Surface et le monde physique

Dès lors qu'on souhaite tirer avantage (et information) des capteurs il est essentiel de poser une convention immuable : un ensemble d'axes alignés avec la tablette qui permet de savoir où est la gauche de la droite, le haut du bas, le devant du derrière.

Bien entendu, à l'œil, cela saute aux …yeux. Mais pour ce qui est des capteurs, du processeur et de l'OS, rien ne saute aux yeux, il faut une convention.

Les axes utilisés par les API sont ainsi définis comme le montre le dessin suivant :



L'axe des Y monte le long du côté gauche de la tablette, celui des X part du même point pour suivre le bord inférieur horizontal de la tablette, et l'axe des Z, partant toujours du même point, s'étend dans la direction du regard au-delà de la tablette. La position 0,0,0 se trouvant donc en bas à gauche. Les valeurs négatives sont autorisées bien entendu.

Des axes et des unités

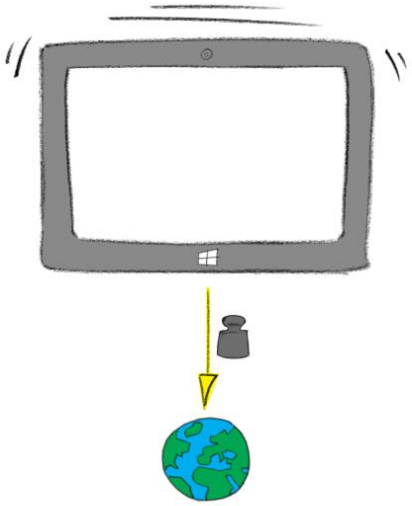
Chaque capteurs, en tout cas cas chaque API de capteur réel ou simulé, retourne des données qui seront toujours relatives au repère cartésien à 3 dimensions tel que montré sur le dessin ci-dessus (sauf pour la lumière ou le compas).

De plus, ces informations sont retournées dans des systèmes d'unité propres à chaque capteur.

L'accéléromètre

Il mesure l'accélération. Et même au repos il ne donnera jamais une lecture à zéro sur les 3 axes à la fois, sauf dans l'espace lointain (loin de toute planète ou étoile et à condition d'être animé d'un mouvement rectiligne uniforme, donc non accéléré). En effet n'oubliez pas que sur notre planète même en train de dormir sur une plage ensoleillée, même à siroter un pastis dans une chaise longue, vous êtes en train de

tomber ! En tout cas c'est ce qui fait vous ne volez pas au-dessus de votre serviette de bain quand vous êtes allongé sur la plage… Notre planète créée un champ gravitationnel permanent dont la valeur moyenne mesurée en G est de 1.



Donc même au repos posé sur votre bureau Surface ne pourra jamais retourner trois valeurs à zéro pour la mesure de l'accéléromètre.

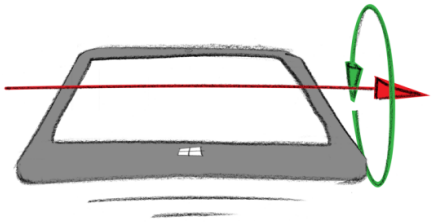
Quoi qu'on pourrait justement construire un jeu dont le but serait d'envoyer la tablette en l'air de telle façon à ce qu'un moment son accélération sur les trois axes, notamment la verticale, soit à zéro… Cela réclamerait pas mal d'entrainement (car il ne faudrait aucune déviation sur les axes X et Y). C'est finalement ce que propose le CNES et "Air Zero G" désormais avec des vols paraboliques ouverts au public pour environ 6000 euros… Là c'est un peu pareil, si vous échouez à rattraper la tablette, hop! C'est 600 à 1000 euros qui partent en fumée Qui aura le cran de créer ce jeu ? (en cas de succès je réclame ma part de royalties pour l'idée, mais en cas de casse je décline toute responsabilité ! – En fait une appli de ce type a été créée pour les smartphones, mais j'ai eu l'idée le premier !).

Donc l'accéléromètre mesure l'accélération sur les 3 axes de façon indépendante et il retourne une valeur `Double` exprimée en `G`.

Le Gyromètre

Comme je le précisais dans le billet précédent sur ce sujet, il ne faut pas confondre gyromètre et gyroscope, les technologies sont différentes, même si quelque part on peut en tirer le même type d'information, c'est à dire une accélération angulaire.

C'est d'ailleurs ce qui diffère de l'accéléromètre, car tous les deux mesures une accélération. Mais le gyromètre s'intéresse à l'accélération angulaire donc autour des 3 axes (rotations).



Le gyromètre retourne une valeur exprimée en degrés par seconde, c'est un Double aussi, et ce, pour les 3 axes.

Le compas

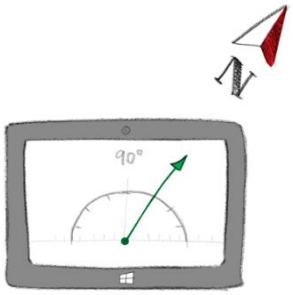
Cette API utilise le magnétomètre embarqué. Par force, ce dernier s'aligne selon le champ magnétique local et pointe donc théoriquement le nord magnétique (sauf perturbations locale du champ magnétique). Toutefois l'API sait retourner le nord géographique.

La donnée retournée est une mesure d'angle en degrés (un `Double`) qui indique la déviation par rapport au nord, soit géographique soit magnétique.

Il faut noter qu'il n'y a pas de calcul savant pour donner le nord géographique d'après le nord magnétique, on ne peut que calculer une approximation mais avec le danger de se tromper fortement si le champ magnétique local est perturbé (aimant, bloc d'alimentation, masses conductrices…). De fait, la mesure du nord géographique qui se doit d'avoir une certaine fiabilité (sinon on ne peut absolument pas s'y fier et cela rend la mesure sans intérêt pratique) utilise généralement les signaux GPS plutôt que le magnétomètre.

Surface n'étant pas équipée de GPS, la valeur du nord géographique n'est pas calculable avec une fiabilité suffisante et il semble que Microsoft a préféré ne rien retourner dans ce cas. Donc avec Surface, le compas ne peut être utilisé que pour lire le nord magnétique, ce qui n'a finalement que peut d'intérêt, du moins en tant qu'instrument de navigation.

On peut en revanche se servir du magnétomètre pour détecter des champs magnétiques… Reste à trouver des idées d'applications qui font un usage décisif de cette information…



Capteur de lumière

Surface se sert de ce capteur pour régler automatiquement la luminosité de l'écran. Il ne retourne pas une information très riche, juste un nombre de Lux (ce coup-ci c'est un `Float`).

Toutefois il ne faut pas minimiser ce capteur qui peut être utilisé par exemple pour basculer automatiquement l'affichage dans un mode "nuit", ce que peuvent exploiter des logiciels de cartographie céleste par exemple (passage de tout l'affichage en nuances de rouges sur fond noir ce qui évite l'éblouissement quand on regarde un ciel nocturne où l'œil finit par être en mydriase).

On peut donc, j'en suis certain, trouver milles et une utilisations astucieuses même si cela ne peut être en soi le cœur d'une application (mais après tout une idée géniale n'est pas interdite !).

Le capteur d'orientation

Cette API retourne une information complexe sous la forme d'un Quaternion et d'une matrice de rotation. Sont exploitation directe est un peu délicate et oblige à entrer dans des calculs spécifiques à la 3D. Les applications de jeu, ou par exemple de pilotage via Wifi d'un hélicoptère radiocommandé peuvent tirer un grand profit de ces données. Mais cela limite tout de même leur champ d'application.

Les lecteurs intéressés par les calculs dans l'espace avec des Quaternions peuvent lire ce papier "Quaternions and their applications to rotation in 3D space" (PDF d'une quinzaine de pages environ).

Le capteur d'orientation simplifiée

Cette API rend l'information précédente plus facilement utilisable dans une application standard (par exemple pour choisir une vue selon l'orientation de la tablette). Elle ne retourne qu'une seule valeur, issue d'une énumération, qui indique la position simplifiée de la tablette.

Simplifiée car il n'y a plus de nuance au degré près, juste une indication "grossière" sur la position : `NotRotated`, `Rotated90DegreesCounterClockwise`, `Rotated180DegreesCounterClockwise`, `Rotated270DegreesCounterClockwise`, `Faceup`, `Facedown`.

Un bon moyen de savoir si la tablette est posée face à l'endroit ou non sur la table par exemple. Une information qui peut être exploitée astucieusement.

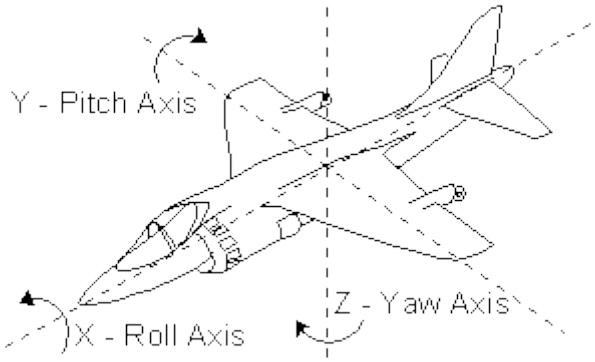
Par exemple sur le Samsung SIII lorsque le téléphone est posé à plat à l'envers (écran vers la table) cela coupe le son des alertes… ultra pratique quand on ne veut pas être dérangé et sans aller bricoler des paramètres qu'il faut ensuite rétablir.

L'inclinomètre

Cette API retourne trois informations qui se réfèrent aux trois degrés de liberté en indiquant la rotation de la machine sur les 3 axes.

L'information est retournée en degrés (`Double`).

On parle ici de Pitch, Roll et Yaw, qui sont des termes d'aviation, ce que le petit dessin ci-dessous résume (il faut imaginer une tablette à la place de l'avion !) :



Le Pitch s'appelle en français le Tangage, le Roll se traduit par Roulis et le Yaw par Lacet.

L'information retournée se base en réalité sur plusieurs capteurs physiques et il semble d'ailleurs que même le compas soit mis à contribution puisque si la tablette est parfaitement alignée sur le nord magnétique on peut arriver à une lecture de zéro sur les trois axes.

Du Code !

Les explications c'est chouette, mais le code c'est bien aussi…

Comme annoncé en début d'article je vais vous proposer de regarder quelques classes que j'ai développées pour tenter d'unifier les capteurs. Car il faut noter que hélas WinRT ne propose pas quelque chose de très unifié et que les API des capteurs ne descendent pas par exemple d'une classe commune, ce qui en faciliterait l'exploitation. Les classes que j'ai écrites simplifient cet accès.

Ce code est simple et peut largement être adapté pour vos propres applications, en l'état il constitue une bonne base pour vos propres réalisations.

Organisation du code

La partie réutilisable, la bibliothèque de code, se trouve dans l'espace de noms "Enaxos.W8.Sensors". Les classes principales reprennent exactement le même nom que celles de WinRT. C'est un choix délibéré : d'abord si on les utilise c'est pour masquer les classes de WinRT, et ensuite puisque .NET gère depuis toujours des espaces de noms, il est très facile de pointer vers mes classes ou celles de WinRT en les préfixant correctement.

Du point vue de la logique générale j'ai tenté d'unifier ce qui ne l'était pas et ce n'est pas facile à faire puisque, comme je le mentionnais plus haut, les API WinRT des différents capteurs ne descendent même pas d'une classe ou d'une interface commune qui pourrait être un "levier" dans cette unification.

Tout en haut de la hiérarchie se trouve `ISensor`, une interface classique qui est définie comme suit :

```
public interface ISensor
    {
        bool Initialized { get; }
        uint MinimumReportInterval { get; }
        uint ReportInterval { get; set; }
        DateTime LastEvent { get; }
        void Open();
        void Close();
        event PropertyChangedEventHandler PropertyChanged;
        string ToString();
    }
```

Cette interface regroupe tout ce que j'ai pu trouver de commun ou que j'ai pu unifier.

`Initialized` est un indicateur booléen qui permet de savoir si l'initialisation d'un capteur s'est déroulée correctement suite à un appel à `Open()`. Par défaut une instance créée n'est pas liée à son capteur, elle peut donc subsister en mémoire et

servir ponctuellement sans que cela ne pose de problème, notamment en utilisant `Close()` qui libère en quelque sorte le capteur.

Il n'y a pas de libération à proprement parlé (de type `Dispose` par exemple) car les capteurs se présentent comme des Singletons qu'on peut utiliser directement. Les classes que j'ai écrites se lient à ce singleton sur `Open()` et s'en détache sur `Close()`, c'est à dire que les évènements sont supprimés et que la référence à la device est passée à `null`.

Tous les capteurs, sauf un ou deux, proposent de régler l'intervalle d'interrogation (`ReportInterval`) et retournent la valeur minimale acceptable pour cet intervalle (`MinimumReportInterval`). C'est pourquoi ces deux propriétés ont pu être fédérées.

Le `PropertyChanged` et le `ToString()` font partie de l'interface car on doit pouvoir les manipuler.

`LastEvent` est une donnée commune à tous les capteurs, elle retourne le stamp de la dernière lecture des données. C'est cette propriété que vous traquerez dans un gestionnaire de `PropertyChanged`, car lorsque cet évènement est déclenché, vous avez l'assurance que toutes les données utiles ont été lues depuis le capteur. Inutile donc de chercher à connaitre le changement de chacune des propriétés.

En revanche, si vous effectuez des bindings directs sur les propriétés d'un capteur cela fonctionnera aussi puisque le `PropertyChanged` de chacune est bien déclenché aussi. On notera qu'ici aussi le code s'assure que toutes les données sont lues avant de déclencher les `PropertyChanged` au lieu de le faire donnée par donnée ce qui pourrait rendre assez complexe une lecture "groupée" des valeurs. Donc pour un capteur retournant 3 valeurs (X,Y,Z) par exemple, qu'on traque le `PropertyChanged` de `X`, de `Y` ou de `Z`, on peut à chaque fois être sûr que si traque `X`, la lecture de `Y` et `Z` est valide, et que si on traque `Y` la lecture de `X` et `Z` sont valides aussi, etc.

En réalité la programmation choisie permet d'avoir l'avantage des deux mondes : un monde dans lequel chaque valeur peut être traquée et lue indépendamment, et un monde dans lequel toutes les valeurs seraient lues en un seul bloc (ce qui est en réalité le fonctionnement des API WinRT).

A partir de `ISensor`, le code classique doit tirer sa révérence pour laisser la place à du code générique. C'est puissant le code générique mais cela ne permet pas d'avoir un ancêtre vraiment commun… D'où la présence de l'interface `ISensor`.

La classe `Sensor` qui servira de base à chaque capteur est ainsi une classe abstraite et générique. Son code supporte `ISensor` :

```
public abstract class Sensor<T> : INotifyPropertyChanged, ISensor where T :
class
    {
        #region fields

        private bool initialized;
        // ReSharper disable InconsistentNaming
        protected DateTime lastEvent = DateTime.MinValue;
        // ReSharper restore InconsistentNaming
        protected T device;
        protected const double Epsilon = 1e-7;

        #endregion

        #region Properties

        /// <summary>
        /// Gets access to the WinRT device API
        /// </summary>
        public T Device
        {
            get { return initialized ? device : null; }
            protected set
            {
                if (initialized) return;
                if (device == value) return;
                device = value;
                OnPropertyChanged();
            }
        }

        /// <summary>
        /// Returns true if the device has been correctly initialized
        /// </summary>
        public bool Initialized
        {
            get { return initialized && Device != null; }
            protected set
            {
                if (value == initialized) return;
                initialized = value;
                OnPropertyChanged();
            }
        }

        /// <summary>
        /// Minimum report interval supported by the sensor
        /// </summary>
        public uint MinimumReportInterval
        {
            get { return GetMinimumReportInterval(); }
        }

        /// <summary>
        /// Get/Set sensor report interval
        /// </summary>
        public uint ReportInterval
        {
            get { return GetReportInterval(); }
            set
            {
```

```
            if (value < MinimumReportInterval) value =
                                            MinimumReportInterval;
            var v = GetReportInterval();
            SetReportInterval(value);
            if (v != value) OnPropertyChanged();
        }
    }

    /// <summary>
    /// Last event date and time
    /// </summary>
    public DateTime LastEvent
    {
        get { return lastEvent; }
        protected set
        {
            if (!initialized) return;
            if (lastEvent == value) return;
            lastEvent = value;
            OnPropertyChanged();
        }
    }

    #endregion

    #region public methods and event

    /// <summary>
    /// Opens the sensor
    /// </summary>
    public void Open()
    {
        if (initialized) return;
        DoOpen();
        if (device == null) return;
        ReportInterval = MinimumReportInterval;
        Initialized = true;
    }

    /// <summary>
    /// Closes the sensor
    /// </summary>
    public void Close()
    {
        if (!initialized) return;
        DoClose();
        Initialized = false;
    }


    /// <summary>
    /// Property changed
    /// </summary>
    public event PropertyChangedEventHandler PropertyChanged;

    #endregion

    #region protected / virtual

    /// <summary>
    /// internal property changed trigger
```

```
        /// </summary>
        /// <param name="propertyName">property name (optional)</param>
        protected virtual void OnPropertyChanged(
                    [CallerMemberName] string propertyName = null)
        {
            var handler = PropertyChanged;
            if (handler != null)
               handler(this, new PropertyChangedEventArgs(propertyName));
        }

        /// <summary>
        /// Opens the device. must be overriden by real implementation
        /// </summary>
        protected virtual void DoOpen()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Closes the device. must be overriden by real implementation
        /// </summary>
        protected virtual void DoClose()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Returns minimum report interval.
        /// must be overriden by real implementation
        /// </summary>
        /// <returns></returns>
        protected virtual uint GetMinimumReportInterval()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Returns report interval.
        /// must be overriden by real implementation
        /// </summary>
        /// <returns></returns>
        protected virtual uint GetReportInterval()
        {
            throw new NotImplementedException();
        }

        /// <summary>
        /// Sets minimum report interval.
        /// must be overriden by real implementation
        /// </summary>
        /// <param name="value"></param>
        protected virtual void SetReportInterval(uint value)
        {
            throw new NotImplementedException();
        }

        public override string ToString()
        {
            return GetType().Name;
        }
```

```
        #endregion

    }
```

Une fois posée les bases de ISensor et de la classe abstraite Sensor, on peut créer des classes spécialisées pour chaque capteur.

Je ne vais pas lister ici le code chaque capteur, cela serait fastidieux d'autant que vous retrouverez le code source en fin de billet… Prenons juste un exemple très simple, le capteur de lumière :

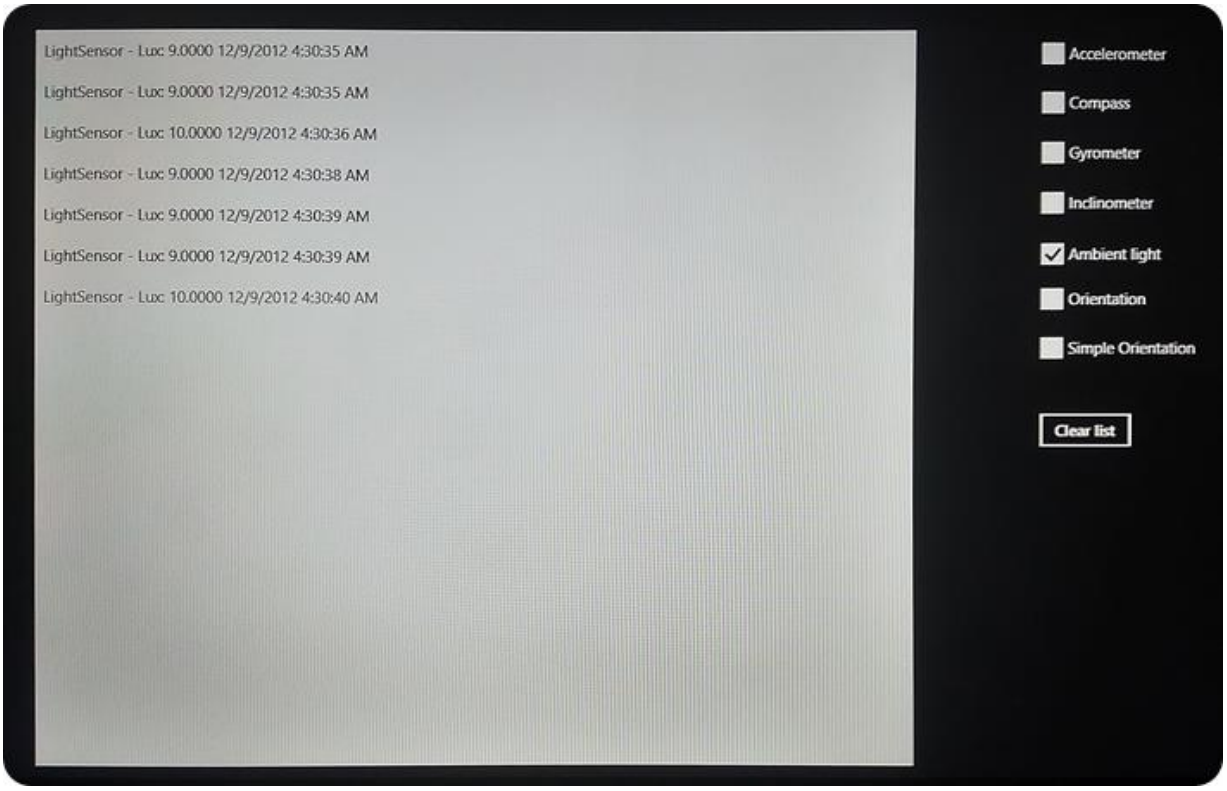
```
public class LightSensor : Sensor<Windows.Devices.Sensors.LightSensor>
    {

        #region fields

        private float lux;

        #endregion

        #region Property

        public float Lux
        {
            get { return lux; }
            set
            {
                if (Math.Abs(value - lux) < Epsilon) return;
                lux = value;
                OnPropertyChanged();
            }
        }

        #endregion



        void deviceReadingChanged(Windows.Devices.Sensors.LightSensor
sender,

Windows.Devices.Sensors.LightSensorReadingChangedEventArgs args)
        {
            lux = args.Reading.IlluminanceInLux;
            lastEvent = args.Reading.Timestamp.DateTime;
            // no matter which event is used,
            // all values are set before the handler is called.
            // ReSharper disable ExplicitCallerInfoArgument
            OnPropertyChanged("Lux");
            OnPropertyChanged("LastEvent");
            // ReSharper restore ExplicitCallerInfoArgument
        }

        protected override void DoOpen()
        {
            device = Windows.Devices.Sensors.LightSensor.GetDefault();
            if (device == null) return;
            device.ReadingChanged += deviceReadingChanged;
            // ReSharper disable ExplicitCallerInfoArgument
            OnPropertyChanged("Device");
            // ReSharper restore ExplicitCallerInfoArgument
        }

        protected override void DoClose()
        {
            if (device == null) return;
            device.ReadingChanged -= deviceReadingChanged;
            device = null;
            // ReSharper disable ExplicitCallerInfoArgument
            OnPropertyChanged("Device");
            // ReSharper restore ExplicitCallerInfoArgument
        }
```

```
        protected override uint GetMinimumReportInterval()
        {
            return device != null ? device.MinimumReportInterval : 0;
        }

        protected override uint GetReportInterval()
        {
            return device != null ? device.ReportInterval : 0;
        }

        protected override void SetReportInterval(uint value)
        {
            if (device == null) return;
            device.ReportInterval = value;
        }

        public override string ToString()
        {
            return base.ToString() + " - "
                        + string.Format("Lux: {0:0.0000} ", lux)
                        + lastEvent.ToString();
        }
    }
```

Ce capteur ne retourne qu'une seule donnée : le nombre de Lux perçus par la sonde de lumière ambiante.

Toutefois elle montre le fonctionnement global d'une classe spécialisée créée à partie de `Sensor`, notamment la surcharge de toutes les méthodes virtuelles et l'exposition des valeurs du capteur.

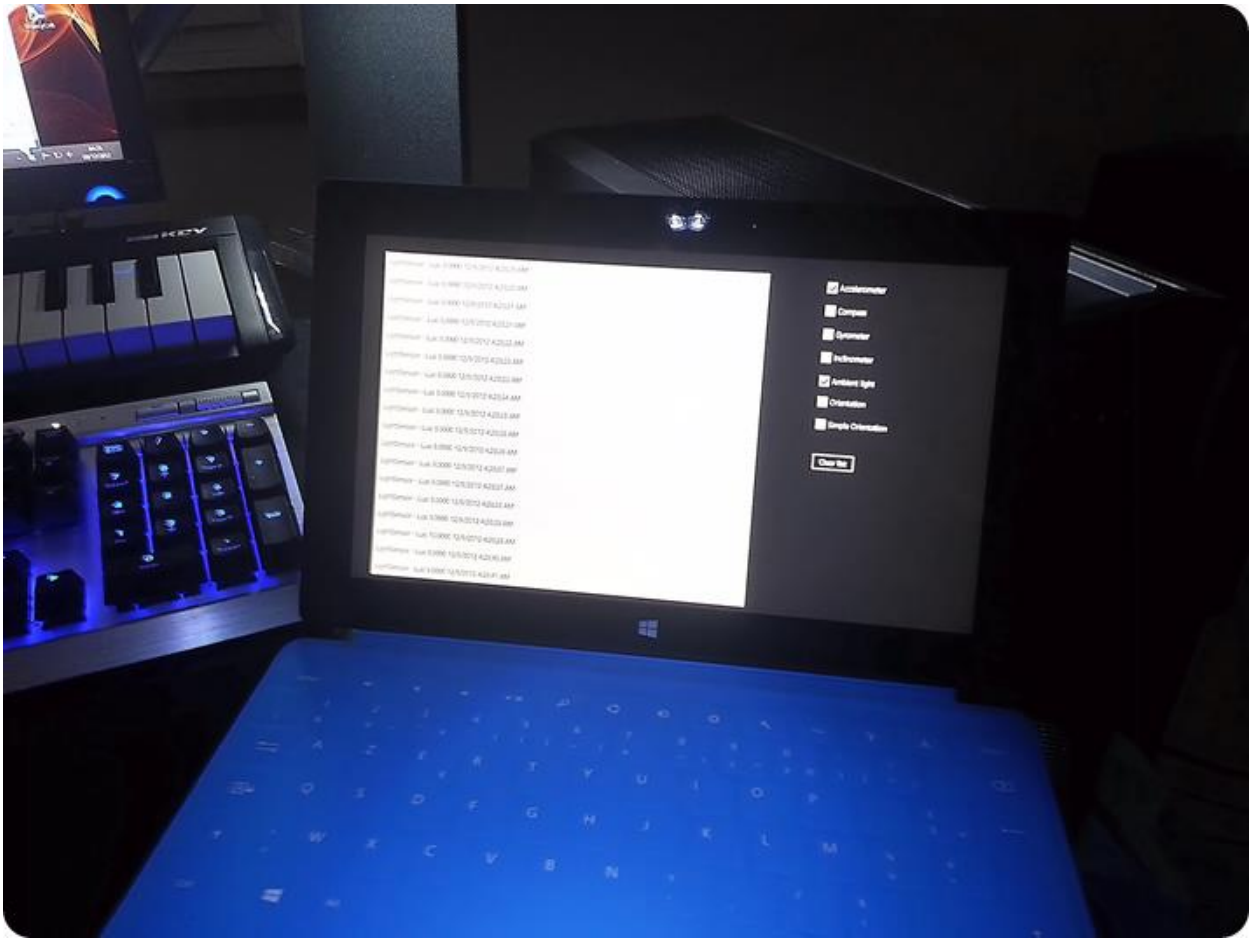
L'application de test en marche



L'application de test est très simple : elle propose des `checkbox` pour chaque capteur et une listbox affichant toutes les lectures. Cet affichage exploite le `ToString()` personnalisé de chaque classe. Plusieurs capteurs peuvent être ouverts en même temps, mais cela devient très vite brouillon !

Pour aider aux tests, il y a un bouton *Clear List* qui efface la liste. On libère en capteur en décochant sa checkbox.

On notera que les captures d'écran sont des photos faites avec mon Galaxy SIII et non des captures depuis le simulateur : Ce dernier ne permet pas de simuler de valeurs pour les capteurs et les API retournent "`null`" systématiquement. Il faut donc utiliser le mode debug distant avec une vraie Surface (très bluffant d'ailleurs, et dont j'ai parlé dans ce billet Debug d'applications WinRT sur Surface)



Ahh travailler la nuit dans un bureau qui clignote de partout de petites leds, illuminé par les écrans et autres zinzins électroniques… Un pur bonheur du geek

Conclusion

Je vous laisse sur cette rêverie de cabine de pilotage de navette spéciale (pas encore spatiale hélas)…

Sans oublier le code complet avec son projet de test :

Enaxos.W8.Sensors

Mock-up gratuit d'applications Windows 8 avec PowerPoint

La bonne conception visuelle des applications Xaml (Silverlight, WPF, Modern UI) est un passage obligatoire pour des applications vendeuses et accrocheuses. Pourquoi ne pas se servir de PowerPoint que tout le monde sait manipuler pour créer des mock-up ? Avec de bon templates tout est possible…

Le tout est dans le tout

Toutes les disciplines, tous les arts, lorsqu'ils sont pratiqués avec professionnalisme, se ressemblent dans leurs fondements : tous nécessitent beaucoup de travail pour avoir l'agilité requise et tous impliquent de préparer soigneusement ses plans avant de se lancer dans la réalisation.

Je le vois tous les jours par exemple entre la musique que je pratique et compose et le travail de ma compagne qui est designer / infographiste. Je peux à chaque fois trouver un parallèle entre ses problèmes de réalisation et ceux que l'on rencontre en musique. La lisibilité d'un bon design est très proche de la "lisibilité" d'une ligne mélodique par exemple. Les différents layers d'un dessin peuvent s'apparenter avec les différentes "couches" d'un morceau à mixer, l'art du mixage lui-même et ses règles subtiles peut s'appliquer presque directement aux mélanges des couleurs, des tonalités, au respect de l'ambiance. Choisir un fond pour un dessin doit savoir le rehausser et mettre en valeur le trait tout comme une "nappe" dans un morceau de musique doit savoir remplir l'espace sonore sans pour autant prendre le pas sur le véritable discours musical qu'elle doit soutenir, etc, etc…

Il s'agit en réalité de "meta patterns", tous les arts ayant une âme commune, un message, une émotion à véhiculer avec force et caractère ou avec douceur et nuance.

Je pourrais aussi vous parler des ressemblances qui existent entre ces "design patterns", entre ceux qu'on utilise en informatique (des "patrons de conception" où le mot "design" n'a rien à voir avec le Design malgré une croyance confuse récente parmi les informaticiens) et ceux utilisés dans l'architecture du bâtiment (c'est d'ailleurs comme ça que sont nés les premiers, en partant des seconds – le fameux livre fondateur de l'architecte Christopher Alexander).

Comme je pourrais rapprocher la conception d'une œuvre de Da Vinci avec celle d'un meuble de la famille d'ébénistes Wassmus (dont Henri-Léonard qui fut fournisseur notamment de Napoléon III) ou même résoudre les problèmes de la mise en page d'une application Modern UI en utilisant le point de vue d'un aquarelliste ou d'un premier violon d'orchestre, voire d'un réalisateur de films…

Toutes ces disciplines ont en commun tant de choses qu'il est même idiot de parler d'elles comme autant d'arts différents. L'Art est un, indivisible. Mais on peut l'aborder par différentes facettes. Un peu comme les cinq théories des cordes sont perçues aujourd'hui comme autant de facettes de la "théorie M", dite aussi "théorie du tout", tentative d'unification du physicien Edward Witten. Une seule théorie régit l'Univers,

une seule essence est à l'origine de tous les arts qui n'en sont que des facettes, des angles de vue différents et partiels, aucun n'arrivant à englober l'Art en entier.

Et le mock-up dans tout ça ?

La réalisation d'un meuble, d'un immeuble, d'une sculpture, d'un morceau de musique ou d'une application informatique se ressemblent.

Il faut à la fois le niveau d'expertise, l'agilité requise pour mener à bien le projet, beaucoup de travail pour tendre vers cette maitrise et bien préparer ses plans pour être sûr de la qualité de l'œuvre finale.

On ne se lance pas dans l'écriture d'une bibliothèque de code sans avoir fait des plans, des diagrammes UML, sans avoir modélisé l'ensemble des comportements des classes et de leurs instances et les conséquences de ces comportements.

Mais aujourd'hui une application informatique n'est plus seulement faite que de code, elle est faite à part égale entre du code et du Design.

Et la partie Design est régit par les mêmes règles, le même besoin de rigueur.

Le Design d'une application ne consiste pas à mettre des boutons ronds ou un dégradé de couleurs en fond de page, ni même à dessiner des petits mickeys pour faire de jolies icônes…

J'ai écrit des billets sur ce sujet auxquels je renvois le lecteur intéressé (Créer des interfaces graphiques efficaces, les bases du Design Windows 8, le design d'interfaces vendeuses et sa conférence TechDays 2012 à Paris sur le même thème, etc.).

Comme tout travail professionnel fait avec art (ou tout art fait avec professionnalisme, cela revient au même), le travail du designer est structuré. Souvent l'Art est perçu comme l'archétype de la liberté, de l'esprit libre, voire de la rêverie, ou même comme le symbole d'un certain sens de la liberté proche de l'anarchie. L'Art serait même de Gauche selon certains. Quelle erreur ! Si l'inspiration est une rêverie (mais pas toujours), si l'idée réclame une écoute du monde (mais pas toujours), si l'envie de faire passer des émotions est une forme d'ouverture aux autres (mais pas toujours) et si ces valeurs sont plus proches en effet d'un Bénabar, ou d'une Balasko voire d'un Tahar Ben Jelloun que de Christian Clavier (tous ces noms ayant été des soutiens de Gauche ou de Droite "officiellement" aux dernières présidentielles), la réalisation

d'une œuvre, elle, est comme la programmation : soumise à des centaines de contraintes, à des codes, des design patterns, des guidelines, à la même rigueur d'analyse, au même besoin de cohérence… Bref, si l'inspiration peut être fantasmatique ou idéologique (et de tout bord), la réalisation n'a qu'une seule valeur en étendard, celle du travail. Bach était un bourgeois sans aucune conscience politique révolutionnaire et pourtant ces œuvres restent de pures merveilles. Mais c'était un bosseur ! Il pensait beaucoup aux meilleures façons d'améliorer son art, sa pratique, le rendu de ces thèmes. Le Design appliqué à l'informatique ce n'est que ça, remettre sans fois sur le métier son ouvrage pour le peaufiner. Ne pas se contenter de la première idée et s'en gargariser au point de se croire un génie.

Et c'est là que les mock-up et le sketching entrent en scène !

Sketcher c'est essayer, peaufiner et prévoir, et prévoir c'est gérer…

Même les applications les plus simples nécessitent un sketching. Les interactions avec l'utilisateur doivent être naturelles, instinctives, toutes les "issues" doivent être prévues exactement comme un diagramme d'activité d'un logiciel ne peut avoir de "branches mortes" de cas de sélection "'débranchés".

Et les choses les plus simples recèlent souvent des subtilités que seule une grande expérience et la sagesse permettent de détecter sans prendre la peine d'un petit dessin… Et encore, la sagesse et l'expérience dictent de se conformer toujours aux règles sous peine d'oublier quelque chose !

Les exemples où l'arrogance se termine toujours mal sont légions. En tant que mycophile ayant longtemps couru les forêts pour y dénicher quelques spécimens rares et délicieux (dont l'Oronge aussi appelée l'Amanite des Césars est une sorte de Graal tout comme la Truffe ou la Morille), je sais que les accidents n'arrivent jamais aux débutant (méfiants) ni aux experts (…experts) mais à tous ceux qui pensent tout savoir. C'est comme cela qu'un jour une amanite phalloïde (qui a la perversité d'être très changeante d'aspect) termine dans leur assiette, mettant un point final à leur arrogance, et à leur vie par la même occasion…

Un mauvais Design peut signer la mort d'une application, telle une phalloïde planquée derrière un Agaric blanc ou une Lépiote pudique au milieu de l'assiette de code…

Ici ce n'est plus seulement le code spaghetti qui tue, c'est sa présentation !

Lire l'article "Le retour du spaghetti vendeur", ça détend …

L'expérience dicte ainsi qu'il n'existe jamais de cas simples, en apparence du moins et qu'il faut de toute façon se méfier justement des apparences ! Que derrière des choses qui paraissent évidentes peuvent se tapir des entortillements pouvant détruire les plus belles idées au moment de leur réalisation.

Dès lors le sketching devient une évidence, une étape absolument nécessaire.

On ne tape pas sous la console un "`Del *.*`" sans avoir vérifié 10 fois le nom du répertoire en cours… De la même façon on ne se lance pas dans la réalisation d'une application à la cinématique parfois complexe, même pour des actions simples, sans avoir pris le temps de la sketcher.

Mocker n'est pas sketcher

Sketcher permet d'aborder certes l'aspect général mais aussi la cinématique. Pour cela on peut utiliser la fonction de sketchflow de Blend, logiciel fantastique mais souvent mal compris par les développeurs.

S'il s'agit de représenter la cinématique d'une application, Blend et son outil de sketching reste l'arme absolue. Il existe toutefois d'autres logiciels spécialisés permettant de modéliser la dynamique d'une application, mais bien souvent ils présentent une complexité et un hermétisme autrement plus troublants que le sympathique Blend !

Créer un Mock-up est une autre affaire puisque là il ne s'agit que de fournir une image "ressemblant" à ce que sera tel ou tel écran.

> *A noter : je sépare le sketching de la création de mock-up pour tenter de clarifier deux aspects différents de la modélisation visuelle d'une application, à savoir les écrans, statiques, et la cinématique, dynamique par nature. Toutefois le sens exact de ces mots récents est fluctuant et la dichotomie que je propose ne reflète pas forcément le sens "officiel" si tant est qu'il y en ait un d'ailleurs. A la base sketcher n'est rien d'autre que produire des mock-up… L'aspect dynamique des choses, apporté notamment par Sketchflow sous Blend, est une nuance essentielle mais qui pourrait paraitre artificielle ou purement ad hoc à certains lecteurs. Je parle donc ici de sketching à la façon de*

Sketchflow (la dynamique) et mock-up à la façon de Photoshop (l'aspect statique).

Le design d'une application passe par plusieurs phases dont le sketching qui généralise la dynamique et l'essentiel de l'aspect général et les mock-up qui précisent l'aspect des écrans de façon plus réaliste.

L'un ne peut aller sans l'autre. On ne produit pas de mock-up au hasard sans connaitre ce qu'il y a en amont et en aval, donc sans la connaissance du logiciel que donne un sketching global et dynamique.

De Photoshop à Visio en passant par PowerPoint

Pour créer des mock-up tous les outils se valent mais certains sont assurément mieux adaptés à certaines situations ou à certains utilisateurs que d'autres.

Partir d'une page blanche sous Photoshop est une option souvent choisie par les infographistes "purs" si on leur demande de créer un "look" pour un site Web par exemple.

D'autres préfèreront le vectoriel d'Illustrator (les plus geeks adorent Expression Design, superbe application vectorielle trop peu connue et hélas abandonnée par Microsoft dernièrement – On utilisera InkScape un fabuleux outil vectoriel qui vaut bien Illustrator dans une majorité de cas). Blend lui-même et ses fonctions vectoriels en font un parfais outil de travail (même hors Sketchflow).

Toutefois, dès qu'il s'agit de représenter des UI standardisées il semble bien plus pratique de pouvoir disposer de "briques visuelles" déjà dessinées. Et si on doit travailler en cross-plateforme cela devient même essentiel pour représenter au mieux le logiciel sous ses diverses mises en page. Dans ce cas on pourra utiliser Visio. Mais il dispose de templates un peu dépassés (notamment pour les applications où les objets ont un look Win32, en tout cas cela est toujours vrai au moment où j'écris ces lignes).

Tous ces logiciels réclament malgré tout une bonne expertise pour être utilisés efficacement.

Un autre logiciel est presque connu de tous : PowerPoint. Tout le monde a déjà fait des "slides" au moins une fois dans sa vie.

Mais hélas PowerPoint n'a pas été conçu à l'origine pour faire du sketching ni même créer des mock-up. Mais tout n'est pas perdu !

PowerPoint : les bons templates

Malgré tout, PowerPoint est un logiciel simple à utiliser, qui présente les choses sont la forme de pages et qui, justement, sert à communiquer, base même du sketching qui sert à nourrir la réflexion d'une équipe en partageant les plans visuels d'une application.

Ne pourrait-on pas utiliser PowerPoint pour créer au moins des mock-up, facilement et sans entrer dans les méandres de logiciels complexes comme Visio ou pire encore Illustrator ou Photoshop, devenus des monstres au fil des ans ?

Un template PowerPoint gratuit

Andreas Wulf a eu la bonne idée de regrouper l'ensemble des principaux éléments visuels d'une application Modern UI dans un template PowerPoint constitué de pages contenant les éléments en vecteur.

Le template est gratuit et peut être utilisé à titre personnel ou commercial, mais à la condition de cité la source (c'est un minimum) et de le faire en respectant la licence originale qui est la Creative Common Attribution – ShareAlike 3.0 Unported License.

Vous pouvez télécharger ce template (un fichier PowerPoint) ci-dessous :

Le Template PowerPoint Windows8 Wireframing

Aller plus loin avec PowerPoint ?

Au-delà de ce template, peut-on aller plus loin avec PowerPoint ?

La réponse est oui !

PowerMockup

Il s'agit d'une extension pour PowerPoint créé par la même personne que le template et qui offre bien plus de facilités et d'options que de simples images vectorielles à copier.

Le produit est réellement intéressant et peut simplifier la vie pour créer et tester visuellement des interfaces Windows 8

Vous trouverez tous les détails sur ce logiciel ici : http://www.powermockup.com/

Pour ne par rendre ce billet trop long, je vous reparlerai prochainement en détail de ses possibilités particulièrement attractives.

Conclusion

Le sketching et les mock-up sont indispensables à une bonne scénarisation de vos applications en général. Sous Modern UI qui met en avant la qualité de l'UI et de l'UX, tout comme les applications Silverlight ou WPF, cela devient totalement incontournable. Si vous travaillez en cross-plateforme, la même attention doit être portée aux affichages (et à la dynamique de l'application) sous Android ou iOS.

Il existe des outils extraordinaires pour vous aider dans cette tâche comme Blend avec l'extension Sketchflow.

Mais il existe aussi des solutions plus simples, comme ce template gratuit pour PowerPoint ou l'extension payante PowerMockup.

Tout cela n'est pas directement interchangeable bien entendu, comme je le disais dans ce billet, chaque outil trouve sa place dans le processus de conception d'une UI et d'une UX réussie, de Photoshop à Blend, d'Illustrator à PowerPoint, du papier et du crayon aux outils graphiques les plus sophistiqués, chaque étape de la réalisation d'un logiciel réclame de la part du Designer un effort constant de mise en adéquation des moyens requis face aux besoins de chaque phase de développement.

L'utilisation d'un template pour PowerPoint, outil généralement bien connu des développeurs, est un moyen simple pour des non Designers de pouvoir produire des mock-up de bonne qualité. L'extension PowerMockup étend encore plus les possibilités.

Alors même si vous n'êtes pas un "artiste", votre métier vous a habitué à suivre des guidelines, des principes rigoureux, le Design c'est la même chose avec des images à la place d'un langage informatique !

Alors n'hésitez pas à modéliser vos applications.

Développement Windows 8 : Le livre

Regrouper dans un même ouvrage les bases du développement des applications Windows Store (Windows 8 sous WinRT et Modern UI) c'est ce que vous propose ce livre paru chez Eyrolles auquel j'ai activement participé…

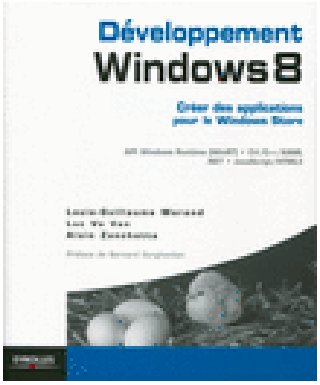
Windows 8 ou WinRT ou Windows Store App ou même Modern UI ?

Comment appeler ces applications particulières depuis que le nom “Metro” a été banni du vocabulaire Microsoft ? Il est dommage qu’aucune appellation aussi “sexy” et facile à retenir n’ait pu remplacer le premier nom connu de tous.

Windows 8 est l’OS. WinRT (Windows RunTime) est le jeu d’APIs, Windows Store App est le nom qu’on donne aux applications vendues sur le Windows Store et conçues pour tourner sous WinRT, donc sous les deux versions de Windows 8 (RT comme sur la première Surface, ou la version “complète” intégrant le bureau dit classique comme sur les PC). Quant à Modern UI c’est le nom officiel qui remplace réellement “Metro”, c’est à dire le langage de design créé pour l’habillage des UI sous Windows 8 et suivants. Modern UI est d’ailleurs, comme l’était Metro, bien plus une question d’UX que d’UI mais c’est une autre affaire !

Pourquoi un livre ?

Le développement d’applications Windows Store, même s’il se rapproche beaucoup



de ce qu’on connaissait sous .NET est malgré tout très différent. Les langages comme C++ et Html/js sont tout aussi à l’honneur que C# par exemple. Si Xaml est intégré à l’OS il n’est plus qu’une option (on peut opter pour HTML), les APIs de WinRT sont très nombreuses et couvrent des domaines plus vastes que celles de .NET, les mécanismes qui animent les applications sont fondés sur d’autres règles que sous WPF ou Silverlight, etc…

Dès lors il devient vite indispensable de faire le point sur ces nouvelles pratiques, ces nouvelles APIs, ce monde totalement asynchrone où les applications sont “tombstonées” comme sur un smartphone et où les capteurs divers peuvent jouer un rôle essentiel à côté du tactile et d’une organisation résolument différente de celle du bureau dit classique.

C’est sur ce constat que les auteurs (A.Zanchetta, L. Vo Van et LG Morand) se sont attelés à l’écriture d’un livre facile à lire et présentant l’essentiel de ce qu’il faut savoir de ce nouvel environnement.

Tous les langages à l’honneur

L'une des particularités de ce livre est que l'ensemble des exemples, sauf quelques exceptions, sont donnés aussi bien en C#/Xaml qu'en Html/Js ou en C++. Chaque développeur y trouvera ainsi son compte et ceux qui ne connaissent pas encore certains langages de la palette offerte auront l'occasion de voir ces autres langages à l'œuvre pour en jauger la syntaxe et, pourquoi pas, s'y intéresser de plus près.

Mon Rôle dans tout cela

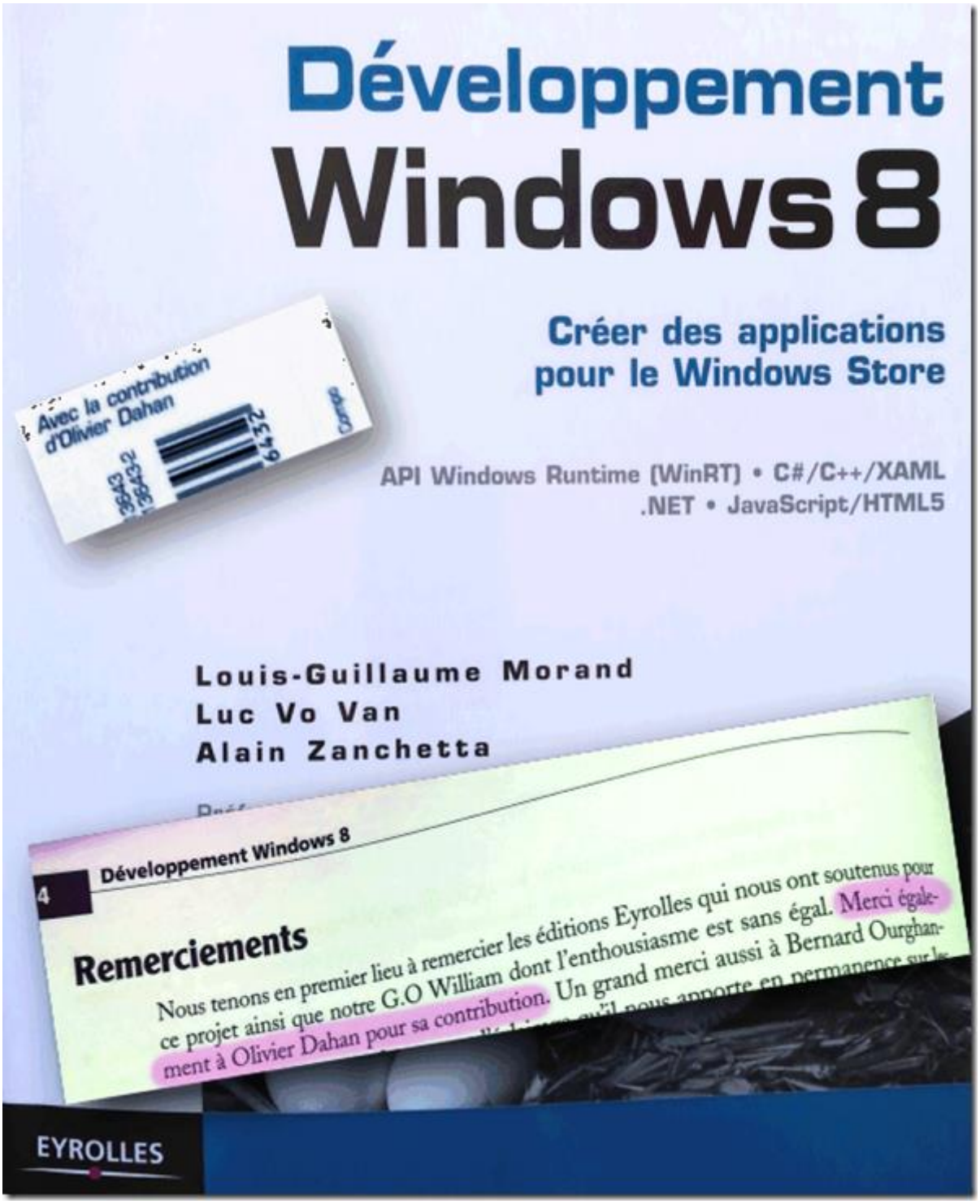
Au départ sollicité par Eyrolles pour effectuer la relecture technique de l'ouvrage, il s'est trouvé que mon expérience d'auteur (trois livres chez Eyrolles, Dot.Blog) et mes compétences sur les produits Microsoft ont fait légèrement dériver cette mission vers celle d'une contribution plus riche faite de compléments d'information et de reformulations pour clarifier encore plus le texte et en faciliter l'approche au lecteur.

Ceux qui ont l'habitude de me lire retrouveront certainement au détour de certains passages la marque de mon style… Mais collaborer, et aussi importante soit cette collaboration, n'est pas être auteur et je laisse volontiers aux trois auteurs tout le mérite qui leur revient n'étant pas frustré de ce genre d'honneur, tant avec les ouvrages que j'ai déjà écrits qu'avec les plus de 650 billets et articles de Dot.Blog qui mis bout à bout font plusieurs livres, écrits seuls, ce que la série ALL DOT BLOG illustre parfaitement !

Pouvoir rendre plus agréable la lecture d'un livre que l'on n'a pas écrit, pouvoir le peaufiner est en fait un plaisir très particulier. En général, arrivé au bout de l'écriture de 800 pages ou plus, et après toutes les relectures que cela réclame (en plus de celles effectuées par des professionnels chez Eyrolles) ont a hâte de voir enfin le bouquin en librairie, de sentir l'odeur du papier… Là c'est avec toute la fraicheur de celui qui arrive après l'écriture principale qu'on peut se lancer dans une vraie relecture productive, gommer les imperfections, déjouer des erreurs de code que ceux qui avaient le nez dedans n'avaient jamais vu, simplifier des tournures qui parfois sentent la fatigue bien naturelle, rajouter quelques précisions là où on se serait arrêté pour boucler un chapitre et passer au suivant.

Cela tient à la fois du travail d'enluminure et d'écriture, doublé ici par l'aspect technique forcément très présent et primordial.

Bref, c'est une collaboration qui m'a plu et qui, je l'espère, permettra à ce livre de connaitre une large diffusion.



Combien ? Où ? et Quand ?

Combient ? Pas très cher : 32 euros.

Où ? : Chez tous les bons libraires ou directement chez Eyrolles (magasin physique à Paris ou sur leur site).

Quand ? Le livre est paru courant Avril 2013.

"Combien ?" pourrait aussi se comprendre d'une autre façon : qu'est ce que cela me rapporte ? N'étant pas auteur sur ce livre mais collaborateur j'ai touché un forfait pour mon travail. Qu'il s'en vende 1 ou 10.000 je ne toucherai rien de plus ou de moins. Ce billet mi-publicitaire pour cet ouvrage est donc totalement désintéressé ! S'il s'en vend 10.000 ou plus je serai heureux pour les auteurs et satisfait du travail accompli qui n'a qu'un but : aider les futurs lecteurs à mieux comprendre WinRT comme je le fais ici sans relâche !

Conclusion

Vous voulez faire un point sur WinRT avant de vous lancer ? Vous souhaitez comparer la syntaxe des différents langages offerts avant de faire votre choix ? Vous aimeriez avoir rapidement un aperçu global des principes essentiels de WinRT ?

… Alors jetez-vous chez votre libraire ou chez Eyrolles !

"**Développement Windows 8 - Créer des applications pour le Windows Store**"
API Windows Runtime (WinRT), C#/C++/XAML, JavaScript/HTML5/CSS3

D'Alain Zanchetta, Luc Vo Van et Louis-Guillaume Morand
Préface de Bernard Ourghanlian - Avec la contribution d'Olivier Dahan
Collection Blanche
Code Geodif = G13643 ; ISBN = 978-2-212-13643-2
Prix 32.00 € - Disponible courant avril 2013

> *Les développeurs sous Windows 8 peuvent exploiter la nouvelle API WinRT qui traite tous les aspects du développement applicatif, de la présentation graphique au dialogue avec le matériel, notamment avec les nombreux capteurs conçus pour Windows 8 (accéléromètre, NFC…), en passant par les fondamentaux tel que les accès réseau et les supports de données. Afin de garantir sécurité et qualité, les applications écrites en respectant les nouveaux standards sont publiés exclusivement via le Windows Store, la plate-forme Microsoft de distribution en ligne d'applications, et doivent répondre à un cahier des charges technique très strict vérifié par Microsoft lors du processus de publication.*
>
> *Mots-clés : C#, C++, XAML, .NET, Windows Runtime, JavaScript, HTML5/CSS3.*

La page où vous pourrez découvrir l'ouvrage et d'autres à paraître sur Windows 8 :
http://www.editions-eyrolles.com/livres/Windows-8-pour-les-professionnels/

Utiliser PathIO sous WinRT

Une petite astuce rapide sur les fichiers texte qui permet de gagner pas mal de lignes de code (et donc de limiter les bugs !).

Faire simple est parfois compliqué !

Plus une API est riche, moins elle est connue. Plus une API est méconnue plus il y a de chance que les solutions que vous trouviez ne soient pas les plus simples.

C'est un raisonnement qui se vérifie à chaque fois.

WinRT possède une API pléthorique bien plus grosse encore que celle de .NET. A la fois cette taille gigantesque et le succès modéré de WinRT font que lorsqu'on cherche une solution à un problème on ne trouve pas grand chose sur le Web. On se contente donc de ce qu'on trouve, qu'il s'agisse d'extraits de documentations, de fils de forums ou d'articles.

Mais rien ne garantit que la solution proposée soit la plus directe…

Chercher cette simplicité qui a elle seule permet d'assurer 80% de la fiabilité d'un code est une tâche autrement plus difficile sous ce type d'environnement que de se lancer sur son clavier et de coder des pages de C# !

PathIO, une des portes vers la simplicité

Au détour de mes visites sur le Web et par pure sérendipité j'ai trouvé un bout de code qui permet sous WinRT de lire un fichier texte. Le code en question (extrait d'une librairie Open Source) faisait usage d'une URI pour accéder à un fichier dans le dossier temporaire de l'application.

Cela donnait ça :

```
var localFolder = Windows.Storage.ApplicationData.Current.TemporaryFolder;
var folder = await localFolder.GetFolderAsync("game");
var file = await folder.GetFileAsync("player.txt");
var fs = await file.OpenAsync(Windows.Storage.FileAccessMode.Read);
var inputStream = fs.GetInputStreamAt(0);
DataReader reader = new Windows.Storage.Streams.DataReader(inputStream);
await reader.LoadAsync((uint)fs.Size);
string data = reader.ReadString((uint)fs.Size);
reader.DetachStream();
```

C'est pas bien compliqué mais pour une tâche aussi simple cela me semblait vraiment trop lourd.

En cherchant un peu, je me suis souvenu de la classe `PathIO` qui devait contenir des choses intéressantes pour simplifier tout ça. Bingo !

Tout le code ci-dessus peut s'écrire :

```
string contenu =
await Windows.Storage.PathIO.ReadTextAsync("ms-
appdata:///temp/game/player.txt");
```

c'est tout…

Conclusion

Bien connaître les API est la clé de la simplicité. Il est vrai que lorsqu'elles deviennent gigantesques cela est plus difficile à faire. Pire, comme en ce moment, où le cross-plateforme s'impose, l'obligation de jongler entre les OS et leurs API ne permet plus vraiment d'atteindre le degré d'expertise et de confort suffisant sur une seule plateforme, et encore moins sur toutes à la fois. Il faut donc toujours se poser la question de savoir s'il n'y a pas "plus simple" et ne pas hésiter à se promener sur le Web, à lire quelques articles. Certes c'est une perte de temps. Mais en réalité écrire un bon code est un gain largement supérieur.

Dans l'exemple ci-dessus, même si une heure a été perdue à se balader sur le Web et les documentations, cela vaut le coup. Le code original est une horreur. A écrire. A comprendre. Et pire, à maintenir. Celui qui le remplace est court, évident, sans risque d'erreur ou de bug ou presque.

D'ailleurs puisqu'on parlait d'elle, la classe `PathIO` possède d'autres méthodes intéressantes comme `AppendLinesAsync`, `AppendTextAsync`, `ReadLinesAsync`, `WriteLinesAsync`, `WriteTextAsync`… Je vous les laisse découvrir mais vous comprenez à leurs noms qu'en une instruction elles font la même chose que plein de lignes de codes éventuellement boguées…

Faire simple est compliqué, mais c'est si efficace…

Control Calendrier pour WinRT

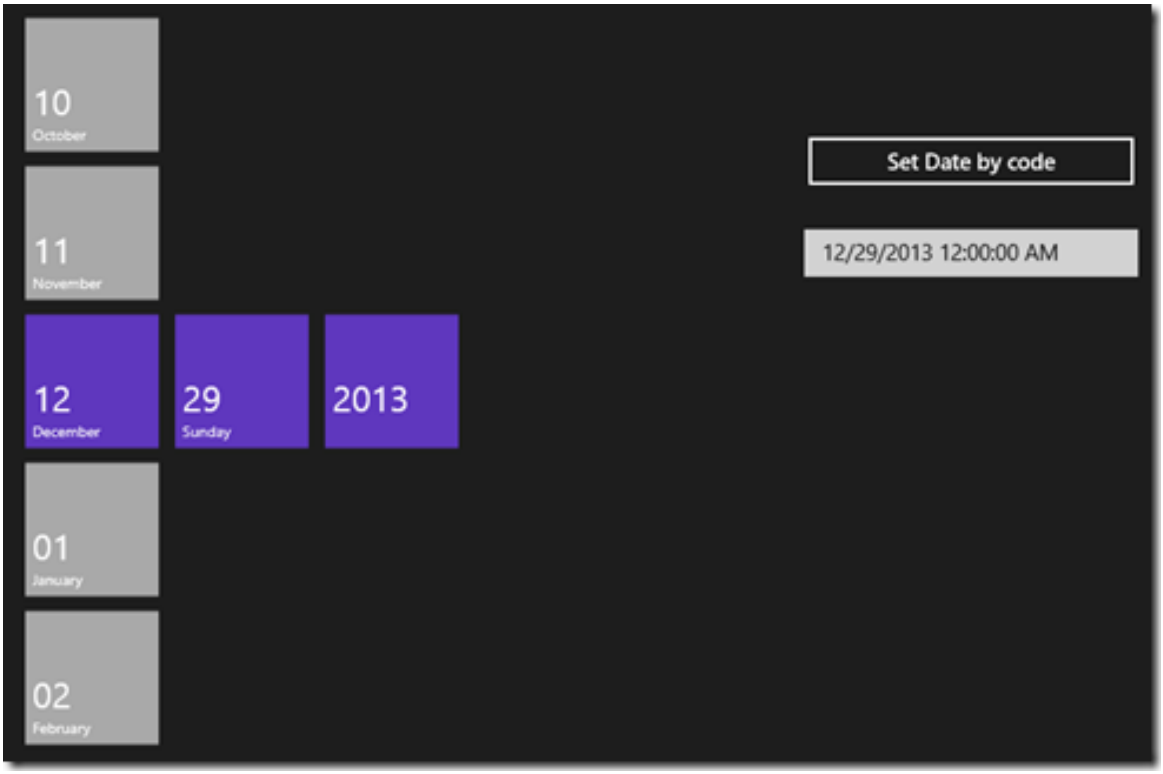
La saisie d'une valeur composite comme la date est souvent fastidieuse pour l'utilisateur. A fortiori dans un environnement tactile où la gestuelle doit être simple

et économe en mouvements (et en saisie clavier jamais très pratique). Voici un petit contrôle sympathique qui simplifie la saisie de dates.

WinRT Xaml Calendar

Sur Codeplex on trouve le Xaml WinRT Calendar qui permet de saisir rapidement une date.

Au-delà de l'utilisation qu'on peut en faire, sa présence en code source sur Codeplex en fait aussi un bon moyen de regarder de plus près comment un contrôle personnalisé peut être créé.



Comme on le voit le contrôle fonctionne de façon naturelle, il utilise des `Listbox` et n'est pas bien compliqué. Ce qui en fait un bon candidat à toutes sortes d'adaptations !

On trouve plus d'explications sur le contenu du contrôle sur le blog de son créateur.

Conclusion

Dans le même esprit que le billet précédent où je louais les vertus de la simplicité du code, ici c'est celle de la simplicité de l'UI qui est mise en avant.

Les deux sont indissociables.

Une belle UI avec du code spaghetti derrière c'est comme une jolie fille bien maquillée qui ne se serait pas lavée… Et un code d'enfer d'expert servi par une UI médiocre, c'est comme le type au corps d'athlète dans la blague dont je ne citerais pas le début et dont la chute est "tant de dynamique pour si petite mèche !" …

Faire simple, du code à l'UI, pour une meilleure UX et une meilleure maintenabilité. Cela doit rester votre objectif premier en toute occasion, quel que soit le logiciel, sa destination, son budget.

WinRT : RoamingSettings, quota et Sérialisation

Dernièrement je suis tombé sur un billet d'un évangéliste Microsoft qui mettait l'accent sur l'importance du stockage itinérant de WinRT. C'est, il est vrai, une feature encore mal exploitée. Je me suis dit qu'en parler cinq minutes ne ferait de mal à personne!

Le stockage itinérant (Roaming)

C'est un des grands avantages de la plateforme unifiée WinRT et pourtant il est peu utilisé par les applications. Parfois on rencontre même des développeurs qui se compliquent la tâche en créant un service Web (ce qui réclame serveur et application) pour qu'un utilisateur puisse retrouver facilement certaines données privées, des réglages de l'application etc, lorsque qu'il se logue sous une autre machine avec son compte.

Or, WinRT comme les autres plateformes possède un mécanisme de stockage centralisé qui est synchronisé automatiquement sur le compte de l'utilisateur lui permettant de retrouver ces fameux réglages d'application ou quelques données privées (en quantité limitée certes) lorsqu'il change de machine. L'espace de cette nature étant compté, pour sauvegarder et synchroniser des données volumineuses il faudra se tourner vers DropBox, Box, Google Drive, SkyDrive ou autres services de Cloud. Mais pour les réglages de base d'une application le système de Roaming est gratuit, facile à utiliser et entièrement automatique.

La bonne utilisation de ce mécanisme rend une application immédiatement plus agréable. Cela créé aussi une surprise favorable chez l'utilisateur, il se logue sur une machine différentes et comme par magie votre logiciel s'ouvre avec les bons réglages ! Je dis bien "magie" car pour un utilisateur de base même assez malin les mécanismes de synchronisation des données itinérantes est quelque chose qui lui échappera techniquement. Et si j'en parle aujourd'hui c'est que cela échappe même souvent aux développeurs !

Proposer une meilleure UX c'est essentiel.

Mais à quel prix ?

Et c'est là que c'est rageant : se servir du stockage itinérant n'est pas plus compliqué qu'autre chose… et ne coûte rien !

Le problème de la sérialisation des données génériques

C'est un problème annexe qui n'a rien à voir directement avec les données itinérantes mais qui se pose souvent aussi dans ce contexte. Autant le régler.

Bref ce problème apparait dans de nombreux cas et donc aussi dans celui de l'utilisation du Roaming car souvent une application doit conserver des éléments de configuration plus complexes qu'un simple `integer` ou une `string`.

Stocker par exemple une "`List<MonType>`" n'est pas directement possible car le mécanisme de sérialisation derrière le Roaming ne sait rien sur la classe "`MonType`". Cela fonctionne bien avec des données simples, des types de base, mais pas avec des types complexes.

Une solution évidente : transformer ce qui est complexe en quelque chose de simple !

Du complexe vers le simple

"`MonType`" est trop complexe pour la sérialisation du `RoamingSettings` ? Qu'à cela ne tienne ! Transformons le complexe en simple, c'est à dire une `string`…

Une simple méthode outil comme la suivante fera le travail :

```
public static string SerializeToString(object obj)
{
   XmlSerializer serializer = new XmlSerializer(obj.GetType());
   using (StringWriter writer = new StringWriter())
   {
      serializer.Serialize(writer, obj);
      return writer.ToString();
   }
}
```

Tout objet qui lui est proposé sera sérialisé en XML et retourné sur la forme d'une `string`. Le complexe devient simple.

Du simple vers le complexe

Bien entendu il faut être capable de récupérer les valeurs ainsi "simplifiées" pour réhydrater des instances de classes plus complexes…

La méthode ci-dessous fait l'inverse de la première :

```
public static T DeserializeFromString<T>(string xml)
{
   XmlSerializer deserializer = new XmlSerializer(typeof(T));
   using (StringReader reader = new StringReader(xml))
   {
      return (T)deserializer.Deserialize(reader);
   }
}
```

A partir d'une chaine contenant la sérialisation d'un objet en XML elle retourne une instance réhydratée. Du simple nous retrouvons le complexe.

Sauvegarder et relire les settings

Armez que nous sommes de ces deux méthodes il est donc possible de sauvegarder un type générique comme "`List<MonType>`" dans les `Settings` de l'application (et de les relire bien entendu) :

```
if (settingsRoaming == null)
   settingsRoaming = ApplicationData.Current.RoamingSettings;
settingsRoaming.Values["myData"] = SerializeToString(myData);

La relecture s'écrit alors :
if (settingsRoaming == null)
   settingsRoaming = ApplicationData.Current.RoamingSettings;
if (settingsRoaming.Values["myData"] != null)
   myData = (List<DataTypes.MyDataType>)
    DeserializeFromString<List<DataTypes.MyDataType>>
              (settingsRoaming.Values["myData"].ToString());
else
   myData = new List<DataTypes.MyDataType>(); // init si rien
```

Cette méthode est parfaite, simple, et permet de sauvegarder des données complexes dans le `RoamingSettings`… mais d'autres aspects sont à prendre en compte.

Les contraintes des données d'itinérance

Lorsqu'on utilise des données d'itinérance il faut garder à l'esprit un certain nombre de choses… La première est que les données peuvent avoir changé après le lancement de l'application, par exemple parce qu'elles n'avaient pas pu être synchronisées avant. Ou même parce que l'application est utilisée ailleurs en même temps.

Pour éviter ce genre de désagrément (qui soulève toutefois d'autres questions) il est possible d'écouter l'évènement `DataChanged` de `ApplicationData`. Les questions soulevées peuvent se résumer à la principale "*qu'est ce que je fais des nouvelles données s'il y en a qui arrivent après le lancement de l'application ?*".

La réponse est "ça dépend"

Selon la nature des réglages sauvegardés on pourra tout aussi bien s'en servir immédiatement sans que l'utilisateur ne le voit, mais parfois (et même assez souvent) ces changements seront visibles et pourront déstabiliser l'utilisateur si on ne le prévient pas. Il faudra donc par exemple afficher un message lui expliquant l'arrivée de nouveaux réglages et lui demander s'il veut les appliquer tout de suite ou non.

Ce qui cache d'autres questions comme "*comment l'utilisateur peut-il prendre une telle décision s'il ne sait rien des données sauvegardées, de leur nature, de leur effet sur son travail ?*". Epineux… Une boîte de dialogue ce n'est pas une formation sur le Roaming… ni un catalogue détaillé du fonctionnement de votre application. *Parfois prévenir l'utilisateur ne sert à rien car il ne pourra pas comprendre les effets de son choix.*

C'est une des bases d'une bonne UX : *ne jamais demander à l'utilisateur quelque chose qu'il ne peut raisonnablement pas comprendre facilement*… Dans un tel cas, c'est au développeur de faire le moins mauvais choix à la place de l'utilisateur (souvent il n'y a pas de "meilleur" choix, juste un "moins mauvais"…).

Le quota

Tout cela est essentiel mais il ne faudrait pas oublier la plus grande des contraintes du `RoamingSettings` : c'est la taille autorisée des données qui y sont stockées !

Microsoft gère cette espace par des copies locales mais aussi par un stockage dans le Cloud ce qui permet justement de synchroniser différentes machines. Et la générosité de ce stockage gratuit dans les nuages à une limite, un chiffre résume cette générosité : **100 Ko.**

Bien sur c'est peu. Voire ridicule.

Mais d'une part cela est fait pour stocker des paramètres d'application (généralement des entiers, des dates, des chemins de fichiers…) qui ne prennent pas tant de place que cela, et d'autre part, il faut s'imaginer que si des centaines d'applications s'autorisaient plus, de grands débats sur la consommation de la bande passante et le

“scandale” d’une telle consommation “à l’insu du plein gréé” de l’utilisateur ne manqueraient d’apparaitre ici et là …

En limitant l’espace à 100Ko Microsoft a bien plus voulu protéger l’utilisateur contre une telle consommation qu’essayer de jouer les radins.

On peut connaître le quota attribué à l’application comme cela :

```
var quota = ApplicationData.Current.RoamingStorageQuota;
```

Ne vous laissez pas abuser par cette API dont le résultat est un “ulong” ! Cela retournera 100Ko ni plus ni moins. Mais peut-être qu’un jour cela évoluera-t-il ? …

Quota dépassé ?

Que se passe-t-il quand le quota est dépassé ?

C’est tout simple : les données qui ne “passent pas” sont uniquement stockées en local et ne sont pas synchronisées.

C’est un bug qui peut être très difficile à découvrir !

Surtout que sur une machine de développement et en debug, le quota sera retourné à zéro (pour éviter de synchroniser des logiciels en cours de développement ce qui serait inutile). Il faudra donc anticiper une valeur de 100 Ko de toute façon. Elle peut même être codée en dur dans votre application, si cela change il sera toujours possible d’en tirer partie et de justifier une mise à jour aux utilisateurs ce qui démontrera votre capacité à vous adapter… (et vous fera un peu de pub. Pousser des mises à jours fréquentes et souvent inutiles est une pratique utilisée par de nombreux logiciels pour se “rappeler aux bons souvenirs” de l’utilisateur dans les environnements mobiles… pas très éthique mais ça existe…).

Prendre en compte le quota

Comme il est difficile de prévoir la taille exacte des données sauvegardées et comme la synchronisation partielle par dépassement de quota est un bug sournois mieux vaut prévenir que guérir…

En s’inspirant du code montré plus haut dans ce billet on peut écrire :

```
private static string SerializeToString(object obj)
{
    XmlSerializer serializer = new XmlSerializer(obj.GetType());
    using (StringWriter writer = new StringWriter())
    {
```

```
        serializer.Serialize(writer, obj);
        return writer.ToString();
    }
}

public void Save()
{
    ApplicationDataContainer settingsRoaming =
        ApplicationData.Current.RoamingSettings;
    this.LastModified = DateTime.Now;
    string serializedData = SerializeToString(this);

    if ((ulong)UnicodeEncoding.Unicode.GetByteCount(serializedData) <=
        ApplicationData.Current.RoamingStorageQuota*1024)
    {
        settingsRoaming.Values["appSettings"] = serializedData;
    }
    else
    {
        //handle the situation
    }
}
```

Les limites dans la limite…

Mais les choses ne sont pas si simples hélas. Il ne faut pas oublier que dans la limite des 100Ko se cachent d'autres limites…

Notamment un paramètre dans le `RoamingSettings` ne peut dépasser 8Ko… Fâcheux si on s'appuie sur une sérialisation XML dont on ne sait rien de la taille finale à l'avance (comme la fameuse "`List<MonType>`" discutée plus haut).

Heureusement il existe les settings "composites". Leur taille peut atteindre 64 Ko, ce qui de toute façon se rapproche de la limite fatidique des 100ko pour la totalité des paramètres.

Leur utilisation complique un peu les choses, pas tant techniquement (le code ci-dessous n'est pas complexe) que conceptuellement (comment construire un composite depuis l'ensemble des paramètres à sauvegarder, quels paramètres regrouper, etc ?).

```
// Composite setting
Windows.Storage.ApplicationDataCompositeValue composite =
   new Windows.Storage.ApplicationDataCompositeValue();
composite["intVal"] = 1;
composite["strVal"] = "string";

roamingSettings.Values["exampleCompositeSetting"] = composite;
```

Conclusion

Il est difficile, voire impossible de prévoir la taille exacte des données itinérantes. Une date est stockée sur 8 octets mais dans le Roaming est-elle sérialisée par WinRT et sous quelle forme ? Les textes sont encodés en unicode avec 2 octets par charactères ou par une autre forme d'encodage ? Bien des questions se posent alors même que la limite des 100Ko est un couperet. Il ne faut pas oublier non plus que le stockage itinérant peut contenir des fichiers et des sous-répertoires … comment cela est-il comptabilisé dans les 100Ko ?

La conclusion à en tirer c'est qu'aux extrêmes on sait mais pas au milieu… je veux dire que si on stocke trois entiers et une chaîne de caractères on sait qu'on sera largement dans les limites de 100Ko, de même si on veut stocker une liste d'images on sait qu'on serait en dehors des clous immédiatement. Mais dans beaucoup de cas on se trouve entre les deux… Comment décider, comment savoir pour éviter le bug ? Il faudra tester à fond l'application comme d'habitude et intégrer dans vos tests celui du dépassement de quota du `RoamingSettings`…

Certains tenteront d'utiliser JSON au lieu de XML pour sérialiser "à l'économie". Dans la pratique je n'ai jamais vu de différence énorme (sauf dans les exemples "pro" JSON, mais je parle de pratique pas de propagande). D'autres seront tentés de zipper le résultat, mais pour le stocker en chaîne il faudra l'encoder en Base64 ce qui fait perdre beaucoup de place… Bref on peut tourner le problème comme on veut, être certain de ne pas dépasser la limite de 100 Ko est très difficile à faire dans certains cas.

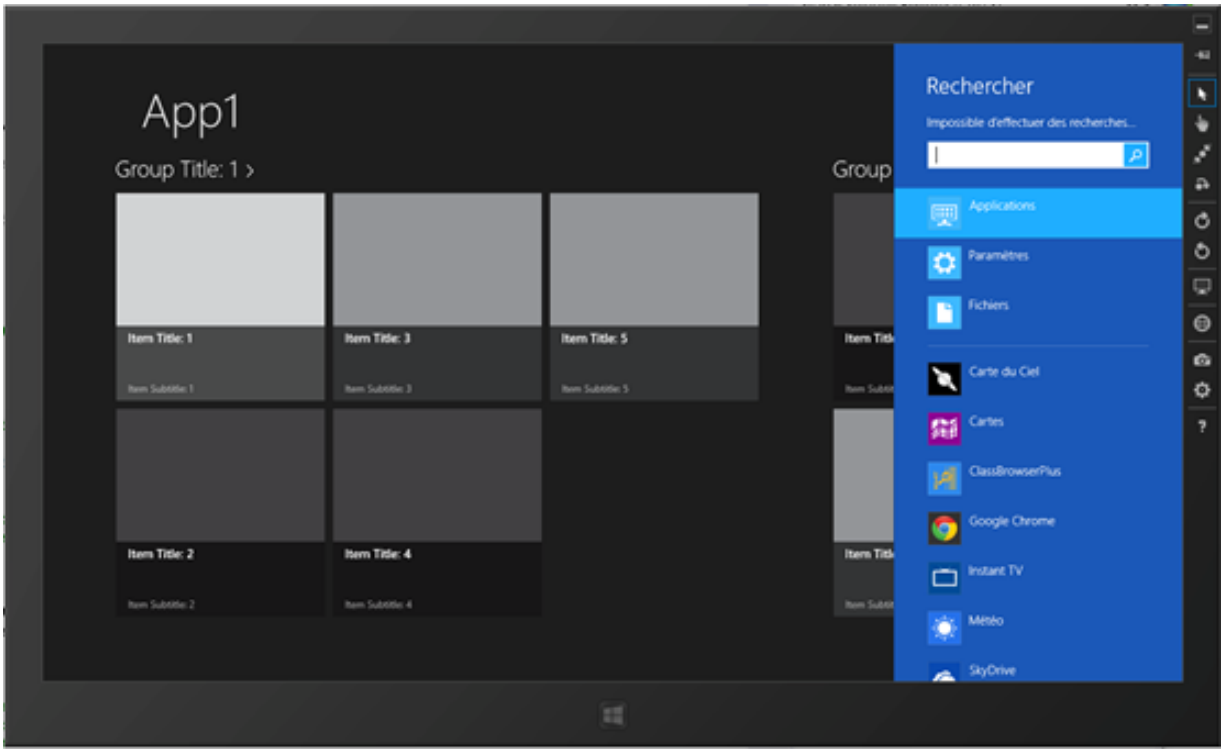
Mais que cela ne vous décourage pas d'utiliser les données itinérantes de WinRT, cela peut grandement participer à l'attrait que suscitera votre application…

WinRT : expressions régulières et panneau de recherche (Windows Store apps)

Créer de bonnes UI est essentiel, le cross-plateforme, sujet qui nous intéresse beaucoup sur Dot.Blog, uniformise le code mais cela ne veut pas dire qu'il est interdit de tirer profit des spécificités de chaque plateforme ! Le panneau de recherche de WinRT pour les applications Windows Store est une de ces spécificités à supporter absolument !

Le panneau de recherche

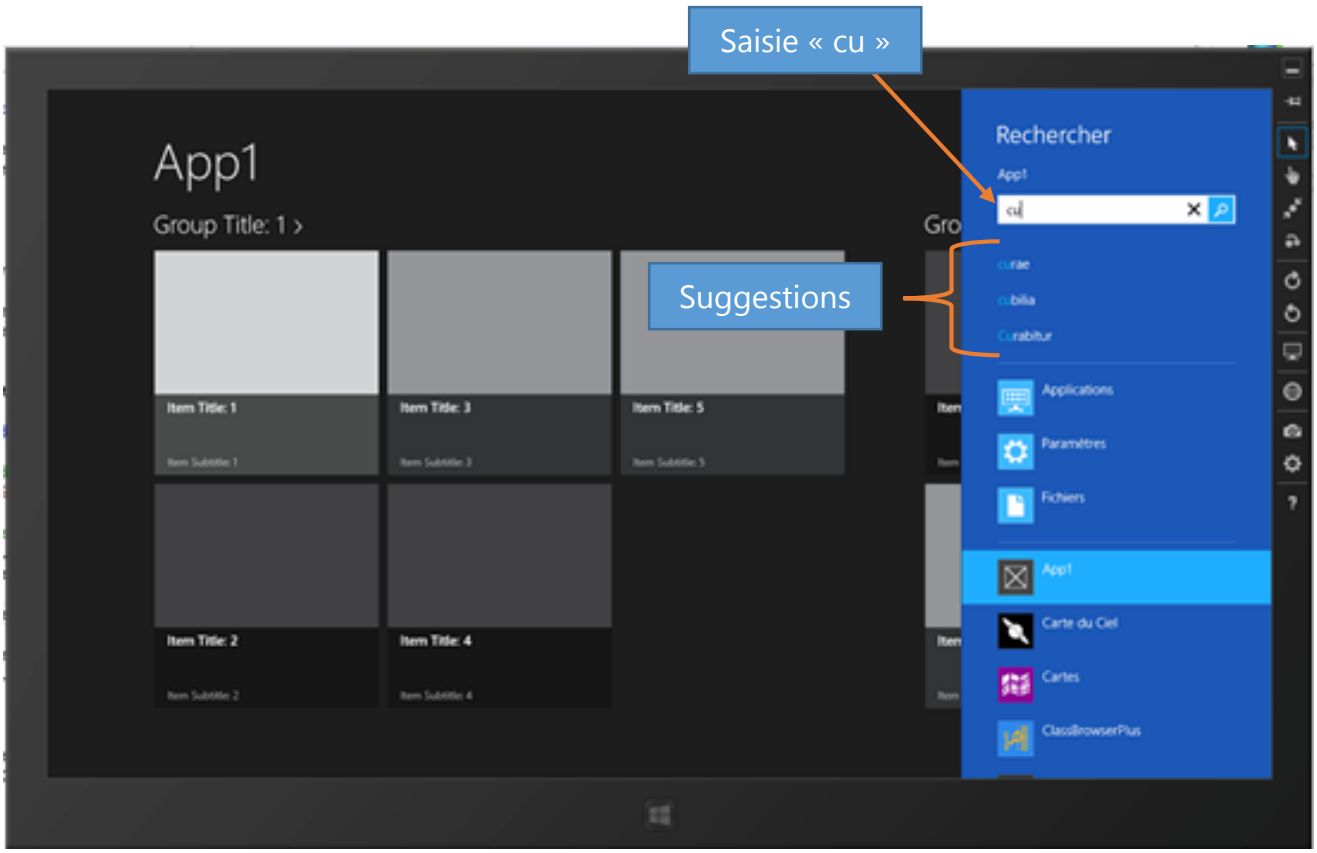
Le panneau de recherche de Windows 8 est particulièrement intéressant car il est capable via un processus unifié de “traverser” les couches applicatives. Chaque application peut retourner des informations en s’abonnant au processus de recherche global du système.



Être fournisseur de données pour des recherches de ce type est assez simple, mais il y a encore plus simple, c’est fournir des propositions (ou suggestions) dans l’écran de recherche.

Suggérer

La capture ci-dessous montre sous la zone de saisie de la recherche les trois mots suggérés par l’application, l’utilisateur ayant tapé “cu” :



Comme je le disais, proposer une recherche complète est un must et n'est guère compliqué, mais suggérer des mots est encore une autre chose. En général on couple les fonctions : on commence par suggérer des réponses possibles à l'utilisateur puis on gère un écran de résultat si ce dernier clique sur l'un des mots suggérés.

Ici nous nous intéresserons uniquement à la partie suggestion car elle est souvent négligée alors que le Charme de recherche a déjà été traité par Dot.Blog (voir l'article avec le code à télécharger : Donner du Charme à vos applications Windows 8)

Cela ne réclame pas grand chose et l'utilisateur pourra bénéficier d'un accès simple aux principales données de l'application.

Cette méthode de recherche est celle qui est montrée dans la capture ci-dessus et qui s'invoque par le raccourci **<touche Windows> Q**.

S'abonner, répondre, proposer…

Je ne referais pas l'article ci-dessus qui montre comment ajouter le support de la recherche et notamment comment activer cette fonction dans le manifeste de l'application (si vous ne le faites pas vous aurez une exception à l'exécution totalement fumeuse et incompréhensible qui ne met absolument pas sur la piste, vous êtes prévenus !).

Une fois l'activation dans le manifeste effectuée tout va se jouer dans `App.xaml.cs`.

D'abord il faut obtenir une référence sur le panneau de recherche puis s'abonner à son évènement `SuggestionsRequested` (suggestions demandées).

C'est dans la méthode `OnWindowCreated` (dont il faut créer une surcharge) que se joue cette partie là, en deux lignes (qu'on pourrait résoudre à une seule en se passant de la variable temporaire) :

```
protected override void OnWindowCreated(WindowCreatedEventArgs args)
{
    var searchPane = SearchPane.GetForCurrentView();
    searchPane.SuggestionsRequested += searchPane_SuggestionsRequested;
}
```

On propose alors un gestionnaire pour l'évènement et c'est naturellement dans ce dernier que va se trouver le code qui recherche dans les données de l'application et qui retournera les suggestions à Windows :

```
        void searchPane_SuggestionsRequested(
              SearchPane sender, SearchPaneSuggestionsRequestedEventArgs
args)
        {
            var results =
              new HashSet<string>(StringComparer.CurrentCultureIgnoreCase);
            // Builds the regular expression.
            var query =
              string.Format(@"(?<!\w){0}(\w+|\s+)", args.QueryText.Trim());
            // Search among all the groups.
            var groups = SampleDataSource.GetGroups("AllGroups");
            foreach (var item in groups.SelectMany(g => g.Items))
            {
                var matches =
                 Regex.Matches(item.Content, query,
RegexOptions.IgnoreCase);
                foreach (Match match in matches)
                    results.Add(match.Value.Trim());
            }
            // Orders the results and add them to the search suggestions.
            var suggestions = results.OrderBy(s => s).Take(5);
            args.
               Request.
                  SearchSuggestionCollection.
                     AppendQuerySuggestions(suggestions);
        }
```

Nul besoin du code de l'exemple car ici nous sommes tout simplement parti du template proposé par Visual Studio sans rien y changer. Il s'agit du template "`Grid App`". Il propose déjà une source de données remplie avec du texte de type *Lorem Ipsum* (d'où l'étrangeté des mots suggérés dans la capture écran plus haut!).

Le mécanisme utilisé consiste à traverser les groupes d'items et de balayer chaque item. La recherche s'effectue via une expression régulière pour ne prendre que des mots commençant par la recherche de l'utilisateur. On peut personnaliser à la fois l'expression pour offrir d'autres types de résultats et la séquence elle-même puisque, bien entendu, elle est fortement liée aux données utilisées par l'application donc ici à celles du template Visual Studio…

Conclusion

Chercher c'est bien, suggérer les bons mots à chercher c'est encore mieux !

Si nous avions déjà vu comment offrir les bénéfices du Charme de recherche la suggestion n'avait pas été traitée. Et je me suis aperçu que ce sujet l'était très rarement.

Il est vrai que tout ce qui concerne WinRT est traité "très rarement" sur Internet, symptôme évident d'une adoption modérée alors qu'on trouve de tout à propos de nombreuses autres plateformes.

Mais WinRT, s'il ne peut représenter l'avenir à lui seul parce que trop normalisateur notamment, et je le pense depuis le départ, n'en est pas moins une plateforme exceptionnelle. Microsoft a raison de continuer sa promotion même si je pense qu'ils devraient clarifier le rôle essentiel de WPF qui reste indispensable dans une vaste majorité de cas.

Dans la démarche cross-plateforme que je vous propose depuis des mois (près de deux ans, c'est à dire avant la sortie de Windows 8 si vous êtes perspicace !) toutes les plateformes comptent, soit parce que l'avenir de telle ou telle autre, comme WinRT ou iOS, est difficile à prévoir, soit parce qu'elles s'imposent à nous par leur succès (comme Android).

C'est justement en adoptant une démarche cross-plateforme qu'on se protège contre tous les doutes et toutes les modes. La démarche est simple : jouer les Madame Irma est un jeu dangereux et plutôt que de faire des choix de développement hasardeux comme on tire une carte de Tarot, nous embrassons toutes les plateformes, ajoutant ou supprimant au fil du temps celles qui s'imposent et celles qui ne manqueront pas de disparaitre dans le même temps, le tout sans toucher ni mettre en cause notre code métier, celui qui coute cher et qui est l'âme de l'application.

Cette démarche évite les choix dangereux et permet d'avancer, maintenant, sans attendre plus. Elle permet aussi d'éviter d'avoir à faire des choix douloureux, à "virer sa cuti" en passant à l'ennemi… C'est une démarche zen, une démarche dans laquelle nous pouvons adorer WinRT tout en louant l'extraordinaire percée d'Android…

Dans ce cadre connaître chaque plateforme pour offrir à l'utilisateur la meilleure UX possible est indispensable. Une bonne UX fait le succès d'une application. Le cross-plateforme uniformise le code mais pas l'UX qui doit rester unique.

Et sous WinRT, la bonne utilisation des Charmes fait partie d'une bonne UX. Puisse ce billet l'avoir rappelé à votre vigilance de développeur soucieux de produire des applications *awesome* !

Transcoder de la vidéo sous WinRT

Intégrer et gérer des vidéos dans un environnement basé sur le visuel semble naturel. C'est pourquoi Windows 8 fournit des API dédiées…

L'espace de nom Transcoding

Microsoft fournir un espace de nom dédié au transcodage, Windows.Media.Transcoding.

Cet espace de noms fournit un ensemble d'API qui va vous aider à transcoder des fichiers audio ou video d'un format à un autre.

Une API simplifiée

L'API mise à disposition est très simple elle est constituée principalement de deux parties :

`MediaEncodingProfile`, qui contient les réglages qui déterminent comment la destination doit être triatée.

`MediaEncoder`, qui effectue le travail de transcodage.

Mise en œuvre

Avant de transcoder un fichier il convient de fixer les paramètres de la conversion, on ne s'occupe que des paramètres de sortie, l'entrée est reconnue ou ne l'est pas, mais aucun réglage ne la concerne :

```
var encodingProfile =
MediaEncodingProfile.CreateMp4(VideoEncodingQuality.HD1080p);
```

Le `MediaEncodingProfile` contient des méthodes Factory comme `CreateMp4` qu'on peut voir ci-dessus ou `CreateWmv`. Le paramètre de la méthode choisie permet de fixer la résolution.

Ensuite il faut appeler `PrepareFileTransCodeAync` pour traiter le fichier (on en profite pour vérifier que la source est acceptée par l'API) :

```
var prepareTranscodeResult =
   await transcoder.PrepareFileTranscodeAsync(
                                sourceFile, destinationFile,
encodingProfile);

if (prepareTranscodeResult.CanTranscode)
{
    await prepareTranscodeResult.TranscodeAsync();
}
```

C'est tout…

Il est bien sûr possible de compliquer un peu plus les choses en chargeant un profile existant depuis les disques ou d'ajouter des opérations de trim (`TrimStartTime` ou `TrimStopTime`) pour extraire une partie seulement du fichier source.

Conclusion

Comme nous avons pu en discuter plusieurs fois, l'API de WinRT est très vaste, bien plus grande que celle de .NET. L'inconvénient majeur est l'apprentissabilité de cet édifice… L'avantage est d'y trouver des solutions simples à des opérations complexes !

WinRT : utiliser le presse-papiers

Le presse-papiers a été la première évolution "collaborative" des OS. La première tentative d'améliorer la coopération entre applications. Si les Charmes de Windows 8

sont les lointains descendants de cette volonté de simplifier la recherche, le partage et les échanges de données, le presse-papiers n'est pas devenu obsolète pour autant ! Au contraire, c'est le B.A-BA d'une bonne UX…

Les API du presse-papiers

WinRT et ses milliers d'API ne pouvait pas oublier le presse-papiers… C'est pourquoi on retrouve une classe statique `Clipboard` dans l'espace de noms `Windows.ApplicationModel.DataTransfer`.

Cette API a sa propre philosophie et elle utilise notamment un objet messager, le `DataPackage`, comme vecteur des informations écrites dans le presse-papiers.

Mais regardons d'abord la classe `Clipboard` :

```
public static class Clipboard
{
    public static DataPackageView GetContent();
    public static void SetContent(DataPackage content);
    public static void Flush();
    public static void Clear();

    public static event EventHandler<object> ContentChanged;
}
```

`GetContent()` est la méthode qui bien entendu permet de récupérer le contenu actuel du presse-papiers (noté PP plus loin) et `SetContent()` celle qui permet d'écrire dans ce dernier.

La méthode `Clear()` vide le PP, `ContentChanged` est un évènement qui est déclenché quand le contenu du PP change et `Flush()` est un peu plus subtile puisque cette méthode ajoute le `DataPackage` au PP tout en le libérant de son application source de telle façon à ce que le contenu partagé puisse être disponible même une fois cette dernière éteinte (supprimée de la mémoire).

La lecture et l'écriture dans le PP se font via des `DataPackage`. En fait rien de bien compliqué puisque la documenation MSDN indique qu'il s'agit simplement d'un `Object` définit en "sealed" sans aucune méthode ou propriété supplémentaire… Ce qui ne s'avère pas tout à fait exact comme nous le verrons plus loin. D'ailleurs elle ne montre pas vraiment non plus comment se servir de `Flush()` ni dans quel contexte précis. Il faudra tâtonner et expérimenter puisque MSDN est si peu bavard…

Le PP de WinRT couvre bien plus que de simples données textes comme le faisait celui des premières versions de Windows… On peut donc y trouver des données de type plus vastes et correspondant mieux aux besoins des utilisateurs et des développeurs d'applications modernes.

Les types supportés vont ainsi de l'incontournable morceau de texte brut jusqu'aux fichiers en passant par les images et le texte mis en forme.

Mais ici point de RTF ou de formats exotiques, WinRT s'est rangé du côté d'un standard, le texte formaté doit être écrit en HTML (et interprété comme tel quand il est lu depuis le PP).

Les principales utilisations

Les écritures

Placer du texte dans le PP est certainement l'opération la plus simple et la plus banale, et heureusement c'est la plus simple :

```
var dataPackage = new DataPackage();
dataPackage.SetText("blabla");
Clipboard.SetContent(dataPackage);
```

Le texte ainsi copié dans le PP est disponible au sein de l'application, des autres applications Windows Store mais aussi des applications standard en bureau classique (WPF par exemple).

Placer du texte formaté est à peine plus compliqué. On passe par une étape supplémentaire qui met en forme le texte HTML via le `HtmlFormatHelper` :

```
var dataPackage = new DataPackage();
var htmlContent = HtmlFormatHelper.CreateHtmlFormat("<b>blabla</b>");
dataPackage.SetHtmlFormat(htmlContent);
Clipboard.SetContent(dataPackage);
```

Placer une image dans le PP est tout aussi direct, du moment qu'on dispose d'une image en mémoire, ce qui implique de passer par une URI. Toutefois si on possède l'image en mémoire on peut directement passer le bitmap bien entendu.

```
    var dataPackage = new DataPackage();
    var storageFile = await
         StorageFile.GetFileFromApplicationUriAsync("Z:\soleil.jpg");
    dataPackage.SetBitmap(
                  RandomAccessStreamReference.CreateFromFile(storageFile));
    Clipboard.SetContent(dataPackage);
```

Si le bitmap est déjà en mémoire on peut donc directement appeler le `SetBitmap()`.

Placer un fichier dans le PP suit un chemin à peine plus compliqué :

```
    var dataPackage = new DataPackage();
    var files = new List<StorageFile>();
    var storageFile = await
         StorageFile.GetFileFromApplicationUriAsync("X:\data.sql");
    files.Add(storageFile);
    dataPackage.SetStorageItems(files);
    Clipboard.SetContent(dataPackage);
```

Comme on le remarque ici l'API fonctionne en réalité sur une liste de fichiers. On peut comme dans l'exemple ci-dessus ne passer qu'un seul fichier ou répéter l'opération `GetFileFromApplicationUriAsync()` pour ajouter les fichiers à la `List<StorageFile>`.

Les lectures

La lecture du PP est extrêmement simple puisqu'il suffit d'écrire un code de ce genre :

```
var dataPackage = Clipboard.GetContent();
```

Il ne reste plus qu'à utiliser l'instance obtenue. Pour cela il est tout de même nécessaire de contrôler ce qu'elle contient…

le `DataPackage` propose une méthode `Contains()` qui permet de tester ce contenu en utilisant l'énumération `StandarDataFormat` qui offre les valeurs : `Text`, `Html`, `Bitmap` et `StorageItems`

Pour savoir si le PP contient du texte brut il suffit donc d'écrire :

```
if (dataPackage.Contains(StandardDataFormats.Text)) ...
```

On opère donc de la même manière pour savoir s'il s'agit d'une image, d'un fichier ou de Html. Charge à l'application de savoir quoi faire des données reçues.

Conclusion

L'API de gestion du presse-papiers n'est pas la plus exaltante de WinRT, c'est une évidence. Mais elle devrait être couverte par toutes les applications, même les plus simples. Vous savez à quel point je suis attaché au Design et à l'UX des applications. Le presse-papiers permet d'enfoncer le clou car souvent les choses les plus simples sont les plus mal supportées par les applications… Copier/Coller des données, l'utilisateur s'attend au minimum à ce niveau zéro de l'UX qu'est le partage d'information via le PP.

Bien entendu l'OS gère de base le Copier/Coller sur les zones de texte. Mais il ne le fait pas avec les autres types de données.

Et même en texte brut il peut être intelligent de proposer des modes de copie un peu avancés (comme copier le nom et l'adresse d'une fiche client en une seule fois au lieu d'obliger l'utilisateur qui veut récupérer ces données pour faire une lettre dans Word à copier chaque champ l'un après l'autre et jurer tout ce qu'il peut à l'encontre du @&@! de développeur qui n'a pas penser à une chose si simple…). Je ne parle pas non plus de la magie qui fera d'un simple utilisateur un fan inconditionnel et qui consiste à savoir découper les données d'autres applications de façon habile pour remplir automatiquement via un Coller toute une série de champs par exemple. C'est difficile à faire dans l'absolu mais dans le cadre d'un environnement logiciel balisé (comme c'est souvent le cas en entreprise) c'est déjà plus facile. Et là le presse-papiers peut se transformer en une arme de séduction redoutable…

Windows.Storage pour Windows Phone 8 et Windows 8

La programmation sous Windows Phone 8 offre l'avantage d'une plateforme unifiée, comment en tirer partie par exemple en matière de stockage local de données. C'est ce que je vous propose de voir…

Le stockage local de données

En programmation Win32/64 le stockage de données locales est excessivement simple : vous écrivez ce que voulez à l'endroit que vous voulez…

Il est sûr que cette liberté et cette apparente simplicité ont ouvert en grand la porte à tous les abus et à ce qu'il faut bien appeler un gigantesque "bordel". Les applications en "mettent" partout, l'utilisateur ne sait jamais, sauf sauvegarde totale de ses disques, s'il ne lui en manque pas un bout lorsqu'il souhaite copier ses données sur une autre machine, la désinstallation des applications n'entraine pas toujours le

ménage des données qui vont avec ce qui pollue à la longue les disques durs, en cas de bug ou de mauvaise saisie de l'utilisateur on ne sait plus où se trouve le fichier enregistré, etc…

WinRT, tout comme le faisait déjà Silverlight, propose un cadre beaucoup plus rigide. Avec la notion d'Isolated Storage (stockage isolé) il n'est plus question d'en "coller partout", chaque application se voit restreinte à une zone privée. Cette organisation est forcément plus saine. Même si la restriction est cette fois-ci très (trop) contraignate pour certaines applications à qui il reste le bureau classique.

Windows Phone 8 et WinRT, un noyau commun

Windows Phone 8 a pour grand avantage de reprendre un noyau WinRT commun avec Windows 8 cela veut dire que des parties importantes de code peuvent désormais être partagées entre une application pour tablette Surface, une version desktop sous Windows 8 et une adaptation pour smartphone.

Si WP8 a créé une rupture de compatibilité avec WP7 paradoxalement cette cassure est moins nette au niveau programmation puisque les applications WP7, en Silverlight, sont supportées par WP8. L'incompatibilité existe entre l'OS et le hardware et beaucoup moins au niveau logiciel.

Deux approches pour les données locales

De cet héritage de WP7, WP8 permet souvent d'utiliser des stratégies différentes pour le même résultat, une qui reste compatible avec WP7, l'autre utilisant les nouvelles API WinRT.

C'est le cas notamment pour l'accès aux données locales.

Quel intérêt pratique ?

Dans l'absolu aucun et nul n'est besoin de reprogrammé ce qui marche déjà en WP7. En revanche c'est pour la création de nouvelles applications qu'il faut se poser la question de la compatibilité du code, soit avec les machines WP7 encore en circulation, soit avec WP8 et donc avec WinRT ce qui permet de partager le même code avec une version pour Surface ou PC. Le succès mitigé de WP7 rend en réalité cette question assez artificielle. Si Microsoft a fait une cassure aussi nette entre WP7 et WP8 c'est bien parce que le nombre des insatisfaits paraissait insignifiant comparé à celui des futurs clients espérés sous WP8… Adoptez la même approche : utilisez systématiquement les nouvelles API.

Ces deux approches peuvent se voir dans le code d'accès à l'espace de stockage de l'application. Soit on y accède via l'Isolated Storage dans un esprit purement Silverlight, soit on utilise les API de l'espace de noms `Windows.Storage`.

Pour illustrer ces deux façons de faire, nous pouvons regarder le code qui suit.

Exemple

Imaginons le besoin de stocker en local des listes d'objets. L'approche générique sera vraisemblablement la meilleure ainsi que la création d'une méthode statique dans une classe de type helper ou bien sous la forme d'un service (approche plus MVVM qu'on retrouve dans MvvmCross par exemple). Il en ira de même pour la lecture de ces listes d'objets.

Dans ce dernier cas on implémentera une méthode qui s'appellera "`LoadList`" (nom arbitraire qui a l'avantage d'être clair quant aux intentions de la méthode) et qui retournera une liste générique donc un `List<T>` ou parfois mieux selon le contexte, un `IList<T>`.

La méthode acceptera en paramètre le nom du sous-répertoire de stockage et le nom du fichier contenant la liste d'objets.

On limitera les objets aux instances de classes (les structures étant exclues donc).

La première version de ce code donnera dans la première approche (compatibilité WP7) quelque chose comme :

```
 public static IList<T> LoadList<T>(string folder, string fileName) where T
: class
{
    var result = new List<T>();
    var isoStore = IsolatedStorageFile.GetUserStoreForApplication();
     if (!isoStore.DirectoryExists(folder))
                isoStore.CreateDirectory(folder);

     var fileStream = string.Format("{0}\\{1}.dat", folder, fileName);

      using (var stream = new
           IsolatedStorageFileStream(fileStream, FileMode.OpenOrCreate,
                                     isoStore))
     {
         if (stream.Length > 0)
        {
             var dcs = new DataContractSerializer(typeof(List<T>));
              result = dcs.ReadObject(stream) as List<T>;
        }
      }
      return result;
 }
```

Pour l'accès à l'Isolated Storage on pourra aussi s'aider d'une classe helper. Un exemple d'une telle classe se trouve ici : https://github.com/eTilbudsavis/native-windows-phone-sdk/blob/master/Esmann.WP.Common/IsolatedStorage/IsoStorageHelper.cs

On peut aussi choisir d'implémenter la méthode dans un style beaucoup plus moderne, donc en asynchrone, et suivant l'API WinRT. Dans ce cas le même code ressemblera au suivant (qui on le remarquera ne simplifie rien ce qui est dommage) :

```
public static async Task<IList<T>> LoadList<T>
                (string folder, string fileName) where T : class
{
    var applicationFolder = ApplicationData.Current.LocalFolder;

    if (!await applicationFolder.FolderExistsAsync(folder))
        applicationFolder.CreateFolderAsync(folder);

    var result = new List<T>();

    var fileStream = string.Format("{0}\\{1}.dat", folder, fileName);

    if (await applicationFolder.FileExistsAsync( fileStream))
    {
        var storageFile = await applicationFolder.GetFileAsync(fileStream);
        using (var inStream = await storageFile.OpenSequentialReadAsync())
        {
            var serializer = new DataContractSerializer(typeof(List<T>));
            result = (List<T>)serializer.
                              ReadObject(inStream.AsStreamForRead());
        }
    }
    return result;
}
```

Conclusion

Il faut faire le bon choix entre toutes ces méthodes qui parfois se superposent, soit dans un style purement Silverlight, soit dans un style purement WinRT.

Le choix de la stratégie dépend grandement du contexte, forcément. Par exemple en cross-plateforme avec MvvmCross on aura intérêt bien entendu à choisir une approche "portable" alors que si on développe une application uniquement pour WP8 et Surface le choix de l'API WinRT sera certainement judicieux.

Mais est-ce judicieux de bloquer son code sur un OS au lieu d'utiliser l'approche cross-plateforme largement présentée cet été dans Dot.Blog, voici une excellente question à se poser !

Windows 8 : Protocole d'activation personnalisé (CPA)

Le protocole d'activation personnalisé est un mécanisme propre à Windows 8 pour permettre le lancement d'une application WinRT comme s'il s'agissait d'un site Web, par une URL. Un peu comme "`mailto:`" ou "`http:`" qui déclenchent les applications correspondantes lorsqu'ils sont exécutés. Vos applications peuvent utiliser ce mécanisme !

Devenir gestionnaire pour un nom de schéma

Le mécanisme consiste à permettre à une application de s'inscrire auprès de l'OS pour devenir gestionnaire par défaut d'un nom de schéma, d'une URI particulière.

Les applications conçues pour le bureau classique autant que les applications Windows Store peuvent utiliser ce mécanisme. Je n'ai pas testé avec une application WPF, mais au minimum le protocole fonctionne lorsqu'il invoqué depuis une telle application (avec un `shellexecute` sur le nom de protocole comme nous le verrons plus loin avec la boîte "*exécuter*" de Windows).

Si l'utilisateur confirme que l'application est bien gestionnaire par défaut de l'URI spécifiée, alors à chaque invocation de cette dernière ce sera cette application qui sera exécutée (avec la possibilité, comme toute URI, de lui passer des paramètres). La confirmation est demandée lorsque le protocole est appelé depuis IE ou Chrome par exemple. Lorsqu'il y a appel direct cette confirmation n'est pas demandée ce qui rend le mécanisme aussi transparent que l'association des extensions de fichier sous Win32.

Bien entendu il est conseillé d'enregistrer une application pour un schéma en rapport direct avec les services qu'elle rend. Si une application s'enregistre pour "`mailto:`" par exemple, l'utilisateur s'attendra à voir une fenêtre de saisie d'un nouvel e-mail s'ouvrir et non une page de publicité ou la création d'une fiche client…

Le CPA doit ainsi être vu comme une sorte d'amélioration du système d'enregistrement des extensions de fichier, revu à la "sauce Web", c'est à dire qu'ici il ne s'agit plus de double-cliquer sur un nom de fichier dans l'explorateur de Windows pour lancer une application mais de permettre à cette dernière d'être exécutée

depuis n'importe où (donc d'autres applications) sous la forme d'une simple URI… Fantastique non ?

Comment faire ?

C'est toujours la question qui se pose avec ces nouveautés, c'est bien sympathique dans l'esprit, mais comment mettre en œuvre la fonctionnalité ? Et comme toujours, la meilleure réponse consiste à montrer un peu de code… Et c'est ce que je vais faire !

Une app Windows Store minimale

Je vais partir d'une application Windows Store "vierge", c'est à dire le modèle "blank".

La première chose à faire après avoir créé le projet et de le modifier très légèrement afin d'extraire par refactoring (avec Resharper c'est bien entendu plus facile) la partie qui s'assure que l'application est créée et activée. A la base ce code se trouve dans `OnLaunched` de `App.Xaml.cs`.

Au passage il sera simplifié pour devenir :

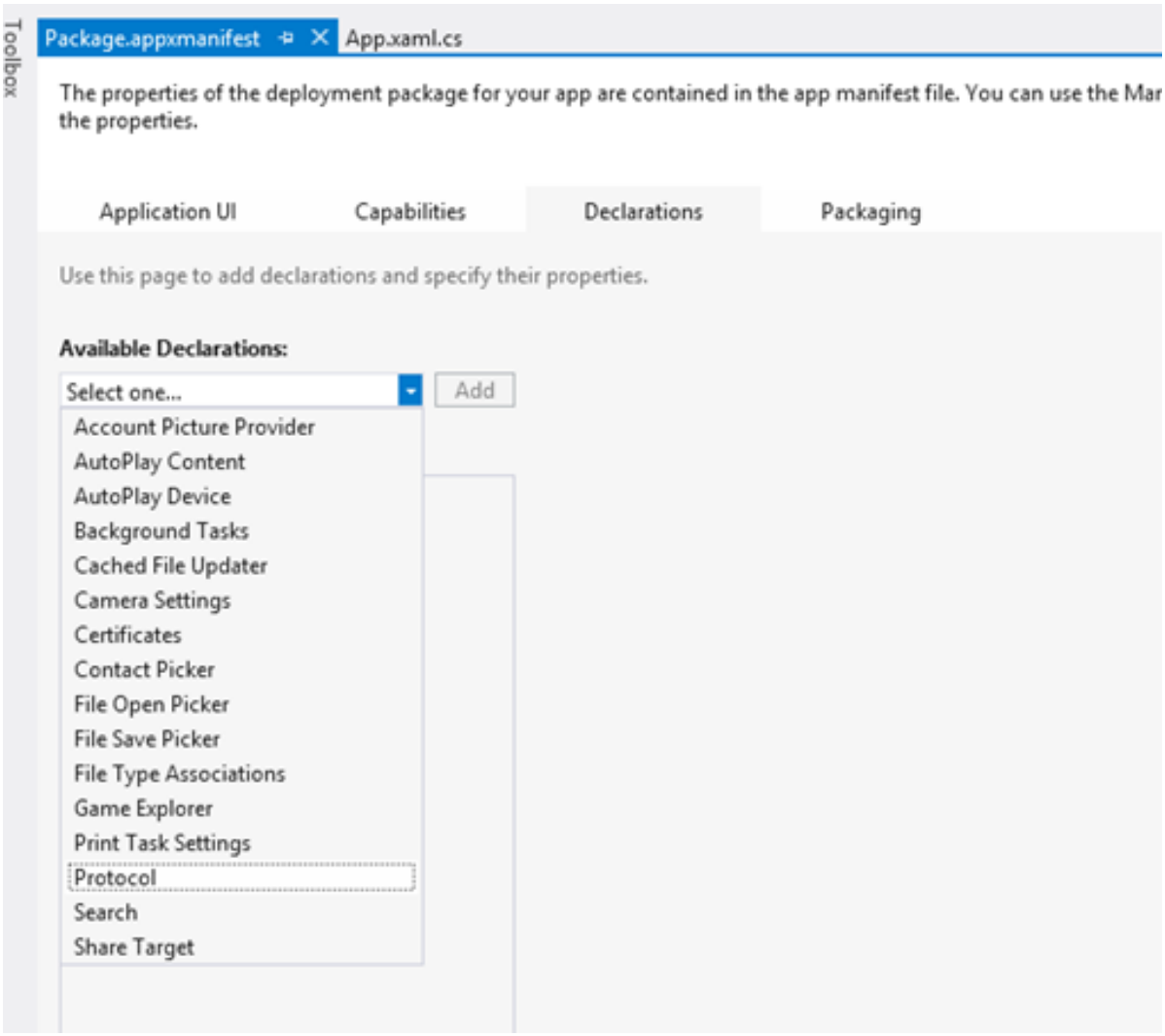
```
        private void EnsureAppIsCreatedAndActivated()
        {
            Frame frame;
            Window.Current.Content = frame =
                (Window.Current.Content as Frame) ?? new Frame();
            if (frame.Content==null)
                if (!frame.Navigate(typeof(MainPage)))
                    throw new Exception("Failed to create initial page");
            Window.Current.Activate();
        }
```

La méthode `OnLaunched` ne fait donc plus qu'un appel à cette nouvelle méthode, ce qui permettra de l'appeler aussi depuis un autre point d'entrée que nous verrons plus bas.

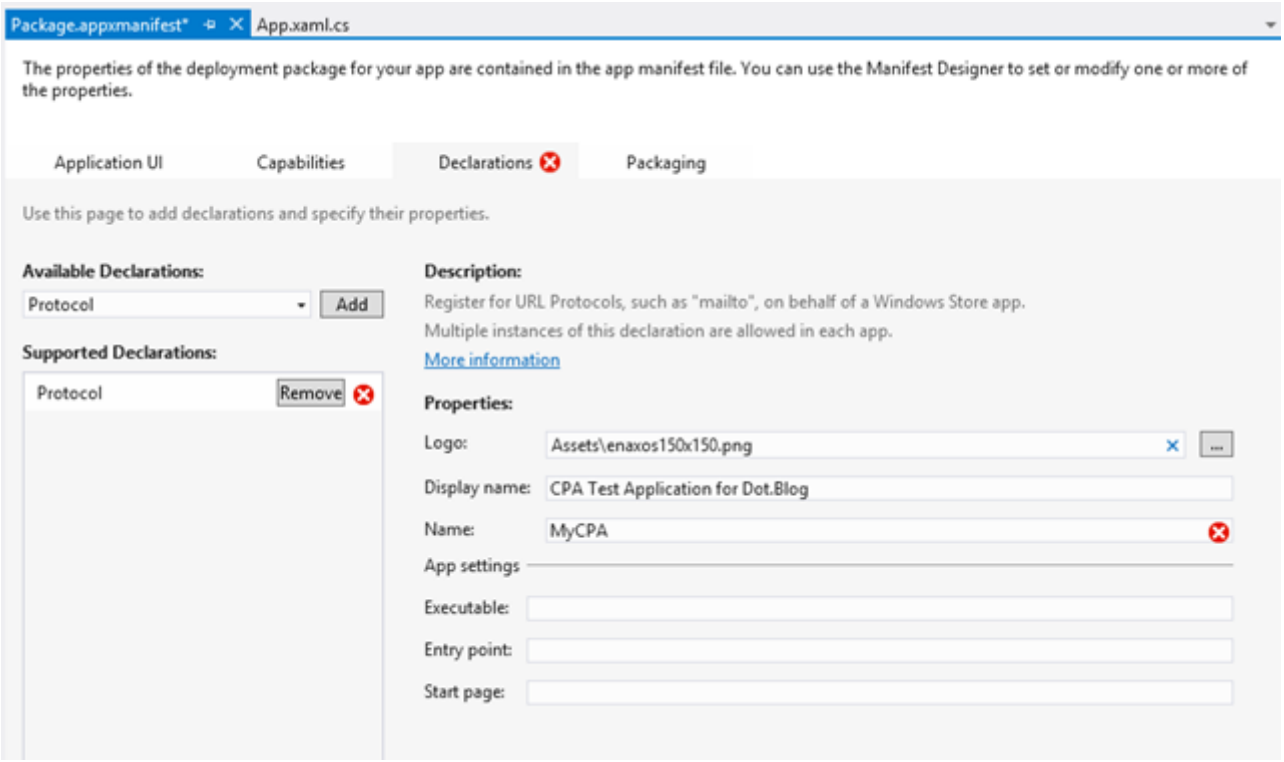
Se déclarer gestionnaire de Protocole personnalisé

La partie la plus importante de l'exemple intervient ici : nous devons déclarer à l'OS que notre application prend en compte la gestion de CPA.

Il suffit pour cela de double-cliquer sur le manifeste et d'aller dans l'onglet "*Declarations*" (comme pour s'intégrer à la chaîne de recherche, ce que je vous ai présenté dans un précédent billet).



Une fois "*Protocol*" sélectionné l'affichage du détail de celui-ci apparait à droite :



L'information la plus importante est "`Name`", c'est le nom de schéma par lequel l'application pourra être appelée.

Comme il est possible que deux applications entrent en conflit sur ce nom (il n'y a pas de réservation comme un nom de domaine…) Windows affichera la liste des applications ayant la même URI, cette liste sera complétée par le logo et le "`display name`", d'où la nécessaire prudente de spécifier ces informations.

Il n'y a pour l'instant aucune "norme" ni aucun répertoire centralisé pour les noms de protocoles personnalisés, et cela ne semble pas prévu. Je vous conseille ainsi d'éviter les noms qui auront toutes les chances d'être choisis par de nombreux développeurs… Evitez d'emblée ces noms et tentez une approche par le nom de votre société ou celui de l'un de vos site Web (sur lequel il y aura la promotion de vos applications par exemple pour être cohérent). Vous aurez beaucoup moins de chance que votre application et son protocole n'entrent en conflit avec d'autres applications.

Comme il est possible de récupérer l'URI entière, avec ses paramètres éventuels donc, tentez une approche de type "`NomDeSociétéOuSiteNomDApplication`", par exemple `EnaxosAppliTest`. Dans l'exemple de code j'ai choisi "`MyCPA`", un truc à éviter en production donc !

Attention ! Même s'il n'existe pas de répertoire central pour les noms de CPA, les noms suivants sont déjà réservés par Microsoft et il ne faut pas les utiliser faute de voir son application se faire rejeter à la validation du Windows Store… La liste actuelle est la suivante : Application.manifest, Application.reference, Batfile, BLOB, Cerfile,Chm.file, Cmdfile, Comfile, Cplfile, Dllfile, Drvfile, Exefile, Explorer.AssocActionId.BurnSelection,Explorer.AssocActionId.CloseSession, Explorer.AssocActionId.EraseDisc, Explorer.AssocActionId.ZipSelection,Explorer.AssocProtocol.search-ms, Explorer.BurnSelection, Explorer.CloseSession, Explorer.EraseDisc,Explorer.ZipSelection, FILE, Fonfile, Hlpfile, Htafile, Inffile , Insfile, InternetShortcut, Jsefile, Lnkfile,Microsoft.PowerShellScript.1, MS-accountpictureprovider, Ms-appdata, Ms-appx, MS-AUTOPLAY, Msi.Package,Msi.Patch, Ms-windows-store, Ocxfile, Piffile, Regfile, Scrfile, Scriptletfile, Shbfile, Shcmdfile, Shsfile, SMB,Sysfile, Ttffile, Unknown, userTileProvider,vbefile, Vbsfile, Windows.gadget, Wsffile, Wsfile, Wshfile.

Attention ! Le nom du CPA doit respecter des règles. Je n'ai pas trouvé (mal cherché ?) la liste exhaustive de ces dernières mais mes tests m'ont prouvé qu'il ne faut absolument pas utiliser de majuscules… Dans l'exemple de ce billet le protocole ne s'appelle donc pas "`MyCPA`", mais "`mycpa`". Sinon il y a une erreur de compilation…

Gérer l'activation via le protocole

Techniquement la déclaration du manifeste suffit pour que notre application puisse désormais être activée via son CPA.

Mais il reste à le matérialiser dans notre code pour que l'utilisateur puisse bénéficier de ce service (l'utilisateur ou les autres applications qui voudront lancer la vôtre).

Comme on peut le voir sur la capture précédente il existe des champs que je n'ai pas saisi comme "`Executable, Entry Point et Start Page`". Ces indications supplémentaires, comme il est facile de le comprendre, vous permettent d'aiguiller l'utilisateur vers une page donnée de votre application lorsqu'elle est activée via le CPA (ou de déclencher un traitement particulier).

Dans `App.Xaml.cs` la méthode "`OnLaunched`" est "*override*", car la classe application fournit une méthode virtuelle pour le lancement. De la même façon elle fournit une méthode virtuelle pour les activations via le CPA. En réalité la méthode virtuelle `OnActivated` est centrale, la façon dont l'application a été activée peut être connue par les arguments passés. C'est de cette façon qu'ici nous allons détecter l'activation

par le CPA et extraire l'URI passée pour la stocker dans une variable statique de l'application en vu de son exploitation :

```
public static string CPAValue { get; private set; }

protected override void OnActivated(IActivatedEventArgs args)
{
    if (args.Kind == ActivationKind.Protocol)
    {
        var protocolActivatedEventArgs =
                args as ProtocolActivatedEventArgs;
        if (protocolActivatedEventArgs != null)
            CPAValue =
             protocolActivatedEventArgs.Uri.ToString() ?? "null";
    }
    EnsureAppIsCreatedAndActivated();
}
```

Il y a bien entendu d'autres façons plus subtiles d'utiliser l'URI passée que de la stocker dans une variable statique. Ce n'est qu'un exemple simplificateur pour le besoin du billet.

La partie visuelle

En réalité l'exemple s'arrête là. Nous avons enregistré le protocole personnalisé de notre application dans l'OS, nous avons surchargé la méthode `OnActivated` pour récupérer cette activation spéciale et la valeur de l'URI, tout est déjà fonctionnel. Ce que vos applications feront du CPA qu'elles auront déclaré est totalement libre !

Mais pour les besoins de l'exemple je vais ajouter un visuel minimaliste à l'application et surtout un moyen de vérifier que nous avons bien récupéré l'URI.

Pour cela j'ajoute quelques éléments visuels pour donner "vie" à la page noire qui sinon s'affichera par défaut et je place un `TextBlock` et un bouton pour afficher l'URI.

Le code du bouton (codé dans le code-behind, pas de MVVM dans cet exemple !) est ultra simple :

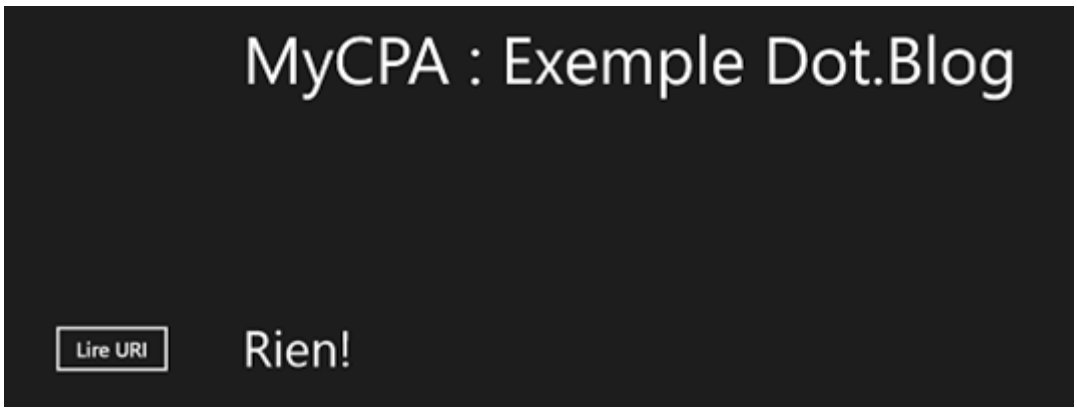
```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    txtURI.Text = App.CPAValue ?? "Rien!";
}
```

Sur le clic du bouton je récupère le contenu de la variable statique CPAValue créée dans `App.Xaml.cs` avec un test de nullité qui affichera "*Rien !*" si l'URI n'est pas définie.

C'est terminé… Reste à voir comment cela fonctionne.

CPA en Action

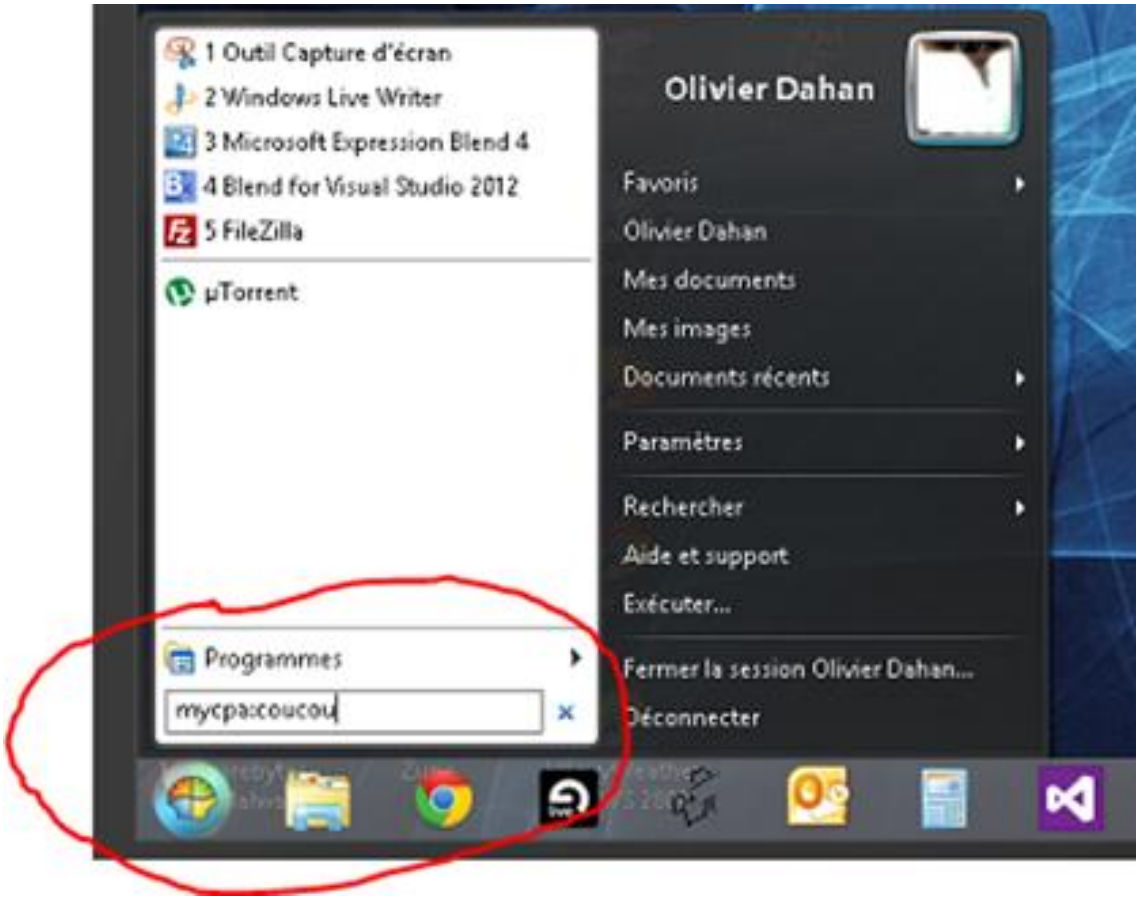
Je vais choisir d'exécuter l'application dans le simulateur. Dans un premier temps l'application sera donc juste lancée par l'environnement de Debug de Visual Studio. Je vais cliquer sur le bouton et voici l'affichage que j'obtiendrai :



Il n'y a rien…

En effet il ne faut pas confondre exécution et activation ! Lorsque mon application est lancée (`OnLaunched` côté événements) elle n'est pas activée par le CPA mais par l'OS de façon classique. Il faudrait d'ailleurs tester les particularités du tombstoning pour savoir s'il faut ou non rétablir le contexte utilisateur par exemple (ce qui n'est pas fait ici).

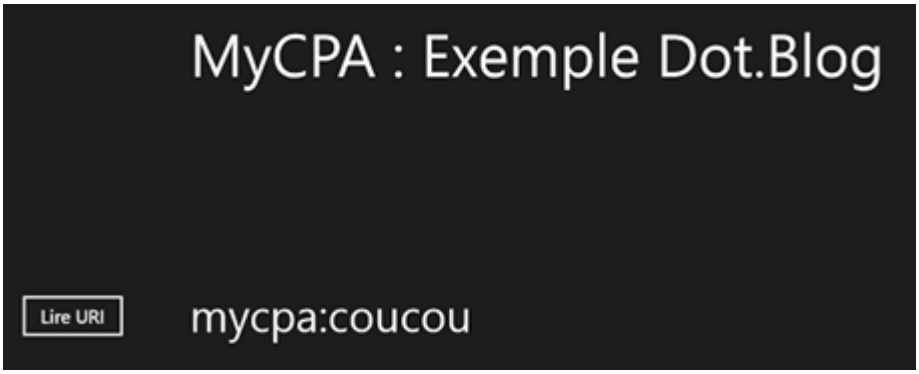
Je vais donc fermer l'application totalement (par ALT-F4) et passer par le bureau classique pour invoquer non plus mon application (qui s'appelle `CPATest.exe`) mais son protocole en lui passant des paramètres…



Les connaisseurs reconnaîtront Classic Shell, un utilitaire open source dont j'ai parlé dans ces colonnes et qui permet de retrouver un bouton "démarrer" à la Windows 7 sur le bureau classique de Windows 8, quelque chose d'indispensable à mon sens.

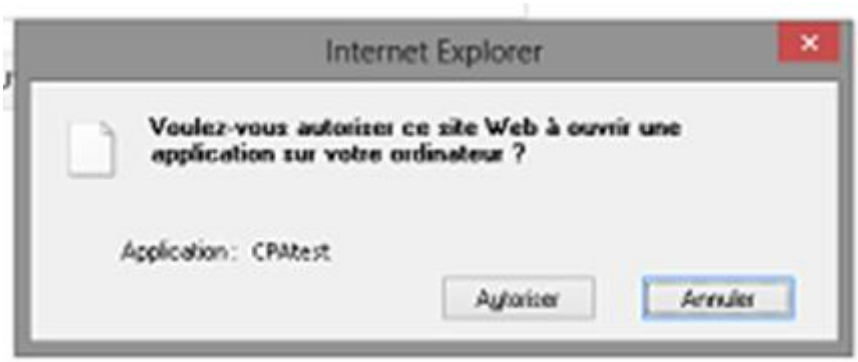
Si vous n'utilisez pas cet outil, vous pouvez exécuter une application en invoquant la boîte de dialogue "`run`" qui existe toujours (je n'utilise plus Windows 8 sans le Classic Shell et j'avoue que je ne pourrais plus dire comment la manip doit être faite en mode "out of the box" … mais vous trouverez j'en suis sûr !).

Bref j'ai tapé "`mycpa:coucou`" et j'ai ensuite validé. On remarquera que je n'appelle pas mon exécutable dont, à la limite, je ne connais pas le nom… J'appelle le nom de mon protocole, "`mycpa`". Il est suivi du symbole deux points comme "`ftp:`" "`http:`" "`mailto:`" etc, puisque de facto c'est bien un nouveau protocole que j'ai créé ! Les deux points sont suivis d'un texte quelconque (les paramètres acceptés par mon protocole). Comme l'exemple ne gère pas du tout les paramètres et ne cherche pas à les extraire, c'est toute la chaîne qui va être réceptionnée tel quel par l'application et affichée sans modification ni traitement, et là quand on clique sur le bouton"*Lire URI*" nous obtenons l'affichage suivant :

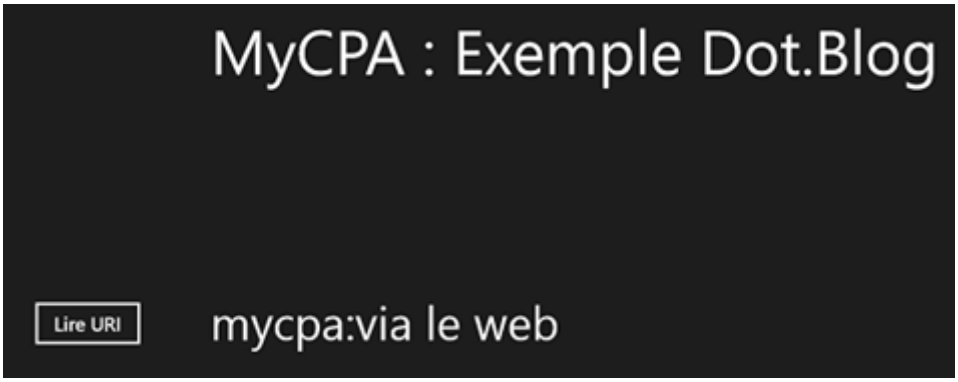


Incroyable non ?

Plus bluffant encore, exécutez Chrome ou IE et dans la barre d'adresse internet tapez "`mycpa:via le web`" et, surprise le dialogue suivant apparait :



La capture est assez mauvaise car dans le simulateur tout est affiché très petit (mon bureau Windows qui est simulé est très large). Mais on peut lire que IE nous demande de confirmer l'exécution de "`CPATest`" (cette fois c'est le nom de l'exécutable). Si on valide (et qu'on clique sur le bouton permettant de connaître le contenu de l'URI passée) :



Notre application devient même une adresse Web qui pourrait être bookmarkée… Génial.

Conclusion

Voici un mécanisme à la fois totalement nouveau, très ouvert, fort simple à implémenter, qui ouvre de réelles perspectives pour la programmation sous WinRT.

S'il est normal de se poser des questions sur l'avenir de Windows 8, comme de tout nouvel OS et même tout nouveau logiciel en général, et s'il est légitime de s'interroger aussi sur le devenir de WinRT sur les PC ou les tablettes, je pense qu'on peut affirmer que Microsoft a réellement produit quelque chose de neuf et de novateur, bourré de bonnes idées.

Il serait vraiment injuste de juger WinRT et Modern UI uniquement sur le battage "grand public" ou sur une simple guerre "pro ou anti Ipad/IPhone".

Apple à toujours volé tous ces concepts aux autres, Apple à volé tous ses designs (vous connaissez certainement cette comparaison entre l'electro-ménager Braun et le design des appareils Apple, confondant non ?), Microsoft innove réellement. Tant sur le design que sur le fond. Ca vaut la peine de donner une chance à WinRT !

Donner du "charme" à vos applications Windows 8

Windows 8 propose des mécanismes nouveaux à l'utilisateur pour interagir avec les applications. Les "Charms" (charmes) sont les outils qui apparaissent à droite de l'écran. Parmi eux le Charme de Recherche. Une fonction simple que Windows 8 sublime en autorisant chaque application à devenir fournisseur de résultats. Tout un monde s'ouvre. Tombons sous le … charme…

Des charmes charmants pour plus de charme

La barre des charmes est affichée à droite de l'écran sur la capture ci-dessus. Elle apparait quand on place la souris dans les coins ou avec une gestuelle de même type sur un écran tactile.

Il y a plusieurs charmes, celui qui nous intéresse aujourd'hui est celui de la recherche.

Rechercher dans les applications

La fonction de recherche en elle-même est assez ancienne sous Windows. Les dernières versions l'avaient encore améliorée mais Windows 8 va beaucoup plus loin dans son esprit d'unification et d'intégration. Vos applications peuvent désormais, et facilement, s'intégrer à la chaîne de recherche et fournir des résultats exploitables immédiatement par l'utilisateur.

On voit ici tout le potentiel d'une telle recherche même pour des applications LOB : depuis le menu principal de Windows 8, toute frappe débute une recherche sans même besoin d'appeler le charme. Imaginez une gestion de clients qui affiche dans les résultats de Windows un accès direct à la fiche du client, sans avoir à lancer l'application, sans avoir à passer par ses écrans de recherche, son menu principal, rien. Juste depuis le menu Windows 8 en tapant quelques lettres l'utilisateur accède par du "*deep linking*" exactement à ce qu'il cherche. Il obtiendra peut-être dans les résultats d'autres réponses d'autres logiciels, le carnet d'adresse pour envoyer un mail à ce client, le compte facebook du contact, les fichiers Excel classés sous ce nom, etc.

C'est tout l'environnement de travail qui se trouve ainsi accessible en quelques touches.

Et vos applications peuvent en profiter.

Pas d'API hyper sophistiquée, pas de header C, pas de combine Javascript, ni de JSon à parser, rien d'exotique…

Le charme de recherche est un élément essentiel de Windows 8 qui permet à vos applications, LOB ou grand public, de se rappeler à la mémoire des utilisateurs… et lui fournir des données utiles !

La dérive… C'est selon votre éthique !

Mais nul doute que des petits malins trouveront vite ici de quoi se faire de la publicité gratuitement !

Ne vous êtes-vous jamais demandé pourquoi sur votre Smartphone ou votre tablette vous recevez tant de mises à jour ?

Les applications sont généralement simples, plus simples que des applications de bureau, seraient-elle si mal programmées qu'il leur faille des mises à jour toutes les semaines ?

Bien sur que non. Pour certaines cela est possible, mais les mauvais programmeurs ne se soucient guère de mettre à jour si souvent leurs applications…

Alors ?

Et bien c'est simple : Sous iOS ou Android pour l'instant (un jour peut-être pour Windows 8) il y a des centaines de milliers d'applications. Vous en téléchargez plein. Vous n'en utilisez finalement que peu. Beaucoup sont oubliées au fond de la mémoire. En fournissant régulièrement des mises à jour, les auteurs vous obligent à voir une notification qui vous rappelle… que cette application est sur votre téléphone : qu'elle EXISTE.

Le B.A.BA de la publicité et de la concurrence c'est d'affirmer son existence, se faire VOIR et être VU.

Pousser régulièrement des mises à jour est une astuce qui détourne un mécanisme technique en un procédé publicitaire, tout simplement…

Le rapport avec les charmes Windows 8 ?

Il est simple : toute application Windows 8 devrait, dans ce même esprit, être fournisseur de recherche. Pour la simple raison que même si elle n'a pas grand chose à proposer, son résultat sera affiché comme les autres et cela rappellera son existence à l'utilisateur…

Du point de vue de l'éthique c'est plus que discutable. Du point de vue marketing dans un cadre ultra concurrentiel cela peut s'avérer payant.

A vous de doser et de retourner des informations qui sont un minimum pertinentes… Sinon l'utilisateur retirera votre application de la chaîne de recherche. Il existe tout de même une sanction au manque d'éthique…

De même pour les mises à jour. Si votre soft doit contenir 20 fonctionnalités phare, implémentez-les toutes mais coupez-en 5 ou 6 en commentaire dans votre source. Faites des mises à jour régulièrement en débloquant une fonction, ce qui justifiera pleinement la mise à jour et ne fera pas trop "fumiste". Cela donnera en plus à l'utilisateur une impression de sérieux : vous écoutez les demandes, vous satisfaites vos clients ! C'est un peu comme en amour, celui (ou celle) qui donne toute sa flamme et son énergie en 5 minutes risque de décevoir dans la durée, il faut savoir ménager des surprises !

Mais bon, cela reste entre nous, je ne vous ai rien dit, vous n'avez rien lu. Ces choses sont en contradiction totale avec mon éthique. Mais cela se pratique et c'est bien d'être au courant !

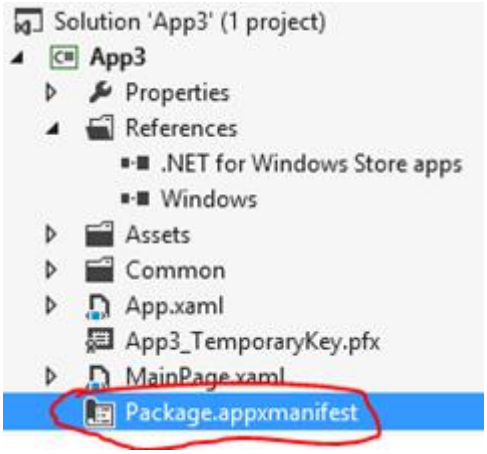
A chacun de voir…

Intégrer son application dans le charme de recherche

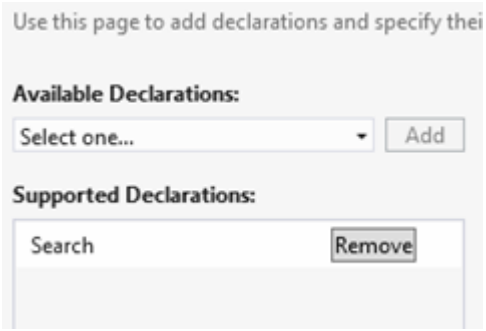
Passons aux choses sérieuses (quoi que je sois certain que beaucoup de lecteurs n'aient pas conscience des petites révélations ci-dessus…).

Pour commencer il faut une application Windows Store et travailler en VS 2012 sous Windows 8 ou VS 2013 pour Windows 8.1. C'est un préalable qui fait sens, il n'y a pas d'émulateur Windows 8.x pour Windows 7, en dehors de Windows 8.x lui-même !

Pour participer aux opérations de recherche il faut le demander. Cela s'effectue via le manifeste de l'application. Double-cliquez sur le fichier `Package.appmanifest` et cliquez sur l'onglet "`Declarations`".



Ouvrez la droplist de combobox et sélectionnez "`Search`" puis cliquez sur "`Add`" pour ajouter la déclaration au manifeste. La recherche apparaitra alors dans la section "Supported Declaraction" (déclarations supportées).

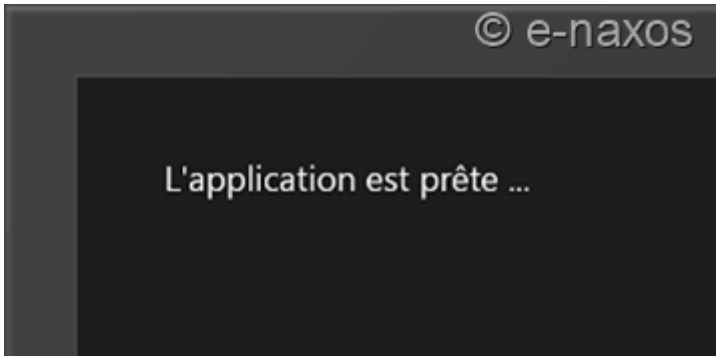


D'autres informations peuvent être précisées mais pour cet exemple nous allons faire très simple.

D'ailleurs maintenant que notre application est fournisseur de résultats de recherche encore faudrait-il savoir ce qu'elle va bien pouvoir répondre… Pour cet exemple elle va répondre qu'elle a trouvé quelle que soit la recherche (un peu comme certains site Web quand vous cherchez quelque chose sous Google… encore une feinte !). Quand l'utilisateur va cliquer, nous n'allons pas lui afficher une liste de sites pornos ni le rediriger vers un site de phishing, laissons cela aux sites Web, nous nous contenterons d'afficher le terme ou l'expression recherchée.

L'effet obtenu

Ci-dessus voici ce que l'application affiche lorsqu'elle est lancée :



Forcément, elle ne fait pas grand chose. On n'est même pas au maximum de ce qu'on pourrait faire avec un simple `TextBlock`…

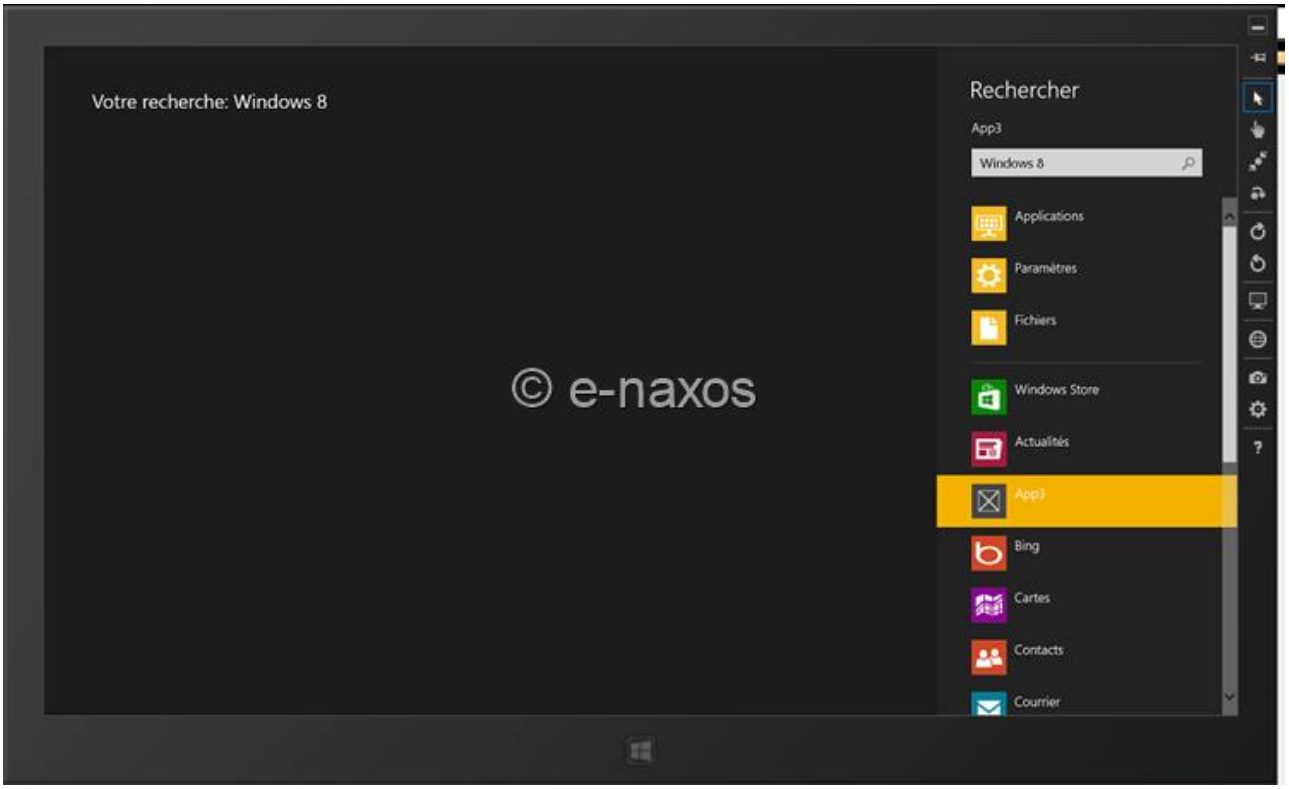
Maintenant revenons au menu Windows 8 et appelons le charme de recherche. Dans la zone de recherche je tape "*Windows 8*". Au fur et à mesure de la frappe on voit dans la partie principale le filtrage s'effectuer (par défaut il s'agit de la liste des applications installées).

Mais le plus important est sous la zone de saisie : à cet endroit on trouve des applications qui peuvent répondre à la recherche. Certaines sont installée de façon prioritaire, les autres sont dans une seconde liste juste en dessous, comme "`App3`" le nom de notre application de test.

En cliquant sur "`App3`" la recherche se poursuit mais Windows active notre application en mode recherche et cette dernière affiche alors ses résultats.

La capture ci-dessous (dans le simulateur) montre tout cela, et on peut voir notre application afficher fièrement "*Votre recherche : Windows 8*".

Cet exemple ne fait rien d'utile il faut l'avouer, mais je suis certain que vous en saisissez tout le potentiel !



Un peu de code

Il en faut tout de même un peu !

Je vous ferai grâce de celui de l'interface qui se limite à un `TextBlock`.

Le code-behind de la `MainPage` n'a rien d'exceptionnel non plus en dehors du fait que nous allons avoir besoin d'accéder à l'instance de la page et de lui envoyer le message à "rechercher", ce qui ce limitera en réalité ici à l'afficher.

Pour ce faire on ajoute une variable statique "`Instance`" initialisée dans le constructeur, comme on le ferait pour un Singleton.

Ensuite on ajoute une méthode publique, par exemple "`ShowMessage(string message)`" dont le seul but est d'afficher le texte qui lui sera passé (en réalité ce devrait être la méthode qui effectue le travail de recherche).

Je passe sur tout cela qui est vraiment trivial (d'autant que les sources du projet sont téléchargeables en fin d'article).

Le "vrai" travail sera accompli dans `App.xaml.cs`. Dans un premier temps il nous faut overrider `OnSearchActivated` pour répondre à l'évènement de recherche :

```
 async protected override void OnSearchActivated(SearchActivatedEventArgs
args)
        {
            await EnsureMainPageActivatedAsync(args);
            if (string.IsNullOrWhiteSpace(args.QueryText))
            {
                // page d'accueil
            }
            else
            {
                MainPage.Instance.ShowMessage("Votre recherche:
"+args.QueryText);
            }
        }
```

C'est vraiment très simple. Si la chaîne de recherche se trouvait être nulle ou vide, on devrait naviguer vers une page neutre indiquant qu'il n'y a rien à chercher. A vous de voir en fonction du contexte ce que votre application peut proposer dans un tel cas.

Dans le cas contraire, c'est là que nous récupérons l'instance statique de `MainPage` pour utiliser la méthode publique de recherche. Ici il s'agit de `ShowMessage` qui ne fait qu'afficher le terme cherché, dans la réalité la méthode appelée fera un vrai de travail de recherche dans son domaine de compétence pour afficher un résultat ayant un sens (liste de clients, d'articles correspondant au terme de recherche avec lien pour voir la fiche détail par exemple).

Vous aurez surement remarqué que la méthode est marquée "`async`"'. Il y a une raison à cela : dans un premier temps nous voulons nous assurer que la `MainPage` a bien été activée. Et comme tout est asynchrone sous Windows 8 ce contrôle sera lui aussi asynchrone, d'où le besoin de le faire précéder de "`await`" pour rendre le code plus simple (merci C# 5 !) ce qui en retour impose donc de marquer la méthode avec "`'async`"…

Mais d'où vient la méthode asynchrone `EnsureMainPageActivatedAsync` ?

De notre propre code…

```
async private Task EnsureMainPageActivatedAsync(IActivatedEventArgs args)
        {
            if (args.PreviousExecutionState ==
                        ApplicationExecutionState.Terminated)
            {
        // restoration de l'état en asynchrone, raison du async plus haut.
            }

            if (Window.Current.Content==null)
            {
                var rootFrame = new Frame();
                rootFrame.Navigate(typeof (MainPage));
                Window.Current.Content = rootFrame;
            }
            Window.Current.Activate();
        }
```

Pourquoi ce méthode est-elle marquée "`async`" alors qu'elle n'utilise pas "`await`" ? Parce que son code n'est pas complet… Vous remarquerez que la première chose qui est faite est de vérifier l'état antérieur de l'application.

Si jamais elle était "*terminated*" il serait nécessaire de la recharger, de remonter son dernier contexte d'exécution, comme pour une application Windows Phone… Et cette initialisation a toutes les chances de faire appel à des traitements asynchrones. Ils seront vraisemblablement précédés de "`await`". D'où la nécessité de marquer "`async`" la méthode. En l'état le "`async`" ne sert à rien et la procédure sera purement synchrone, ce que VS nous précise à la compilation.

Dans le cas où il n'y aurait aucune Frame, il sera aussi nécessaire d'en créer une et d'activer la `MainPage` (et sa Frame).

Cette méthode a donc pour but de s'assurer que la `MainPage` est bien chargée, dans une Frame reconnue et non vide, elle s'applique aussi à recharger le contexte de l'application si celle-ci a été terminée par l'OS.

Ces mécanismes sont nouveaux pour de la programmation Windows, il est donc essentiel de bien en comprendre le fonctionnement et la raison d'être…

C'est tout pour le code. Il n'y a vraiment pas grand chose, mais la recherche ne fait aucun vrai travail, nous avons tout juste créée un perroquet qui répète ce qui est cherché.

A vous de mettre du vrai code qui fournit de vraies réponses !

Conclusion

Si les tuiles que nous avons vues dans un précédent billet sont des éléments de base de Windows 8, les Charmes le sont tout autant. Bien comprendre la programmation des tuiles et des charmes est même la base de la programmation de Windows 8. Le reste n'est finalement que du Silverlight…

Bien entendu il reste malgré tout beaucoup de choses à savoir sur WinRT. Cette API est gigantesque et Windows 8 est un OS plein de surprises (généralement bonnes). Alors pour être sûr de ne rien louper, abonnez-vous au flux de Dot.Blog.

WinRT réinvente les Ria Services (les nouveaux WCF Data Services)

Les Ria Services, cette merveille de sophistication et de simplicité qui permet sous Silverlight d'écrire des applications orientées données en un claquement de doigts… WinRT en C#/Xaml si proche de Silverlight… Finalement la fusion se fait : tout ce qu'il y avait de bon dans Silverlight se retrouve dans WinRT, même les Ria Services, sous le nom de Data Services. Un exemple vous en dira plus long…

Le tooling

La première des choses à considérer c'est le tooling. Voici notre trousse à outils pour accomplir le miracle :

- Windows 8
- Visual Studio 2012
- Les WCF Data Services 5 (à télécharger ici : http://www.microsoft.com/fr-fr/download/details.aspx?id=29306)
- Mais aussi les Data Services Tools for Windows Store Apps (ici : http://www.microsoft.com/en-us/download/details.aspx?id=30714)
- SQL Serveur 2012
- Une base de données de test (je vous conseille Adventure Works for SQL Server 2012, http://msftdbprodsamples.codeplex.com/releases/view/55330)

Si vous utilisez Visual Studio 2013 ou Windows 8.1, il faudra certainement utiliser des versions plus récentes de ces librairies. WCF Data Services 5.6.0 est disponible et des mises à jour existent sous la forme de paquets Nuget.

L'exemple développé ici permet de comprendre l'esprit de la librairie sous réserve de « breaking changes » intervenus dans les mises à jour récentes.

Grâce à cet ensemble nous allons pouvoir travailler presque comme avec les Ria Services sous Silverlight, et si c'est un petit pas pour le développeur c'est un pas de géant pour l'avenir des applications LOB sous Windows 8, et pour Windows 8 lui-même peut-être !

En passant, trouvez-vous aussi un endroit calme pour travailler (je viens de refaire mon bureau, ça aide à la concentration ! un petit aperçu d'une partie des installations que je profite d'exhiber avec fierté tant que c'est rangé ☺).

Construire l'exemple

Plaisanter est une chose, faire quelque chose d'utile avec cet attirail de geek en est une autre, donc au boulot !

Créer le projet Windows Store

Première étape de cet exemple dont le seul but est de montrer que tout cela fonctionne d'ores et déjà (même si le manque de docs et d'exemples rend la tâche plus ardue) : créer un nouveau projet C#/Xaml pour Windows Store.

Seconde étape : personnaliser le manifeste de l'application pour lui donner les droits d'accès réseau. C'est un détail à ne pas oublier ou rien ne marchera.

J'ai choisi de déboguer en mode simulateur plutôt qu'en mode machine locale, cela évite de polluer la machine (et puis il existe des petits bugs liés à des handles de fichiers non relâchés qui empêchent parfois le déploiement, sur le simulateur il suffit de tout fermer et de rouvrir, en local il faut faire un reboot plus contraignant).

Vous pouvez aussi préférer travailler dans une machine virtuelle, mais Windows 8.1 est aujourd'hui une version mature utilisable en production et un vrai setup est toujours plus efficace qu'une émulation. Si vous préférez une VM je vous conseille Virtual Box, ça marche très bien avec Windows 8.x.

Pour l'instant nous allons laisser ce projet en l'état, nous n'y avons pas beaucoup touché mais sans les données nous ne pouvons pas avancer plus.

Créer le projet Web

Dans la solution qui a été ajoutée automatiquement autour du précédent projet nous ajoutons maintenant un projet Web application, le plus simple possible sans rien dedans.

Les données

Il faut disposer d'un serveur et d'une base de données. Vous pouvez prendre ce que vous voulez pour les tests si vous avez déjà une base dédiée à ce genre de travail. Sinon installez la base exemple indiquée dans les liens en début de billet.

Comme tout fonctionne pour le moment comme sous Ria Services, je ne vais pas entrer dans les détails.

Il faut bien entendu ajouter au projet Web un modèle Entity Framework. Je n'ai utilisé qu'une seule table de la base de données.

Ensuite il faut ajouter un WCF Data Service.

Le squelette créé par défaut mérite une petite adaptation pour que notre exemple fonctionne : exposer le Dataset "`Employees`" du modèle EF :

```
namespace WebApplication1
{
    public class WcfDataService1 : DataService< AdventureWorks2012Entities >
    {
        // This method is called only once to initialize service-wide policies.
        public static void InitializeService(DataServiceConfiguration config)
        {
            config.SetEntitySetAccessRule("Employees", EntitySetRights.AllRead);
            config.DataServiceBehavior.MaxProtocolVersion =
                DataServiceProtocolVersion.V3;
        }
    }
}
```

C'est peu de choses, mais c'est essentiel pour exposer les données en OData. Vous remarquerez qu'ici je n'ai donné que les droits en lecture sur la table.

Il est possible de lancer en mode debug le projet Web pour tester le service (clic droit sur le projet puis exécuter en debug).

Le navigateur affichera une erreur 403 car le répertoire n'est pas browsable et que nous n'avons aucune page par défaut dans ce projet. Cela permet toutefois de noter l'adresse du service (par exemple `localhost:65373`) ce qui nous servira pour l'application cliente.

On peut aussi vérifier que l'accès à la ressource déclarée est possible.

Dans l'exemple qui utilise des noms par défaut, l'accès se fait par l'adresse suivante :

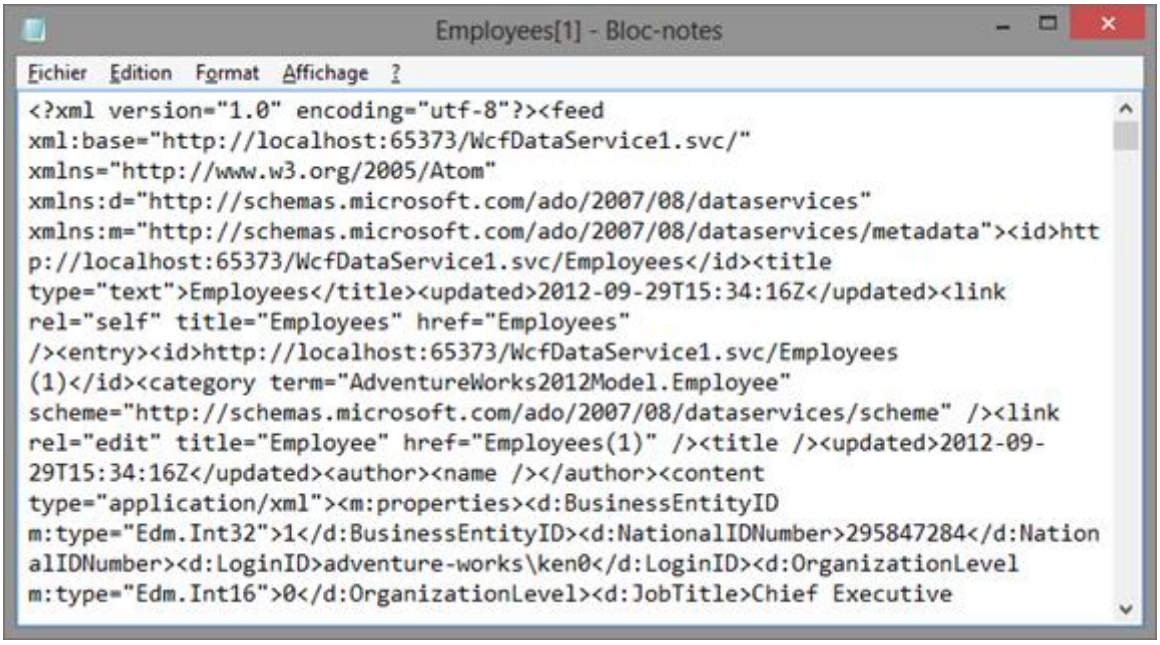
`http://localhost:65373/WcfDataService1.svc/Employees`

"`Employees`" étant le dataset unique du modèle EF.

S'agissant d'un flux OData, le navigateur le prend pour un flux RSS et passe dans un mode d'affichage peu pratique :

Cela suffit pour voir que le service répond, qu'il renvoie bien un flux OData reconnu comme tel, et on peut même voir qu'il y a 290 enregistrements retournés.

En demandant l'affichage du code source de la page on peut s'assurer que toutes les données sont bien renvoyées (ici en xml) :



C'est loin d'être lisible mais on y retrouve les données réclamées.

La partie cliente

La mise en page d'une application Modern UI se rapproche beaucoup de Silverlight en plus "calibré" et plus simple (si on suit les directives de MS). Je vais donc faire très simple pour l'exemple : une `Listbox` avec une petit `DataTemplate` pour afficher la liste des enregistrements et une poignée de `TextBlock` et `TextBox` pour le détail de l'enregistrement sélectionné. Un bouton "`Load`" permettra de charger les données.

Les informations de détail sont placées dans une `Grid` dont le `DataContext` est lié à l'item sélectionné de la `ListBox`. Tout le reste n'est que Binding rudimentaire sur lequel je passerai.

Une fois terminé cet écran ressemble à cela :

On notera l'observance la plus stricte du minimalisme cher à Modern UI ! (En réalité on peut faire de très belles choses en Modern UI, à condition de ne pas respecter toutes les règles édictées par MS, c'est donc un choix à faire…).

Mais l'essentiel de notre affaire n'est pas dans la mise en page…

Il nous manque le principal : les données.

Ajouter le service

C'est là que les outils indiqués en début de billet vont produire leur effet magique… Notamment les tools pour WCF Data Services 5.

On procède en réalité de la même façon que pour ajouter un service Web sous Silverlight : dans les références on ajoute un service, ce qui amène un dialogue classique pour cette fonction. En cliquant sur le bouton de découverte VS va trouver le service exposée par le projet Web qui se trouve dans la même solution et va nous permettre de l'importer.

Une fois le service référencée, "y'a plus k'a !".

C'est là que ça se corse, car en absence d'exemples fiables et d'une documentation explicite, c'est un peu sportif : le principe est le même que les Ria Services mais avec suffisamment de différences pour que rien de ce qu'on essaye ne fonctionne :-)

Il faut donc retrousser ses manches et plonger dans le code, investiguer les types retournées ici et là, tenter d'interpréter les exceptions. Bref le travail de débroussaillage classique du pionnier qui essuie les plâtres. J'ai l'habitude, c'est un avantage certain.

Je vais passer sur ces recherches et sur le tâtonnement ainsi que sur les explications complètes du pourquoi du comment. Je publierai d'autres billets qui détailleront les mécanismes en jeu une fois que j'aurai la certitude d'avoir tout compris et de n'avoir rien loupé. Ce billet est un avant goût, une preuve fonctionnelle de faisabilité, pas un article sur les détails de WCF Data Services 5. J'y reviendrai mais pour ce que j'en ai vu OData rend les choses très complexes. Je préfère mille fois le fonctionnement des RIA Services « classiques », j'avoue même avoir abandonné un projet modeste commencé en OData sous Silverlight pour le refaire « proprement » avec les RIA Services tellement chaque petite chose devenait un enfer à coder… Mais bon, je parle de cela il y a un an, les choses ont certainement évolué et OData peut offrir des avantages dans certains cas. Je ne voudrais pas vous écoeurer avant même d'avoir terminé l'exemple !

Accéder aux données

Donc, comme nous l'avons vu sur la mise en page, il y a un bouton "`Load`". Ce bouton va déclencher le chargement des données.

Tout est asynchrone sous Windows 8, et si nous étions déjà habitué à ce mode de fonctionnement avec Silverlight et les Ria Services, c'est dans la façon de mettre en œuvre cet asynchronisme que ce trouve les pièges de WinRT…

Au début c'est facile, il suffit de créer un contexte, exactement comme avec les Ria Services.

Mais en fait ça se complique tout de suite un peu puisqu'il faut passer une URI.

C'est là qu'il est intéressant d'avoir testé le service (voir un peu plus haut) et d'avoir noté l'adresse…

Pour créer le contexte on procédera donc comme suit :

```
 if (context == null) context =
      new AdventureWorks2012Entities(
           new Uri("http://localhost:65373/WcfDataService1.svc/",
UriKind.Absolute));
```

La variable “context” a été déclarée de cette façon :

```
private ServiceReference1.AdventureWorks2012Entities context;
```

Le premier appui sur le bouton Load créera le contexte, les appuis suivants, s’il y en a, récupéreront le contexte déjà créé.

Tout va résider maintenant dans la façon d’envoyer la requête au server et d’interpréter le résultat.

```
async private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            try
            {
                if (context == null) context =
                 new AdventureWorks2012Entities(
                    new Uri("http://localhost:65373/WcfDataService1.svc/",
                           UriKind.Absolute));

                var query =
                 (DataServiceQuery<Employee>)
                     (from ee in context.Employees
                      orderby ee.BirthDate select ee);

                var data = await query.ExecuteAsync();

                lb1.ItemsSource = data;
            }
            catch (DataServiceQueryException ex)
            {
                throw new Exception("Erreur: " + ex.Message);
            }
        }
```

On pense au départ qu’une requête Linq fera l’affaire… C’est vrai, mais ce n’est pas suffisant.

Comme on le remarque dans le code ci-dessus, la requête Linq est transtypée en “`DataServiceQuery<Employee>`”. Resharper m’indique qu’il s’agit d’un transtypage un peu étrange car il ne trouve pas ce qu’il y a de commun entre un résultat de requête Linq et le type `DataServiceQuery`.

J'avoue que je reste aussi perplexe que lui pour l'instant faute d'avoir vraiment bien compris le jeu entre ces différentes classes. Mais cela viendra et je vous dirai tout !

L'avantage de ce transtypage est de pouvoir accéder à certaines fonctions qui permettent d'exécuter la requête en mode asynchrone.

Toutefois je voulais me servir des nouveaux modes de C# 5 (`await` et `async`) plutôt que de programmer un Callback à "l'ancienne".

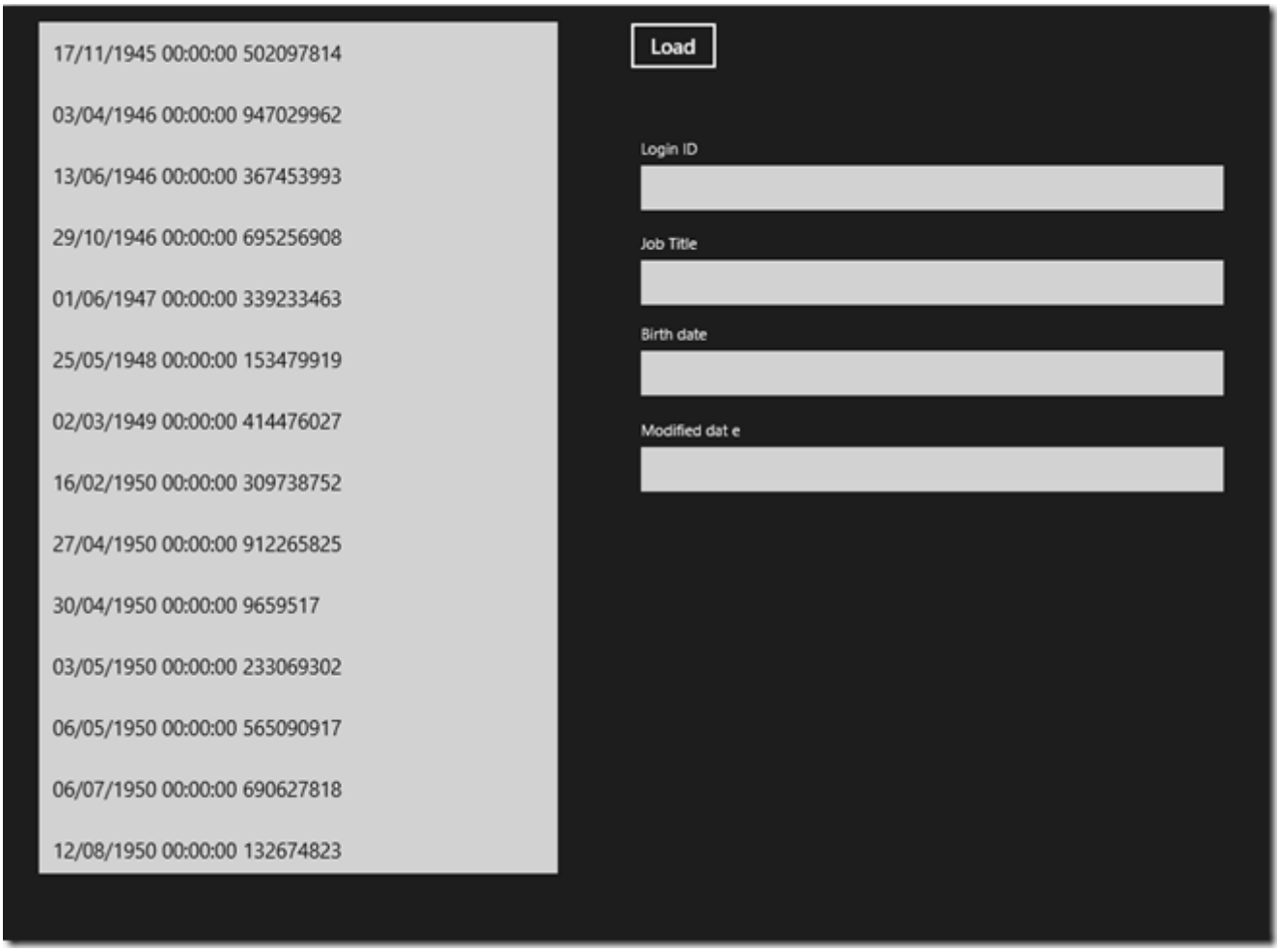
C'est pourquoi la méthode `Button_Click` est précédée du marqueur `async` car je vais utiliser `await` dans son corps.

La variable "`data`" est ainsi assignée via un `await` sur l'opération d'exécution de la requête.

En réalité, `ExecuteAsync()` est une extension que j'ai ajoutée. Elle permet d'appeler l'exécution de la requête asynchrone sans se soucier des détails et d'obtenir un code lisible grâce à `await`.

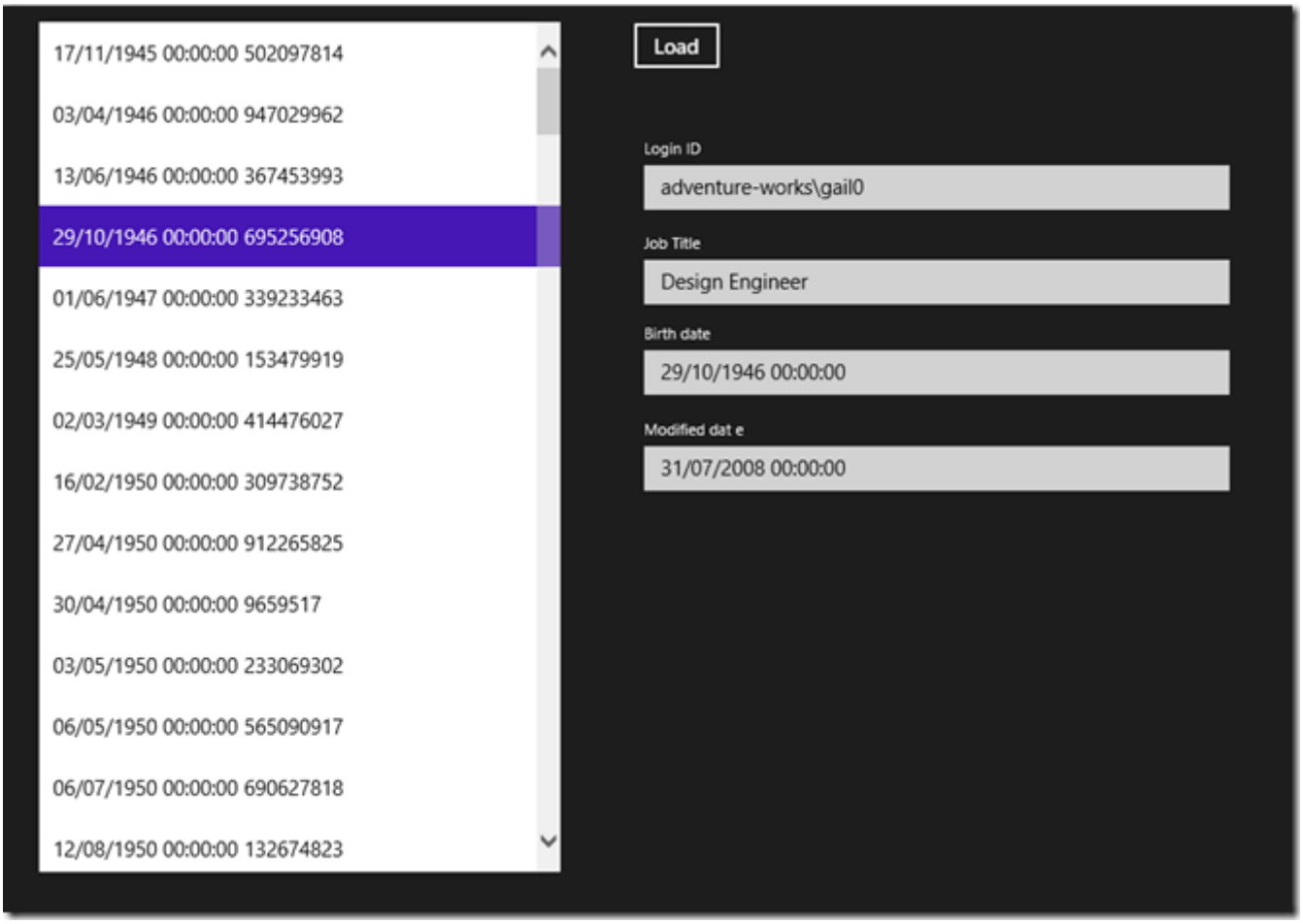
Une fois la requête exécutée, la variable "`data`" est affectée tout simplement à `l'ItemsSource` de la `Listbox`.

L'affichage se produit alors :



Mon DataTemplate de la `ListBox` est minimaliste, je reprends pour chaque ligne uniquement la date de naissance (puisque la requête réclame un tri sur ce champ) et l'ID de l'employé.

Lorsqu'on clique sur un item de la liste, la partie détail à droite se synchronise :



Je n'ai pas exposé les données en mode écriture, même si le `TextBox` le laisse croire, on ne peut donc pas modifier les données. Si on le fait, ces modifications sont purement locales en mémoire. Il suffirait bien entendu d'ajouter un autre bouton avec un appel `SaveChanges` pour rendre l'ensemble de données modifiable (à condition d'ajouter ce droit lors de son exposition par le service).

Cool non ?

Ah oui… Il reste un détail, le code de la fameuse extension qui permet d'utiliser `await :`

```
public static class WcfDataServicesExtensions
    {
        public static async Task<IEnumerable<TResult>>
                                    ExecuteAsync<TResult>
                                                (this DataServiceQuery<TResult> query)
        {
            var queryTask =
                 Task.Factory.FromAsync<IEnumerable<TResult>>
                        (query.BeginExecute(null, null), (asResult) =>
            {
                var result = query.EndExecute(asResult).ToList();
                return result;
            });

            return await queryTask;
        }
    }
```

Ce n'est pas bien compliqué mais ça simplifie l'écriture du code comme nous l'avons vu.

Conclusion

Je ne vous joins pas le projet au billet car il ne comporte rien de spécial en dehors de ce qui est publié ici, le reste n'est que Binding ou plomberie habituelle. De plus l'exemple fonctionne avec une base de données que vous n'aurez peut-être pas envie d'installer, ou pas dans la même version de SQL Server, etc…

Ce qui comptait dans ce billet était de montrer que le tooling pour WinRT se met aussi en ligne sur ce qu'il y avait de mieux dans Silverlight et que dès maintenant il est possible d'intégrer WinRT dans une logique LOB.

Cela est vraiment crucial tellement Windows 8 a été présenté par MS et d'autres, à tord ou à raison, comme un produit grand public.

Il n'en est rien, Windows 8 et WinRT forment une plateforme sérieuse et mature.

Silverlight n'est pas mort, il a juste changé de nom. Toute sa puissance, tant pour les UI que pour l'accès aux données est toujours vivante et peut-être encore plus qu'avant.

Le mode OData est à mon sens un peu lourd et le nouveau framework tout autant. Certes il y a eu des progrès depuis la version que j'ai testée mais je ne suis pas convaincu totalement et les RIA Services sous Silverlight restent pour moi une référence d'efficacité.

Si vous ne le savez pas Microsoft a arrêté les RIA Services. Beaucoup se sont battus pour les voir être libérés en Open Source. C'est chose faite. Le projet Open RIA Services existe. Le fait que Microsoft ait accepté de libérer le produit est à la fois une mauvaise nouvelle (Microsoft lâche tous les bons produits pour des nouveautés qui ne sont pas forcément de la même classe) et une excellente nouvelle (les RIA Services vont survivre aux mauvaises décisions de Microsoft). Il ne reste plus qu'à ce que Microsoft libère aussi Silverlight en Open Source et nous pourrions presque rêver en un monde meilleur… Dans tous les cas étudiez les nouveaux WCF Services en OData, ils apportent quelque chose indéniablement, mais restez à l'écoute de Open RIA Services !

Essayez aussi de reproduire l'exemple de ce billet, vous serez peut-être séduit par les nouveaux services WCF OData.

Pour information :

- L'annonce du 3 juillet 2013 à propos des RIA libérés en Open Source sur le blog de Jeff Handley, Lead developer chez Microsoft sur le projet Nuget Gallery et sur les RIA Services : http://jeffhandley.com/archive/2013/07/03/ria-services-is-getting-open-sourced.aspx
- Le site officiel des nouveaux Open RIA Services : http://www.openriaservices.net
- La page officielle CodePlex des Open RIA Services : http://openriaservices.codeplex.com/

Avertissements

L'ensemble de textes proposés ici est issu du blog « Dot.Blog » écrit par Olivier Dahan et produit par la société E-Naxos.

Les billets ont été collectés en septembre 2013 pour les regrouper par thème et les transformer en document PDF cela pour en rendre ainsi l'accès plus facile. Les mois d'octobre et novembre ont été consacrés à la remise en forme, à la relecture, parfois à des corrections ou ajouts importants comme le livre PDF sur le cross-plateforme par exemple (TOME 5).

Les textes originaux ont été écrits entre 2007 et 2013, six longues années de présence de Dot.Blog sur le Web, lui-même suivant ses illustres prédécesseurs comme le Delphi Stargate qui était dédié au langage Delphi dans les années 90/2000.

Ce recueil peut parfois poser le problème de parler au futur de choses qui appartiennent au passé… Mais l'exactitude technique et l'à propos des informations véhiculées par tous ces billets n'a pas de temps, tant que les technologies évoquées existeront …

Le lecteur excusera ces anachronismes de surface et prendra plaisir j'en suis certain à se concentrer sur le fond.

E-Naxos

E-Naxos est au départ une société éditrice de logiciels fondée par Olivier Dahan en 2001. Héritière de *Object Based System* et de *E.D.I.G.* créées plus tôt (1984 pour cette dernière) elle s'est d'abord consacrée à l'édition de logiciels tels que la suite Hippocrate (gestion de cabinet médical et de cabinet de radiologie) puis d'autres produits comme par exemple MK Query Builder (requêteur visuel SQL).

Peu de temps après sa création E-Naxos s'est orientée vers le Conseil et l'Audit puis s'est ouverte à la Formation et au Développement au forfait. Faisant bénéficier ses clients de sa longue expérience dans la conception de logiciels robustes, de la relation client, de la connaissance des utilisateurs et de l'art, car finalement c'en est un, de concevoir des logiciels à la pointe mais maintenables dans le temps.

C#, Xaml ont été les piliers de cette nouvelle direction et Olivier a été récompensé par Microsoft pour son travail au sein de la communauté des développeurs WPF et Silverlight. Toutefois sa première distinction a été d'être nommé MVP C#. On ne construit pas de beaux logiciels sans bien connaître le langage…

Aujourd'hui E-Naxos continue à proposer ses services de Conseil, Audit, Formation et Développement, toutes ces activités étant centrées autour des outils et langages Microsoft, de WPF à WinRT (Windows Store) en passant par Silverlight et Windows Phone.

A l'écoute du marché et offrant toujours un conseil éclairé à ses client, E-Naxos s'est aussi spécialisée dans le développement Cross-Plateforme, notamment dans le mariage des OS Microsoft avec Android, les deux incontournables du marché d'aujourd'hui et de demain.

N'hésitez pas à faire appel à E-Naxos, la compétence et l'expérience sont des denrées rares !