Développement Windows 8

Créer des applications pour le Windows Store

API Windows Runtime (WinRT) • C#/C++/XAML
.NET • JavaScript/HTML5

Louis-Guillaume Morand
Luc Vo Van
Alain Zanchetta

Préface de Bernard Ourghanlian

ght © 2013 Eyrolles.

EYROLLES

Développement Windows 8

Maîtrisez les nouvelles API de développement Windows 8 pour le Windows Store

Windows 8 ouvre de nouvelles opportunités aux développeurs de tous horizons : usage en mobilité, interfaces intuitives et performantes, accès aux capteurs, et bien d'autres. Que votre expérience s'appuie sur JavaScript/HTML5, XAML, C# ou C++, cet ouvrage met à profit vos connaissances pour créer des applications Windows Store, tout en couvrant les aspects ergonomiques liés à la diversité des matériels et la monétisation via le Store.

Que vous ciblez une plateforme standard (x86) ou bien ARM, au travers de cas pratiques et d'exemples didactiques, vous découvrirez les nombreuses possibilités qui s'offrent à vous en termes d'intégration au sein du système et de fonctionnalités UX.

Au sommaire

Windows 8, un système pour tous types de matériels • Orienté tactile • **Applications Windows Store** • Expérience utilisateur • Distribution et déploiement • Sécurité • Performances et autonomie • **Windows 8 et Windows RT** • **Plate-forme de développement** • Windows Runtime • Développement Windows en HTML5 et JavaScript • .NET et XAML • C++ et XAML • Pour les développeurs d'applications Bureau • **Outillage du développeur** • Windows 8 SDK • Visual Studio 2012 • Blend pour Visual Studio • MSDN • Composants tiers • **Développement d'une première application** • **Ergonomie** • Zoom sémantique • Scénarios d'utilisation • États d'affichage • Maîtriser le tactile • Concepts de navigation • Langage d'interaction tactile • Structure des écrans types • Éléments d'interface • Langage visuel • **Modèle, liaisons et accès aux données** • WCF • Pattern MVVM (Model-View-View-Model) • Liaisons de données en XAML avec .NET • Accès aux données • **Programmation asynchrone** • Interface utilisateur réactive • IAsyncOperation • Async et await • Tâches • Pool de threads • Appels asynchrones en JavaScript • Appels asynchrones en C++ • Bonnes pratiques • **Cycle de vie applicatif** • Persistance des données • **Vignettes et notifications** • Modern UI • Tirer profit des vignettes dynamiques • Windows Push Notification Service • **Utilisation des charmes** • Partage, recherche, paramétrage • Associations applicatives • **Techniques d'animation** • **Animations** • Transitions de thème (XAML) • Transition de contenu • Listes • Animations libres • **Animations indépendantes et dépendantes** • Storyboard • États visuels • Transitions en CSS • **Gestion de l'interface tactile** • Pointeurs • Mouvements et manipulations • GestureRecognizer • **Interfaces avec les capteurs et périphériques** • Géolocalisation • Capteurs • Technologie NFC • Gestion des périphériques • **Windows Store** • Fiche détaillée d'une application Windows 8 • Publier sur le Store • Règles à respecter • Augmenter la visibilité de l'application • Gestion des applications publiées • **Accéder à Internet** • Service REST • JSON • Données applicatives • **Windows Runtime avancé** • Processus WinRT • Accéder au Cloud • Métadonnées • Projections.

L. Morand

Auteur de plusieurs ouvrages autour de Windows, **Louis-Guillaume Morand** a longtemps partagé son savoir, notamment en tant que responsable de publication sur Developpez.com. Consultant au sein de Microsoft, il accompagne les entreprises dans leur utilisation des nouvelles technologies telles que Windows Phone ou Windows 8.

L. Vo Van

Consultant Application Platform/UX chez Microsoft Consulting Services, **Luc Vo Van** accompagne des entreprises dans leurs projets de développement sur les plates-formes .NET et Azure. Il est spécialisé sur les sujets liés à l'expérience utilisateur (UX) et sur les technologies de présentation, ainsi que sur les problématiques d'entreprise qui leur sont liées.

A. Zanchetta

Alain Zanchetta a rejoint en 2012 les équipes de développement de Windows Phone à Redmond où il participe à l'évolution de la plate-forme XAML, après quinze années passées au sein de Microsoft France en tant que consultant, où il a accompagné de nombreux clients lors des évolutions de l'environnement de développement Microsoft de Win32 et COM vers .NET et les technologies de présentation comme WPF ou Silverlight.

À qui s'adresse cet ouvrage ?

- Aux développeurs Windows souhaitant porter leurs applications vers les nouvelles architectures mobiles et souhaitant distribuer leurs applications via le Windows Store ;
- Aux développeurs .Net souhaitant tirer profit de la nouvelle plateforme WinRT ;
- Aux développeurs web qui doivent pour la première fois développer une application native sur Windows.

Code éditeur : G13643
ISBN : 978-2-212-13643-2
Conception : Nord Compo

Copyright © 2013 Eyrolles.

Développement Windows 8

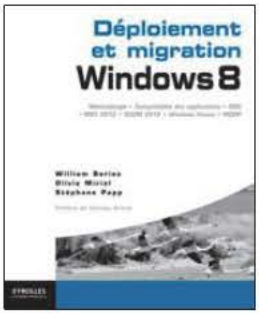
Créer des applications pour le Windows Store

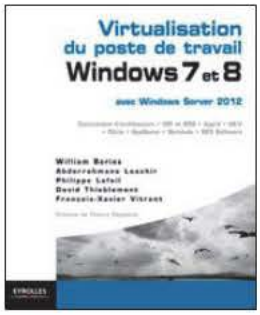
API Windows Runtime (WinRT) • C#/C++/XAML
.NET • JavaScript/HTML5

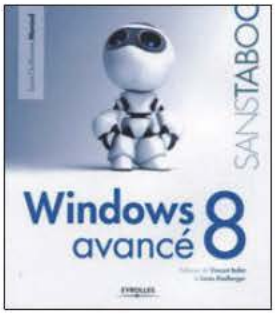
Copyright © 2013 Eyrolles.

Chez le même éditeur

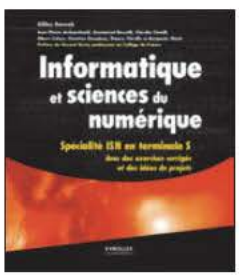
Ouvrages Windows 8

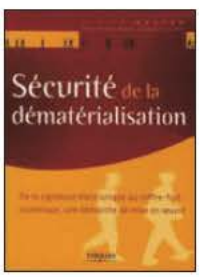


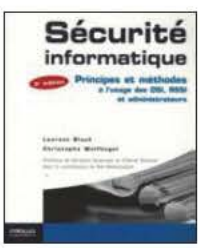


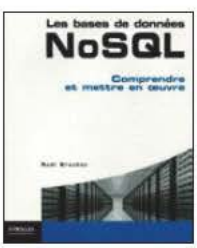


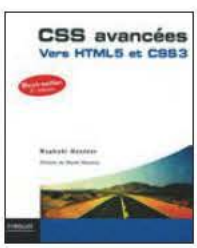
Autres ouvrages

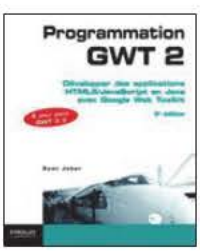












Copyright © 2013 Eyrolles.

Développement Windows 8

Créer des applications pour le Windows Store

API Windows Runtime (WinRT) • C#/C++/XAML
.NET • JavaScript/HTML5

Louis-Guillaume Morand
Luc Vo Van
Alain Zanchetta

Préface de Bernard Ourghanlian
Avec la contribution d'Olivier Dahan

Copyright © 2013 Eyrolles.

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Remerciements à Anne Bougnoux pour sa relecture.

Copyright © 2013 Eyrolles.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.
© Groupe Eyrolles, 2013, ISBN : 978-2-212-13643-2

Préface

8… Un chiffre qui porte chance dans la culture chinoise, car il est homophone du mot « prospérité » en cantonais. Ce n'est pas un hasard si la cérémonie d'ouverture des Jeux olympiques de Pékin a eu lieu le 8 août 2008 à 20 heures et 8 minutes (08/08/08, 08:08) précises…

Ce chiffre 8 portera-t-il chance à Windows et à Microsoft ? Il est trop tôt pour le dire, et pourtant Windows 8 est probablement la version la plus importante depuis Windows 1.0. Qu'il me soit permis de tenter de le démontrer brièvement par une histoire (très abrégée) de ce système d'exploitation.

La première version de Windows a été rendue disponible le 20 novembre 1985. À cette époque, le véritable système d'exploitation était MS-DOS, l'interface graphique étant matérialisée sous la forme d'un *shell* graphique s'exécutant au-dessus du véritable système d'exploitation. Une telle approche avait du sens : elle permettait d'améliorer l'interface homme-machine sans remettre en cause l'essentiel, à savoir la compatibilité avec les applications MS-DOS. Puis est venu Windows NT (appelé Windows NT 3.1, qui est devenu disponible le 27 juillet 1993) dont l'objectif initial était de remplacer OS/2 version 2, et dont l'interface de programmation initiale – d'aucuns l'ont oublié aujourd'hui – était l'API du Presentation Manager d'OS/2.

En raison du succès initial de Windows 3.0, puis de Windows 3.1, Microsoft changea son fusil d'épaule et fit de Windows NT le futur remplaçant des versions Windows 3.1, Windows 95, Windows 98 et Windows Millenium Edition. Pour cela fut créée une nouvelle API, Win32, qui serait compatible avec les noms de fonctions, la sémantique et l'utilisation des types de données de l'API Windows 16 bits existante. En ce sens, Win32 peut être vu comme une extension 32 bits de Win16, visant à faciliter le portage des applications Windows 16 bits existantes vers Windows NT. Cette API fut également rendue disponible sur les différentes versions de systèmes d'exploitation fondées sur MS-DOS – à commencer par Windows 95. Tout cela finit

Copyright © 2013 Eyrolles.

par converger en une seule version, qui devait être initialement Windows 2000, mais qui fut finalement Windows XP qui vit le jour officiellement le 25 octobre 2001. Sont alors sortis successivement Windows Vista et Windows 7, ces versions continuant de s'appuyer largement sur l'API Win32 (Win64 étant la variante de cette API implémentée pour les plateformes 64 bits).

Ce qui est intéressant, quand on considère cette histoire très brièvement résumée, c'est que pendant ces quelque 27 années, jamais Microsoft n'a changé à la fois l'interface homme-machine, l'environnement d'exécution et les API utilisées par les développeurs. Avec Windows 8, c'est ce qui se passe pour la première fois !

En fait, si vous m'autorisez à filer cette métaphore, ce sont les planètes Microsoft qui sont en train de s'aligner… Tout a commencé avec l'interface homme-machine (IHM) et Windows Phone 7 qui a introduit pour la première fois la nouvelle IHM, anciennement appelée METRO et désormais appelée « Microsoft Design Language » ; puis la Xbox lui a emboité le pas et enfin Windows avec Windows 8 qui partagent la même interface homme-machine. Pour le système d'exploitation, c'est l'inverse : Windows Phone 8 a emprunté à Windows 8 ses technologies de base telles que son noyau, son système de fichiers, sa pile réseau, sa sécurité, son support multimédia et son navigateur. Il ne manque désormais plus que le noyau de la Xbox pour que les planètes Microsoft soient complètement alignées mais seul l'avenir pourra nous en dire davantage sur ce dernier sujet… En l'état, sur Windows 8 et Windows Phone 8, bien que les applications ne soient pas identiques, il est relativement trivial de construire des expériences cohérentes : mêmes outils (Visual Studio et Blend), partage d'un ensemble significatif de code entre les projets Windows 8 et Windows Phone 8 : C#/XAML, C++/DirectX, contrôles Web HTML 5… D'où, pour les développeurs, un regain d'intérêt certain pour la plateforme Windows.

Mais revenons-en à Windows 8… Comme on vient de le dire, Windows 8 dispose de la même interface homme–machine que celle que l'on trouve depuis Windows Phone 7 ; celle qui s'instancie dans ce que l'on appelle désormais les « Windows Store Apps ». Ces applications s'exécutent au-dessus d'un nouveau moteur d'exécution appelé Windows Runtime ou WinRT[1] qui offre aux développeurs toute une série d'API permettant le support des communications et des données, du graphique et du multimédia, des terminaux, du stockage et de l'impression, plus un composant d'infrastructure de taille relativement modeste.

Appesantissons-nous un court instant sur WinRT. Pendant toutes ces années, les applications présentes sur le bureau Windows ont été développées en utilisant, plus ou moins directement, les API Win32 (ou Win64), que ce soit pour afficher des gra-

1. Ne pas confondre WinRT, l'environnement d'exécution des applications Windows Store avec WinRT, l'abréviation de Windows RT, version de Windows 8 qui s'exécute sur les processeurs ARM.

Copyright © 2013 Eyrolles.

phiques, dialoguer avec l'utilisateur, communiquer à travers le réseau, etc. Bien entendu, Windows 8 ne renie pas son passé, ne serait-ce que pour assurer une compatibilité transparente des applications Windows 7. WinRT, quant à lui, est constitué d'une collection de nouvelles API permettant la création d'un nouvel univers d'applications immersives s'exécutant en mode plein écran (sans plus de décoration de fenêtres) et appelées Windows Store Apps. Bien entendu, les applications Windows dites « de bureau » sont toujours disponibles et restent toujours pertinentes dans de nombreux cas. C'est notamment le cas des applications Office telles qu'Excel, Word ou PowerPoint. Ces applications peuvent d'ailleurs tirer parti de l'API WinRT, par exemple pour communiquer avec des capteurs.

Les Windows Store Apps ont été conçues pour communiquer avec WinRT à travers une couche dite de « projection » indépendante des langages qui permet à des applications d'être écrites à la fois dans des langages typés de manière statique tels que VB, C++ ou C# mais aussi dans des langages dynamiques et faiblement typés tel que JavaScript.

Comme on le verra dans la suite de cet ouvrage, WinRT repose sur une évolution extrêmement significative de COM à travers l'ajout d'une seconde interface à l'inusable `IUnknown`, appelée `IInspectable`. Mais ce nouveau COM facilite le partage des données entre différents langages ; il permet même l'utilisation de métadonnées, comme en environnement .NET. Ainsi, les métadonnées WinMD constituent l'un des éléments fondateurs du support multi-langage de WinRT ; c'est un élément commun à tous les langages qu'utilise d'ailleurs l'infrastructure de WinRT pour générer les projections dont nous parlions plus haut.

Windows 8 apporte donc, de manière évidente, tout un ensemble de nouveautés : nouvelle interface homme-machine, nouvel environnement d'exécution, nouvelles API. Un tel renouveau n'a jamais eu de précédent dans l'histoire de Microsoft. Windows 7, à titre de comparaison, est construit au-dessus du même environnement d'exécution « Explorer » que toutes les versions de Windows avant lui depuis Windows 2000 et offre au développeur les mêmes API Win32 (avec quelques nouvelles possibilités, évidemment). Bien entendu, WinRT reste, en interne, un client de Win32, comme n'importe quelle autre application ; mais ceci est rendu complètement transparent pour le développeur, quel que soit son langage.

En ce sens, Windows 8 constitue certainement un pari important pour Microsoft. Un véritable changement de paradigme… Mais pourquoi un tel pari ? Tout simplement parce que, s'il est vrai que Windows a donné vie aux ordinateurs personnels qui sont aujourd'hui utilisés par plus d'un milliard de personnes sur terre, Windows est resté un système d'exploitation assez proche des professionnels de l'informatique et des entreprises. Ainsi, l'utilisation de la plupart des fonctions du système nécessitaient – jusqu'à Windows 8 – un certain niveau de familiarité avec des concepts tels que les fichiers, les répertoires, les permissions, les partages, etc.

Copyright © 2013 Eyrolles.

Pourtant la démarche d'Apple centrée sur le consommateur à travers des produits tels que l'iPhone et l'iPad a démontré au monde qu'une autre approche était possible, à travers une interaction intuitive avec l'ordinateur, sans nécessiter le moins du monde la connaissance préalable du concept de fichier, de répertoire ou de procédure d'installation d'application. Windows 8 constitue à n'en pas douter la réponse de Microsoft à cette nouvelle approche, reprenant à son compte la « consumérisation de l'informatique », ce phénomène sociologique irrépressible qui procède d'une démocratisation sans cesse plus grande du monde numérique qui nous entoure.

Windows 8 est donc une aventure. Aventure intellectuelle tout d'abord, car il a commencé comme dans l'un de ces cercles occultes où, tout d'abord, seuls quelques initiés sont admis à partager la connaissance. Puis, cette connaissance s'est diffusée à un cénacle un peu plus large où seuls les femmes et les hommes dont la compétence était reconnue par leurs pairs étaient admis. Enfin, le temps est venu de la dissémination beaucoup plus large de ce savoir, par laquelle on cherchait à convaincre et à « évangéliser ». Aventure industrielle aussi dans laquelle Microsoft a investi des milliards et jeté toutes ses forces pour donner vie à ce qui n'était, au moins au début, qu'un rêve dans l'esprit de quelques ingénieurs.

Aujourd'hui, cette aventure a pris forme, avec un système d'exploitation, Windows 8, disponible sur de nombreux environnements matériels très divers : PC de bureau traditionnel, portable, tablette, *ultrabook*, hybride, renforcé, etc.

Nul ne sait quelle sera la fin de cette aventure. De nombreuses pages restent encore à écrire. Mais, comme dans le merveilleux roman de Michael Ende, *L'Histoire sans fin*, le lecteur a sa part de l'histoire à écrire…

« Ceux qui comprennent ne comprennent pas qu'on ne comprenne pas » nous disait Paul Valéry. Ce n'est pas le cas de ceux qui ont choisi d'écrire cet ouvrage. Ils ont chaussé leurs bottes de pédagogues et mis tout leur talent pour mettre à la portée du plus grand nombre des concepts parfois opaques.

Car Windows 8 (et Windows Phone 8) est ouvert à tous les styles de développements et à tous les langages, quels que soient leurs types : procéduraux, orientés objets, fonctionnels, de script, typés fortement ou non, de manière dynamique ou statique… et cet ouvrage se devait de respecter toute cette diversité en étant accessible à tous.

Développeurs, empressez-vous de tourner la première page de cet ouvrage, vous êtes ici chez vous !

Bernard Ourghanlian,
CTO de Microsoft France

Copyright © 2013 Eyrolles.

Table des matières

Avant-propos **1**
Pourquoi ce livre ? 1
À qui s'adresse ce livre ? 2
Structure de cet ouvrage 2
Remerciements 4
Remerciements de Louis-Guillaume Morand 4
Remerciements de Luc Vo Van 4
Remerciements d'Alain Zanchetta 4

CHAPITRE 1
Applications Windows Store **5**
Qu'est-ce qu'une application Windows Store ? 6
Une expérience utilisateur plus fluide 6
Un cycle de vie plus contrôlé 6
Distribution et déploiement 8
Distribution via le Windows Store 8
Déploiement interne en entreprise 8
Déverrouillage pour les développeurs 9
Sécurité 9
Performances et autonomie 10
Windows 8 et Windows RT 10
Windows 8 sur processeurs x86 et x64 10
Windows RT pour les processeurs ARM 11
Une plate-forme de développement 11
Des API accessibles avec le Windows Runtime (WinRT) 12
HTML5 et JavaScript 13
.NET et XAML (eXtensible Application Markup Language) 15
C++ et XAML 16
Outillage du développeur 18
Le SDK Windows 8 18
Visual Studio 2012 19

Copyright © 2013 Eyrolles.

Blend pour Visual Studio 19
MSDN, la référence documentaire 19
Composants tiers 20
Une première application 20
XAML 20
JavaScript et HTML 22

CHAPITRE 2
Éléments d'ergonomie 25
Une philosophie de design 26
Scénarios d'utilisation 30
États d'affichage 31
Plein écran 31
Ancré 32
Remplissage 32
Concepts ergonomiques de navigation 33
Navigation hiérarchique 34
Navigation linéaire 34
Navigation hybride 35
Langage d'interaction tactile 35
Mouvements 35
Manipulations 36
Structure des écrans types 37
Pages 37
Panoramiques 39
Éléments d'interface 40
Vignettes dynamiques 40
Notifications 42
Icônes et contrats 44
Barres d'applications 44
Zoom sémantique 46
Langage visuel 47
Absence de contours 47
Polices 47
Alignement sur une grille 48
Authentiquement numérique 48
Branding et identité visuelle 48
Style visuel et ergonomie 49

Copyright © 2013 Eyrolles.

CHAPITRE 3
Modèle, liaisons et accès aux données 51
Principes généraux 51
Le pattern MVVM (Model-View-ViewModel) 54
Liaison de données en XAML avec .NET 55
Liaison simple 55
Liaison de listes 61
Liaison de données en C++ 64
Liaison de données en HTML5 et JavaScript 68
Accès aux données 73
Lecture et écriture dans un fichier 74
Principes de base 74
Utilisation de fichiers textes 76
Accès simplifié : la classe FileIO 78
Utilisation d'un fichier XML 78
Accès à des données distantes 81
Service WCF simple 81
Utilisation d'une bibliothèque d'interface 82
Utilisation de WCF Data Services 86

CHAPITRE 4
Programmation asynchrone 89
Asynchronisme et interface utilisateur réactive 89
IAsyncOperation 93
Async et await 96
Tâches 99
Pool de threads 101
Appels asynchrones en JavaScript 102
Appels asynchrones en C++ 103
Bonnes pratiques 104

CHAPITRE 5
Cycle de vie des applications 105
Le principe de cycle de vie 106
Le principe de suspension 109
Responsabilités d'une application 109
Sauvegarde et restauration d'état 109
Sauvegarde de l'état applicatif en C# 111
Sauvegarde de l'état applicatif en JavaScript 114
Sauvegarde de l'état applicatif en C++ 115

Copyright © 2013 Eyrolles.

Reprise des données 116
La reprise en C# 116
La reprise en JavaScript 117
La reprise en C++ 117
L'activation 117
L'activation en C# 118
L'activation en JavaScript 119
L'activation en C++ 120
Conclusion 121

CHAPITRE 6
Les vignettes et notifications 123
Vignettes dynamiques 123
Organisation d'une vignette 125
Mise à jour d'une vignette 128
Mise à jour du badge 129
Le respect des bonnes pratiques 130
Vignettes secondaires 130
Création d'une vignette secondaire 131
Transmission de contexte 132
Bonnes pratiques 133
Utilisation du Windows Push Notification Service 133
Conclusion 137

CHAPITRE 7
Tirer profit de la puissance des charmes 139
La notion de contrat 139
Le charme de partage 140
Source de partage 142
Partager autre chose que du texte 144
Déclenchement du partage 146
Cible de partage 147
Le charme de recherche 150
La suggestion de résultat 153
Les templates de contrat de recherche 155
Le charme de paramétrage 156
Personnalisation de l'écran Paramètres 157
Affichage d'un écran personnalisé 158
Les associations applicatives 160
L'association de fichier 160

Copyright © 2013 Eyrolles.

Chargement du fichier 162
L'association de protocole 164
Utilisation du protocole. 165
Les API de lancement 166
Ouverture de l'application, par défaut ou non, d'un fichier 167
Ouverture de l'application par défaut d'un protocole 169
Conclusion 170

CHAPITRE 8
Techniques d'animation 171
Rôle et importance des animations 172
Guider l'utilisateur 172
Renforcer la présentation 172
Animations types 173
Transitions de thème (XAML) 173
Transition de contenu 175
Implémentation XAML 176
Implémentation JavaScript 176
Listes 178
Implémentation XAML 178
Implémentation JavaScript 178
Animations libres 180
Animations indépendantes et dépendantes 181
Animations indépendantes 181
XAML 181
HTML/CSS 181
Animations dépendantes 182
XAML 182
HTML/CSS 182
Storyboard (XAML) 182
Définition 183
Utilisation 183
États visuels (XAML) 184
Définition des états visuels 185
Appliquer un état visuel 187
Édition des états visuels avec Blend 187
Transitions CSS 190
Événement DOM transitionend 191
Utilisation conjointe avec l'extension ms-view-state 191

Copyright © 2013 Eyrolles.

CHAPITRE 9
Gestion de l'interface tactile 193
Interactions utilisateur sous Windows 193
Considérations pour les interfaces tactiles 194
Les pointeurs 195
Détection des capacités tactiles 197
Mouvements et manipulations 198
GestureRecognizer 198
Utilisation du GestureRecognizer (JavaScript) 198
Événements de manipulation (XAML) 200
Conclusion 203

CHAPITRE 10
Interfaces avec les capteurs et périphériques 205
La géolocalisation 205
Accès ponctuel 207
Accès continu 209
Bonnes pratiques de la géolocalisation 210
Les capteurs 211
Utilisation de la technologie NFC 213
Envoyer ou recevoir des informations via NFC 215
Utiliser NFC sans NFC 216
La gestion des périphériques 217
Lister les périphériques disponibles 217
Interagir avec les périphériques 218
Conclusion 222

CHAPITRE 11
Le Windows Store 223
Le Windows Store 223
Une découverte facilitée 224
Publier son application sur le Store 226
Licence développeur 226
Procédure d'enregistrement et de validation 227
Manifeste et capacités 227
Soumission de l'application 229
Les principales règles à respecter 232
Augmenter la visibilité de votre application 233
Noter l'application 233
Faire télécharger l'application depuis un site web 234
Gestion des applications publiées 235

Copyright © 2013 Eyrolles.

CHAPITRE 12
Accéder à Internet 237
Données applicatives 237
Données itinérantes d'une application 238
Sécurité 240
Accès à des services Internet personnalisés 241
Utilisation de sockets 242
Service HTTP « brut » 246
Service REST et sérialisation JSON 249
Service WCF 251
Accéder à des services standards - SkyDrive 251
Développer plus vite grâce aux services mobiles Windows Azure 254
Création d'un service mobile 255
Accès aux données stockées sur Azure 256

CHAPITRE 13
Windows Runtime avancé 259
Processus WinRT 259
Composants WinRT 261
Évolution des composants logiciels 261
Anatomie d'un composant WinRT 264
Classe et interfaces 264
Modèle de threading 266
Métadonnées 267
Activation et fabriques 270
Système de types WinRT et projections 273

Conclusion 279

Index 281

Copyright © 2013 Eyrolles.

Avant-propos

L'interface de Windows 8, au premier abord déroutante, cache une révolution : celle des interfaces de programmation (API). Si les technologies de développement Windows classiques restent disponibles dans l'environnement du bureau, de nombreuses API apparaissent. La plus importante, WinRT, permet de développer pour le nouveau Windows Store.

Pourquoi ce livre ?

Le nouveau visage de Windows, c'est WinRT, une nouvelle API très riche conçue pour le développement d'applications vendues via le nouveau Windows Store. WinRT touche à tous les aspects du développement applicatif, de la présentation graphique au dialogue avec les matériels, notamment les nombreux capteurs conçus pour Windows 8 (accéléromètre, NFC…), en passant par les fondamentaux tel que les accès réseau et la lecture ou écriture de fichiers.

Afin de garantir sécurité et qualité, les applications écrites en respectant les nouveaux standards sont publiées via le Windows Store, la plate-forme en ligne de distribution Microsoft ; elles doivent répondre à un cahier des charges technique très strict vérifié par Microsoft lors du processus de publication.

Ce livre vous guide à travers les principaux concepts de programmation de cette nouvelle génération d'applications, et décrit leur mise en œuvre à travers de nombreux exemples de code. Il distille également des bonnes pratiques, allant de certains principes fondamentaux de programmation, au respect des normes conditionnant la publication dans le Windows Store, en passant par des recommandations en vue de favoriser l'adoption des applications par leurs utilisateurs – but ultime du développement applicatif orienté vers le grand public.

Copyright © 2013 Eyrolles.

À qui s'adresse ce livre ?

Écrit par des développeurs, ce livre s'adresse aux programmeurs qui connaissent déjà certaines technologies de développement Windows ou web (.NET, C++, HTML5, JavaScript…) mais qui peuvent se sentir démunis devant la nouvelle interface de programmation. Pourtant, quelle que soit l'ampleur de l'évolution des API, Microsoft a toujours privilégié la capitalisation des connaissances, plutôt que de forcer le développeur à abandonner tout ce qu'il sait et tout ce qu'il a écrit. Ainsi les applications du Windows Store peuvent-elles être programmées avec les langages de .NET tels C# et Visual Basic .NET, comme en JavaScript et HTML5, voire en C++.

Ce livre s'inscrit dans cette approche. Il peut être lu par tout programmeur maîtrisant l'une de ces technologies, grâce au socle commun que constitue WinRT.

Les exemples de code sont ainsi donnés tant en C# ou C++ qu'en JavaScript. Outre qu'elle permet de s'adresser à un public élargi, cette triple approche permet au lecteur, soucieux de s'informer pour faire les bons choix, de faire la part des choses entre le concept lui-même et les éventuelles difficultés de syntaxe liées à tel ou tel langage. D'ailleurs, pourquoi n'en profiterait-il pas pour découvrir d'autres technologies ou langages lui permettant de mieux réaliser ses idées ?

Structure de cet ouvrage

Chaque chapitre traite un pan de la programmation d'applications pour le Windows Store :

- Le **premier chapitre** présente les principes fondamentaux des applications Windows Store, ce qui les rassemble et ce qui les différencie des applications Windows classiques.
- Le **chapitre 2** aborde l'ergonomie des applications – les utilisateurs étant de plus en plus exigeants. Une application moderne se doit d'offrir un excellente « expérience utilisateur », d'autant plus que l'ergonomie fait partie des critères de validation avant publication.
- Le **chapitre 3** décrit comment modéliser les données utilisées par une application, comment les afficher et les mettre à jour de manière automatique et comment les rendre persistantes.
- Le **chapitre 4** aborde une technique de développement omniprésente depuis la sortie de Windows 8 : la programmation dite asynchrone, qui garantit à l'utilisateur une interface toujours réactive – mais au prix d'un changement de paradigme pour le développeur.
- Le **chapitre 5** détaille le cycle de vie des applications qui a vu son modèle modifié pour que le système puisse proposer une expérience utilisateur optimale basée sur la fluidité et l'accessibilité sur les unités mobiles (tablettes et smartphones, principalement).

Copyright © 2013 Eyrolles.

- Le **chapitre 6** décrit comment les applications s'intègrent avec l'écran d'accueil et ses vignettes dynamiques, en abandonnant la métaphore du bureau si longtemps présente sous Windows.
- Le **chapitre 7** présente les nouveaux moyens standardisés de communication et d'échange entre applications comme le partage de données ou la recherche d'information.
- Le **chapitre 8** explique les techniques d'animation qui accompagnent l'utilisateur dans sa découverte de l'environnement d'une application.
- Le **chapitre 9** décrit un aspect essentiel de la nouvelle interface de Windows : la gestion du tactile.
- Le **chapitre 10** présente les moyens d'interagir avec le matériel et notamment avec les nombreux capteurs qui équipent les tablettes, les smartphones et autres matériels modernes.
- Le **chapitre 11** décrit la procédure de soumission d'une application au Windows Store et les bonnes pratiques associées pour aider le développeur dans cette nouvelle phase assez technique.
- Le **chapitre 12** décrit les moyens d'accéder au réseau Internet, omniprésent dans notre vie et souvent crucial pour le fonctionnement des applications modernes.
- Le **chapitre 13** propose au développeur expérimenté un voyage au centre des mécanismes faisait vivre les applications du Windows Store pour compléter ses connaissances et mieux en maîtriser les rouages sous-jacents.

L'histoire d'une aventure de 13 passionnés !

Nous ne pouvions pas vous laisser démarrer la lecture de cet ouvrage sans vous expliquer comment ce projet a démarré. Ce livre fait partie d'un projet de rédaction de quatre ouvrages sur la thématique du poste de travail, essentiellement sur Windows 8. Ce projet a réellement démarré au début de l'année 2012. Les 13 auteurs de ces ouvrages sont des passionnés et experts reconnus dans leurs domaines respectifs.

- 🕮 *Sécurité et mobilité du poste de travail Windows 8,* Arnaud Jumelet, Stanislas Quastana et Pascal Sauliere, Eyrolles 2013
- 🕮 *Développement Windows 8,* Louis-Guillaume Morand, Luc Vo Van et Alain Zanchetta, Eyrolles 2013
- 🕮 *Virtualisation du poste de travail Windows 7 et 8,* William Bories, Abderrahmane Laachir, Philippe Lafeil, David Thieblemont et François-Xavier Vitrant, Eyrolles 2013
- 🕮 *Déploiement et migration Windows 8,* William Bories, Olivia Mirial et Stéphane Papp, Eyrolles 2013

D'autres passionnés chez Microsoft, pour certains des experts reconnus, ont apporté une précieuse contribution à ces ouvrages par une relecture profonde et pragmatique, au cours de laquelle ils ont pu mettre les auteurs à l'épreuve. Enfin, ceux qui n'avaient pas d'expertise technique particulière n'ont pas hésité à relire des dizaines de pages pour améliorer la qualité de ces ouvrages. Bref, c'est un projet atypique auquel nous avons tous pris beaucoup de plaisir à participer !
En espérant que vous partagerez ce plaisir, je vous souhaite une excellente lecture !

William Bories,
Coordinateur du projet.

Copyright © 2013 Eyrolles.

Remerciements

Nous tenons en premier lieu à remercier les éditions Eyrolles qui nous ont soutenus pour ce projet ainsi que notre G.O William dont l'enthousiasme est sans égal. Merci également à Olivier Dahan pour sa contribution. Un grand merci aussi à Bernard Ourghanlian pour cette belle préface et tout l'éclairage qu'il nous apporte en permanence sur les nouvelles technologies et les choix effectués par Microsoft.

Merci aussi à tous nos relecteurs sans lesquels cet ouvrage n'aurait pas la même qualité : Myriam Delesalle avant tout, mais aussi Christophe Nasarre-Soulier, Patrice Manac'h et bien sûr Michel Molongo, Audrey Petit, Nicolas Antoine, Sébastien Bovo, David Coppet, Isabelle Leboucher et André Zambalas, tous volontaires pour relire sans relâche notre prose pendant leur temps libre afin de donner à ce livre la qualité qu'il possède aujourd'hui.

Remerciements de Louis-Guillaume Morand

Ce livre, mon troisième chez Eyrolles, fut pour moi le plus plaisant à écrire du fait de l'avoir réalisé avec Alain et Luc, deux grosses pointures de qui j'ai toujours beaucoup appris. C'est donc eux que je tiens à remercier en premier. Puis vient Sophie ma moitié qui a gentiment accepté de me voir passer mes soirées et nuits sur l'ordinateur, à écrire ; enfin, je remercie tous les copains dont la présence permet de s'aérer la tête et de profiter de la vie.

Remerciements de Luc Vo Van

Je suis, depuis très jeune, un fervent lecteur de livres liés à l'informatique. Être auteur a toujours été l'un de mes rêves et c'est donc avec enthousiasme que j'ai rejoint Louis-Guillaume et Alain dans la rédaction de cet ouvrage. Je tenais à remercier tout particulièrement ma femme Yseult et ma fille Clémentine pour leur patience ; Clémentine ayant su, de concert avec la rédaction de cet ouvrage, subtiliser chaque instant de sommeil potentiel.

Remerciements d'Alain Zanchetta

Je tiens à remercier ma femme et mes enfants pour leur patience et leur soutien : en plus de ce livre qui m'a occupé un certain nombre de soirées, je leur ai fait subir un déménagement de 8 000 km et de 360 cartons… 2012 aura décidément été une année bien remplie. Je remercie bien évidemment aussi mes deux compagnons de route pour cette expérience très enrichissante.

Copyright © 2013 Eyrolles.

1

Applications Windows Store

Ces dernières années ont vu naître de nombreuses innovations technologiques. Accessibles au plus grand nombre, ces dernières ont entraîné l'émergence de nouveaux usages. Il est désormais possible de se servir de matériel léger, autonome et connecté pour créer librement l'information et y accéder, où que l'on soit. De nouvelles façons d'interagir avec les applications sont maintenant largement répandues : interface tactile, géolocalisation, connectivité permanente et services en ligne ouvrent la voie à de nouveaux scénarios d'utilisation. Windows 8 a été ré-imaginé en intégrant ces évolutions dans son cœur, donnant naissance à un nouveau type d'applications : les applications Windows Store.

À l'heure de la rédaction de cet ouvrage, Windows 8 représente la toute dernière génération de Windows. Plus qu'une simple évolution du système d'exploitation de Microsoft, Windows a été ré-imaginé afin de fournir aux utilisateurs de meilleures performances, plus de flexibilité et une productivité améliorée.

Les applications jouent un rôle majeur dans cette démarche : alors que le système d'exploitation fournit des services transverses et la gestion des composants bas-niveau de la machine, c'est avec les applications que l'utilisateur réalise les tâches qui sont pour lui les plus importantes. La qualité des applications et la synergie entre ces dernières forment un tout avec le système d'exploitation et en définissent ainsi la richesse et la puissance.

Copyright © 2013 Eyrolles.

Qu'est-ce qu'une application Windows Store ?

Bien plus qu'une évolution de la plate-forme de développement, les applications Windows Store sont un nouveau type d'applications Windows à part entière. Cette section décrit quelques-unes des différences les distinguant des applications « classiques », dites Bureau, utilisées jusqu'alors.

Une expérience utilisateur plus fluide

Windows a changé du point de vue de « l'expérience utilisateur » (on parle le plus souvent d'UX pour *User eXperience*), avec son interface utilisateur baptisée *Modern UI* et ses applications dédiées, les applications Windows Store. Les interfaces utilisateur de ce nouveau type d'applications se caractérisent par leur fluidité, par la mise en avant des contenus utiles, différents modes d'affichage, une grande lisibilité due à l'utilisation très limitée du chrome (contours), ou encore une exploitation prononcée de la typographie. Pour être cohérentes avec le système, les applications Windows Store se doivent d'être, elles aussi, rapides, fluides et modernes.

> Le chapitre 2, consacré à l'ergonomie, présente de manière non technique les principaux aspects liés à l'expérience utilisateur des applications Windows Store.

Un cycle de vie plus contrôlé

Les applications Windows traditionnelles sont destinées à être exécutées sur le Bureau. Il s'agit de programmes généralement implémentés avec des technologies telles que MFC, Windows Forms ou encore WPF. Leur cycle de vie est relativement simple :

Figure 1–1
Cycle de vie d'une application Windows classique

Les applications Bureau sont globalement libres de gérer leur propre cycle de vie et s'exécutent tant qu'elles ne sont pas explicitement fermées. Le système d'exploitation leur alloue les ressources nécessaires en fonction de leurs besoins et, sauf cas exceptionnels, n'a pas vocation à déterminer si une application doit être suspendue ou arrêtée.

Copyright © 2013 Eyrolles.

Alors que le modèle lié aux applications Bureau a été conçu pour une utilisation en continu, les applications Windows Store ciblent des usages où l'interactivité et la mobilité ont un rôle prépondérant. Ces deux facteurs ont contribué à la conception d'un cycle de vie applicatif très différent de celui utilisé jusque-là, qui donne au système d'exploitation la possibilité de contrôler de manière plus fine le cycle de vie des applications (voir figure 1-2).

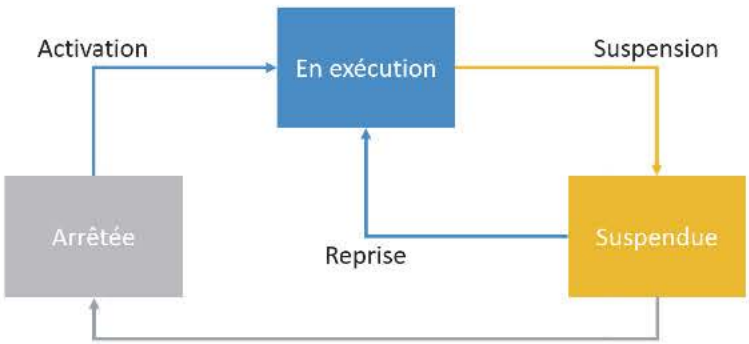


Figure 1–2
Cycle de vie d'une application Windows Store

Un nouvel état, « suspendu », a donc été ajouté au cycle de vie classique. Dans cet état, généralement atteint lorsque l'application n'est plus affichée à l'écran, l'application réside toujours en mémoire mais son code ne s'exécute plus. Le système d'exploitation peut également décider d'arrêter (fermer) une application si les ressources système viennent à manquer, et ce sans intervention de l'utilisateur. L'application est avertie par le système d'exploitation lorsque son état change et dispose alors d'un temps limité pour, par exemple, sauvegarder ses données applicatives.

Figure 1–3
Applications en mode remplissage (à gauche) et ancré (à droite)

Copyright © 2013 Eyrolles.

Les applications affichées à l'écran, deux au maximum, se partagent la quasi-totalité des ressources système disponibles, tandis que les autres applications Windows Store non affichées sont suspendues, ne sollicitent pas de ressource processeur et peuvent être supprimées de la mémoire vive si le système d'exploitation en voit la nécessité. Ce modèle met en avant les principes de conception liés à la fluidité et à la mobilité : les ressources sont dédiées aux applications affichées à l'écran afin de leur assurer une interactivité et une fluidité maximales, tout en optimisant l'utilisation de la batterie. Plus encore que pour les applications Bureau, le système d'exploitation joue un rôle central dans la gestion du cycle de vie des applications Windows Store, car il impose aux applications de passer d'un état du cycle à un autre.

Les deux types d'applications proposées par Windows, Bureau et Windows Store, diffèrent donc grandement dans la manière dont sont gérés leurs cycles de vie respectifs. Ces différences ne sont pas que techniques : elles influent également sur l'utilisation qui pourra être faite des applications. Certains usages pourraient nécessiter une exécution en tâche de fond et auraient alors plus de sens comme applications Bureau, tandis que d'autres, nécessitant des spécificités réservées aux applications Windows Store (gestion des modes d'affichage, mode de distribution) et une meilleure autonomie de la batterie, pourraient être implémentés en tant qu'applications Windows Store.

Distribution et déploiement

Le mode de distribution des applications Windows Store diffère considérablement du mode de distribution classique : en accord avec l'évolution des usages, et mis à part le cas particulier du poste développeur, leur déploiement se fait de manière centralisée, que ce soit pour le grand public ou en entreprise. Il existe trois manières de déployer une application Windows Store, décrites ci-après.

Distribution via le Windows Store

Sans aucun doute la façon la plus répandue de déployer une application, la distribution via le Windows Store de Microsoft sert à publier et à monnayer un programme à grande échelle. Le chapitre 11 décrit en détail le processus de soumission d'une application, ainsi que les nombreuses possibilités offertes par ce mode de distribution.

Déploiement interne en entreprise

Le Windows Store public n'est pas le seul mode de publication. En effet, une entreprise peut déployer une application en interne au sein de son parc informatique, sans avoir à passer par le Windows Store public. De nombreuses applications métier n'ont en effet pas vocation à être distribuées publiquement. Autre aspect important : les déploiements internes sont intégralement sous le contrôle de l'entreprise. Ce n'est

Copyright © 2013 Eyrolles.

pas le cas des applications distribuées sur le Windows Store public, qui doivent d'abord être validées par Microsoft avant d'être publiées.

Déverrouillage pour les développeurs

Le développeur d'applications a bien entendu besoin de déployer localement l'application qu'il développe. C'est le déverrouillage qui lui permet de le faire, moyennant l'ouverture d'un compte développeur Microsoft et le déploiement des certificats adéquats sur son poste. Le chapitre 11 décrit les modalités d'obtention d'un compte développeur Windows Store.

Sécurité

Au fil des années, la technologie prend une place grandissante dans notre quotidien. En conséquence, les problématiques liées à la sécurité et au respect de la vie privée sont devenues une priorité de premier ordre dans le développement logiciel. Ces considérations sont prises en compte dans Windows en donnant aux utilisateurs la possibilité de connaître et de gérer les capacités des logiciels dédiés à la nouvelle plate-forme. Le développeur de l'application se doit de déclarer les besoins de son application (accès aux contacts, géolocalisation, Internet et bien d'autres) lors de la publication du logiciel sur le Windows Store. Afin de valider la mise à disposition de l'application sur le Store, Microsoft vérifie que les fonctionnalités (API) utilisées sont bien celles déclarées par le développeur. Ces fonctionnalités sont ensuite présentées à l'utilisateur sur la page dédiée à l'application au sein du Windows Store. Lors du premier lancement de l'application, l'utilisateur devra donner explicitement son consentement sur l'utilisation de ces dernières. Les différentes fonctionnalités ont été identifiées en amont par Microsoft, sur des critères généralement liés aux aspects financiers (coûts de connexion réseau), à la protection de la vie privée et de la machine de l'utilisateur.

Il est demandé aux développeurs de prévoir et gérer dans leur code le cas où l'utilisateur refuse l'utilisation de certaines capacités. Une application peut par exemple continuer à fonctionner même si l'utilisateur refuse la géolocalisation.

La déclaration des besoins de l'application, quant à elle, se fait par le biais d'un fichier XML, dit `manifest`, que le développeur peut éditer manuellement ou par le biais d'une interface graphique dans Visual Studio.

> Le chapitre 11 décrit en détail la déclaration de ces fonctionnalités.

Copyright © 2013 Eyrolles.

Performances et autonomie

Les ambitions en termes d'expérience utilisateur et les contraintes de la mobilité doivent s'allier de manière élégante.

Comme mentionné dans la section traitant du cycle de vie applicatif, au maximum deux applications Windows Store peuvent s'exécuter simultanément. Cette contrainte permet au système d'exploitation à la fois de fournir une expérience immersive et de concentrer les ressources machines sur les applications que l'utilisateur est réellement en train d'utiliser. L'effet immédiat est une fluidité et un confort d'utilisation optimal, ne consommant les ressources que lorsque les applications présentes à l'écran en ont besoin.

Les applications Windows Store savent bien entendu exécuter des processus en tâches de fond. Cependant, ces dernières sont strictement supervisées par le système d'exploitation afin d'optimiser l'usage des ressources, prolongeant l'autonomie du système. Windows alloue un temps d'exécution et un quota de ressources, notamment CPU ou réseau, pour les exécuter : celles-ci peuvent à un moment donné être regroupées afin que le processeur reste en veille sur de plus longues périodes, et pour rassembler l'exécution des tâches quand celui-ci est réactivé.

Le rendu visuel et la fluidité sont des éléments essentiels de toute expérience utilisateur de qualité. Toutes les applications Windows Store, quelle que soit la technologie choisie pour leur implémentation, bénéficient de l'accélération graphique matérielle. Les processeurs graphiques spécialisés sont en charge de l'affichage, tandis que le processeur principal traite en parallèle des données non graphiques. La collaboration de ces deux éléments matériels est transparente pour le développeur et conduit à des applications performantes et généralement moins consommatrices en énergie.

Windows 8 et Windows RT

La sortie de la dernière version de Windows arrive avec une nouveauté majeure : une toute nouvelle déclinaison du système d'exploitation nommée Windows RT.

Windows 8 sur processeurs x86 et x64

Succédant au très populaire Windows 7, la version 8 du système cible les processeurs x86 et x64. Elle en assure la compatibilité ascendante, tant pour le parc logiciel que matériel, et met à disposition des fonctionnalités comme l'intégration dans des domaines d'entreprises ou la virtualisation. Windows 8 est disponible en trois éditions : standard, Professionnelle ou Entreprise.

Copyright © 2013 Eyrolles.

Windows RT pour les processeurs ARM

Cette version de Windows est conçue pour fonctionner sur processeurs ARM, à ce jour plus économes en énergie et plus abordables que les processeurs basés sur les architectures x86 ou x64. Pour l'utilisateur classique, l'usage de Windows RT est très similaire à celui de Windows 8 : même interface graphique et même ergonomie. Une exception notable existe cependant entre les deux versions : les applications Bureau tierces ne peuvent s'exécuter sous Windows RT. En effet, ces dernières sont à ce jour toutes compilées pour les architectures x86/x64, les rendant incompatibles avec les processeurs ARM. Certaines applications Bureau très spécifiques et fournies par Microsoft existent cependant et sont livrées directement avec Windows RT, comme l'Explorateur de fichiers, Internet Explorer ou Office 2013. Il est à noter que, contrairement à Windows 8, Windows RT ne peut être acheté séparément et sera toujours livré préinstallé sur un appareil.

Les applications Windows Store sont les seules qui soient compatibles avec les deux versions du système d'exploitation et elles sont compilées par défaut pour les trois architectures x86, x64 et ARM. L'audience des applications Windows Store est donc très large car elle regroupe tous les utilisateurs de Windows 8 et Windows RT.

> **ATTENTION Ne pas confondre Windows RT et WinRT**
>
> Afin d'éviter toute confusion, il est important de noter que Windows RT est la version du système d'exploitation dédiée aux architectures ARM, tandis que l'abréviation WinRT représente le Windows Runtime, qui est la plate-forme technique sur laquelle reposent les applications Windows Store. Le Windows Runtime est détaillé au chapitre 13.

Une plate-forme de développement

Windows est accompagné historiquement d'une plate-forme de développement riche et variée. Le développeur d'applications dispose d'un grand choix d'outils de qualité, quel que soit l'archétype de ses programmes : client riche, web, mobile ou service. Les applications Windows Store ne dérogent pas à la règle et peuvent être implémentées par des développeurs de tous horizons en se basant sur des technologies éprouvées telles que Visual Studio, .NET, C++, XAML ou encore HTML5 et JavaScript.

Tout comme le système d'exploitation sur lequel elle s'appuie, la plate-forme de développement soutenant les applications Windows Store a été complètement repensée, en capitalisant sur le savoir-faire de Microsoft.

Copyright © 2013 Eyrolles.

Des API accessibles avec le Windows Runtime (WinRT)

Le Windows Runtime est la plate-forme sur laquelle s'appuient les applications Windows Store. Il s'agit pour Microsoft de fournir aux développeurs des API conçues pour faciliter le développement d'applications natives performantes, mobiles, contextuelles, fluides et sécurisées. Un soin tout particulier a également été apporté à l'ouverture de ces API : il est possible de les utiliser depuis de nombreuses technologies telles que le code natif C++, le code managé .NET ou encore JavaScript.

Dans les coulisses, le Windows Runtime et ses API sont implémentés en C++ en se basant sur l'évolution d'un composant fondamental et historique de Windows : le Component Object Model ou COM (dont la première version date de 1993 !). Sa remarquable flexibilité rend possible et naturelle l'utilisation des nouvelles API depuis des technologies aussi variées que JavaScript, C# ou C++, et ce grâce à un mécanisme de projection transparent. Le développeur peut également écrire ses propres composants Windows Runtime de bas niveau, réutilisables depuis n'importe laquelle des technologies citées précédemment. Pour l'utilisateur, la technologie d'implémentation n'a finalement pas d'importance : à l'utilisation, il s'agira dans tous les cas d'une application Windows Store.

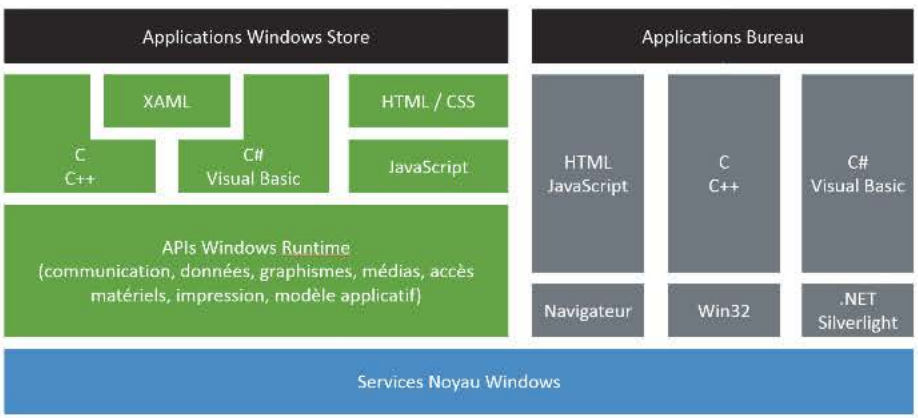


Figure 1–4
Plate-forme applicative

Voici un exemple d'appel Windows Runtime (mise à jour d'une vignette secondaire) dans trois langages :

JavaScript

```
var tileTextAttributes = tileXml.getElementsByTagName("text");
tileTextAttributes[0].appendChild(tileXml.createTextNode("Mon texte !"));
var updater =
notifications.TileUpdateManager.createTileUpdaterForSecondaryTile(appbarTileId);
```

Copyright © 2013 Eyrolles.

C++

```
XmlNodeList^ tileTextAttributes = tileXml->GetElementsByTagName("text");
tileTextAttributes->Item(0)->InnerText = "Mon texte !";
TileUpdater^ secondaryTileUpdater =
TileUpdateManager::CreateTileUpdaterForSecondaryTile(appbarTileId);
```

C#

```
XmlNodeList tileTextAttributes = tileXml.GetElementsByTagName("text");
tileTextAttributes[0].InnerText = "Mon texte !";
TileUpdater secondaryTileUpdater =
TileUpdateManager.CreateTileUpdaterForSecondaryTile(appbarTileId);
```

Le chapitre 13 détaille le fonctionnement interne du Windows Runtime, et notamment le mécanisme de projection.

HTML5 et JavaScript

La possibilité de recourir à des technologies traditionnellement réservées au Web pour le développement d'applications natives est un des éléments remarquables de la nouvelle plate-forme Windows. Les applications Windows Store implémentées en HTML et JavaScript s'exécutent au sein d'un processus nommé `WWAHost` qui fournit aux applications hébergées un environnement basé sur les fondations d'Internet Explorer 10, assurant au développeur une compatibilité complète avec les standards de ce navigateur pour les technologies suivantes :

- HTML5 ;
- CSS3 ;
- SVG ;
- JavaScript/ECMAScript.

Le moteur de rendu d'Internet Explorer est directement utilisé pour afficher le contenu HTML5/CSS3/SVG ; ainsi, les applications bénéficient pleinement de l'accélération matérielle par le biais de DirectX. En d'autres mots, le ou les processeur(s) graphique(s) est(sont) mis à contribution lors de l'exécution des applications HTML5, résultant en des performances d'affichage optimales.

Les composants déjà développés avec ces technologies web peuvent ainsi être réutilisés au sein d'une application Windows, souvent sans changement de code. Une disparité fondamentale existe cependant entre une application web et une application Windows locale : son contexte d'exécution.

Copyright © 2013 Eyrolles.

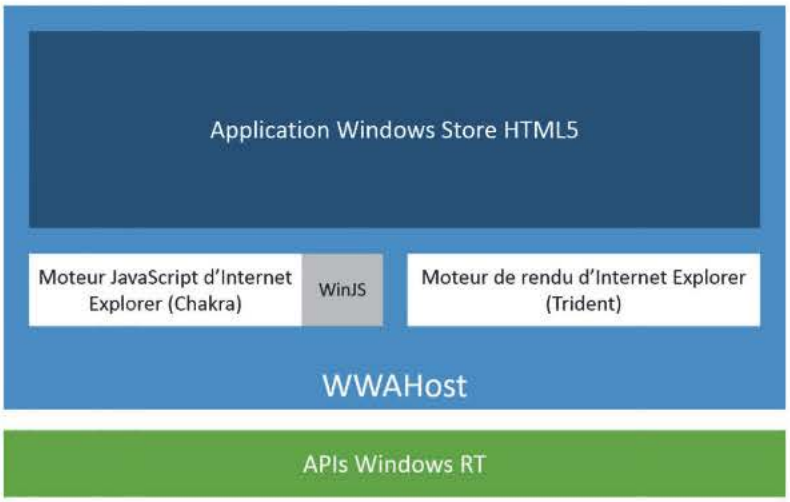


Figure 1–5
Environnement d'exécution WWAHost

Ce dernier, appelé contexte local, semble parfois déroutant dans un premier temps pour le développeur web : l'application Windows Store ne s'exécutant pas au sein d'un serveur web, le concept de session locale est différent de celui d'une session web et l'application dispose d'un cycle de vie sans équivalent sur le Web. Les principales différences entre les contextes web et locaux sont les suivantes :

- Le contexte local donne accès à certaines fonctionnalités spécifiques de Windows, et tout particulièrement aux API du Windows Runtime, par le biais de la bibliothèque Windows pour JavaScript (WinJS). Cette dernière autorise les applications HTML5 à communiquer directement avec le système d'exploitation, donnant ainsi potentiellement accès aux capacités physiques de la machine, aux contrats Windows, à la gestion des manipulations tactiles, au multithreading et à bien d'autres fonctionnalités. Ces ajouts ouvrent la voie à des scénarios jusqu'alors impossibles avec les standards que sont HTML, CSS ou JavaScript.
- Le contexte local ne supporte pas les extensions ou plug-ins (par exemple Flash ou Silverlight).
- Les accès aux ressources machine, notamment l'accès réseau, sont soumis aux déclarations des besoins de l'application, décrites dans le chapitre 11.
- Le développement d'applications HTML5 est une solution particulièrement séduisante pour les développeurs disposant d'une expérience dans le développement web, car elle allie des standards reconnus et maîtrisés aux capacités offertes par le système d'exploitation Windows via le Windows Runtime. Cette intégration de HTML5 et de JavaScript au Windows Runtime crée implicitement une exception technique qui peut surprendre : les applications de ce type, bien qu'utilisant les standards du Web sont intimement liées à Windows et ne peuvent pas s'exécuter au sein d'un navigateur web. Ce nouveau mode de développement s'apparente donc plus à une ouverture supplémentaire offerte par Microsoft dans

Copyright © 2013 Eyrolles.

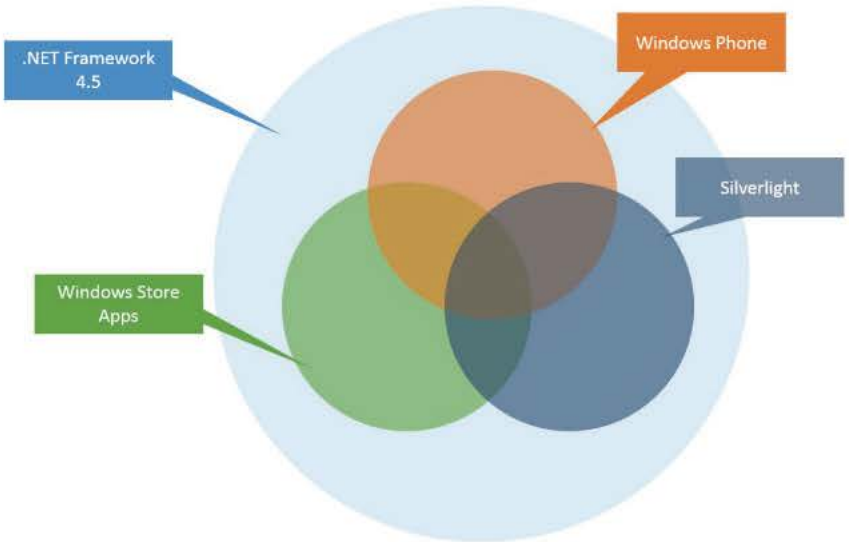
le cadre de la ré-exploitation d'un savoir-faire, plutôt que dans celui d'une réutilisation du code entre application native et application web.

.NET et XAML (eXtensible Application Markup Language)

Les développeurs .NET, en particulier ceux familiers avec WPF et Silverlight, peuvent réutiliser leurs compétences afin de développer des applications Windows Store. En effet, Windows 8 et Windows RT leur offrent la possibilité d'écrire leurs applications en se basant sur le framework .NET 4.5 (Visual C# et Visual Basic) couplé à un moteur de rendu XAML.

Depuis sa version 3.5, le framework .NET cible différentes plates-formes par le biais d'un mécanisme de profils. Ces derniers définissent des sous-ensembles du framework complet, afin de donner ou non l'accès à certaines API en fonction de la plate-forme cible. Par exemple, le profil .NET standard donne accès aux API disponibles sur les postes de travail Windows, tandis que le profil Windows Phone donne accès à celles s'exécutant sur les smartphones équipés du système d'exploitation de Microsoft.

Figure 1–6
Profils .NET



Le profil Windows Store donne accès à un sous-ensemble du framework .NET ainsi qu'à des API spécifiques au Windows Runtime. Le mécanisme de projection rend également certaines API du Runtime accessibles en .NET, facilitant leur utilisation par le développeur et son application.

XAML (eXtensible Application Markup Language) est un langage de description applicatif conçu par Microsoft et basé sur le standard XML. Également utilisé pour décrire des workflows dans Workflow Foundation, son usage le plus courant consiste

Copyright © 2013 Eyrolles.

en la description d'interfaces graphiques. C'est dans ce cadre que le XAML joue un rôle important dans le développement d'applications Windows Store .NET ou C++ car il permet aux développeurs de décrire des interfaces graphiques complexes en bénéficiant des avantages suivants :

- Basé sur du XML, ce langage autorise les développeurs et les intégrateurs graphiques à utiliser des outils spécialisés comme Blend pour construire des interfaces graphiques de manière intuitive et visuelle.
- La séparation est propre entre la description des écrans (XAML) et le code (C#/VB/C++) qui les contrôle.
- À l'exécution, l'utilisation d'un moteur de rendu performant réécrit en code natif utilisant DirectX transforme le XAML en primitives graphiques vectorielles comprises par le GPU.

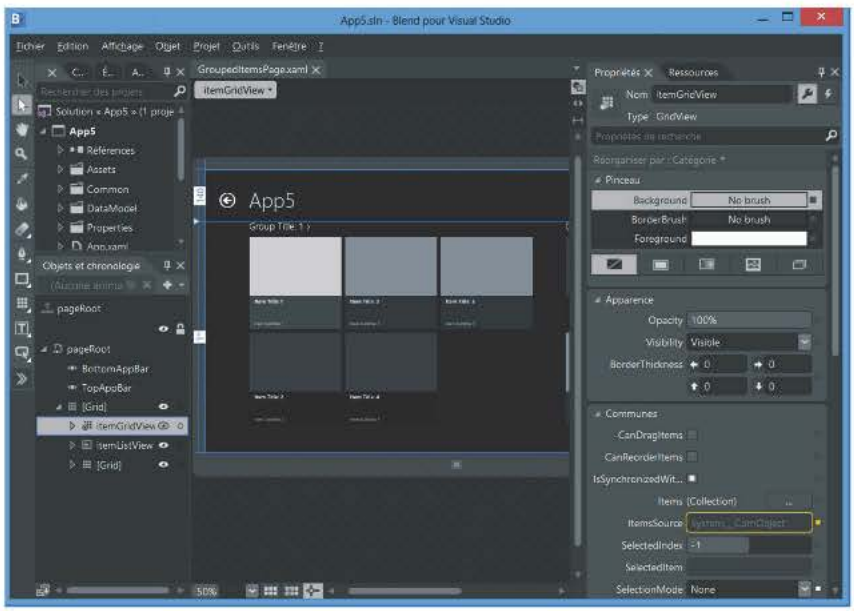


Figure 1–7
Blend pour Visual Studio permet d'éditer le XAML en WYSIWYG

.NET permet également l'implémentation de composants Windows Runtime bas niveau en C# ou Visual Basic. Ces composants, au même titre que ceux écrits en C++, sont réutilisables depuis les autres langages reconnus par le Windows Runtime.

C++ et XAML

Avec le Windows Runtime, le langage C++ revient sur le devant de la scène ; il s'agit d'ailleurs de la technologie utilisée pour implémenter le Windows Runtime lui-même. La couche de projection est donc quasi inexistante, d'où une interaction

Copyright © 2013 Eyrolles.

encore plus transparente avec le Runtime. Le développement d'applications en C++ dispose d'atouts spécifiques le rendant particulièrement intéressant dans certains cas :

- Techniquement, il s'agit de la plate-forme de développement la plus performante. Bien que les applications s'exécutent en grande majorité de manière parfaitement fluide quelle que soit la technologie de développement choisie, les applications en code natif C++ tirent au maximum parti des capacités de la machine.
- Certaines API ne sont disponibles que pour ce langage, notamment celles liées à DirectX. Elles sont de plus combinables avec le moteur de rendu XAML, alliant ainsi performance et productivité lors du développement d'interfaces graphiques complexes.
- Le C++ permet la réutilisation de code portable (propre à l'application ou appartenant à des bibliothèques Open Source) au cœur de l'application, la couche de présentation « supérieure » étant nécessairement liée à Windows.

Il est à noter que dans sa version 2012, Visual C++ implémente l'essentiel de la norme C++ 11, qui rend ce langage plus accessible, notamment au niveau de la gestion de la mémoire grâce aux notions de `shared_ptr<T>` ou `unique_ptr<T>`.

Le standard C++ implémenté dans Visual Studio 2012 est décrit dans la norme ISO/CEI 14882:2011.

Considérations pour les développeurs d'applications Bureau

Pour le développeur ayant déjà une expérience dans le développement d'applications clientes Windows (C++, WPF ou même Silverlight), les privilèges octroyés aux applications Windows Store peuvent sembler extrêmement réduits. Ces dernières s'exécutent en effet dans des contextes contrôlés restrictifs (dits *sandbox*), où les fonctionnalités susceptibles d'avoir un effet sur la sécurité et les ressources sont déclarées à l'avance dans le manifeste de l'application. Voici quelques exemples d'actions qu'une application Windows Store ne pourra pas effectuer :

- accéder sans consentement explicite de l'utilisateur à des dossiers ou fichiers du disque local ;
- lire ou écrire en base de registre ;
- interagir avec l'interface utilisateur hors du périmètre de la zone occupée par l'application elle-même ;
- avoir accès à des informations système locales, journal des événements, processus en cours d'exécution, etc.

Comme mentionné précédemment, le cycle de vie applicatif est lui aussi particulier, ce qui semble complexifier le développement. Il faut cependant garder en tête que ces restrictions garantissent un niveau élevé de sécurité et une confiance accrue de la part de l'utilisateur, tout en laissant le système d'exploitation exercer un contrôle optimal sur l'utilisation des ressources. Pour ces raisons, de nombreux scénarios auront davantage de sens s'ils sont implémentés en tant qu'applications Bureau, tandis que d'autres tireront un grand bénéfice à profiter d'une intégration avec le Windows Runtime.

Copyright © 2013 Eyrolles.

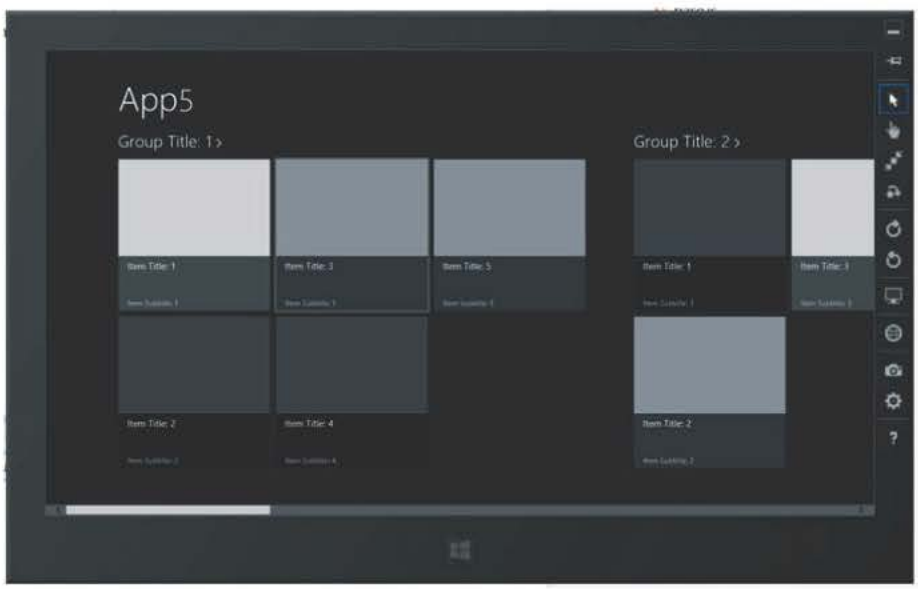
Outillage du développeur

Microsoft a toujours fourni des outils de développement en phase avec l'évolution de ses plates-formes. Les applications Windows Store ne dérogent pas à cette règle et la gamme Visual Studio a été largement enrichie afin de fournir une plate-forme de développement complète pour les applications Windows Store. Mieux encore, certaines versions de ces outils sont disponibles gratuitement, pour que le plus grand nombre, aussi bien professionnels que développeurs occasionnels, soit en mesure de développer des applications Windows Store.

Le SDK Windows 8

Élément fondamental, le SDK Windows 8 (Software Development Kit) fournit les outils nécessaires au développement d'applications Windows Store. En plus de nombreuses bibliothèques, le SDK propose également un simulateur Windows 8 pour tester les applications Windows Store dans différentes résolutions et configurations et pour simuler des événements tactiles sur des machines n'étant pas dotées de ces capacités matérielles.

Figure 1–8
Le simulateur Windows 8



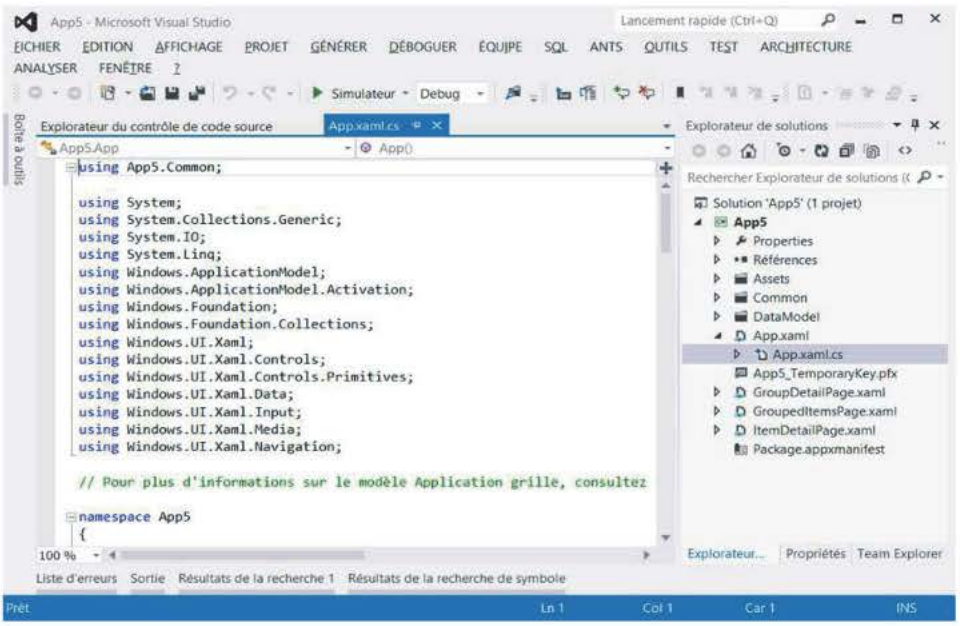
Le SDK étant intégré par défaut à Visual Studio, son installation indépendante est donc un geste rare généralement réservé à la mise en place de machines dédiées à l'intégration continue.

Copyright © 2013 Eyrolles.

Visual Studio 2012

La nouvelle version de l'IDE emblématique de Microsoft permet évidemment de développer des applications Windows Store, mais propose également des centaines d'autres nouveautés, dont une nouvelle interface graphique fluide et épurée, des outils de recherche sophistiqués ou encore de tous nouveaux outils destinés à faciliter le développement d'interfaces graphiques XAML et HTML5.

Figure 1–9
Visual Studio 2012



Blend pour Visual Studio

Expression Blend est un outil familier aux développeurs d'applications WPF et Silverlight. Outil par excellence de conception d'interfaces graphiques XAML, sa nouvelle déclinaison nommée Blend pour Visual Studio sert à concevoir des applications Windows Store aussi bien en XAML qu'en HTML5.

Expression Blend reste disponible pour les applications WPF et Silverlight, tandis que Blend pour Visual Studio est une édition distincte du logiciel, livrée en standard avec Visual Studio et ne ciblant que les applications Windows Store.

MSDN, la référence documentaire

MSDN (Microsoft Developer Network) est la référence pour tous les développeurs sur plate-forme Microsoft. Il regroupe des centaines de milliers de pages de documentation traitant des différentes technologies de développement Microsoft, dont

Copyright © 2013 Eyrolles.

une large section dédiée aux applications Windows Store. Des chapitres entiers sont consacrés à l'expérience utilisateur et à l'ergonomie, et des milliers de pages décrivent en détail chaque API du Windows Runtime.

> **RESSOURCES MSDN pour les applications Windows Store**
>
> ▸ http://msdn.microsoft.com/fr-FR/windows/apps/

Composants tiers

De nombreux éditeurs tiers se lancent dans la commercialisation de composants pour les applications Windows Store. Il s'agit généralement de fournir des bibliothèques et API facilitant l'intégration des contrôles utilisateur (graphiques, grilles, panoramas…) ou encore de traiter/afficher différents formats de fichiers spécifiques.

Bien que la qualité des composants disponibles sur le marché soit inégale, il est tout de même recommandé d'étudier l'intégration de tels composants dans les applications, car elle résulte parfois en une considérable réduction du temps de développement global d'un logiciel.

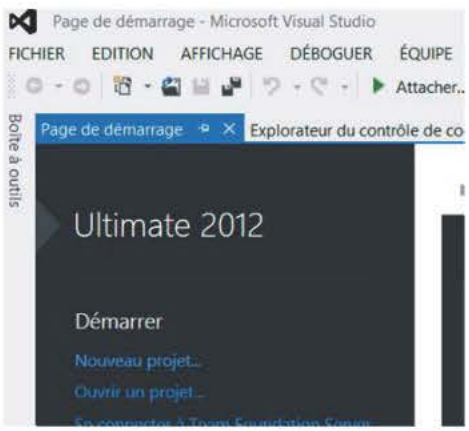
Une première application

Cette section décrit pas à pas la mise en œuvre des outils mentionnés afin de créer une première application Windows Store.

XAML

1 Démarrer Visual Studio 2012. Sur la partie gauche de l'écran, sélectionner *Nouveau projet...*

Figure 1–10



Copyright © 2013 Eyrolles.

2 Dans les modèles préinstallés, déplier l'arborescence de gauche. Sous l'en-tête *Visual C#* ou *Visual Basic* en fonction du langage cible, sélectionner la section dédiée aux projets *Windows Store*. Sélectionner *Application grille (XAML)*.

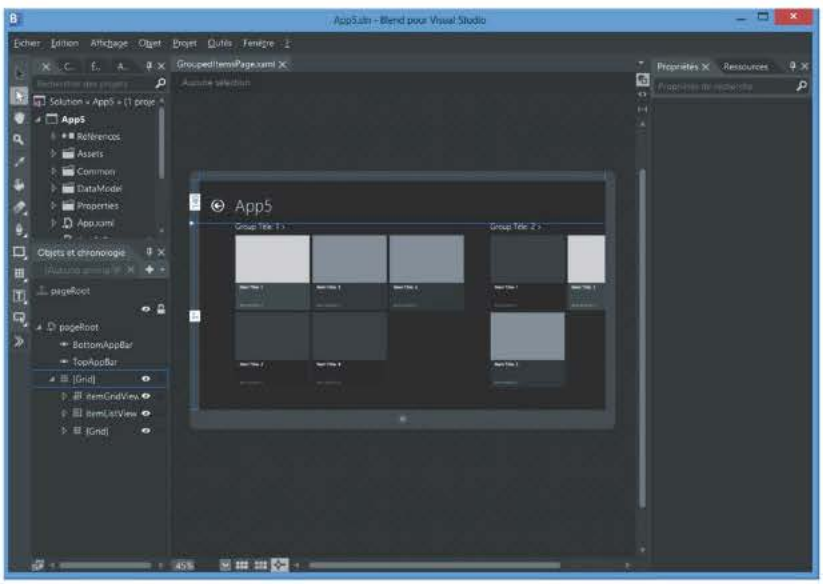
Figure 1–11

3 Nommer le projet dans la zone basse de l'écran et cliquer sur *OK*. Visual Studio crée alors les fichiers nécessaires à la compilation d'un projet d'application Windows Store basé sur .NET et XAML.

4 Visual Studio et Blend sont des outils conçus pour être utilisés de concert. Depuis l'explorateur de solution, un clic droit sur un des fichiers XAML affiche un menu contextuel donnant la possibilité de l'ouvrir dans Blend. Blend pour Visual Studio démarre et le fichier XAML sélectionné s'affiche dans la surface de conception, donnant accès à une palette impressionnante d'outils qui permettent aux développeurs et aux intégrateurs de parfaire les aspects visuels des applications.

Figure 1–12

Figure 1–13



Copyright © 2013 Eyrolles.

5 Revenir dans Visual Studio 2012. L'application telle que créée par l'assistant est prête à s'exécuter. Le bouton de lancement laisse choisir dans quel contexte l'application sera déboguée : dans l'environnement Windows de la machine locale, sur une machine distante, ou au sein de l'émulateur du SDK. Sélectionner l'un d'eux pour démarrer cette première application.

Figure 1–14

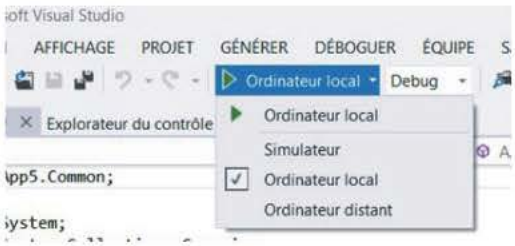
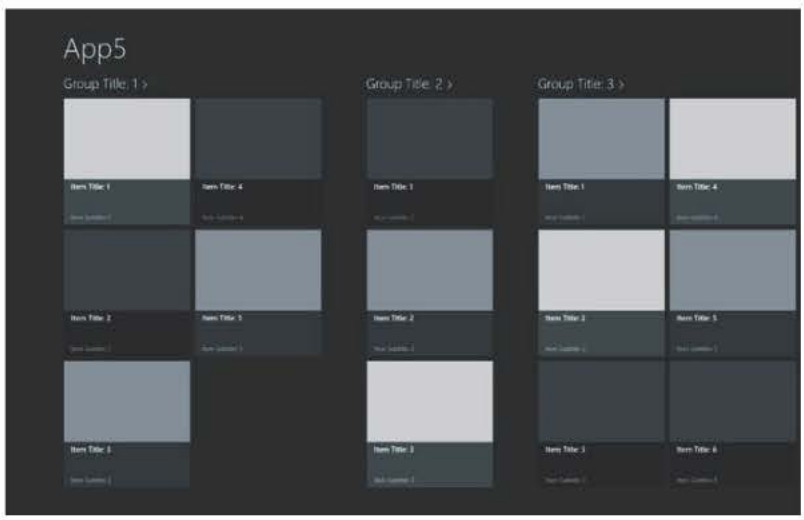


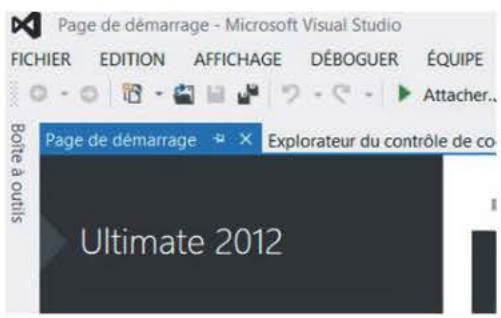
Figure 1–15



JavaScript et HTML

1 Démarrer Visual Studio 2012. Sur la partie gauche de l'écran, sélectionner *Nouveau projet...*

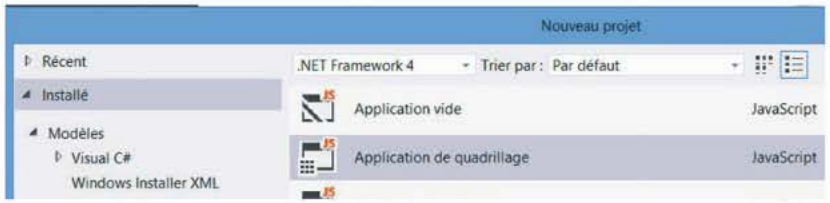
Figure 1–16



Copyright © 2013 Eyrolles.

2 Dans les modèles préinstallés, déplier l'arborescence de gauche. Sous l'en-tête *JavaScript*, sélectionner la section dédiée aux projets *Windows Store*. Sélectionner *Application de quadrillage*.

Figure 1–17



3 Nommer le projet dans la zone basse de l'écran et cliquer sur *OK*. Visual Studio crée alors les fichiers nécessaires à la compilation d'un projet d'application Windows Store basé sur JavaScript et HTML.

Visual Studio et Blend sont des outils conçus pour être utilisés de concert. Depuis l'explorateur de solution, un clic droit sur un des fichiers HTML affiche un menu contextuel donnant la possibilité de l'ouvrir dans Blend. Blend pour Visual Studio démarre et le fichier HTML sélectionné s'affiche dans la surface de conception, donnant accès à une palette impressionnante d'outils qui permettent aux développeurs et aux intégrateurs de parfaire les aspects visuels des applications.

Figure 1–18

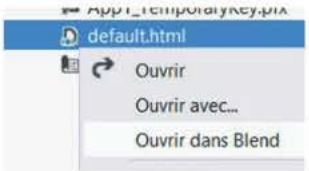
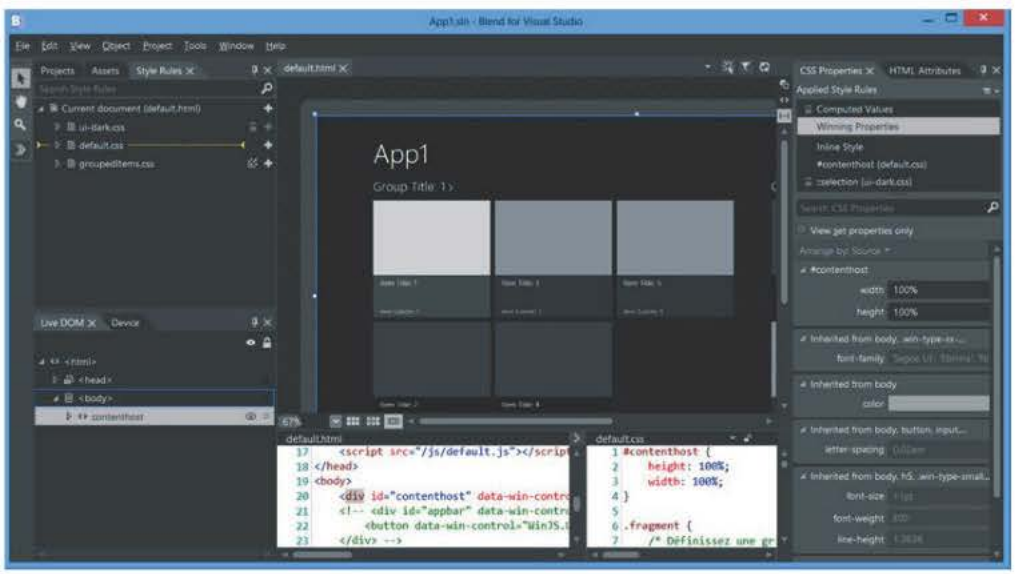


Figure 1–19



Copyright © 2013 Eyrolles.

4 Revenir dans Visual Studio 2012. L'application telle que créée par l'assistant est prête à s'exécuter. Le bouton de lancement laisse choisir dans quel contexte l'application sera déboguée : dans l'environnement Windows de la machine locale, sur une machine distante, ou au sein de l'émulateur du SDK. Sélectionner l'un d'eux pour démarrer cette première application.

Figure 1–20

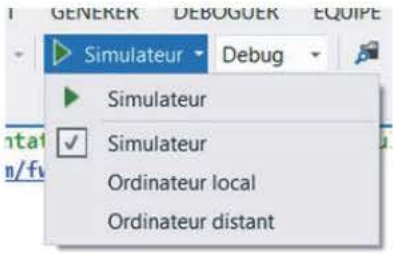


Figure 1–21

Comme l'ont montré les étapes précédentes, la création d'applications Windows Store est facilitée par les fonctionnalités et les outils de développement fournis par Visual Studio. En quelques clics seulement, il est ainsi possible d'obtenir une application Windows Store fonctionnelle !

Copyright © 2013 Eyrolles.

2

Éléments d'ergonomie

L'élément le plus remarquable de Windows 8, celui qui marque le plus les nouveaux utilisateurs, est sans conteste sa nouvelle interface utilisateur, fruit d'une profonde réflexion de la part des équipes de Microsoft. Les applications y jouent un rôle primordial dans l'expérience utilisateur : leur parfaite intégration avec le système d'exploitation est fondamentale dans l'appréciation que l'utilisateur portera à la fois sur les applications et sur le système, les deux formant un tout indissociable.

Ce chapitre a pour objectif d'initier et de sensibiliser le développeur aux principaux concepts ergonomiques liés aux applications Windows Store.

Figure 2–1
L'écran d'accueil Windows

Copyright © 2013 Eyrolles.

Une philosophie de design

La conception des interfaces utilisateur sur les plates-formes et produits de Microsoft obéit à des principes de cohérence et de confort d'utilisation (qui doit être familier, rassurant et évocateur). Cinq principes sont au cœur de cette philosophie et il est conseillé de s'en imprégner pour concevoir des applications Windows Store de qualité.

- **Peaufiner les détails.** Dédier du temps et de l'attention aux détails et à la finition se traduit souvent par des applications de qualité supérieure et des utilisateurs satisfaits. Il y a beaucoup à gagner à parfaire l'utilisation, en particulier pour les éléments les plus vus et les plus utilisés.
- **En faire plus avec moins.** Se concentrer sur le contenu principal, éviter les distractions et laisser la liberté à l'utilisateur d'explorer les fonctionnalités. Les applications Windows Store furent d'ailleurs pendant un moment qualifiées d'immersives pour souligner cet aspect essentiel.
- **Rapidité et fluidité.** Favoriser l'interaction directe et rapide avec le contenu de l'application. Utiliser les animations de manière subtile et élégante pour que l'application réponde aux actions de l'utilisateur tout en le guidant et en l'informant des activités en cours.
- **Authentiquement numérique.** Tirer pleinement parti des médias et de la culture numériques en s'affranchissant au maximum des contraintes et des analogies du monde physique. Assumer pleinement un design digital, des couleurs vives, des alignements parfaits et des interactions rendues possibles par les médias numériques.
- **Gagner en équipe.** Utiliser l'écosystème des applications, les possibilités intrinsèques de la plate-forme et du matériel. Réduire la redondance graphique et fonctionnelle tout en capitalisant sur les connaissances de l'utilisateur. Penser plate-forme.

Cette philosophie de design est implémentée dans Windows 8 et Windows RT par le biais de l'interface utilisateur moderne *(Modern UI)*. À l'exception des jeux, les applications Windows Store doivent en respecter les principes tout en s'intégrant à l'interface utilisateur moderne. Sur ce point, Windows est aujourd'hui plus normatif qu'il ne l'a jamais été.

Copyright © 2013 Eyrolles.

UN PEU D'HISTOIRE

Les premières traces de la philosophie de design Microsoft (*Microsoft design language*) datent de Windows Media Center pour Windows XP. Ce style fut largement enrichi et réutilisé dans la gamme de produits Zune : le baladeur, particulièrement le Zune HD en 2009, et son lecteur multimédia associé sont les premières applications des principes fondamentaux de cette philosophie. Bien que la réception commerciale de ces produits ne fût pas à la hauteur des espérances de l'éditeur, leur influence en termes de design ne fit que croître au sein de Microsoft.

Figure 2–2 Media Center pour Windows XP

Figure 2–3 Le logiciel et le baladeur Zune

Le langage de design Microsoft a été dévoilé publiquement en 2010 sous le nom de « Metro », dans le cadre des annonces liées à la nouvelle génération de système d'exploitation pour téléphone : Windows Phone 7. Suite à sa popularisation par le biais de Windows Phone, ce langage a vu son influence croître au sein de Microsoft, au point de devenir un élément fédérateur et identitaire pour l'éditeur. En 2012, il a ainsi été utilisé pour concevoir les expériences utilisateur de nombreux produits :

- Windows 8 et Windows RT ;
- Windows Phone 8 ;
- Xbox 360 ;

Copyright © 2013 Eyrolles.

- Office 2013 ;
- Visual Studio 2012 ;
- de nombreux sites web et services en ligne Microsoft.

Le terme « Metro » voit son origine dans les transports : des icônes stylisées et explicites, un usage fort de la typographie et une dynamique constante. Le terme sera utilisé en interne et en externe pendant plusieurs années avant d'être renommé « *Microsoft design language* » peu avant la sortie de Windows 8, le 26 octobre 2012.

Ce langage se base sur le style typographique international, aussi connu sous le nom de style suisse, dont les principes fondateurs sont la lisibilité, l'objectivité et la simplicité. L'usage intensif de la typographie et des couleurs uniformes dans les interfaces des produits cités précédemment reflète parfaitement cette influence. Appliqué en informatique, ce style tranche radicalement avec les interfaces dites skeuomorphes, qui visent à reproduire fidèlement des qualités physiques (matériaux, icônes photo réalistes…).

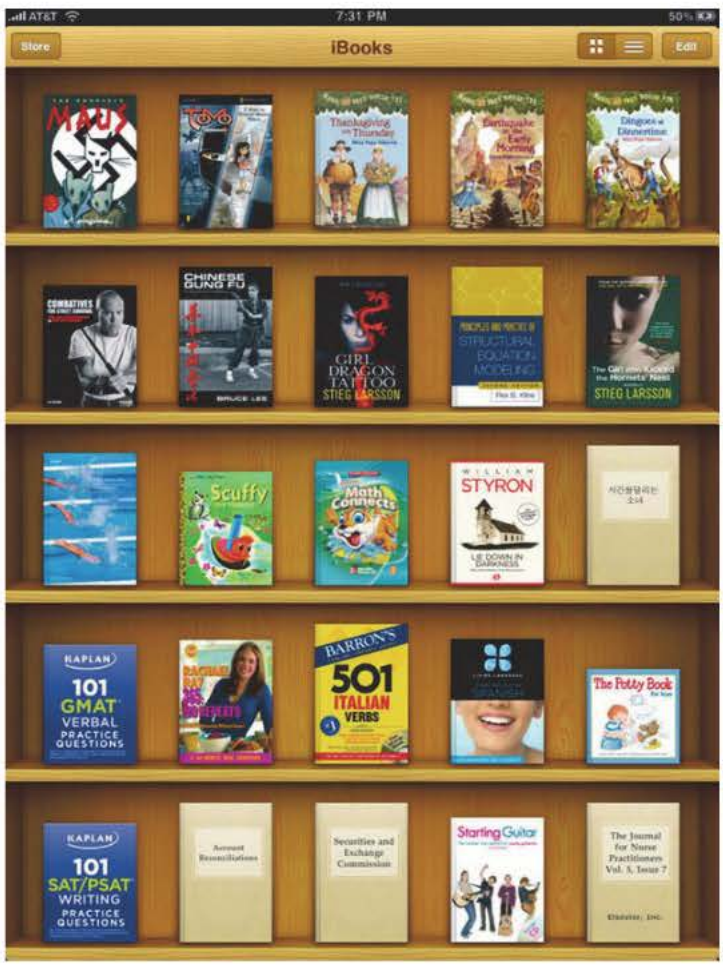


Figure 2–4
iBooks sur iOS imite une bibliothèque physique, et ce jusqu'à l'animation d'ouverture d'un livre.

Copyright © 2013 Eyrolles.

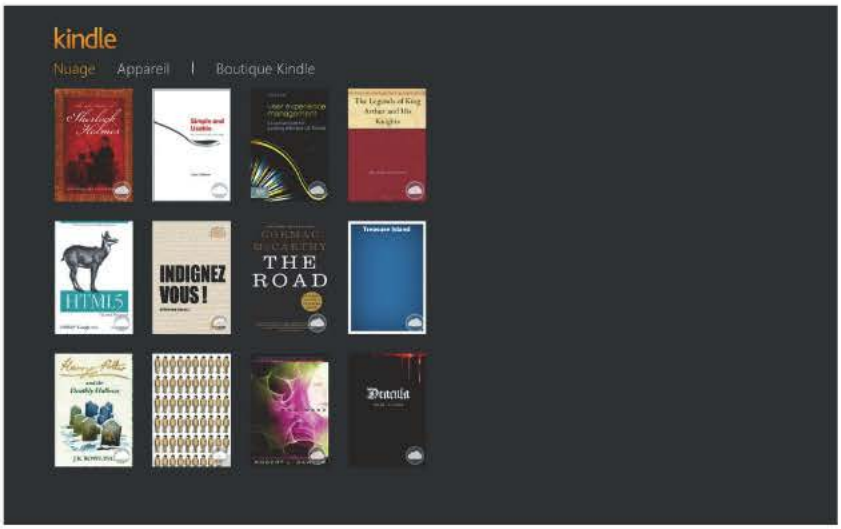


Figure 2–5
L'application Kindle d'Amazon pour Windows 8 représente également des ouvrages, mais sans rappeler ni livre ni bibliothèque.

Le langage de design Microsoft s'avère particulièrement pertinent aujourd'hui. En effet, la quantité croissante d'informations demandées par les utilisateurs a fait naître des besoins importants en termes de synthèse des données et de simplicité d'affichage dans les applications. Plusieurs traits et éléments graphiques récurrents permettent d'exposer le sujet et façonnent ainsi l'expérience utilisateur Microsoft, par exemple :

- l'agencement des éléments et l'alignement du contenu sur une grille virtuelle ;

Figure 2–6
Les éléments sont alignés, créant implicitement un effet de cohésion.

- la diminution radicale des contours, ou *chromes* ;
- l'utilisation de techniques typographiques avec des combinaisons de polices, tailles, couleurs ;

Copyright © 2013 Eyrolles.

- l'emploi avisé des animations qui renforcent l'organisation et le flux visuel des données ;
- les vignettes dynamiques : bien qu'elles ne fassent pas strictement partie du langage, elles sont des éléments importants et distinctifs de l'implémentation du langage sur les plates-formes Windows et Windows Phone.

Ces quelques traits, dont aucun n'est indispensable, soulignent la personnalité particulière du langage de design Microsoft. Ce dernier donne une cohérence à l'ensemble de l'expérience utilisateur Windows, tout en mettant rapidement les utilisateurs à l'aise par la réutilisation de concepts familiers : il est donc essentiel que les applications soient développées en le prenant en compte.

Scénarios d'utilisation

Comme mentionné dans le premier chapitre, Windows et sa nouvelle plate-forme ont été repensés pour intégrer de nouveaux scénarios. La plate-forme historique, basée sur la métaphore du bureau, a également été améliorée et reste pertinente. Il est donc important qu'en amont même de la réalisation d'une application, les équipes de développement prennent en compte les spécificités de chacune des plates-formes afin de déterminer laquelle serait la plus appropriée aux cas d'usage envisagés.

L'expérience utilisateur des applications Windows Store se prête particulièrement bien aux usages en mobilité, tactiles et connectés. Le mode de déploiement via le Windows Store ou en déploiement interne en entreprise est également un facteur important à prendre en compte. Le tableau qui suit présente une liste non exhaustive de certains facteurs d'utilisabilité à prendre en compte lors du choix de la plate-forme applicative.

Tableau 2–1 Comparatif des applications Windows Store versus applications bureau

Applications Windows Store	Applications Bureau
Gestion de facto des interfaces tactiles	Saisie en masse, souris et clavier
API simplifiée pour l'utilisation des capteurs standards (accéléromètre, GPS, gyroscope…)	Accès direct aux matériels : spécifique métier, Kinect…
Plein écran ou modes ancré/remplissage	Fenêtrage
Compatibilité Windows 8, RT	Compatibilité avec Windows 8 et les versions antérieures, en fonction de la plate-forme de développement utilisée. Incompatibilité avec Windows RT.
Importantes restrictions liées à la sécurité. Applications s'exécutant dans une *sandbox*, susceptibles de nécessiter des confirmations explicites de l'utilisateur.	Sécurité relative. Les contraintes de sécurité dépendent des droits de l'utilisateur et de la technologie utilisée pour implémenter l'application.

Copyright © 2013 Eyrolles.

États d'affichage

L'interface de Windows 8 et Windows RT propose à l'utilisateur trois manières différentes d'afficher ses applications Windows Store. Avant d'être publiées, ces dernières doivent obligatoirement passer une série de tests, dont certains consistent à vérifier qu'elles gèrent correctement ces états d'affichage.

Quelques principes ergonomiques régissent l'utilisation des états d'affichage :

- **L'utilisateur doit rester maître de son écran**. Bien que des API telles que `Application.TryUnsnap` autorisent l'application à changer elle-même l'état d'affichage, il est recommandé de ne les invoquer que suite à une action explicite de l'utilisateur. Il est considéré comme malvenu de changer l'agencement de l'écran de l'utilisateur sans que ce dernier ne l'ait demandé explicitement.
- Dans le même ordre d'idée, l'utilisateur est libre de choisir quelles applications s'afficheront simultanément sur son écran. Il ne faut donc pas supposer que deux applications données seront toujours affichées simultanément.
- **Garder les fonctionnalités quel que soit l'état d'affichage appliqué**. En accord avec le point précédent, l'utilisateur peut à tout moment décider de modifier l'état d'affichage d'une application. Cette dernière se doit alors de conserver son contexte de navigation et ses fonctionnalités, et ce même si l'espace d'affichage disponible a changé.
- **Adapter l'affichage à l'espace disponible**. Une application dispose d'un espace d'affichage variant en fonction de son ancrage courant. Il est important d'adapter le contenu de l'application à cet espace, par exemple en ré-agençant les différentes zones ou en modifiant la représentation des données. Les applications Windows Store ont accès à de nombreuses API mettant en œuvre des mécanismes de mise en page adaptatifs (media queries CSS, panels XAML).

Les applications Windows Store disposent d'API leur permettant de fonctionner sur de nombreuses résolutions et DPI (densité de pixels par pouce) différentes. L'adaptation du contenu à l'espace d'affichage est donc primordiale.

> Les exemples de périphériques d'affichage pour les appareils Windows sont nombreux, de la résolution quad HD (3840x2160) jusqu'à la résolution minimale acceptée par les applications Windows Store, 1024x768. Le DPI varie également, y compris au sein des produits d'une même gamme, comme pour les tablettes Surface et Surface Pro de Microsoft dont les écrans 10,6 pouces affichent respectivement des résolutions de 1 366 × 768 et de 1 920 × 1 080.

Plein écran

Si l'application occupe l'intégralité de l'écran, elle doit donc agencer son contenu de manière intelligente afin de s'adapter aux nombreuses résolutions possibles.

Copyright © 2013 Eyrolles.

Figure 2–7
Affichage en mode plein écran

L'absence totale d'éléments graphiques appartenant au système d'exploitation (bordures, titre, menu) en accord avec le principe « plus avec moins », souligne l'aspect immersif des applications Windows Store.

Ancré

Figure 2–8
Affichage en mode ancré
(à gauche ou à droite)

L'application occupe une zone verticale de 320 pixels de large à gauche ou à droite de l'écran. Un séparateur interactif de 22 pixels permet à l'utilisateur de basculer vers le mode plein écran ou le mode ancré. Il s'agit du seul état d'affichage dont une des dimensions, la largeur, est fixe quelle que soit la résolution utilisée. La proportion de cet espace par rapport aux dimensions de l'écran est donc variable en fonction de la résolution horizontale de ce dernier.

La surface disponible relativement restreinte dans cet état limite les interactions possibles ; par conséquent, il est recommandé d'optimiser l'affichage en mode ancré pour des scénarios de supervision passive de données (indicateurs, discussions en ligne, musique). Dans la vaste majorité des cas, c'est l'application adjacente, en mode remplissage, qui sera utilisée et disposera d'un espace d'affichage plus important.

Remplissage

L'application remplit l'essentiel de l'écran, c'est-à-dire la résolution horizontale moins les 22 pixels du séparateur et les 320 pixels du contenu ancré.

Copyright © 2013 Eyrolles.

Figure 2–9
Affichage en mode remplissage (à gauche ou à droite)

En termes de conception et de développement, l'état remplissage est très similaire à l'état plein écran. En effet, et contrairement à l'état ancré, la taille de l'espace disponible n'est pas connue à l'avance et il est donc conseillé d'agencer le contenu dynamiquement. Dans la grande majorité des cas, cet état ne requiert pas de traitement spécifique puisqu'il est finalement assez proche de l'état « plein écran », lui-même adaptatif.

Il est à noter que les modes d'affichage ancré et remplissage requièrent une résolution horizontale minimale de 1 366 pixels. En dessous, Windows n'autorisera pas l'utilisateur à ancrer ses applications.

Concepts ergonomiques de navigation

Les applications Windows Store disposent d'une ergonomie de navigation qui leur est propre. Elle mêle des notions de navigation web à d'autres plus spécifiques à Windows, comme l'utilisation de menus contextuels (barres d'application) ou l'interaction avec le système d'exploitation via les icônes (*Rechercher*, *Partager*, *Paramètres*). Comme dans les applications web, la navigation s'effectue de page en page, avec un rôle important donné au bouton *Retour*, situé en haut à gauche de l'écran. D'autres particularités, comme l'usage intensif de panoramiques et la quasi-absence de fenêtres modales ou d'onglets, contribuent à rendre particulière la navigation au sein des applications Windows Store.

Il est important de maîtriser les concepts de navigation des applications Windows Store, car ils contribuent dès la phase de conception à structurer fortement les écrans et les scénarios d'utilisation : une navigation non intuitive ou illogique est suffisante pour bloquer l'adoption d'une application par les utilisateurs.

Afin de faciliter la compréhension des trois prochaines sections, des analogies avec des applications aux principes et contenus connus de tous seront utilisées :

- analogie « Actualités » : une application servant à naviguer dans des articles de presse et à les lire ;
- analogie « Magasin » : une application servant à explorer le catalogue d'un magasin en ligne.

Copyright © 2013 Eyrolles.

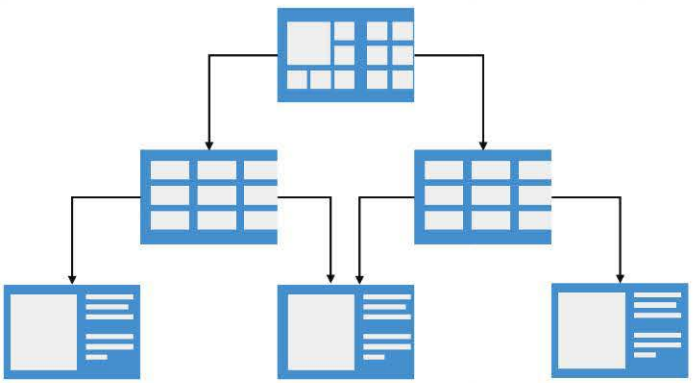
Navigation hiérarchique

La navigation hiérarchique est utilisée pour évoluer parmi les écrans de l'application : de l'écran contenant les informations les moins détaillées vers les écrans affichant les informations les plus précises. Bien qu'il n'y ait pas de règle spécifique à ce sujet, trois écrans types sont mentionnés de manière récurrente par Microsoft :

- Le *hub* présente macroscopiquement les données et est généralement utilisé comme point d'entrée, ou en tant qu'écran d'accueil. Dans l'analogie « Actualités », il s'agirait de l'écran « À la Une » affichant les articles les plus importants du jour. Dans l'analogie « Magasin », il s'agirait d'afficher tous les rayons.
- La *section* décrit un sous-niveau hiérarchique, mais pas un élément particulier. Dans l'analogie « Actualités », il pourrait s'agir d'une thématique comme les sports ou l'international. Dans l'analogie « Magasin », il pourrait s'agir d'un rayon ou d'un sous-rayon comme les livres ou les disques.
- Le *détail* décrit un élément particulier, n'ayant pas d'enfant. Dans l'analogie « Actualités », il s'agirait d'un article, tandis que dans l'analogie « Magasin », il pourrait s'agir d'un produit.

Dans la navigation hiérarchique, le bouton *Retour* est utilisé pour revenir au niveau hiérarchique supérieur, par exemple du détail vers la section, ou de la section vers le hub.

Figure 2–10
Navigation hiérarchique



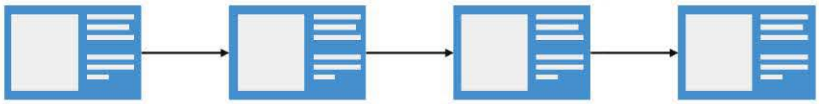
Navigation linéaire

Contrairement à la navigation hiérarchique qui consiste à descendre et remonter les branches d'une arborescence, la navigation linéaire consiste à se déplacer latéralement entre des écrans de même niveau sémantique. Dans l'analogie « Actualités », elle consisterait à naviguer entre deux thématiques ou deux articles. Dans l'analogie « Magasin », elle consisterait à naviguer entre deux rayons ou deux produits d'un même rayon.

Dans la navigation linéaire, le bouton *Retour* est utilisé pour revenir à la page précédente, quelle qu'elle soit.

Copyright © 2013 Eyrolles.

Figure 2–11
Navigation linéaire

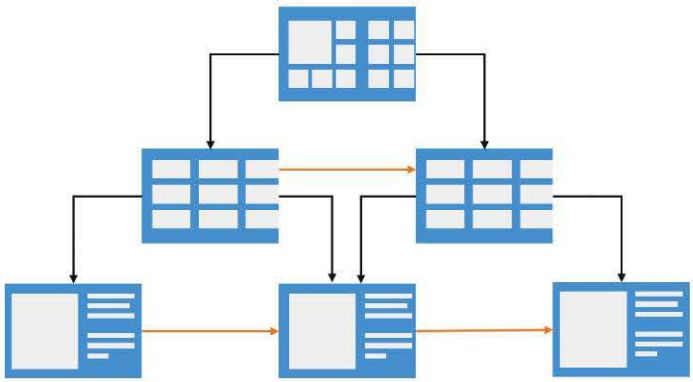


Navigation hybride

La navigation hybride combine la structuration de la navigation hiérarchique à la facilité d'exploration de la navigation linéaire. L'utilisateur explore l'arborescence hiérarchique et peut à tout moment naviguer vers un écran de même niveau en utilisant la liste déroulante associée au titre de la page ou la barre d'application.

Dans la navigation hybride, le bouton *Retour* est utilisé pour revenir au niveau hiérarchique supérieur, et la liste déroulante du titre ou la barre applicative sert à naviguer linéairement.

Figure 2–12
Navigation hybride



Langage d'interaction tactile

La formalisation d'un langage d'interaction tactile est une des grandes nouveautés de Windows 8 et Windows RT. Microsoft a conçu ce langage en alliant richesse fonctionnelle et simplicité : les utilisateurs interagissent de manière cohérente et prédictible avec les applications et leur contenu, et ce avec un court temps d'apprentissage et d'adaptation. Le langage d'interaction tactile fait partie intégrante du système d'exploitation et les API exposées par le Windows Runtime exploitent ses possibilités.

Il est important de souligner que toutes les actions du langage d'interaction tactile sont également exécutables par le biais de la souris et du clavier, conférant un niveau fonctionnel équivalent aux appareils sans capacités tactiles.

Mouvements

Les mouvements sont des actions effectuées physiquement par le ou les doigt(s) de l'utilisateur. Ils sont reconnus par le système d'exploitation et éventuellement trans-

Copyright © 2013 Eyrolles.

formés en manipulations en fonction du contexte. Le tableau qui suit dresse la liste des mouvements reconnus, telle que décrite par Microsoft.

Tableau 2–2 Liste des mouvements reconnus

Mouvement	Description
Appuyer	Brève pression de l'écran avec un doigt.
Appuyer et maintenir	Pression prolongée de l'écran avec un doigt.
Faire glisser	Pression de l'écran avec un ou plusieurs doigt(s) et déplacement dans une seule direction.
Balayer	Pression de l'écran avec un ou plusieurs doigt(s) et déplacement sur une courte distance dans une seule direction.
Tourner	Pression de l'écran avec deux doigts ou plus et mouvement en arc de cercle de haut en bas ou de bas en haut.
Pincer	Pression de l'écran avec deux doigts ou plus, puis rapprochement des doigts.
Étirer	Pression de l'écran avec deux doigts ou plus, puis écartement des doigts.

Manipulations

Les manipulations résultent de la détection des mouvements décrits précédemment. Ils engendrent une interaction contextuelle avec le contenu. Le tableau qui suit dresse la liste des manipulations disponibles par défaut.

Tableau 2–3 Liste des types de manipulations et mouvements associés

Manipulation	Mouvement	Action
Apprentissage	Appuyer et maintenir	Affiche des informations détaillées ou didactiques concernant l'élément sélectionné (par exemple, une infobulle).
Commande	Appuyer	Invoque l'action primaire liée à l'élément sélectionné.
Panoramique	Faire glisser horizontalement	Fait glisser le contenu panoramique.
Zoom optique et redimensionnement	Pincer ou étirer	Modifie le facteur de grossissement de la zone de contenu sélectionnée.
Zoom sémantique	Pincer ou étirer	Pincer bascule généralement vers une vue synthétique affichant simultanément un nombre plus élevé d'éléments, avec moins de détails. Étirer bascule généralement vers une vue détaillée, avec un nombre moins élevé d'éléments, chacun affiché avec plus de détails.

Copyright © 2013 Eyrolles.

Tableau 2–3 Liste des types de manipulations et mouvements associés

Manipulation	Mouvement	Action
Rotation	Tourner	Effectue une rotation plane du contenu sélectionné, autour du point central du segment défini par les deux doigts. Le contenu est déplacé de manière à garder les doigts toujours au même endroit relativement à l'objet manipulé.
Déplacement	Faire glisser	Déplace (translate) le contenu sélectionné sur une surface plane.
Sélectionner	Balayer	Sélectionne un item dans une liste sélectionnable. La direction du balayage est orthogonale au sens de défilement de l'écran (par exemple, verticale dans un panoramique). Le maintien et le déplacement du doigt sur un objet provoquent son déplacement au sein de la liste parente, si cette dernière accepte cette opération. Un effet élastique est utilisé pour signifier le début de l'opération de déplacement.
Affichage des barres de commande	Faire glisser (depuis les bords haut ou bas de l'écran)	Affiche les barres d'application, si elles existent. Les barres apparaîtront quel que soit le bord d'origine de la manipulation.

Le chapitre 9, « Gestion de l'interface tactile », décrit les aspects techniques liés aux mouvements et aux manipulations.

Structure des écrans types

Deux types d'écrans sont utilisés de manière récurrente dans les applications Windows Store.

Pages

La page est le type d'écran le plus commun. La partie haute de l'écran est réservée au bouton *Retour* et au titre, tandis que la zone située en dessous du titre est utilisée pour afficher le contenu. Les dimensions se mesurent en unités qui valent par défaut 20 pixels, tandis que les sous-unités sont égales à 5 pixels. Attention, ces valeurs ne sont pas absolues et varient en fonction de la densité de pixels de l'écran cible (*dots per inch*, *dpi* pour *points par pouce*).

L'utilisation d'unités de mesure abstraites autorise à penser les interfaces en termes de taille, plutôt qu'en nombre de pixels physiques. Il est à noter que lors de la sortie de Windows 8, très peu d'écrans rapportaient à Windows leur *dpi* réel. En XAML comme en HTML, on utilisera de préférence des indications de taille relatives plutôt que des valeurs fixes.

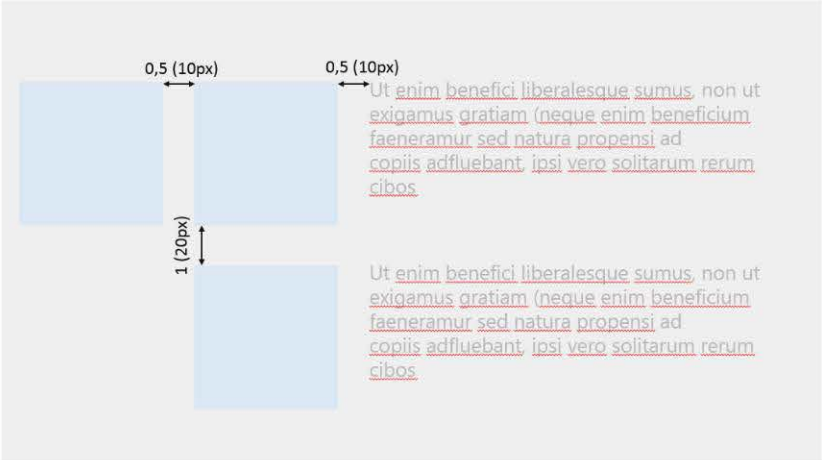
Copyright © 2013 Eyrolles.

La disposition des éléments dans une page est relativement stricte. Une marge gauche de 6 unités est appliquée au titre de la page et à son contenu. Le titre repose sur une ligne virtuelle (marge haute) de 5 unités. Le contenu principal de la page respecte la marge gauche et se situe 2 unités en dessous du titre, à 7 unités verticalement.

Figure 2–13
Espacements des zones principales

Au sein d'une liste, les éléments sont séparés de 0,5 unités horizontalement et d'une unité verticalement. Deux listes, par exemple des sections dans un panoramique, sont séparées de 4 unités horizontalement. Les schémas suivants illustrent ces règles d'affichage.

Figure 2–14
Espacements au sein d'une liste



Copyright © 2013 Eyrolles.

Figure 2–15
Espacements entre listes/sections

Panoramiques

Un des éléments graphiques remarquables des applications Windows Store est l'utilisation prononcée des panoramiques. Implémentés dans un premier temps sur le baladeur Zune et les téléphones Windows Phone 7, les panoramiques se présentent comme des pages dont le contenu s'étale horizontalement en dépassant l'étendue de l'écran, et souvent organisé en sections. L'utilisateur est invité à faire glisser le contenu de droite à gauche sur son écran, telle une fenêtre coulissant sur un parchemin étalé horizontalement.

Figure 2–16
Les panoramiques invitent l'utilisateur à parcourir le contenu horizontalement.

Les écrans de ce type gardent généralement la structure d'une page classique, avec un usage plus prononcé des sections pour organiser les informations, plus nombreuses.

Copyright © 2013 Eyrolles.

Quelques recommandations d'usage supplémentaires optimisent l'expérience liée aux panoramiques.

- Les contenus tronqués à droite sont acceptés ; ils sont même recommandés. En effet, ils invitent l'utilisateur à faire défiler l'écran pour découvrir le reste du contenu disponible.
- Les composants réagissant aux manipulations horizontales, comme les ascenseurs *(sliders)*, sont à proscrire au sein des panoramiques car ils introduisent une ambiguïté dans les interactions.
- Pour les mêmes raisons, les défilements verticaux sont à proscrire. Si de nombreux items doivent être affichés dans une même liste, il est préférable d'afficher seulement les premiers et de renvoyer vers une page dédiée.
- Les panoramiques affichant un contenu majoritairement textuel peuvent implémenter un effet d'aimant sur les sections, afin que le mouvement « balayer » aligne automatiquement le début d'une section avec la marge gauche. Cette interaction aide l'utilisateur dans le parcours d'information, plus difficile à effectuer lorsqu'il est majoritairement constitué de texte.

Éléments d'interface

Certains éléments graphiques sont récurrents au sein des applications Windows Store et il est bon de les considérer comme des implémentations du langage de conception Microsoft. Employer ces éléments à bon escient permet aux utilisateurs de prendre en main les applications plus rapidement en leur fournissant une expérience familière et rassurante.

Vignettes dynamiques

Les vignettes dynamiques, souvent appelées par leur nom anglais *tiles*, sont les éléments emblématiques des interfaces modernes Microsoft. Inaugurées avec Windows Phone 7, elles composent un affichage synthétique et vivant des données tout en fournissant aux plates-formes Windows un élément graphique unique et différentiateur. Elles disposent de certaines caractéristiques qui, une fois bien assimilées, aideront à concevoir des expériences personnalisées s'intégrant harmonieusement à l'ensemble du système.

Les vignettes sont dites « vivantes » car leur contenu peut être mis à jour régulièrement. Lorsque le contenu s'y prête, des animations dynamisent la vignette tout en alternant le contenu affiché pour profiter au maximum de l'espace alloué. Cette vivacité invite les utilisateurs à interagir et à explorer leur contenu.

Copyright © 2013 Eyrolles.

Figure 2–17
Windows 8 et Windows Phone font des vignettesdes éléments centraux de leurs interfaces.

Les vignettes sont interactives et leur action primaire, le « toucher » ou le simple clic, doit engendrer une action de navigation grâce à laquelle l'utilisateur explorera plus en détail le contenu qu'elles proposent. De par leur taille et leur utilisation au sein de Windows, elles représentent toujours un contenu et ne doivent pas être considérées comme de simples boutons.

Il est d'usage d'avoir plusieurs vignettes sur un même écran, pas forcément toutes de la même taille, mais toujours alignées les unes avec les autres. Différentes combinaisons de taille, d'agencement et de différenciations graphiques représentent une hiérarchie d'informations.

Figure 2–18
Les vignettes varient en termes de formes et de dispositions pour s'adapter au support et au type d'information à afficher.

Copyright © 2013 Eyrolles.

Il est important de noter que, hormis une interaction avec la vignette affichée dans l'écran d'accueil Windows, aucun contrôle utilisateur « vignette » prêt à l'emploi n'est disponible dans le SDK Windows. L'implémentation d'une vignette personnalisée n'est pas aussi simple qu'elle peut le sembler de prime abord. Quelques recommandations garantissent qu'une implémentation est cohérente avec le reste du système d'exploitation et les attentes qu'en auront les utilisateurs.

- Baser le comportement et les fonctionnalités sur les vignettes présentes dans l'écran d'accueil Windows. Créer des vignettes dont les agencements, comportements ou fonctionnalités sont particuliers à une application peut être déstabilisant pour l'utilisateur.
- Le catalogue de modèles de vignettes sur MSDN fournit des informations détaillées sur ces dernières : http://msdn.microsoft.com/fr-fr/library/windows/apps/Hh761491.aspx.
- Fournir un retour visuel à l'utilisateur lorsqu'il manipule la vignette, par exemple un effet d'enfoncement lors du toucher et un léger liseré lors du survol de la souris.
- Ne pas afficher simultanément un trop grand nombre de vignettes. Ces dernières disposent d'un contenu riche et dense ; en afficher trop à la fois noie rapidement l'utilisateur sous l'information. Il est recommandé d'en afficher peu et en plus grand format, tout en invitant l'utilisateur à explorer le reste du contenu par le biais de sections, de panoramiques ou de pages dédiées.

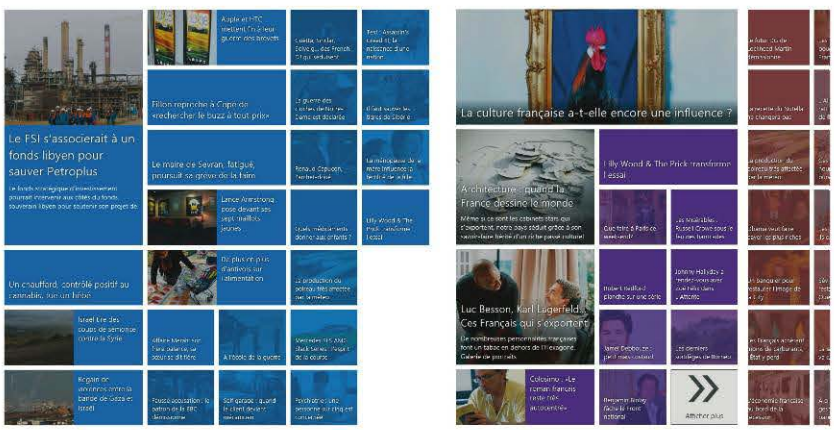


Figure 2–19
Un trop grand nombre de vignettes diminue la lisibilité et rend l'interface confuse.

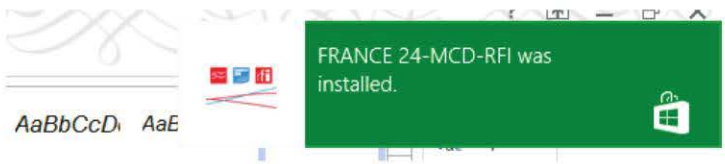
Notifications

Windows met à la disposition des applications Windows Store plusieurs mécanismes de notification à l'utilisateur. Chaque mécanisme a ses particularités et choisir le plus approprié fournira aux utilisateurs une expérience naturelle et intuitive. A contrario, un mauvais choix est susceptible de provoquer incompréhension, confusion, voire agacement.

Copyright © 2013 Eyrolles.

Les trois mécanismes de notification (vignettes, badges et toasts) sont décrits ci-après, par degré d'intrusion croissant.

Tableau 2–4 Description des vignettes, badges et toasts

Mécanisme	Description et scénario	Exemple d'utilisation
Contenu de vignette	Photos ou textes affichés dans l'écran d'accueil Windows, optionnellement animés. S'applique au contenu ne demandant pas d'action immédiate. 	Aperçu du contenu d'un nouveau message, ou d'un nouvel article.
Badge	Petite icône ou nombre, généralement en bas d'une vignette. Indique l'état actuel du contenu (icône), ou le nombre de nouveaux objets depuis le dernier lancement. Lorsqu'un nombre est utilisé pour un badge, il doit représenter le nombre d'éléments à traiter, et non le nombre global d'éléments dans une liste. 	Nombre de messages non lus, de tâches ou d'alertes à traiter. Icône représentant l'état de disponibilité d'un objet.
Toast	Les toasts, ou languettes, s'affichent progressivement en haut à droite de l'écran lorsqu'une information importante doit être communiquée à l'utilisateur. Les toasts sont à utiliser avec parcimonie car relativement intrusifs. En effet, ils apparaissent quelle que soit l'application actuellement affichée à l'écran et émettent un son susceptible de déranger l'utilisateur. Il est recommandé aux applications de fournir une option pour désactiver ce type de notification. 	Réception d'information demandant une attention immédiate : alerte, nouveau message important

Copyright © 2013 Eyrolles.

Icônes et contrats

Une importante partie de l'intégration des applications Windows Store avec le système d'exploitation se fait via les contrats. Les plus importants sont accessibles à tout moment, représentés par des icônes *(charms)* accessibles par le biais de la barre des icônes à la droite de l'écran : *Recherche*, *Partage*, *Périphériques* et *Paramètres*.

Si l'application doit fournir des fonctionnalités couvertes par l'un de ces contrats, il est alors fortement recommandé de les intégrer dans les mécanismes existants, plutôt que d'implémenter des procédés spécifiques. L'application y gagnera en facilité d'utilisation et en ouverture (car elle sera alors en capacité d'interagir avec le système et les autres applications, comme c'est le cas du contrat de recherche ou du contrat de partage de données).

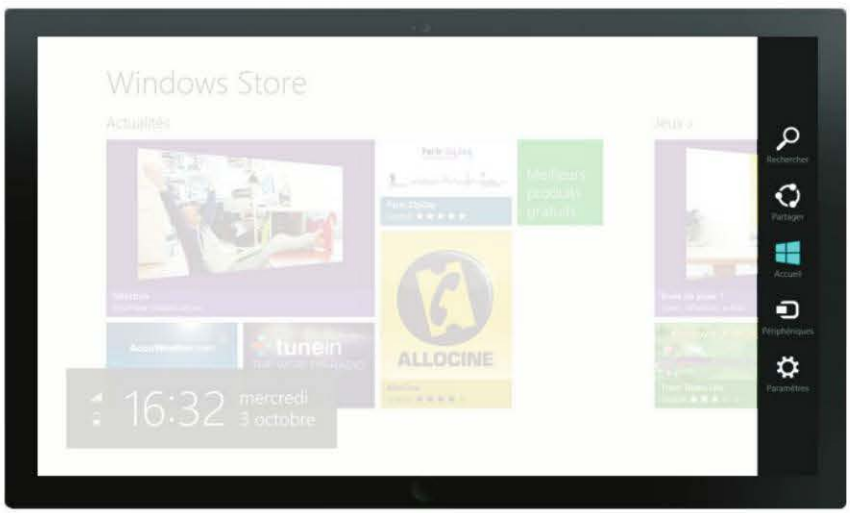


Figure 2–20
Les contrats Rechercher, Partager, Périphériques et Paramètres sont accessibles à tout moment via leurs icônes.

Barres d'applications

Les barres d'applications se positionnent en haut ou en bas de l'écran. Elles donnent accès à des actions contextuelles et apparaissent par un glisser (en partant de l'un des bords haut ou bas de l'écran), un clic droit, ou lors de la sélection d'un ou plusieurs objet(s). Si les deux barres existent, elles apparaissent alors toujours simultanément.

Les barres d'applications ne doivent pas donner accès aux fonctionnalités principales de l'application, mais sont réservées aux actions contextuelles. Dans le même ordre d'idée, elles ne doivent pas être visibles par défaut.

Copyright © 2013 Eyrolles.

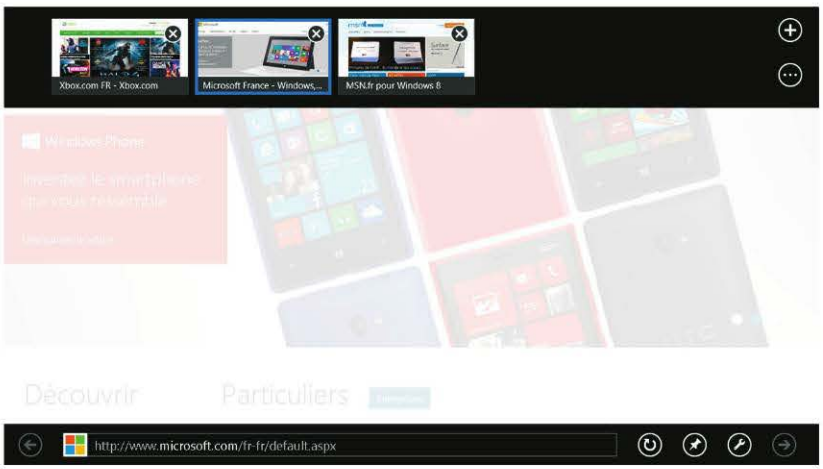


Figure 2–21
Internet Explorer utilise les barres d'applications haute et basse.

Barre d'applications haute

Elle est généralement utilisée pour basculer l'application d'un contexte de navigation à un autre. Elle simule par exemple un système d'onglets dans Internet Explorer, ou permet de basculer entre différentes thématiques dans Bing Actualités.

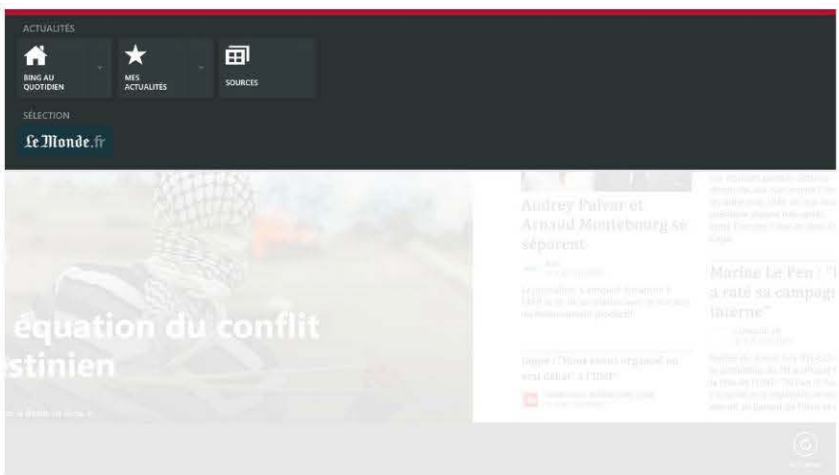


Figure 2–22
La barre d'applications haute facilite souvent l'accès aux thématiques et catégories principales.

Barre d'applications basse

Habituellement dédiée aux commandes contextuelles, elle affiche des boutons d'action. La barre s'affiche automatiquement lors des sélections et propose les actions disponibles en fonction du contexte actuel. Par exemple, l'écran d'accueil Windows affiche des actions différentes en fonction du nombre de vignettes sélectionnées.

Copyright © 2013 Eyrolles.

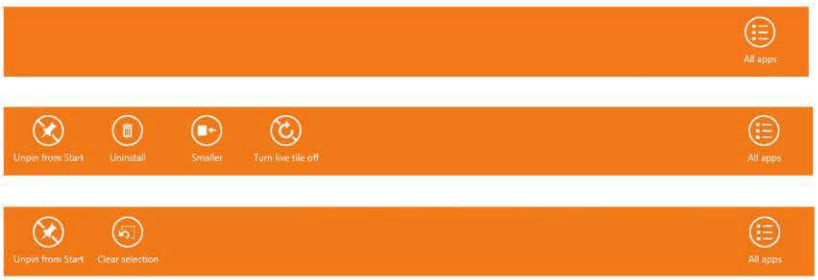


Figure 2–23
L'écran d'accueil de Windows fait varier le contenu de sa barre d'applications basse en fonction du nombre de vignettes sélectionnées.

Le placement des boutons dans la barre d'applications basse doit respecter certaines conventions :

- les boutons se placent à partir des bords gauche ou droit, ce qui facilite leur accessibilité pour les utilisateurs de tablettes tactiles ;
- les boutons liés à une création (*Créer*, *Nouveau*, *Ajouter*…) se placent à droite ;
- les boutons de confirmation (*Oui*, *Accepter*, *OK*…) sont situés à gauche des boutons de refus (*Non*, *Refuser*, *Annuler*…) ;
- l'usage de menus déroulants est recommandé si des actions complexes liées aux commandes sont nécessaires.

Figure 2–24
OneNote complète certaines des actions de sa barre d'applications basse par l'ajout d'un menu déroulant.

Zoom sémantique

Le zoom sémantique fait varier la densité d'information à l'écran sur deux niveaux, de manière à la fois élégante et ludique. Il est dit « sémantique » par opposition à un zoom visuel qui grossit ou diminue la taille d'un objet à l'écran. Le zoom sémantique offre quant à lui, en plus d'un effet visuel de zoom, une autre représentation des informations affichées. Les figures qui suivent présentent deux exemples classiques de zoom sémantique, le premier proche d'un zoom visuel, le second avec une représentation des données bien différente.

Copyright © 2013 Eyrolles.

	Zoomé	Dézoomé
Capsules	Détail de chaque élément	Vue synthétique de chaque élément
Index	Les items catégorisés	Les catégories

Langage visuel

En plus de la philosophie de design et de l'implémentation qui en est faite au sein de Windows, les applications Windows Store respectent un langage visuel particulier. Bien que sa mise en œuvre ne soit pas obligatoire, il s'agit d'une implémentation de référence qu'il est bon d'assimiler.

Absence de contours

Application directe du principe « Faire plus avec moins », l'absence de contours, bordures et autres cadres donne aux applications Windows Store un style élégant, moderne et mettant le contenu en avant.

Polices

Segoe, utilisée depuis 2004 par l'éditeur, est la famille de polices de référence du langage visuel Microsoft, à tel point qu'elle est utilisée pour son nouveau logo, publié en 2012. Elle joue également un rôle central dans Windows 8, les applications Windows Store et Windows Phone.

Copyright © 2013 Eyrolles.

Les recommandations typographiques mentionnent l'utilisation d'au maximum deux polices simultanément au sein d'un écran, en trois tailles différentes. L'utilisation d'un nombre plus élevé de polices et de tailles de police entraîne souvent une réduction de la lisibilité.

Alignement sur une grille

L'alignement des éléments, en particulier celui des vignettes dynamiques, fournit aux écrans applicatifs une structure et une lisibilité accrue. Elle structure aussi les informations de manière hiérarchique sans utiliser de connecteurs ou de contours.

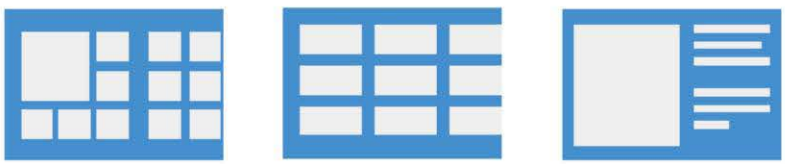


Figure 2–25
Différentes dispositions des éléments permettent de s'affranchir des artefacts visuels pour représenter une organisation d'informations.

Authentiquement numérique

Comme mentionné précédemment, le style visuel des applications Windows Store se veut numérique, avec un usage très limité du skeuomorphisme (la reproduction du monde réel). L'utilisation de couleurs vives, d'icônes stylisées et de contenu photographique est ainsi recommandée. À l'inverse, les graphismes type bandes dessinées et l'usage intensif de dégradés ne sont pas préconisés.

Branding et identité visuelle

Le *branding* consiste à personnaliser les aspects graphiques d'une application afin de laisser transparaître l'image et la personnalité d'une marque. Tout en respectant les principes de la philosophie de design Microsoft, les axes principaux sur lesquels le graphiste peut jouer pour mettre en exergue cette personnalisation sont les suivants.

Tableau 2–5 Les axes pour définir une identité visuelle

Elément visuel	
Couleurs et typographie	Les chartes graphiques définissent toujours l'usage d'au moins un jeu de couleurs et d'une police de caractères. Ces deux points sont applicables à de nombreux endroits dans les écrans d'applications Windows Store. La zone haute, bouton *Retour* et titre, sont de bons candidats à l'application de ces éléments.
Disposition	La disposition des données, notamment en ce qui concerne les textes et les images, fait souvent partie des chartes graphiques des marques.

Copyright © 2013 Eyrolles.

Tableau 2–5 Les axes pour définir une identité visuelle (suite)

Elément visuel	
Graphismes et logo	L'utilisation de dessins, icônes et logos existants rassure l'utilisateur déjà familiarisé avec une marque. Dans le respect du principe « authentiquement numérique », certains graphismes seront adaptés (stylisés), ce qui facilitera leur cohérence visuelle avec l'écosystème *Modern UI*.
Photographies	Certaines marques disposent de banques d'images ayant des styles et codes particuliers, qu'il peut être bon de réutiliser, par exemple au sein de vignettes.

Style visuel et ergonomie

Les recommandations mentionnées dans cette section sont générales et, dans certains cas, l'usage d'un graphisme particulier différencie efficacement les applications. En d'autres mots, l'application stricte du style visuel *Modern UI* n'est pas obligatoire, mais recommandée.

Certaines applications, comme Fresh Paint, démontrent l'utilisation d'un style visuel différent qui respecte néanmoins les principes fondamentaux du langage de design Microsoft, et ce malgré des métaphores réalistes habituellement proscrites (fond de style « bois », pinceaux représentés de façon réaliste, etc). On mesure ici toute la difficulté de cet exercice dont il faut bien peser les risques, notamment en termes de cohérence graphique et d'ergonomie, à la fois au sein d'une même application mais aussi avec le reste du système.

L'implication de spécialistes en ergonomie et en graphisme au sein des équipes de développement s'avère souvent très bénéfique, tout particulièrement dans le cadre des applications Windows Store. Réaliser une application de qualité requiert des compétences très variées qui, mises en commun, réussissent à concilier utilisabilité, esthétisme et excellence technique.

Copyright © 2013 Eyrolles.

3

Modèle, liaisons et accès aux données

L'interface présentée par une application à ses utilisateurs leur sert à consulter et éventuellement modifier un contenu, qui peut être une simple image ou un ensemble assez complexe de données structurées. Ce chapitre présente les principaux mécanismes que le développeur va être amené à mettre en œuvre autour de données structurées, c'est-à-dire :

- *leur modélisation en mémoire ;*
- *leur affichage et leur modification à travers l'interface utilisateur de l'application ;*
- *leur persistance.*

> **REMARQUE Différences de langages**
>
> Certains mécanismes sont ici assez différents entre HTML5/JavaScript et XAML/C# (ou VB.Net) et seront, le cas échéant, présentés dans des sections distinctes.

Principes généraux

Bien que ce ne soit pas une caractéristique propre au développement d'applications pour Windows, il faut rappeler combien il est important de modéliser soigneusement les données que l'application manipule : stocker de simples paires clé–valeur de type chaînes de

Copyright © 2013 Eyrolles.

caractères dans un dictionnaire rend service dans certains cas, mais conduit le plus souvent à un code peu fiable, truffé d'erreurs qui ne seront détectées qu'à l'exécution.

Le **premier principe** à appliquer lors du développement d'une application Windows Store consiste donc à adopter une **approche orientée objet**, avec des classes, des méthodes et des propriétés aux responsabilités bien définies. La modélisation orientée objet sort du cadre de ce livre, néanmoins l'implémentation du modèle ainsi défini doit respecter certaines contraintes imposées par la technologie d'interface utilisateur, afin d'exploiter pleinement celle-ci. Il s'agit typiquement d'implémenter certaines interfaces bien définies ou de s'appuyer sur des classes du framework qui les implémentent : par exemple, pour que les mécanismes de liaison de données fonctionnent correctement et pour obtenir que les modifications des propriétés d'un objet soient automatiquement prises en compte par l'interface utilisateur, on ne pourra pas se contenter de l'implémentation des propriétés intrinsèques au langage C# (les mécanismes à mettre en place seront décrits plus loin).

D'autres contraintes ont un effet sur l'implémentation du modèle de données : par exemple, des considérations de portabilité nécessitent parfois l'utilisation de deux niveaux dans l'implémentation du modèle. C'est typiquement le cas lors de l'utilisation du C++ comme langage de programmation :

- le cœur du modèle est généralement implémenté de manière totalement indépendante de la plate-forme, en utilisant uniquement les types et classes portables du C++ (la Standard Template Library est très riche et il serait dommage de s'en passer) ou de bibliothèques Open Source disponibles pour d'autres environnements ;
- une fine couche d'encapsulation va transformer les types standards C++ en types compris par les classes de WinRT avec lesquelles l'interface utilisateur est implémentée : ainsi, les chaînes de caractères seront des `std::string` au cœur du modèle, mais exposées comme des `Platform::String` dans la partie « supérieure » de celui-ci.

Le **deuxième principe** récurrent dans les applications Windows Store est l'utilisation quasiment systématique du **mécanisme de liaison de données** ou *DataBinding* (on exclut ici les jeux DirectX, qui suivent une logique différente). À quoi ce mécanisme correspond-il ? Il s'agit en fait d'automatiser les échanges bidirectionnels (ou non) entre le modèle et les composants graphiques.

Soit par exemple une classe `Personne` possédant une propriété `Nom` de type chaîne de caractères :

```
public partial class Personne {
   public string Nom
   {
      get { … }
```

Copyright © 2013 Eyrolles.

```
        set { … }
…
}
```

Soit un contrôle de type `TextBox` chargé d'afficher cette propriété en permettant à l'utilisateur d'en modifier la valeur :

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

Avec un mode de programmation traditionnel, on utiliserait le code suivant pour afficher le nom dans le contrôle (en supposant ici que `joueur` est une variable de type `Personne`) :

```
NameTextBox.Text = joueur.Nom;
```

Ensuite, lors du clic sur un bouton ou de toute autre action nécessitant la mise à jour de l'objet, la propriété `Nom` de la variable `joueur` serait mise à jour par un code similaire :

```
joueur.Nom = NameTextBox.Text;
```

Cette approche présente plusieurs inconvénients. Tout d'abord, le code est fortement dépendant du contrôle graphique utilisé pour afficher le nom de la personne. Par exemple, si l'écran évolue et si ce nom n'est plus modifiable, il est souhaitable de remplacer la `TextBox` par un `TextBlock` ; toutefois, cela demande alors de modifier non seulement le XAML, mais aussi le code C# associé. Ensuite, au début de chaque action, il faut mettre à jour tous les objets du modèle affichés dans la page, puis mettre à jour celle-ci à la fin de l'action, ou tout au moins faire ces opérations pour les champs dont cette action va lire ou modifier les valeurs. C'est un travail fastidieux et source d'erreurs.

Le mécanisme de liaison de données permet d'automatiser les transferts entre l'interface utilisateur et le modèle des données à l'aide d'une approche purement déclarative :

```
<TextBox Text="{Binding Nom, Mode=TwoWay}"/>
```

Cette déclaration est assez dépouillée car elle part du principe qu'une instance de la classe `Personne` constitue le contexte de données (`DataContext`) du contrôle graphique et car le type chaîne de caractères est quant à lui le plus simple à lier. Dans d'autres cas, la syntaxe d'une liaison de données sera plus riche. La suite de ce chapitre détaillera l'ensemble des possibilités offertes par la liaison de données au travers d'exemples plus sophistiqués.

Copyright © 2013 Eyrolles.

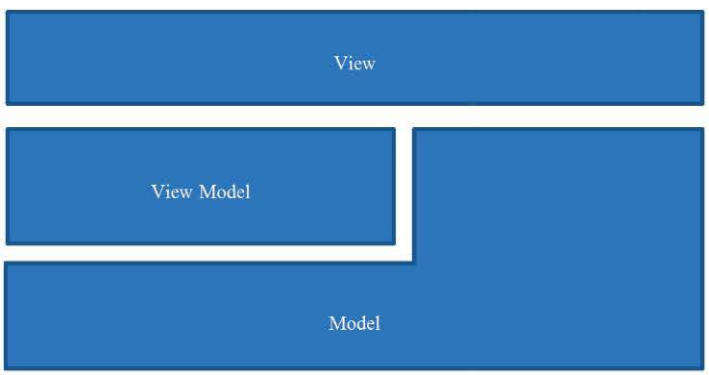
Le pattern MVVM (Model-View-ViewModel)

Une implémentation à deux niveaux du modèle de données comme celle présentée précédemment correspond typiquement à un schéma d'implémentation populaire dans les environnements WPF et Silverlight, le MVVM, pour Model-View-ViewModel :

- Modèle *(Model)* – Comme décrit plus haut, c'est la représentation interne des données affichées et manipulées par l'utilisateur.
- Vue *(View)* – C'est la partie graphique constituée d'un assemblage de contrôles.
- Modèle visuel *(View Model)* – Il constitue une vue des données prêtes à être affichées par les mécanismes de liaison, mais conserve malgré tout une certaine indépendance vis-à-vis des détails de l'affichage. Par exemple, dans le cas d'une liste d'éléments à afficher, le modèle visuel ajoute souvent au modèle la notion d'élément sélectionné, mais il n'impose pas pour autant l'utilisation d'un contrôle graphique particulier pour afficher cette liste.

  De la même manière, le modèle visuel expose des commandes, actives ou non en fonction de son état et de celui du modèle associé. Ces commandes seront déclenchées aussi bien par des boutons que par d'autres manipulations effectuées par l'utilisateur.

Figure 3–1
Modèle MVVM



La figure 3-1 présente une possibilité d'interaction directe entre la vue et le modèle. En effet, dans beaucoup de cas, les classes appartenant au modèle sont directement exploitables par la vue et il est donc inutile d'ajouter une encapsulation qui aurait un coût en termes de développement et de performances sans apporter de plus-value. Néanmoins, cette interaction doit être limitée au mécanisme de liaison de données et exclure l'exécution d'actions, celles-ci devant être contrôlées par le modèle visuel.

Copyright © 2013 Eyrolles.

> **ALLER PLUS LOIN** **L'approche MVVM**
>
> MVVM est une approche bien particulière dont la richesse et les subtilités ne peuvent être décrites en quelques lignes et sortent du cadre de cet ouvrage. Le lecteur intéressé trouvera sur le Web de nombreux articles présentant en profondeur cette méthode, ainsi que des bibliothèques de code qui en simplifient la mise en œuvre.

Liaison de données en XAML avec .NET

Liaison simple

La liaison de données est initialisée par l'utilisation de l'extension XAML `{Binding}` pour préciser la valeur d'une propriété comme dans l'exemple donné en début de chapitre :

```
<TextBox Text="{Binding Nom, Mode=TwoWay}"/>
```

L'extension `{Binding}` déclenche la création d'une instance de la classe `Binding` au même titre que les autres éléments du langage XAML, tel `TextBox`. L'objet `Binding` ainsi créé est responsable des transferts de données entre le contrôle, constituant la cible de la liaison, et la source de données. Si le type de données exposé par la source n'est pas exactement celui attendu par la cible, un convertisseur de type est utilisé : ce dernier peut être déterminé automatiquement par le moteur ou être précisé explicitement dans le XAML. Dans l'exemple de code précédent, on peut se demander d'où vient la propriété `Nom` qui est liée à la propriété `Text` du `TextBox`… Elle provient de l'objet source, non représenté dans ce code, qui a été lié à la propriété `DataContext` d'un des éléments supérieurs au `TextBox` dans la hiérarchie visuelle, généralement un conteneur de type grille, `Canvas` ou autre.

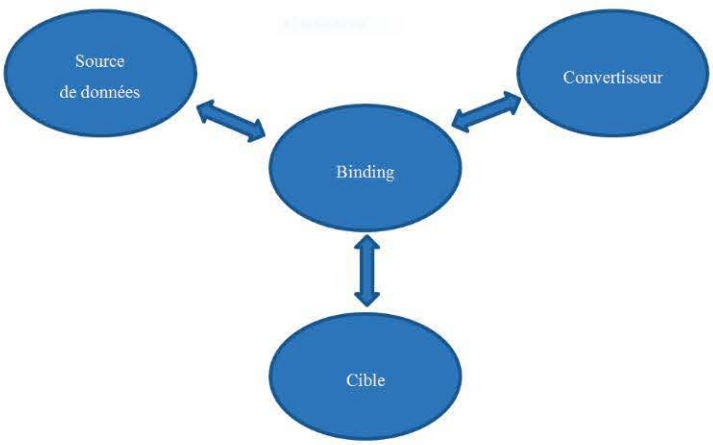


Figure 3–2
Liens entre Binding, source, cible et convertisseur

Copyright © 2013 Eyrolles.

Si la source de données peut être définie comme une propriété standard C# (en ajoutant la prise en charge de `INotifyPropertyChanged`), la cible de la liaison de données doit être une propriété de type « propriété de dépendance » (`DependencyProperty`). Le runtime WinRT fournit un système de propriétés particulièrement adapté aux mécanismes de liaison de données et aux animations. De plus, ce système est extensible : la classe `Grid` définit par exemple des propriétés de dépendances (ligne, colonne, étendue en lignes, étendue en colonnes) qui peuvent être associées à des contrôles insérés dans cette grille, même si en termes d'héritage de classes, ces contrôles n'ont pas de lien avec le contrôle `Grid` et ne possèdent pas directement ces propriétés.

La source de la liaison de données doit être une propriété appartenant à un objet .NET (ou un objet C++ WinRT avec l'attribut `Bindable` ou implémentant l'interface `ICustomPropertyProvider`). Il peut s'agir d'un objet du modèle ou d'un objet graphique, voire la cible elle-même (ou plus précisément une autre propriété de la cible).

Si la source implémente l'interface `INotifyPropertyChanged`, l'objet `Binding` va s'abonner à l'événement `PropertyChanged` de celle-ci et transmettra les mises à jour de la propriété liée à la cible. Les modèles de projet utilisés par Visual Studio contiennent pour la plupart une classe `BindableBase` qui expose cette interface ainsi que deux méthodes facilitant le déclenchement de l'événement `PropertyChanged` :

- `OnPropertyChanged` effectue ce déclenchement ;
- `SetProperty<T>` s'appuie sur une nouveauté du langage C# grâce à laquelle une méthode récupère le nom de la méthode appelante. Si la valeur passée en paramètre est différente de la valeur courante de la propriété, celle-ci est remplacée et l'événement `PropertyChanged` est déclenché.

```
protected bool SetProperty<T>(ref T storage, T value,
                    [CallerMemberName] String propertyName = null)
…
```

Avec ces deux méthodes, l'implémentation d'une propriété conforme à ce qu'attend le moteur de liaison de données devient assez simple, par exemple :

```
private string _nom;
public string Nom
{
    get { return _nom; }
    set { SetProperty(ref _nom, value); }
}
```

Copyright © 2013 Eyrolles.

Si une propriété calculée est affichée, il suffit de déclencher l'événement `PropertyChanged` correspondant pour que le moteur de liaison la rafraîchisse : on utilise ici le booléen retourné par `SetProperty`. Voici par exemple l'implémentation d'une propriété `NomComplet` et la modification associée sur l'implémentation de la propriété `Nom` :

```
public string NomComplet
{
    get { return Prenom + " " + Nom; }
}

private string _nom;
public string Nom
{
    get { return _nom; }
    set {
      if (SetProperty(ref _nom, value)) OnPropertyChanged("NomComplet");
    }
}
```

Les propriétés implémentées de manière automatique en C# (`public string Nom {get; set ;}`) ou en VB.NET (`public property Nom as string`) peuvent être affichées par liaison de données. Néanmoins, le moteur de liaison ne sera pas notifié lors de leurs modifications et ne mettra pas à jour la cible, c'est-à-dire l'élément affiché.

Dans une liaison de données, le convertisseur est facultatif ; il n'est par exemple pas nécessaire dans le cas simple d'un contrôle de type `TextBox` lié à une propriété de type chaîne de caractères. Cependant, il se révèle très utile dans un certain nombre de cas : le mécanisme de liaison de données peut par exemple servir à colorer un contrôle graphique en fonction de la valeur d'une propriété qui n'est pas du tout graphique et c'est alors le convertisseur qui effectuera la correspondance. En termes de code, un convertisseur est simplement une classe implémentant l'interface `IValueConverter` et est généralement déclaré en tant que ressource XAML. Dans l'exemple qui suit, la couleur du bloc de texte est liée à la propriété `Age` de l'instance de `Personne` affichée :

```
<Page.Resources>
   <local:AgeToColorConverter x:Key="AgeToColorConverter"/>
</Page.Resources>
…
<TextBlock Text="{Binding Nom}"
   Foreground="{Binding Age, Converter={StaticResource
  AgeToColorConverter}, ConverterParameter=30}"/>
```

Copyright © 2013 Eyrolles.

```
public class AgeToColorConverter : IValueConverter
{
   private static int GetValue(object value, int defaultValue)
   …
   private static Brush AboveBrush = new SolidColorBrush(Colors.Green);
   private static Brush BelowBrush = new SolidColorBrush(Colors.Red);

   public object Convert(object value, Type targetType,
                         object parameter, string language)
   {
      int intValue = GetValue(value, 0);
      int intTrigger = GetValue(parameter, 0);
      return (intValue >= intTrigger) ? AboveBrush : BelowBrush;
   }
…
}
```

Dans cet exemple, l'âge de la personne à partir duquel le convertisseur change la couleur du pinceau retourné est passé en paramètre au niveau de la balise `{Binding}` (c'est la valeur « 30 » dans le code exemple). Cela implique de convertir une chaîne de caractères en entier à chaque fois que la valeur de la propriété `Nom` est modifiée. Une manière légèrement plus efficace de gérer ce seuil consiste à le passer une fois pour toutes au convertisseur et à ne pas utiliser le paramètre `ConverterParameter` ; il suffit pour cela d'ajouter une propriété de type `int` au convertisseur et de la positionner dans les ressources de la page (cette approche nécessite cependant que le paramètre soit identique pour toutes les utilisations du convertisseur au sein de la page).

```
<Page.Resources>
   <local:AgeToColorConverter x:Key="AgeToColorConverter"
                              Trigger="30"/>
</Page.Resources>
…
<TextBlock Text="{Binding Nom}" Foreground="{Binding Age,
               Converter={StaticResource AgeToColorConverter}}"/>


public class AgeToColorConverter : IValueConverter
{
   public int Trigger { get;set;}
   private static int GetValue(object value, int defaultValue)
   …
   private static Brush AboveBrush = new SolidColorBrush(Colors.Green);
   private static Brush BelowBrush = new SolidColorBrush(Colors.Red);

   public object Convert(object value, Type targetType,
                         object parameter, string language)
```

Copyright © 2013 Eyrolles.

```
    {
        int intValue = GetValue(value, 0);
        return (intValue >= Trigger) ? AboveBrush : BelowBrush;
    }
...
}
```

Avant de détailler complètement la syntaxe de la balise `{Binding}`, il faut noter que la source des données n'est pas nécessairement un objet du modèle, comme une `Personne`, mais peut être un contrôle graphique : dans l'exemple qui suit, la hauteur du rectangle est directement liée à la valeur du curseur :

```
<Slider x:Name="TheSlider" Minimum="1" Maximum="100"
      Orientation="Horizontal" Value="10"/>
<Rectangle Fill="Blue" Height="{Binding Value, ElementName=TheSlider}"/>
```

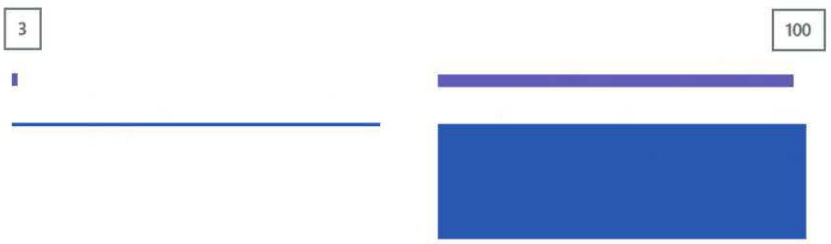


Figure 3–3
Liaison entre contrôles graphiques

Le tableau suivant dresse la liste détaillée des paramètres qu'il est possible de préciser lors de la mise en place d'une liaison de données.

Tableau 3–1 Paramètres pour définir une liaison de données

`Path`	Chemin de la propriété, c'est le paramètre principal de la liaison. Il peut être omis : `{Binding Value}` est en effet équivalent à `{Binding Path=Value}`.
`ElementName`	Nom de l'élément dont on va lire (et éventuellement écrire) la propriété désignée par le paramètre `Path`. S'il est omis, la propriété est lue dans le contexte de données de l'élément XAML (voir plus bas).
`Mode`	Type énuméré indiquant dans quels sens s'effectuent les mises à jour : `OneTime` = une lecture unique au moment de l'initialisation de la liaison, `OneWay` = de la source vers l'élément XAML, `TwoWay` = dans les deux sens (pour un `TextBox` par exemple).
`Source`	Référence vers la source de données. En XAML, cela sera souvent utilisé pour accéder à une propriété d'un objet déclaré dans un dictionnaire de ressources.

Copyright © 2013 Eyrolles.

Tableau 3–1 Paramètres pour définir une liaison de données (suite)

`RelativeSource`	Indique une source de données relative à l'élément XAML courant.
	`Self` désigne l'élément lui-même :
	`<Rectangle …`
	`Height="{Binding ActualWidth, RelativeSource={RelativeSource Self}}"/>`
	crée un carré
	`TemplatedParent` désigne l'élément dans le modèle duquel on effectue la liaison de données. Par exemple, si on définit un modèle de bouton ainsi :
	`<ControlTemplate x:Key="ButtonControlTemplate1" TargetType="Button">`
	`  <Grid>`
	`  <Ellipse Fill="{Binding Background, RelativeSource={RelativeSource TemplatedParent}}"/>`
	`  <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>`
	`  </Grid>`
	`</ControlTemplate>`
	l'ellipse prendra la couleur de fond précisée lors de l'instanciation de ce modèle de bouton :
	`<Button Background="Red" Width="100" Content="test" Template="{StaticResource ButtonControlTemplate1}"/>`
	Remarque : la balise `TemplateBinding` réalise la même liaison avec une syntaxe plus succincte :
	`<Ellipse Fill="{TemplateBinding Background}" />`

`Converter`	Convertisseur (voir paragraphes précédents).
`ConverterParameter`	Paramètre passé dans les méthodes `Convert` et `ConvertBack` de l'interface `IValueConverter` implémentée par le convertisseur. Ce paramètre est passé sous forme de chaînes de caractères.
`ConverterLanguage`	Paramètre de langue passé au convertisseur : c'est particulièrement utile pour les affichages et saisies de nombres ou dates sous forme de chaînes de caractères.

Si aucune source n'est précisée, c'est le contexte de données de l'élément XAML qui est utilisé pour accéder à la propriété liée. Ce contexte est alimenté de différentes manières :

- en le précisant explicitement au niveau d'un élément : ceci est écrit soit dans le code C# associé à la page ou au contrôle, soit dans le XAML lui-même. Dans le premier cas, il s'agit de tout objet .NET existant au moment de l'affectation de la propriété `DataContext` ; dans le deuxième cas, il s'agit d'un objet référencé explicitement (typiquement un objet déclaré en ressources) ou d'un objet retourné par liaison de données ;

Copyright © 2013 Eyrolles.

- par héritage du contexte d'un objet parent : tant qu'on ne remplace pas le contexte hérité à l'aide d'une affectation explicite, les composants XAML héritent du contexte de données de leur parent.

L'exemple qui suit illustre ces deux modes d'alimentation : le contexte de données du `StackPanel` `MySP` correspond à l'instance de `Personne` sélectionnée dans la liste `TheListBox` et les différents contrôles incorporés dans le `StackPanel` héritent automatiquement de ce contexte :

```
<ListBox x:Name="TheListBox"
         ItemsSource="{Binding Joueurs, ElementName=ThePage}"/>
<StackPanel DataContext="{Binding SelectedItem, ElementName=TheListBox}">
    <TextBlock Text="{Binding Nom}" Foreground="Black" FontSize="24"/>
```

Dans un modèle d'affichage de données (`DataTemplate`), le contexte de données correspond à l'objet représenté par le modèle. Dans l'exemple précédent, on peut préciser un modèle de données pour les éléments de la liste ; à l'intérieur de ce modèle, le contexte de données sera l'instance de `Personne` à afficher :

```
<ListBox x:Name="TheListBox"
   ItemsSource="{Binding Joueurs, ElementName=ThePage}">
   <ListBox.ItemTemplate>
      <DataTemplate>
         <StackPanel Orientation="Vertical">
            <TextBlock Text="{Binding Nom}" FontWeight="Bold"/>
            <TextBlock Text="{Binding Points}"
                       FontStyle="Italic" Margin="8,0,0,0"/>
         </StackPanel>
      </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox >
```

Liaison de listes

Outre les trois modes d'alimentation du contexte de données (explicite, hérité, modèle), l'exemple précédent exploite aussi la liaison de données sur un type non élémentaire, c'est-à-dire une liste d'objets : la propriété `Joueurs` de la page sert à alimenter la propriété `ItemsSource` de la `ListBox`.

Il existe différents contrôles qui possèdent une propriété `ItemsSource`. Dans tous les cas, il s'agit de contrôles capables d'afficher une liste d'éléments : `ItemsControl`, `ListBox`, `ListView`, `ComboxBox`, etc. La valeur affectée à cette propriété doit être un type énumérable, pour réaliser la liaison initiale et afficher les éléments correspondants ; néanmoins, si ce type est trop simpliste, le composant graphique sera rempli avec le contenu initial de l'énumération et ne reflétera pas les mises à jour ultérieures.

Copyright © 2013 Eyrolles.

Pour obtenir un comportement plus dynamique, il faut que la collection d'éléments passée à la propriété `ItemsSource` implémente l'interface `INotifyCollectionChanged` : celle-ci permet alors au moteur de liaison de s'abonner à l'événement `CollectionChanged` pour mettre à jour les éléments affichés lors des modifications de la liste.

La classe générique `ObservableCollection<T>` est idéale pour stocker des listes d'objets à afficher car elle implémente cette interface. Par exemple, tout appel à sa méthode `Add` se traduira par l'émission de l'événement `CollectionChanged` avec la valeur `NotifyCollectionChangedAction.Add` comme propriété `Action` du paramètre associé à l'événement.

Les énumérations simples comme `IEnumerable<T>`, les tableaux ou les listes sont aussi affichables dans les contrôles de type liste. Néanmoins, leurs modifications ne seront pas prises en compte par le moteur de liaison.

COMPRENDRE DataTemplate

Lorsqu'un ensemble d'objets est affiché dans un contrôle XAML, il est souvent nécessaire de passer par un modèle (template) pour décrire comment ces objets doivent être affichés.
Par exemple : les éléments propres au contrôle parent, comme une barre de défilement, sont uniques et peuvent être décrits directement dans le XAML, mais ceux qui affichent les objets de la liste doivent être répétés autant de fois que celle-ci contient d'éléments. C'est pour cette raison qu'il est nécessaire de passer par la définition d'un modèle, appelé `DataTemplate`. Celui-ci est instancié autant de fois qu'il y a d'éléments dans la liste et le contexte de données de chacune de ces instances correspond alors à un élément de la liste. À l'intérieur de chaque instanciation du `DataTemplate`, les mécanismes d'alimentation du contexte de données de chaque élément XAML s'appliquent normalement (y compris s'il contient lui-même des listes).

Beaucoup d'applications Windows Store présentent des listes d'éléments regroupés selon certains critères, comme dans l'écran suivant.

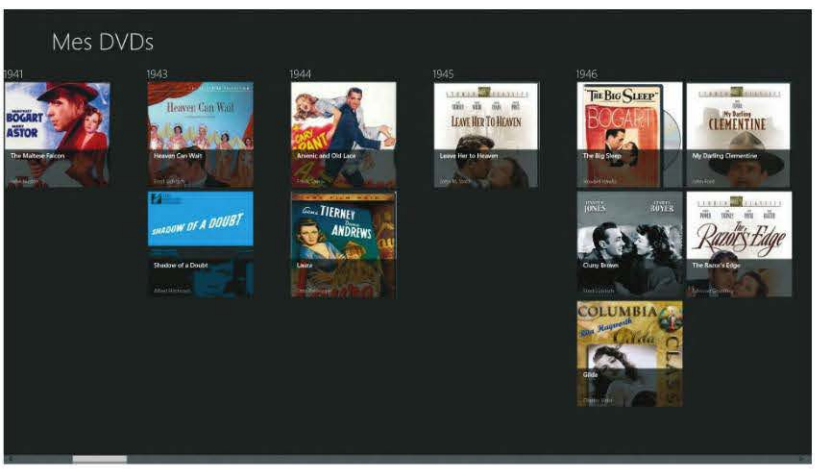


Figure 3–4
Affichage dans une grille

Copyright © 2013 Eyrolles.

Ce type d'affichage se réalise en plusieurs temps. Tout d'abord, le modèle de données doit représenter cette hiérarchie à plusieurs niveaux, par exemple une liste de genres cinématographiques et, pour chaque genre, une liste de films. Une possibilité est d'utiliser une requête LINQ pour rassembler des entités provenant d'une énumération simple dans des groupes correspondant au critère désiré. L'écran de l'exemple, qui groupe les films par année de sortie, peut ainsi être obtenu à l'aide de la requête LINQ suivante :

```
MovieList liste =…
var result = from m in liste
group m by m.Year into g
orderby g.Key
select g;
```

Ensuite, le contrôle XAML `GridView` est lié à un objet `CollectionViewSource`. Par exemple :

```
<Page.Resources>
<CollectionViewSource x:Name="groupedItemsViewSource" IsSourceGrouped="true"/>
…
</Page.Resources>
…

<GridView …
ItemsSource="{Binding Source={StaticResource groupedItemsViewSource}}"
```

Dans le cas où le modèle possède deux niveaux, il faut indiquer à l'objet `CollectionViewSource` la propriété de premier niveau qui retourne la liste des objets de second niveau. Il s'agit de la propriété `movies` dans l'exemple qui suit :

```
<Page.Resources>
<CollectionViewSource x:Name="groupedItemsViewSource" IsSourceGrouped="true"
ItemsPath="movies"/>
…
```

Contrairement à ce que l'exemple laisse penser, les groupes d'objets affichés dans une `GridView` n'ont pas nécessairement besoin d'être homogènes : un Outlook version Windows Store pourrait par exemple afficher un premier groupe d'éléments correspondant aux prochains rendez-vous, puis un groupe avec les messages non lus, etc.

Pour réaliser un tel affichage hétérogène, il est possible de se baser sur une approche totalement non typée dans laquelle on ne stocke que des collections d'objets, mais ceci est une source d'erreurs qui ne sont pas détectées à la compilation ; par exemple, le compilateur ne peut pas empêcher le développeur de stocker un message au milieu d'une liste de rendez-vous. Une approche plus satisfaisante consiste à se baser sur des

Copyright © 2013 Eyrolles.

collections génériques et la notion d'héritage : la collection affichée dans chaque groupe devient alors fortement typée, même si elle manipule des objets différents de ceux des collections des autres groupes.

Ensuite, on peut s'appuyer sur le type de chaque élément pour personnaliser la manière dont il est affiché : il suffit pour cela de remplacer la propriété `ItemTemplate` de la `GridView` par la propriété `ItemTemplateSelector` et d'implémenter une classe dérivant de `DataTemplateSelector`. C'est alors celle-ci qui va choisir le modèle à utiliser en fonction du type effectif de l'objet sous-jacent, ou de toute autre propriété qui a du sens.

```
public class BookItemTemplateSelector : DataTemplateSelector
{
    protected override DataTemplate SelectTemplateCore(object item,
                                Windows.UI.XAML.DependencyObject container)
   {

    DataTemplate ret=null;
    if (item is ProductItem)
        {
        ret = App.Current.Resources["Product250x250ItemTemplate"] as
DataTemplate;
        }
    …
    return ret;
    }
}
```

Liaison de données en C++

Les composants WinRT écrits en C++ peuvent aussi être liés aux contrôles XAML, au même titre que les composants écrits en VB.NET ou en C#. Néanmoins, même si le compilateur et les classes de base effectuent un certain nombre d'opérations de bas niveau pour le développeur, il reste quelques spécificités du langage à prendre en compte pour être en mesure de réaliser ces liaisons, car le C++ ne dispose pas de mécanismes d'introspection et d'invocation dynamique comme les langages du Framework .Net.

Tout d'abord, il faut noter que seuls les composants WinRT peuvent être liés à une propriété de contrôle XAML, pas une classe C++ normale.

Ensuite, il convient de rappeler (mais c'est aussi le cas pour les objets C# ou VB.NET) que seules les propriétés sont liées : les champs ne sont pas pris en compte par le mécanisme de liaison de données (on notera que, de toute manière, l'exposition de champs à l'extérieur d'un objet ne constitue pas une bonne pratique de programmation).

Enfin, il faut ajouter l'attribut `[Windows::UI::XAML::Data::Bindable]` aux classes dont on veut lier les propriétés : grâce à cet attribut, le compilateur créera le code donnant au moteur de liaison l'accès aux propriétés de la classe.

Copyright © 2013 Eyrolles.

Voici la définition d'une classe C++ assez simple mais illustrant ces différentes contraintes.

```
[Windows::UI::XAML::Data::Bindable]
public ref class Vin sealed : TestBindingCPP::BindableBase
{
public:
      Vin(Platform::String^ nom, Platform::String^ appellation,
          int millesime, int nbBouteilles);

      property Platform::String^ Nom {
          Platform::String^ get();
          void set(Platform::String^ value);
     }
      property Platform::String^ Appellation {
          Platform::String^ get();
          void set(Platform::String^ value);
      }
      property int Millesime {
          int get(); void set(int value);
      }
      property int NbBouteilles {
          int get(); void set(int value);}
private:
      Platform::String^ _nom;
      Platform::String^ _appellation;
      int _millesime;
      int _nbBouteilles;
};
```

Et voici un aperçu de son implémentation :

```
Vin::Vin(Platform::String^ nom, Platform::String^ appellation,
         int millesime, int nbBouteilles)
{
      nom = nom;
      _appellation = appellation;
      _millesime = millesime;
      _nbBouteilles = nbBouteilles;
}

String^ Vin::Nom::get()
{
      return _nom;
}

void Vin::Nom::set(String^ value)
{
```

Copyright © 2013 Eyrolles.

```
    if (_nom != value)
    {
        _nom = value;
        OnPropertyChanged("Nom");
    }
}
…
```

Habituellement, cette classe hérite d'une classe de base implémentant l'interface `INotifyPropertyChanged`. Cette dernière est nécessaire pour que le moteur de liaison prenne en compte les modifications des propriétés intervenant « côté code » :

```
[Windows::Foundation::Metadata::WebHostHidden]
public ref class BindableBase : Windows::UI::XAML::Data::INotifyPropertyChanged
{
public:
    virtual event Windows::UI::XAML::Data::PropertyChangedEventHandler^
                                                                PropertyChanged;
protected:
    virtual void OnPropertyChanged(Platform::String^ propertyName);
};

void BindableBase::OnPropertyChanged(String^ propertyName)
{
    PropertyChanged(this, ref new PropertyChangedEventArgs(propertyName));
}
```

Une telle classe est utilisable en XAML de la manière habituelle. Dans l'extrait qui suit, la classe `MainWindow` expose une propriété de type `Vin` qui est déclarée comme contexte de données d'un `StackPanel` afin d'avoir ensuite un référencement aisé de ses propriétés à l'intérieur de celui-ci (il est donc nécessaire que `MainWindow` possède aussi l'attribut `Bindable` présenté précédemment).

```
<StackPanel Grid.Row="1" Grid.Column="1"
          HorizontalAlignment="Center" VerticalAlignment="Top"
DataContext="{Binding VinCourant, ElementName=ThePage}"
>
…
    <TextBlock Text="Nom" Margin="0,20,0,0"/>
    <TextBox Width="200" HorizontalAlignment="Left"
            Text="{Binding Nom, Mode=TwoWay}" />
    <TextBlock Text="Millésime" Margin="0,20,0,0" HorizontalAlignment="Left"/>
    <TextBox Width="50" HorizontalAlignment="Left"
             Text="{Binding Millesime, Mode=TwoWay}" />
…
```

Copyright © 2013 Eyrolles.

Dans une vraie application, ce sera plutôt le `ViewModel` qui exposera la propriété `VinCourant`, la fenêtre XAML se contentant de gérer l'affichage à travers la liaison de données.

Pour lier une collection d'objets à un composant XAML comme une `ListBox`, le plus simple est comme en C# d'utiliser une classe générique qui implémente les interfaces attendues par le moteur de liaison de données.

La classe la plus naturelle est alors `Platform::Collections::Vector<T>` qui implémente l'interface `Windows::Foundation::Collections::IObservableVector<T>`.

L'exemple précédent sera ainsi enrichi en laissant choisir l'appellation du vin dans une liste prédéfinie à l'aide d'une `ComboBox`.

- La fenêtre (ou le `ViewModel`) se voit doter d'une propriété indiquant quelles sont les appellations disponibles :

```
// Header
public:
   property Windows::Foundation::Collections::IObservableVector
                                          <Platform::String^>^ Appellations {
    Windows::Foundation::Collections::IObservableVector<Platform::String^>^
get();
}
private:
   Platform::Collections::Vector<Platform::String^>^ _appellations;

// Implémentation
Windows::Foundation::Collections::IObservableVector<Platform::String^>^
    MainPage::Appellations::get()
{
    return _appellations;
}
```

- La `ComboBox` insérée dans le `StackPanel` précédent fait l'objet deux liaisons de données, vers deux sources différentes. D'une part, sa propriété `ItemsSource` est liée à la propriété `Appellations` de la fenêtre pour afficher l'ensemble des appellations disponibles (lecture seule). D'autre part, sa propriété `SelectedItem` est liée à la propriété `Appellation` du contexte de données courant, donc de la propriété `VinCourant` de la fenêtre (lecture et écriture) :

```
<ComboBox HorizontalAlignment="Left" Width="200"
          ItemsSource="{Binding Appellations, ElementName=ThePage}"
          SelectedItem="{Binding Appellation, Mode=TwoWay}"/>
```

Copyright © 2013 Eyrolles.

Liaison de données en HTML5 et JavaScript

Les mécanismes de liaison de données disponibles pour les applications Windows Store écrites en HTML5 et JavaScript sont assez différents des mécanismes présentés précédemment, puisque les technologies de présentation et d'implémentation des classes sont toutes les deux remplacées (les moteurs HMTL5 et XAML sont deux composants bien distincts).

Cette section va présenter les mécanismes de base, c'est-à-dire accessibles sans utilisation de bibliothèque complémentaire. Néanmoins, il faut noter que l'aspect dynamique de JavaScript autorise la création ou l'enrichissement de classes à la volée et des bibliothèques tierces enrichissant les mécanismes décrits ici sont ou seront disponibles rapidement.

Les paragraphes suivants illustrent la liaison d'un objet `Vin` défini comme suit dans une page HTML :

```
(function () {
    "use strict";

    WinJS.Namespace.define("Cave", {

        Vin: function (n, a, c) {
            this.nom = n;
            this.appellation = a;
            this.couleur = c;
        },
…
    });
})();
```

La liaison la plus simple consiste à afficher dans un objet HTML la valeur d'une propriété d'un objet JavaScript. Pour cela, il faut :

- ajouter l'attribut `data-win-bind` à l'élément HTML :

```
<input type="text" id="appellationTextBox1"
      data-win-bind="innerText:appellation" />
```

- puis appeler la méthode JavaScript `WinJS.Binding.processAll` pour lier les deux objets :

```
var vin1 = new Cave.Vin("Amiral de Beychevelle", "Saint-Julien", "rouge");
var ctl1 = document.getElementById("appellationTextBox1");
WinJS.Binding.processAll(ctl1, vin1);
```

Copyright © 2013 Eyrolles.

Avec ce code, la liaison n'a lieu qu'une seule fois, lors de l'appel à `processAll` : ni les modifications de la propriété `appellation` effectuées en JavaScript, ni les saisies réalisées dans le champ HTML ne sont transmises à l'autre partie de la liaison. Une telle liaison présente donc assez peu d'intérêt par rapport à une affectation manuelle de la propriété cible, puisqu'elle demande à la fois l'ajout d'un attribut à l'objet HTML et une exécution de code JavaScript. Néanmoins, c'est en enrichissant ce mécanisme de base qu'il est possible de mettre en place une liaison complète.

Tout d'abord, l'attribut `data-win-bind` va être complété :

- il peut contenir plusieurs paires propriété HTML/propriété JavaScript, ce qui est typiquement utilisé quand la mise en forme du contrôle dépend aussi des données affichées, comme un affichage en rouge d'une valeur erronée ;
- il est possible de fournir une fonction de conversion. Le principe est le même que pour la liaison de données XAML/C# : par exemple, le fait que la valeur d'une propriété soit erronée concerne le modèle, mais le fait qu'on veuille alors afficher cette valeur en rouge est une considération de présentation, qui doit donc être prise en charge à ce niveau. La fonction de conversion appartient donc plutôt à la couche de présentation.

  Cette fonction est simple : elle prend pour paramètre la valeur de la propriété du modèle et retourne la valeur à attribuer à la propriété de l'objet HTML :

```
(function () {

    "use strict";

    WinJS.Namespace.define("Convertisseurs", {

        couleurToImageConverter: WinJS.Binding.converter(function (c) {
            if (c == "blanc") {
                return "/images/blanc.png";
            } else {
                return "/images/rouge.png";
            }
        })
    });

})();
```

Ensuite, le nom de cette fonction est ajouté à l'attribut `data-win-bind` :

```
<img id="couleurImage3"
    data-win-bind="src:couleur Convertisseurs.couleurToImageConverter" />
```

Copyright © 2013 Eyrolles.

Les objets JavaScript n'ont pas de mécanisme intrinsèque de notification des changements de leurs propriétés : c'est tout à fait comparable au C# où les propriétés « simples » ne suffisent pas à la liaison de données et où il faut implémenter explicitement l'interface `INotifyPropertyChanged` pour que le moteur de liaison XAML mette l'interface à jour.

Les bibliothèques JavaScript de WinRT contiennent l'infrastructure nécessaire à ces notifications : la méthode `as` crée et retourne une encapsulation de la classe passée en paramètre qui est compatible à la fois avec celle-ci (i.e. elle implémente les mêmes membres) et avec la liaison de données, en signalant ses changements d'une manière compatible avec le moteur de liaison de HTML 5.

La mise en place de la liaison devient alors :

```
var vin3 = new Cave.Vin("Château Larcis Ducasse", "Saint-Emilion grand cru",
                        "rouge");
var obs3 = WinJS.Binding.as(vin3);
var ctl3 = document.getElementById("appellationTextBox3");
WinJS.Binding.processAll(ctl3, obs3);
ctl3 = document.getElementById("couleurImage3");
WinJS.Binding.processAll(ctl3, obs3);
```

Figure 3–5
Liaison JavaScript

Le code modifiant la propriété est alors « normal » :

```
obs3.appellation = "Bordeaux";
obs3.couleur = "blanc";
```

Figure 3–6
Liaison JavaScript modifiée

Ici, on atteint un peu les limites de la liaison de données en JavaScript. En effet, pour avoir une liaison à double-sens, c'est-à-dire qui répercute sur l'objet du modèle les modifications effectuées sur un objet de l'interface, il faut écrire une méthode abonnée aux événements signalant les changements réalisés via l'interface utilisateur

Copyright © 2013 Eyrolles.

(i.e. l'événement `change` pour un bon nombre d'éléments HTML) et qui affecte explicitement la nouvelle valeur à la propriété du modèle :

```
var vin4 = new Cave.Vin("Clos des Porrets-St-Georges",
                        "Nuits St-Georges 1er cru", "rouge");
var obs4 = WinJS.Binding.as(vin4);
var ctl4 = document.getElementById("appellationTextBox4");
WinJS.Binding.processAll(ctl4, obs4);
ctl4.addEventListener("change",
                      function (evt){obs4.appellation = evt.target.value; });
```

Comme le montre cet exemple, les fonctions anonymes sont d'un grand secours afin de ne pas polluer le code par la multiplication de fonctions ne contenant qu'une seule instruction.

Pour finir ce tour d'horizon de la liaison de données en JavaScript, voici comment afficher une liste d'objets en répétant un modèle graphique pour chaque objet. Cette opération nécessite trois composants :

- tout d'abord, une collection d'objets JavaScript. Pour que le modèle graphique puisse être répété, il faut bien évidemment que cette collection soit homogène et que tous ses éléments disposent des propriétés qui seront liées :

```
<script type="text/javascript">
var listeVins = [
   WinJS.Binding.as(new Cave.Vin('Gewurztraminer vendanges tardives',
                                 'Alsace',
                                 'blanc')),
   WinJS.Binding.as(new Cave.Vin('Grand cru Muenchberg',
                                 'Alsace grand cru',
                                 'blanc')),
   WinJS.Binding.as(new Cave.Vin('Château Les Tours de Laroche',
                                'Côtes de Bourg',
                                 'rouge')),
   WinJS.Binding.as(new Cave.Vin('Château Ferrande',
                                 'Graves',
                                 'rouge')),
…
```

- ensuite, il faut définir le modèle qui sera répété pour chaque élément de la liste. Il s'agit d'un élément HTML décoré par l'attribut `data-win-control="WinJS.Binding.Template"` indiquant que cet élément ne doit pas être affiché directement dans la page, mais sera instancié par ailleurs :

```
<table id="templateDiv" data-win-control="WinJS.Binding.Template">
<tr>
  <td>
```

Copyright © 2013 Eyrolles.

```
    <img data-win-bind="src:couleur Convertisseurs.couleurToImageConverter" />
  </td>
  <td>
    <span data-win-bind="textContent: nom" style="font-weight:bold"></span>
    <br />
    <span data-win-bind="textContent: appellation"></span>
  </td>
</tr>
</table>
```

- enfin, il faut insérer dans la page un objet `div` qui représente l'endroit où la liste d'objets sera affichée par expansion du modèle défini précédemment :

```
<div id="templateControlRenderTarget"></div>
```

Une fois ces trois éléments disponibles, l'expansion des modèles se fait à l'aide de la méthode `render` appelée pour chaque objet de la liste à afficher :

```
// Modèle à appliquer aux éléments de la liste
var templateElement = document.getElementById("templateDiv");
// Point d'ancrage
var renderHere = document.getElementById("templateControlRenderTarget");
renderHere.innerHTML = "";

// Parcours de la liste et expansion des modèles
WinJS.UI.process(templateElement).then(function (templateControl) {
    listeVins.forEach(function (v) {
       templateControl.render(v, renderHere);
   });
});
```

Le résultat de ce code est présenté sur la figure suivante.

Figure 3–7
Liste liée en JavaScript

Copyright © 2013 Eyrolles.

Accès aux données

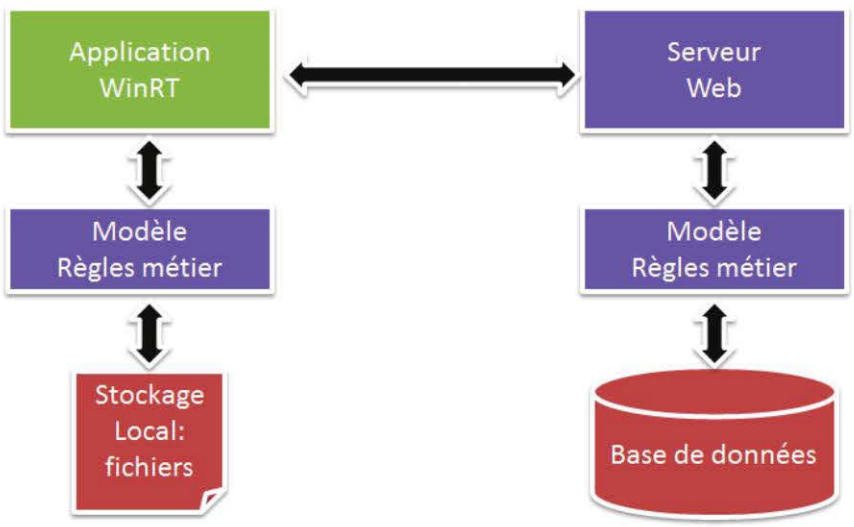
Les sections précédentes ont décrit comment définir et afficher un modèle de données au sein d'une application Windows Store. Ces données peuvent provenir de différentes sources : il s'agit par exemple de fichiers dans le cas d'un tableur, ou d'une base de données comme dans la plupart des applications de gestion. Pour des raisons essentiellement liées à la sécurité, les applications Windows Store imposent certaines contraintes qui n'existent pas dans l'environnement de bureau classique.

En termes de fichiers, une telle application peut accéder uniquement à un espace qui lui est propre, ainsi qu'aux bibliothèques de documents de l'utilisateur à condition d'avoir déclaré préalablement cet accès dans le fichier manifeste de l'application. L'accès aux autres répertoires, `C:\Windows` par exemple, lui est interdit.

Il n'y a pas aujourd'hui d'équivalent des classes ADO.NET dans l'interface de programmation WinRT : cela signifie qu'il n'est pas possible d'écrire des applications de type client-serveur au sens classique (accès direct à une base de données telle que SQL Serveur) et qu'il faut se diriger soit vers des bibliothèques tierces, soit vers des architectures orientées services.

En pratique, il n'y a pas de vrai blocage car ce dernier type d'architecture, basé uniquement sur des communications HTTP, a depuis plusieurs années remplacé les architectures à deux niveaux.

Figure 3–8
Modèle d'architecture



Copyright © 2013 Eyrolles.

Lecture et écriture dans un fichier

Principes de base

Dans l'ensemble, les entrées et sorties fichiers sont assez proches de ce qui existe depuis de nombreuses années dans le framework .NET. Les principales différences sont liées à la sécurité et au caractère asynchrone des méthodes d'accès aux fichiers.

Les premières classes à manipuler pour accéder aux fichiers sont proches, mais néanmoins différentes des classes de .NET : elles s'appellent `StorageFolder` et `StorageFile` qui, comme l'indiquent leurs noms, correspondent respectivement aux répertoires et aux fichiers.

Les principaux répertoires accessibles à une application Windows Store sont :

- des répertoires de données propres à l'application : un répertoire de stockage local à la machine, un répertoire de stockage dont l'utilisateur retrouvera le contenu sur les différentes machines sur lesquelles il installera l'application, un répertoire temporaire dont le contenu sera régulièrement vidé ;
- les répertoires classiques d'une installation Windows, comme celui des documents de l'utilisateur.

Une référence sur un `StorageFolder` représentant un des répertoires du premier groupe s'obtient à l'aide de la classe `ApplicationData` via le singleton `Current` qui expose les propriétés `LocalFolder`, `TemporaryFolder` et `RoamingFolder`.

Une référence sur un `StorageFolder` représentant un des répertoires classiques de Windows s'obtient par exemple à l'aide de la classe `KnownFolders` :

- `KnownFolders.DocumentsLibrary` représente le répertoire contenant les documents de l'utilisateur courant.
- `KnownFolders.PicturesLibrary` représente le répertoire contenant les images de l'utilisateur courant.
- `KnownFolders.HomeGroup` donne accès aux documents du groupe résidentiel.

Pour ces répertoires, il faut déclarer la capacité correspondante dans le manifeste de l'application. IntelliSense aide le développeur ici car cette capacité est indiquée dans l'info-bulle associée à la propriété statique qu'elle documente, comme le montre la figure suivante.

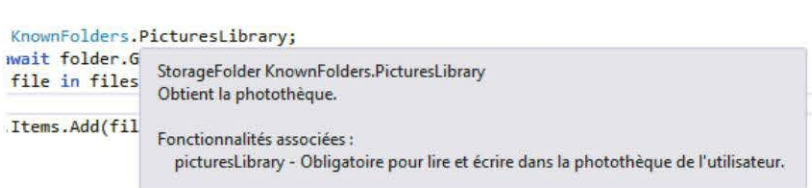


Figure 3–9
IntelliSense et indication des capacités nécessaires

Copyright © 2013 Eyrolles.

Si cette capacité n'est pas déclarée, une exception `Access Denied` sera levée dès l'appel à la propriété statique. Il suffit de cocher la case correspondante après avoir ouvert le fichier `Package.appxmanifest` pour éviter cette exception.

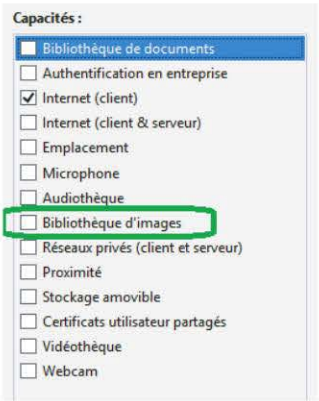


Figure 3–10
Accès à la bibliothèque d'images

Il est aussi possible d'accéder à un répertoire à partir de son nom à l'aide de la fonction `StorageFolder.GetFolderFromPathAsync`.

Une fois une instance de la classe `StorageFolder` récupérée, les méthodes de cette classe permettent d'accéder aux sous-répertoires, aux fichiers ou encore de lire les propriétés du répertoire représenté par cette instance.

Trois groupes de méthodes présentent des fonctions similaires : celles dont le nom contient `File` travaillent sur les fichiers, celles dont le nom contient `Folder` sur les répertoires et celles dont le nom contient `Item` accèdent indifféremment aux deux types d'entités. Voici par exemple les principales méthodes permettant d'obtenir une ou plusieurs instances de la classe `StorageFile` :

- `GetFilesAsync` sert à énumérer les fichiers du répertoire. Appelée sans paramètres, elle retourne tous les fichiers du répertoire, mais il est possible d'appeler une surcharge acceptant un paramètre de type `CommonFileQuery` pour incorporer les fichiers situés dans des sous-répertoires.
- `CreateFileQueryWithOptions` exécute une requête de recherche plus sophistiquée, en indiquant par exemple les extensions de fichiers à prendre en compte. Le code suivant n'énumère ainsi que les fichiers avec l'extension `.jpg` :

```
StorageFolder folder = KnownFolders.PicturesLibrary;
string[] suffixes = { ".jpg" };
var files = await folder.CreateFileQueryWithOptions(
                          new QueryOptions(CommonFileQuery.OrderByName,
                                           suffixes)).GetFilesAsync();
```

- `CreateFileAsync` crée un fichier.

Copyright © 2013 Eyrolles.

Avant de présenter différentes techniques de lecture et d'écriture de fichiers, il faut noter au passage que les classes `StorageFolder` et `StorageFile` donnent accès aux propriétés des objets manipulés : non seulement aux propriétés classiques, comme la taille ou la date de dernière modification, mais aussi à des métadonnées incorporées dans les en-têtes d'un certain nombre de fichiers usuels.

Par exemple, le code qui suit obtient quelques métadonnées associées à une image JPEG, comme la date de prise de vue, le modèle d'appareil-photo ou encore la géolocalisation de la prise de vue, sans avoir à ouvrir et décoder le fichier. Il faut bien évidemment que l'appareil-photo ayant enregistré cette image ait inséré les métadonnées correspondantes dans celle-ci.

```
StorageFolder folder = KnownFolders.PicturesLibrary;
var file = await folder.GetFileAsync("WP_000008.jpg");
var basicProps = await file.GetBasicPropertiesAsync();
var otherProps = await file.Properties.GetImagePropertiesAsync();
var datePriseDeVue = otherProps.DateTaken;
var modele = otherProps.CameraModel;
var latitude = otherProps.Latitude;
var longitude = otherProps.Longitude;
```

Lire ou écrire les données d'une application dans un fichier s'effectue par l'intermédiaire d'un objet de type flux (`Stream`) obtenu à partir d'un `StorageFile` et d'un appel à une des méthodes suivantes : `OpenAsync`, `OpenReadAsync`, `OpenSequentialReadAsync`, `OpenStreamForReadAsync`, `OpenStreamForWriteAsync`, `OpenTransactedWriteAsync`.

Les flux s'emploient de différentes manières. En utilisant les différentes méthodes de la classe `Stream` ou en s'appuyant sur un `StreamWriter`, il est aisé de contrôler précisément le format du fichier écrit.

Ainsi, cette approche est particulièrement intéressante lorsque ce contenu est « standardisé » et que le fichier est destiné à être échangé avec d'autres applications : outre les classiques formats multimédia des images, il peut par exemple s'agir de feuilles de calcul ou de parties d'échecs.

Toutefois, beaucoup de méthodes des classes standards acceptent des objets `Stream` en paramètre et effectuent des opérations de plus grande portée, comme la sérialisation en un seul appel d'une grappe d'objets. Cette approche est utile lorsque le développeur a besoin de sauvegarder et de restaurer un graphe d'objets, mais ne s'intéresse en revanche pas au format précis du fichier ainsi créé.

Utilisation de fichiers textes

L'exemple suivant illustre l'écriture d'un fichier PGN (*Portable Game Notation*, format de fichier utilisé fréquemment sur Internet pour échanger des parties d'échecs) en enchaînant les différentes étapes évoquées précédemment :

Copyright © 2013 Eyrolles.

- accès au répertoire de documents de l'utilisateur → `StorageFolder` ;
- création du fichier → `StorageFile` ;
- création d'un flux en écriture → `Stream` ;
- utilisation d'un `StreamWriter` dans la mesure où il s'agit d'un fichier texte.

Pour que cet exemple s'exécute correctement, il faut que le programme correspondant ait indiqué dans son manifeste les accès aux fichiers dont il a besoin.

Comme décrit précédemment pour les images, il faut demander l'accès à la bibliothèque de documents de l'utilisateur dans la section *Capacités* du fichier manifeste de l'application. Cela n'est pas suffisant : une application Windows Store n'a le droit d'accéder qu'aux types de fichiers déclarés dans la section *Déclarations* de son fichier manifeste : dans *Association de Type de Fichier*, il faut ajouter le type `.pgn` dans la liste des types de fichiers acceptés.

Sauvegarde d'un fichier sur le disque dur

```
class PgnFile
{
    public async void SaveGame(string fileName, Game game)
    {
        StorageFolder folder = KnownFolders.DocumentsLibrary;
        StorageFilefile = await folder.CreateFileAsync(fileName);
        using (Stream stream = await file.OpenStreamForWriteAsync())
        using (StreamWriter writer = new StreamWriter(stream,
                                                      Encoding.UTF8))
        {
            writer.WriteLine("[Event \"{0}\"]", game.Event);
            writer.WriteLine("[Site \"{0}\"]", game.Site);
            writer.WriteLine("[Date \"{0:dd.MM.yyyy}\"]", game.Date);
            writer.WriteLine("[Round \"{0}\"]",game.Round);
            writer.WriteLine("[White \"{0}\"]", game.WhitePlayer);
            writer.WriteLine("[Black \"{0}\"]", game.BlackPlayer);
            writer.WriteLine("[Result \"{0}\"]", game.Result);
            foreach (var move in game.Moves)
            {
                writer.Write(move);
            }
            writer.WriteLine(game.Result);
            writer.WriteLine();
        }
    }
…
}
```

Il est à noter que les déclarations des variables `stream` et `writer` sont effectuées à l'intérieur d'une instruction `using`. Cela signifie que, quelle que soit la manière dont

Copyright © 2013 Eyrolles.

on sortira du bloc correspondant, il sera fait appel à la méthode `Dispose` de l'interface `IDisposable` que les classes `Stream` et `StreamWriter` implémentent. On s'assure ainsi que toutes les écritures dans le fichier seront finies à la sortie de ce bloc :

- le fichier n'est pas verrouillé plus longtemps que nécessaire ;
- même en cas de sortie intempestive du programme, le fichier est complet.

Accès simplifié : la classe FileIO

Parfois, une application manipule en mémoire un objet peu structuré, comme une simple liste de chaînes de caractères ou un tableau d'octets (représentant par exemple une image JPEG). Dans ce cas, le chargement et la sauvegarde de cet objet sont réalisables en une seule opération d'entrée/sortie avec la classe `FileIO`. Celle-ci expose un ensemble de méthodes statiques pour des accès simples, présentées dans le tableau suivant.

Tableau 3–2 Méthodes statiques de la classe FileIO

Méthode	Description
`WriteBufferAsync`	Écrit un fichier à partir d'un tampon d'octets.
`WriteLinesAsync`	Écrit un fichier à partir d'un ensemble de lignes représentées par une énumération de chaînes de caractères.
`WriteTextAsync`	Écrit un fichier à partir d'une unique chaîne de caractères.
`AppendLinesAsync`	Ajoute à un fichier un ensemble de lignes représentées par une énumération de chaînes de caractères.
`AppendTextAsync`	Ajoute à un fichier une unique chaîne de caractères.
`ReadBufferAsync`	Remplit un tampon à partir du contenu d'un fichier.
`ReadLinesAsync`	Lit le contenu d'un fichier texte sous la forme d'une énumération de chaînes de caractères.
`ReadTextAsync`	Lit le contenu d'un fichier texte en une seule chaîne de caractères.

Ces méthodes ouvrent, écrivent (ou lisent) puis referment le fichier. Appeler de manière répétée `AppendTextAsync` pour ajouter un ensemble de chaînes à un fichier constitue une mauvaise pratique en termes de performances (ainsi que du point de vue de la gestion des accès concurrents) et il est donc préférable d'utiliser les mécanismes présentés précédemment pour n'ouvrir le fichier qu'une seule fois avant d'y effectuer des écritures répétées puis de le fermer.

Utilisation d'un fichier XML

Pour enregistrer des données structurées dans un format propre à l'application, XML reste une des options les plus courantes car il est particulièrement bien adapté à cette tâche. Il y a plusieurs manières de s'appuyer sur ce format :

Copyright © 2013 Eyrolles.

- manipuler en mémoire un objet de type `XmlDocument` ;
- utiliser les classes `XmlReader` et `XmlWriter` pour lire et écrire le fichier élément par élément ;
- s'appuyer sur les mécanismes de sérialisation .NET pour écrire ou lire en une seule opération un modèle de données en mémoire.

Un `XmlDocument` se lit typiquement à partir de méthodes statiques de la classe. Il peut être récupéré localement ou depuis un serveur en utilisant le protocole HTTP :

```
StorageFolder folder = KnownFolders.DocumentsLibrary;
StorageFile file = await folder.GetFileAsync("MaCave.xml");

XmlDocument caveDoc = await XmlDocument.LoadFromFileAsync(file);

Uri refUri = new Uri(".../referentiel.xml");
XmlDocument referentielDoc = await XmlDocument.LoadFromUriAsync(refUri);
```

Les classes `XmlReader` et `XmlWriter` travaillent sur des `Streams` :

```
using (Stream stream = await file.OpenStreamForReadAsync())
using (XmlReader reader = XmlReader.Create(stream))
{
    while (reader.Read())
    {
    …
    }
}
```

En .NET, il existe trois manières de sérialiser un modèle de données constitué d'instances se référençant les unes aux autres. Elles sont toutes disponibles pour les applications Windows Store. Chacune a ses avantages et ses inconvénients.

- La sérialisation XML contrôle finement la forme du document XML produit ; elle est à privilégier lorsque les fichiers créés sont échangés avec d'autres applications et doivent respecter un schéma précis.
- La sérialisation JSON est plus compacte que la précédente. Elle présente peu d'intérêt pour des sauvegardes autonomes, mais est en revanche très intéressante dans le cas d'échanges avec des serveurs web exposant des services REST.
- La sérialisation XML à l'aide du `DataContractSerializer` fait partie de la bibliothèque de communication WCF, mais est utilisable pour des sauvegardes locales. Le document XML produit n'est pas toujours facile à lire, mais cette sérialisation offre une possibilité unique : sérialiser des graphes d'objets contenant des cycles.

  Beaucoup de modèles objets contiennent par exemple des références réciproques entre entités. C'est le cas dans une relation parent-enfant : par exemple, une

Copyright © 2013 Eyrolles.

région vinicole va contenir une collection d'`Appellation` et chaque `Appellation` va à son tour contenir une référence vers sa région vinicole.

La sérialisation à l'aide d'un `DataContractSerializer` s'assure que les entités ne sont écrites qu'une seule fois dans le fichier (lors de leur première occurrence). C'est donc un mode de sauvegarde très pratique pour toutes les applications manipulant de nombreuses entités reliées entre elles : comme il n'y a pas de système de bases de données dans les API standards WinRT, il est possible (si la quantité de données est raisonnable) de charger en mémoire un référentiel assez riche fonctionnellement, puis de l'interroger à l'aide de requêtes LINQ ou d'en parcourir les relations.

Les trois sérialisations s'utilisent sensiblement de la même manière : obtention d'un `Stream` sur le fichier à lire ou écrire, création d'une classe de sérialisation en passant à son constructeur le type de l'objet racine du modèle à sauvegarder, puis appel d'une méthode de sérialisation pour l'écriture ou de désérialisation pour la lecture. Voici par exemple le code de sauvegarde et de restauration d'un objet de la classe `Cellar` à l'aide d'un `DataContractSerializer` :

```
async void SaveCellarAsXmlWithDataContractSerializer(string fileName,
                                                     Cellar cellar)
{
    StorageFolder folder = KnownFolders.DocumentsLibrary;
    StorageFile file = await folder.CreateFileAsync(fileName);
    DataContractSerializer ser = new DataContractSerializer(typeof(Cellar));
    using (Stream stream = await file.OpenStreamForWriteAsync())
    {
        ser.WriteObject(stream, cave);
    }
}

async Task<Cellar> LoadCellarAsXmlWithDataContractSerialiser(string fileName)
{
    StorageFolder folder = KnownFolders.DocumentsLibrary;
    StorageFile file = await folder.GetFileAsync(fileName);
    DataContractSerializer ser = new DataContractSerializer(typeof(Cellar));
    using (Stream stream = await file.OpenStreamForReadAsync())
    {
        return ser.ReadObject(stream) as Cellar;
    }
}
```

Le code pour les deux autres méthodes de sérialisation est quasiment identique. Pour la sérialisation JSON, il suffit de remplacer `DataContractSerializer` par `DataContractJsonSerializer`. Pour la sérialisation XML simple, il faut remplacer `DataContractSerializer` par `XmlSerializer`, `WriteObject` par `Serialize` et `ReadObject` par `Deserialize`, mais la structure du code reste identique.

Copyright © 2013 Eyrolles.

Accès à des données distantes

Comme indiqué précédemment, une application Windows Store ne dispose pas d'un ensemble de classes comme ADO.NET pour accéder directement à un serveur de bases de données ; il est nécessaire de passer par l'intermédiaire de services web au sens large, c'est-à-dire de services accessibles via HTTP, quel que soit le protocole utilisé au-dessus (SOAP, JSON…).

Les types de services utilisés couramment dans l'environnement .NET sont pour la plupart disponibles pour les applications Windows Store. La principale différence avec une application ciblant le bureau classique est que seuls les appels asynchrones sont disponibles. Cependant, c'était déjà une caractéristique rencontrée avec les applications Silverlight.

Les sections suivantes vont présenter trois cas d'utilisation de services WCF assez courants dans les applications .NET.

> Les services WCF n'ont à subir aucune modification pour être appelés par les applications Windows Store ; leurs implémentations ne sont donc pas décrites ici. Seul l'aspect client sera détaillé.

Service WCF simple

L'interface suivante définit un service WCF très simple :

```
[ServiceContract]
public interface IArithmetics
{
   [OperationContract]
   bool IsPrime(long n);
}
```

Pour accéder à ce service depuis une application Windows Store, le plus direct est d'utiliser la boîte de dialogue d'ajout de référence de service (appelée en cliquant-droit sur le projet et en choisissant l'option *Ajouter une Référence de Service* dans le menu contextuel) et de préciser l'URL du fichier `.svc` représentant le service (si ce dernier est défini dans le même projet, le bouton *découvrir* y donnera accès sans lancer le serveur correspondant).

Visual Studio crée alors les classes donnant accès au service de la manière habituelle. Néanmoins, comme les applications Windows Store ne peuvent effectuer que des appels HTTP en mode asynchrone, les classes créées n'exposent que des méthodes de ce type. Par ailleurs, elles s'appuient sur le modèle basé sur les mots-clés `async/await` de programmation asynchrone de .NET. La figure qui suit illustre le fait que seule la méthode `IsPrimeAsync` est exposée par le proxy créé automatiquement par Visual Studio.

Copyright © 2013 Eyrolles.

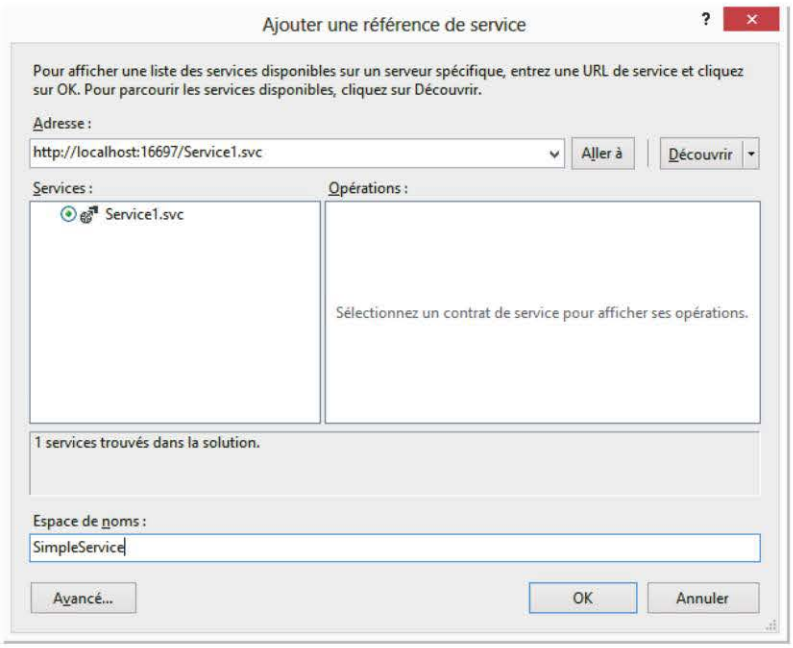


Figure 3–11
Ajout d'une référence vers un service WCF

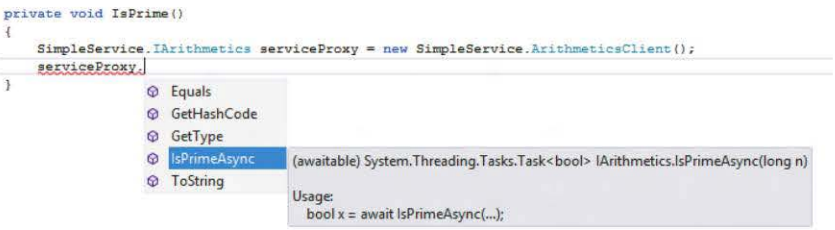


Figure 3–12
Seules les versions asynchrones des opérations WCF sont accessibles.

La méthode cliente réalisant l'appel du service et la mise à jour de l'interface est donc sans surprise :

```
async private void IsPrime()
{
   SimpleService.IArithmetics serviceProxy =
                                    new SimpleService.ArithmeticsClient();
   Result = string.Format(await serviceProxy.IsPrimeAsync(N)
                        ? "{0} est premier" : "{0} est composé", N);
}
```

Utilisation d'une bibliothèque d'interface

WCF permet la création de services accessibles via des protocoles standards et qui peuvent donc appelés par des clients divers, et en particulier non Windows. Dans le cas des applications Windows en général et WinRT en particulier, les couches tech-

Copyright © 2013 Eyrolles.

niques de .NET redonnent un aspect orienté objet au proxy appelant le service : les mêmes entités (celles identifiées par l'attribut `[DataContract]`) sont alors disponibles sur le serveur et le client.

Si le proxy est entièrement créé par Visual Studio, les entités ne sont pas tout à fait identiques : elles exposent les mêmes données membres (celles identifiées à l'aide de l'attribut `[DataMember]`), mais les entités produites pour le proxy ne possèdent aucune méthode.

Dans beaucoup de cas, les entités créées par Visual Studio sont suffisantes au développement de l'application cliente. Néanmoins, dans un certain nombre de scénarios, le partage de code entre services web et client est intéressant : c'est par exemple le cas des propriétés calculées.

Bien qu'une application Windows Store et un service WCF ne s'exécutent pas tout à fait dans le même environnement .NET, il est possible de partager du code entre les deux ; un mécanisme va même plus loin qu'un simple partage de code source et va jusqu'à mettre des DLL en commun.

Pour partager les entités utilisées dans l'interface d'un service WCF entre celui-ci et ses clients, il faut donc commencer par créer un projet de type « bibliothèque de classes portable » (*Portable Class Library*) et sélectionner les deux types d'environnements correspondant au service et à l'application Windows Store.

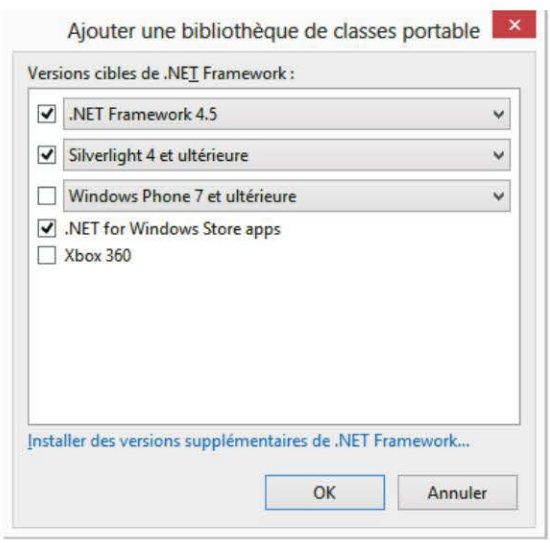


Figure 3–13
Création d'une bibliothèque portable

La deuxième étape consiste à déclarer et implémenter les entités à partager dans cette DLL. Comme elle a par construction vocation à être portable dans différents environnements (dont la liste est visible dans la figure précédente), tous les types .NET ne sont pas nécessairement disponibles.

Copyright © 2013 Eyrolles.

Néanmoins, dans le cas d'un partage entre .NET 4.5 et .NET pour les applications Windows Store, toutes les interfaces et classes de base qui servent à la liaison des données sont effectivement disponibles dans les deux environnements. Et si les entités servent essentiellement à transporter des données, elles peuvent proposer quelques services complémentaires simples qui ne poseront pas de difficulté particulière lors du partage de code :

Définition de la classe Vin

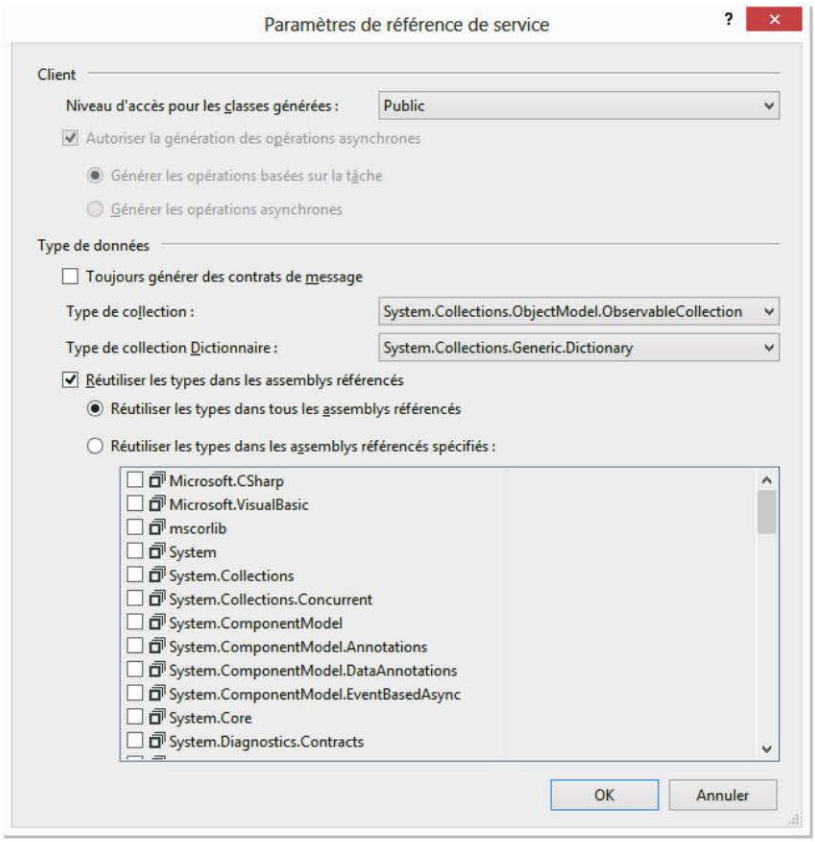
```
[DataContract(IsReference=true)]
public class Vin : BindableObject
{
    private string _nom;
    [DataMember]
    public string Nom
    {
        get { return _nom; }
        set { if (SetProperty(ref _nom, value))
                OnPropertyChanged("NomComplet"); }
    }

    private int _millesime;
    [DataMember]
    public int Millesime
    {
        get { return _millesime; }
        set { if (SetProperty(ref _millesime, value))
                 OnPropertyChanged("NomComplet"); }
    }

    public string NomComplet
    {
        get { return (_millesime == 0)
                ? _nom
                : (_nom + " " +     _millesime.ToString()); }
    }
…
}
```

Le service WCF peut alors être développé en s'appuyant sur cette DLL d'interface partagée. Lors de l'ajout de la référence vers celui-ci au sein de l'application cliente, il faut être vigilant sur deux points :

- ajouter la DLL d'interface aux références du projet WinRT **avant d'ajouter la référence vers le service WCF** ;
- vérifier que la case à cocher *Réutiliser les types dans les assemblys référencés* est bien sélectionnée dans la boîte de dialogue *Paramètres de référence de service* (accessible via le bouton *Avancé* lors de l'ajout de la référence de service).

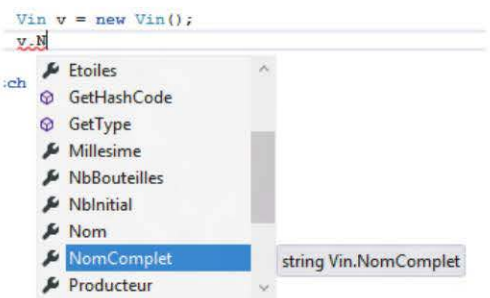
Copyright © 2013 Eyrolles.

Figure 3–14
Réutilisation des types connus



Une fois la référence ajoutée, seule la classe correspondant au proxy sera réellement nouvelle (elle l'est nécessairement à cause de l'aspect asynchrone donné aux opérations vues côté client), mais ses méthodes utiliseront les classes de la bibliothèque partagée, comme on le voit dans la figure suivante où la propriété calculée `NomComplet` est effectivement disponible pour le code de l'application cliente.

Figure 3–15
Appel à la bibliothèque portable



Copyright © 2013 Eyrolles.

Utilisation de WCF Data Services

WCF Data Services est une technologie qui sert à exposer simplement via WCF un modèle d'entités reliées par des relations. Le protocole sous-jacent s'appelle OData et s'appuie sur des appels de type REST, mais ces détails sont masqués aux applications, qui peuvent utiliser une API de plus haut niveau comme LINQ pour interroger le service.

Les architectures basées sur ce type de services seront très orientées « données » et se rapprocheront un peu des applications qui accèdent directement à une base de données relationnelles ; c'est pratique dans bien des cas, mais doit être néanmoins utilisé avec précautions, surtout pour les accès en écriture où il est souhaitable d'avoir un maximum de contrôle côté serveur.

Pour utiliser WCF Data Services dans une application Windows Store, il faut bien évidemment commencer par créer le service WCF :

- création du modèle d'entités (fichier EDMX) à partir d'une base relationnelle ;
- création et configuration du service WCF, en particulier de la sécurité de celui-ci.

Une fois le service créé, il est ajouté aux références de l'application Windows Store : ceci va déclencher la création d'une classe dérivant de `DataServiceContext` et exposant l'équivalent du modèle EDMX au sein du service web. Cette classe peut alors servir de base à l'exécution de requêtes LINQ, comme dans l'exemple qui suit :

```
Uri serviceUri = new Uri("http://localhost:51062/DionysosService.svc");
DionysosEntities context = new DionysosEntities(serviceUri);
var query = from app in context.APPELLATIONs
            where app.REGION.NOM == "Bordeaux"
            select app;
```

Il reste néanmoins une petite subtilité : en effet, dans cet exemple, la variable `Query` est de type `IQueryable<APPELLATION>` et il est tentant de la lier directement à un objet graphique comme une ListBox :

```
TheListBox.ItemsSource = query;
```

Pourtant, l'exécution de cette instruction déclenche la levée d'une exception :

```
System.NotSupportedException: Silverlight does not enable you to directly
enumerate over a data service query. This is because enumeration automatically
sends a synchronous request to the data service. Because Silverlight only
supports asynchronous operations, you must instead call the BeginExecute and
EndExecute methods to obtain a query result that supports enumeration.
```

Copyright © 2013 Eyrolles.

Bien que l'exception mentionne de manière erronée Silverlight, le message est clair : il ne faut pas essayer d'exécuter la requête de manière synchrone. Pour avoir accès à sa version asynchrone, il faut la transtyper dans un objet `DataServiceQuery<APPELLATION>`, puis appeler `BeginExecute` en passant une fonction de rappel qui sera exécutée lorsque le résultat sera disponible. Au sein de celle-ci, il faut penser à « revenir sur le thread de l'IHM » pour affecter la liste obtenue à l'objet graphique :

```
void GetAppellations()
{
  Uri serviceUri = new Uri("http://localhost:51062/DionysosService.svc");
  DionysosEntities context = new DionysosEntities(serviceUri);
  var query = from app in context.APPELLATIONs
              where app.REGION.NOM == "Bordeaux"
              select app;
  DataServiceQuery<APPELLATION> dqs = (DataServiceQuery<APPELLATION>)(query);
  dqs.BeginExecute(QueryExecuted, dqs);
}

async private void QueryExecuted(IAsyncResult iar)
{
    DataServiceQuery<APPELLATION> dqs =
                    iar.AsyncState as DataServiceQuery<APPELLATION>;
    IEnumerable<APPELLATION> apps = dqs.EndExecute(iar);
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        TheListBox.ItemsSource = apps;
    });
}
```

Cet exemple fonctionne correctement, mais il n'est pas totalement satisfaisant, car l'appel asynchrone est effectué en utilisant un *callback*, ce qui complique le code notamment si plusieurs appels doivent être exécutés en se basant sur leurs résultats successifs. En utilisant la classe `TaskFactory`, il est possible de s'appuyer sur les mots-clés `async` et `await` pour retrouver une programmation proche de la version synchrone :

Appel d'une méthode Web Service

```
async void GetAppellations()
{
    Uri serviceUri = new Uri("http://localhost:51062/DionysosService.svc");
    DionysosEntities context = new DionysosEntities(serviceUri);
    var query = from app in context.APPELLATIONs
                where app.REGION.NOM == "Bordeaux"
                select app;
    DataServiceQuery<APPELLATION> dqs =
                                (DataServiceQuery<APPELLATION>)(query);
```

Copyright © 2013 Eyrolles.

```
    TaskFactory<IEnumerable<APPELLATION>> tf =
                                new TaskFactory<IEnumerable<APPELLATION>>();
    IEnumerable<APPELLATION> appellations =
              await tf.FromAsync(dqs.BeginExecute(null, null),
                                  iar => dqs.EndExecute(iar));
    TheListBox.ItemsSource = appellations;
}
```

Le chapitre sur les appels asynchrones décrit plus en détails ces mécanismes d'appels autour des mots-clés `async` et `await`, ainsi que leurs équivalents dans les autres langages de programmation.

Copyright © 2013 Eyrolles.

4

Programmation asynchrone

Dans les environnements modernes comme WinRT, les utilisateurs s'attendent à une très grande réactivité de la part du système, mais aussi de chaque application. Ceci se traduit pour le développeur par la nécessité d'exécuter tous les traitements longs de manière asynchrone afin de laisser l'interface utilisateur réagir immédiatement à toutes les actions de celui-ci.

Ce chapitre détaille les mécanismes sous-jacents, le modèle de programmation asynchrone proposé par WinRT pour chacun des langages de programmation, l'évolution des mécanismes couramment utilisés en .NET, ainsi que les principales pratiques de programmation asynchrone.

Asynchronisme et interface utilisateur réactive

Pour qu'une interface utilisateur soit agréable à utiliser, elle doit être non seulement esthétique, mais surtout réactive. Il arrive fréquemment que des utilisateurs reviennent à d'anciennes versions d'une application parce que la dernière est jugée trop lente et au final peu agréable à utiliser. Les applications étant de plus en plus souvent connectées à Internet avec des accès et des débits très différents, les temps de réponse sont variables et pas toujours sous le contrôle du programmeur : une application qui tarde à donner un résultat n'est pas nécessairement mal programmée (contrairement à une application autonome qui ne dépend de personne).

Copyright © 2013 Eyrolles.

Néanmoins, pour l'utilisateur, le résultat est le même : il appuie sur un bouton et s'attend à une réaction immédiate de l'application. Si cette réaction tarde trop, il doit pouvoir passer à une autre activité, au sein même de l'application ou non.

Les navigateurs internet intègrent ce concept depuis de nombreuses années : pendant qu'une page est en cours de téléchargement, l'application affiche un retour à l'utilisateur (logo animé, barre de progression, messages) et reste opérationnelle. Ainsi, l'utilisateur peut annuler la navigation en cours, retourner en arrière ou simplement ouvrir un nouvel onglet dans lequel il va afficher et lire une autre page pendant que la première finit de se charger.

Cette volonté de réactivité permanente n'est pas limitée aux applications : si l'utilisateur est bloqué, c'est toute la perception du système d'exploitation qui en pâtit. Qui n'a jamais fulminé contre une machine ne redémarrant pas parce qu'une application affiche une fenêtre demandant la confirmation d'une sortie sans sauvegarde ?

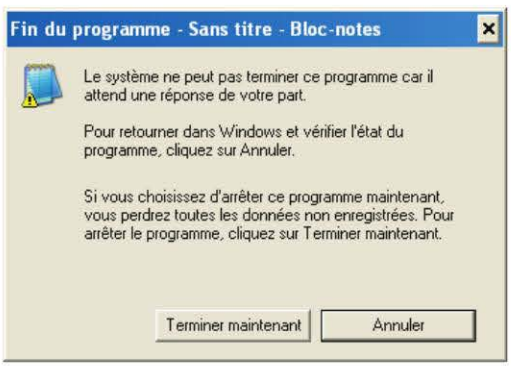


Figure 4–1
Application bloquée sous Windows XP

Dans de nombreux cas d'appels externes, l'application se trouve ainsi bloquée inutilement : elle n'est pas en mesure de répondre aux sollicitations de l'utilisateur et n'effectue pas non plus de réelle tâche de fond. Cette situation est illustrée dans le schéma qui suit.

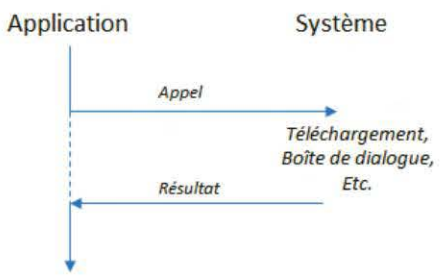


Figure 4–2
Appel Synchrone

Copyright © 2013 Eyrolles.

Ce schéma correspond par exemple au code suivant dans lequel `GetResponse` est l'appel long :

```
void TelechargementSynchrone(Uri serviceUri)
{
    WebRequest request = HttpWebRequest.Create(serviceUri);
    WebResponse response = request.GetResponse();
    …
}
```

Pour éviter les blocages liés à des opérations longues, la solution générale consiste à faire exécuter celles-ci par un thread différent de celui qui gère l'interface utilisateur : celle-ci reste alors réactive, puisque les actions de l'utilisateur sont prises en charge immédiatement, même si elles peuvent n'être exécutées qu'un peu plus tard. Certaines applications existantes mettent en œuvre ce principe (par exemple, les navigateurs internet). Néanmoins la programmation multitâche est délicate et un principe général sans implémentation standardisée se retrouvera nécessairement décliné avec plus ou moins de succès, voire ignoré dans les diverses applications.

Pour résoudre ces problèmes tout en homogénéisant l'implémentation, la vraie solution consiste à modifier les API elles-mêmes, afin d'inciter voire d'obliger le développeur à utiliser un modèle de programmation cohérent et éprouvé. Depuis longtemps, le framework .NET offre des versions asynchrones de toutes les méthodes susceptibles d'avoir un temps d'exécution à la fois long et consacré pour l'essentiel à l'attente de ressources externes (au sens large, c'est-à-dire aussi bien une base de données locale qu'un service web). Chaque méthode répondant à ces caractéristiques est alors coupée en deux parties :

- la première déclenche l'appel et le traitement associé sur un thread d'arrière-plan ;
- la seconde en récupère le résultat.

Par ailleurs, la méthode d'appel accepte en paramètre une fonction de rappel optionnelle (*callback* en anglais) qui est invoquée automatiquement lorsque le résultat est disponible.

Le schéma d'appel présenté plus haut devient le suivant.

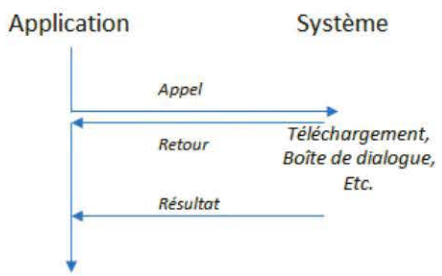


Figure 4–3
Appel asynchrone avec rappel

Copyright © 2013 Eyrolles.

Ce fonctionnement est obtenu avec un code comme celui-ci :

```
void TelechargementAsynchrone(Uri serviceUri)
{
    WebRequest request = HttpWebRequest.Create(serviceUri);
    request.BeginGetResponse(FinTelechargementAsynchrone, request);
}
void FinTelechargementAsynchrone(IAsyncResult ar)
{
    WebResponse response = ((WebRequest)ar.AsyncState).EndGetResponse(ar);
…
}
```

Ce mécanisme est disponible depuis de nombreuses années pour les développeurs .NET, mais il ne résout pas tous les problèmes.

- Le rappel du délégué survient généralement sur un thread différent de celui qui a réalisé l'appel initial : cela pose quelques difficultés lorsque le traitement associé doit être réalisé sur le thread qui gère l'interface utilisateur. En effet, Windows (comme la plupart des systèmes d'exploitation) impose qu'un composant graphique soit uniquement sollicité par le thread qui l'a créé. Il faut donc recourir de nouveau à un appel asynchrone pour exécuter les mises à jour de l'interface sur le thread adéquat. Le corps de méthode unique se transforme en un groupe de trois blocs de code séparés mais qui doivent néanmoins se transmettre des données : le traitement initial qui se solde par l'appel asynchrone, la récupération du résultat de celui-ci et enfin la mise à jour de l'interface utilisateur.
- Comme l'appel asynchrone est compliqué par ces blocs multiples et que les méthodes synchrones sont disponibles, beaucoup de programmeurs choisissent la voie de la facilité et utilisent ces dernières.
- Les schémas un peu plus sophistiqués, comme le déclenchement puis l'attente de plusieurs appels enchaînés ou simultanés, sont rapidement complexes à mettre en œuvre.
- Les appels asynchrones ne sont pas disponibles pour les boîtes de dialogue modales : celles-ci peuvent donc toujours bloquer la fermeture de l'application.

Pour résoudre ces difficultés, l'API WinRT fournit un nouveau modèle pour les appels asynchrones, suffisamment simple pour que les développeurs s'y adaptent facilement et qui autorise de nouveaux scénarios :

- Toutes les méthodes potentiellement longues sont rendues asynchrones selon un nouveau schéma qui sera décrit plus loin et **elles n'existent pas en version synchrone**. Par convention, le nom des méthodes asynchrones se termine par le suffixe `Async`.

Copyright © 2013 Eyrolles.

Durée d'une longue méthode

Le temps d'exécution à partir duquel une méthode est considérée comme longue pour WinRT est assez réduit puisqu'il a été fixé à cinquante millisecondes seulement. Tout ce qui est susceptible de dépasser cette durée est servi par une API asynchrone.

- Les méthodes asynchrones retournent toutes un objet implémentant l'interface `IAsyncOperation` (ou une de ses variantes telles que `IAsyncAction` ou `IAsyncOperationWithProgress`).
- Le langage C# tout comme VB.NET, ainsi que leur compilateur, intègrent le concept d'asynchronisme afin de faciliter l'écriture et la relecture du code asynchrone en prenant en charge toutes les opérations de bas niveau associées à ce type d'appel.
- Dans l'environnement .NET, les méthodes `async` retournent des objets de type `Task<T>`.
- Les méthodes asynchrones existantes du framework .NET sont conservées lorsque les classes restent disponibles pour les applications Windows Store, mais leurs versions synchrones ont disparu.
- Dans beaucoup de cas, une méthode correspondant au nouveau standard d'appel asynchrone a été ajoutée à côté de la méthode existante, par exemple :
  - `WebRequest.BeginGetResponse` est conservée.
  - `WebRequest.GetResponse` a été supprimée.
  - `WebRequest.GetResponseAsync` a été ajoutée et retourne un objet `Task<T>`.
- Les boîtes de dialogue s'affichent maintenant uniquement via des appels asynchrones basés sur le même schéma que les autres appels longs.

IAsyncOperation

Les méthodes asynchrones de WinRT retournent une des interfaces suivantes.

`IAsyncAction`	Traitements asynchrones sans valeur de retour
`IAsyncActionWithProgress<TProgress>`	Traitements asynchrones sans valeur de retour mais avec suivi de la progression
`IAsyncOperation<TResult>`	Traitements asynchrones avec valeur de retour
`IAsyncOperationWithProgress<TResult, TProgress>`	Traitements asynchrones avec valeur de retour et avec suivi de la progression

Copyright © 2013 Eyrolles.

Voici un exemple volontairement simple d'un code qui procède au téléchargement d'un flux RSS et en affiche l'avancement (les implémentations des appels asynchrones en JavaScript et en C++ seront présentées plus loin dans ce chapitre) :

Téléchargement d'un flux RSS avec affichage de l'avancement

```
IAsyncOperationWithProgress<SyndicationFeed, RetrievalProgress> _enCours;

void LancerTelechargement(string url) 1
{
    var client = new SyndicationClient();
    _enCours = client.RetrieveFeedAsync(new Uri(url)); 4
    _enCours.Progress = ProgressCB;
    _enCours.Completed = CompletedCB;
}

private void CompletedCB(3
    IAsyncOperationWithProgress<SyndicationFeed, RetrievalProgress> asyncInfo,
    AsyncStatus asyncStatus)
{
    switch (asyncStatus)
    {
        case AsyncStatus.Completed:
            DisplayMessage("Completed");
            SyndicationFeed feed = asyncInfo.GetResults();5
            DisplayFeed(feed);
            break;
        default:
            DisplayMessage(asyncStatus.ToString());
            break;
    }
    _enCours = null;
}

private void ProgressCB(2
    IAsyncOperationWithProgress<SyndicationFeed, RetrievalProgress> asyncInfo,
    RetrievalProgress info)
{
    DisplayMessage(string.Format("{0} / {1}",
                info.BytesRetrieved,
                 info.TotalBytesToRetrieve));
}

void AnnulerTelechargement()
{
    if (_enCours != null)
    {
```

Copyright © 2013 Eyrolles.

```
            _enCours.Cancel();
        }
    }
```

Dans cet exemple (qui, pour rester simple, ne gère pas les appels multiples) :

1. la méthode `LancerTelechargement` appelle `SyndicationClient.RetrieveFeedAsync` qui retourne une interface `IAsyncOperationWithProgress<TResult, TProgress>`. Celle-ci possède deux propriétés de type délégué qui enregistrent deux fonctions de rappel ;
2. `ProgressCB` sera appelée régulièrement pour indiquer l'avancement de la tâche ;
3. `CompletedCB` sera appelée en fin de traitement ;
4. par ailleurs, l'interface retournée par l'appel de `RetrieveFeedAsync` peut être utilisée pour annuler l'opération en cours à l'aide de sa méthode `Cancel` ;
5. enfin, la méthode `GetResults()` récupère le résultat du traitement asynchrone, un objet `SyndicationFeed` dans cet exemple.

La notification de la progression et la possibilité d'annuler le traitement en cours sont deux nouveautés du schéma des appels asynchrones de l'API WinRT par rapport à ce qui est habituellement proposé par .NET. Il faut noter néanmoins que pour la plupart des appels, la notion d'avancement ne fait pas de sens et n'est donc pas disponible.

Dans ce schéma, les fonctions de rappel peuvent être exécutées sur un thread différent de celui qui a effectué le premier appel : si l'interface utilisateur doit être mise à jour à l'intérieur de ces fonctions, il est donc nécessaire de revenir sur le thread principal pour effectuer ces traitements, sous peine de voir apparaître l'exception suivante :

```
The application called an interface that was marshalled for a different thread.
(Exception from HRESULT: 0x8001010E (RPC_E_WRONG_THREAD))
```

Pour cela, il faut s'appuyer sur un objet `Dispatcher` et la méthode asynchrone `RunAsync` :

```
private void DisplayMessage(string message)
{
    Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
    {
        DebugMessage = message;
    });
}
```

Copyright © 2013 Eyrolles.

> **EN COULISSE Liaison de données**
>
> Bien qu'elle semble parfois traitée de manière « magique » en arrière-plan, la liaison de données est en fait un mécanisme synchrone déclenché via l'événement `PropertyChanged` de l'interface `INotifyPropertyChanged` : la mise à jour d'une propriété liée à un élément d'interface doit donc elle aussi être réalisée sur le thread de l'interface utilisateur.

Si l'interface `IAsyncOperation` et ses consœurs apportent quelques nouveautés par rapport au schéma d'appel asynchrone classique de .NET, l'exemple précédent montre qu'elle n'est pas suffisante pour rendre les appels asynchrones faciles à programmer : c'est ici qu'intervient la modification du langage C# et de son compilateur avec l'intégration de l'asynchronisme au cœur du langage.

Async et await

Comme il a déjà été mentionné, les fonctions potentiellement longues de l'API WinRT n'existent qu'en version asynchrone et l'exemple précédent a montré qu'un appel asynchrone se trouve découpé en plusieurs blocs de code. Voici un petit exemple de lecture et affichage de fichier texte :

Lire et afficher un fichier texte

```
void LireFichier(string nom)
{
    var operation = KnownFolders.DocumentsLibrary.GetFileAsync(nom);
    operation.Completed = LireFichierPartie2;
}

void LireFichierPartie2(IAsyncOperation<StorageFile> asyncInfo,
                        AsyncStatus asyncStatus)
{
    StorageFile fichier = asyncInfo.GetResults();
    var operation = FileIO.ReadTextAsync(fichier);
    operation.Completed = LireFichierPartie3;
}

private void LireFichierPartie3(IAsyncOperation<string> asyncInfo,
                                AsyncStatus asyncStatus)
{
    string contenu = asyncInfo.GetResults();
    Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
                        () => TextContent = contenu);
}
```

Copyright © 2013 Eyrolles.

Une telle écriture de code est assez laborieuse et difficile à déboguer… Il est possible d'utiliser des délégués anonymes pour rendre le code plus compact mais le gain en lisibilité n'est pas toujours évident (sans compter que l'exemple qui suit ne gère pas les erreurs) :

Lecture depuis un fichier

```
void LireFichierCompact(string nom)
{
  KnownFolders.DocumentsLibrary.GetFileAsync(nom).Completed = (info,status) =>
  {
    FileIO.ReadTextAsync(info.GetResults()).Completed = (info2, status2) =>
    {
      Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
      {
          TextContent = info2.GetResults();
      });
    });
  )};
}
```

Comme les appels asynchrones ne sont plus une option dans la programmation WinRT, le langage C# intègre cette notion et propose deux nouveaux mots-clés pour simplifier la vie du programmeur :

- `await` « attend » la fin d'un appel asynchrone.
- `async` marque une méthode comme pouvant utiliser le mécanisme `await`.

L'exemple de lecture de fichier retrouve alors la simplicité des appels synchrones (le code écrit par le développeur est simple, mais celui créé par le compilateur est d'une grande sophistication) :

```
async void LireFichierAsync(string nom)
{
  StorageFile fichier = await KnownFolders.DocumentsLibrary.GetFileAsync(nom);
  string contenu = await FileIO.ReadTextAsync(fichier);
  TextContent = contenu;
}
```

En effet, l'exécution de la méthode est suspendue à chaque mot clé `await`, puis elle reprend lorsque le résultat de l'appel asynchrone correspondant est disponible. Cependant, contrairement à ce qui se passe dans le cas d'appels synchrones, le thread qui a commencé à exécuter la méthode n'est pas bloqué pendant cette suspension. Il va alors effectuer d'autres traitements : c'est ainsi que l'interface utilisateur reste réactive, puisque le thread responsable demeure disponible pour traiter les événements qu'elle déclenche (ou plutôt, il n'est jamais très longtemps indisponible). En outre,

Copyright © 2013 Eyrolles.

une fois l'appel asynchrone terminé, le code qui suit est exécuté par le même thread, évitant l'emploi du `Dispatcher`.

Il ne s'agit pas non plus d'une simple boucle de messages « masquée » incorporée dans le code créé, car si l'utilisateur ou le système décide de terminer l'application, le code qui suit le mot-clé `await` ne sera pas exécuté.

Les exceptions qui surviennent avant, pendant ou après les appels asynchrones peuvent toutes être capturées et traitées uniformément, et ce de manière naturelle, c'est-à-dire à l'aide d'un bloc `try/catch` dans la méthode appelante :

```
async void LireFichierAsync(string nom)
{
  try
  {
    StorageFile fichier = await KnownFolders.DocumentsLibrary.GetFileAsync(nom);
    string contenu = await FileIO.ReadTextAsync(fichier);
    TextContent = contenu;
  }
  catch …
  {
…
  }
}
```

Si le mot-clé `await` et l'appel de la méthode asynchrone sont souvent immédiatement enchaînés comme dans les exemples précédents, ce n'est pas obligatoire.

Puisque la méthode asynchrone retourne toujours une interface de type `IAsyncOperation`, il est donc possible d'enregistrer des fonctions de rappel sur cette interface ou de la stocker dans une variable de classe (s'il est nécessaire d'autoriser l'annulation de l'appel, une variable locale ne suffira pas car elle n'a pas une portée suffisante).

Et il est possible d'appeler `await` sur l'interface après avoir effectué de telles opérations, par exemple :

```
var client = new SyndicationClient();
_request = client.RetrieveFeedAsync(new Uri(Url));
_request.Progress = ProgressCB;
SyndicationFeed result = await request;
…
```

Dans un exemple comme celui-ci, la variable `result` ne peut évidemment pas contenir un objet valide si l'opération asynchrone (ici un téléchargement) est annulée, par exemple en appelant la méthode `Cancel()` sur la variable membre `_request`. En cas d'appel de cette méthode, une exception est levée sur l'instruction `await` : `TaskCanceledException`, dont le message est *A task was canceled*.

Copyright © 2013 Eyrolles.

Quelles contraintes une méthode asynchrone impose-t-elle ? Tout d'abord, un constructeur ne peut pas être asynchrone : la suspension de l'exécution d'un constructeur créerait des problèmes sur le cycle de vie des objets, certes intéressants, mais difficiles à appréhender pour le programmeur (et probablement pour les développeurs du compilateur).

Ensuite, l'asynchronisme est « viral » : cela est visible dans les exemples précédents où une méthode doit être déclarée `async` pour utiliser le mot-clé `await`. Une méthode .NET asynchrone ne peut pas avoir n'importe quelle signature et le compilateur C# l'indique clairement : « *error CS1983 : The return type of an async method must be void, Task or Task<T>* ». De la même manière, une méthode asynchrone ne peut pas avoir de paramètres de type `ref` ou `out`.

Si une méthode renvoie des données acquises de manière asynchrone, elle aura donc elle-même une signature asynchrone. C'est en fait naturel puisque l'exécution de la méthode appelée va être découpée (de manière transparente pour le développeur) en plusieurs étapes et la méthode appelante va nécessairement hériter de ce découpage.

Tâches

Comme indiqué plus haut, les méthodes asynchrones du framework .NET accessibles aux applications Windows Store retournent des objets de type `Task` pour les méthodes `void` asynchrones et `Task<T>` pour les méthodes retournant un objet de type `T`.

Les mécanismes décrits précédemment au sujet des méthodes asynchrones qui s'appuient sur le mot-clé `await` pour rendre la programmation asynchrone accessible à tous, sont applicables aux méthodes retournant des `Task/Task<T>` aussi bien qu'aux méthodes de WinRT qui renvoient des interfaces `IAsyncOperation` :

- `SyndicationClient.RetrieveFeedAsync` retourne un `IAsyncOperationWithProgress<SyndicationFeed, RetrievalProgress>` et
- `WebRequest.GetResponseAsync` retourne un `Task<WebResponse>`, mais les deux méthodes s'utilisent de la même manière :

```
var client = new SyndicationClient();
SyndicationFeed result = await client.RetrieveFeedAsync(new Uri(url));

var request = WebRequest.CreateHttp(new Uri(url));
WebResponse response = await request.GetResponseAsync();
```

Les tâches offrent un certain nombre de fonctions intéressantes pour le développeur.

Tout d'abord, elles permettent d'utiliser les anciennes API, basées sur l'interface `IAsyncResult`, de la même manière que les nouvelles API : ainsi, le développeur a un

Copyright © 2013 Eyrolles.

mode de programmation asynchrone uniforme, quels que soient les composants utilisés. Par exemple, si un service WCF Data Service est ajouté aux références de service d'un projet d'application Windows Store C#, il n'est pas possible d'exécuter les requêtes sur le contexte de données correspondant de manière synchrone et la version asynchrone de ces requêtes est décomposée en `DataServiceQuery<T>.BeginExecute` et `DataServiceQuery<T>.EndExecute`. La classe générique `TaskFactory<T>` et sa méthode `FromAsync` permettent d'utiliser `await` et de simplifier l'utilisation de toutes les méthodes basées sur le schéma `BeginXXX/EndXXX` et `IAsyncResult` :

```
var request = WebRequest.CreateHttp(new Uri(url));
TaskFactory<WebResponse> tf = new TaskFactory<WebResponse>();
WebResponse wr = await tf.FromAsync(request.BeginGetResponse(null, null),
                                    iar => request.EndGetResponse(iar));
…
```

Ensuite, elles autorisent à configurer certains comportements des appels asynchrones. Par exemple, il n'est pas toujours nécessaire que le traitement du résultat d'une méthode asynchrone soit pris en charge par le même thread que celui qui a lancé l'opération : cela n'est souvent indispensable que lorsqu'une mise à jour de l'interface est effectuée à partir du résultat reçu. Dans les autres cas, autoriser un autre thread à traiter le résultat d'un appel asynchrone améliore souvent les performances de l'application en réduisant les délais liés aux synchronisations. Ceci s'effectue en appelant `ConfigureAwait` sur l'objet `Task` :

```
var request = WebRequest.CreateHttp(new Uri(url));
WebResponse wr = await request.GetResponseAsync().ConfigureAwait(false);
```

Le paramètre passé à `ConfigureAwait` indique si le rappel doit être exécuté dans le même contexte que l'appel ou non.

Les tâches permettent de coder des schémas de programmation asynchrone plus sophistiqués qu'une simple attente non bloquante tels que :

- une attente temporisée :

```
Task<WebResponse> t1 = …
t1.Wait(TimeSpan.FromMilliseconds(20.0));
```

- une attente simultanée de plusieurs tâches lancées en parallèle, avec reprise soit quand toutes les tâches sont terminées, soit quand une d'entre elles l'est :

```
Task<WebResponse> t1 = …
Task<WebResponse> t2 = …
Task.WaitAll(t1, t2); // Attend la fin de t1 et de t2
Task.WaitAny(t1, t2); // Attend la fin de t1 ou de t2
```

Copyright © 2013 Eyrolles.

> **ATTENTION Attentes bloquantes**
>
> Il faut noter que ces derniers exemples constituent des attentes réellement bloquantes, au même titre que des appels synchrones classiques. Ils sont donc à proscrire complètement de tous les traitements appelés depuis l'interface utilisateur. Ils trouvent cependant parfaitement leur place dans des traitements déjà effectués en arrière-plan.

Enfin, les tâches sont créées explicitement, typiquement pour réaliser des traitements d'arrière-plan, tout en autorisant les différents mécanismes de synchronisation déjà présentés.

Pool de threads

La création d'un thread a toujours été une opération coûteuse ; pour cette raison, le framework .NET met depuis longtemps à la disposition des applications un ensemble de threads prêts à l'emploi (appelé *pool de threads*) auxquels il est possible de confier des actions à exécuter via la classe `System.Threading.ThreadPool`. Ces threads prennent aussi en charge l'exécution des fonctions de rappels des méthodes asynchrones, ainsi que l'exécution des tâches présentées dans la section précédente.

Avec WinRT, un nouveau pool de thread devient accessible aux applications .NET, mais la méthode utilisée pour lancer un traitement d'arrière-plan sur un de ces threads évolue pour être conforme au modèle de programmation asynchrone de WinRT :

- `Windows.System.Threading.ThreadPool.RunAsync` retourne une interface `IAsyncAction` pour attendre la fin de l'exécution du traitement en arrière-plan ou l'annuler ;
- cette interface est aussi passée au délégué à exécuter en arrière-plan : cela permet de tester dans le corps du traitement l'éventuelle annulation de celui-ci. En cas d'annulation, il n'y a en effet pas d'interruption brutale d'un code en cours d'exécution, mais un arrêt « en douceur ».

Voici par exemple un petit code créant des nombres premiers en arrière-plan, mais interruptible à l'aide des mécanismes standards de WinRT :

Pilotage de méthode asynchrone

```
IAsyncAction _generationPremiers;

private void GenerePremiers(IAsyncAction aa)
{
    AutoResetEvent evt = new AutoResetEvent(false);
    int n = 3;
```

Copyright © 2013 Eyrolles.

```
    while (aa.Status != AsyncStatus.Canceled)
    {
        if (EstPremier(n))
        {
            AffichePremier(n);
        }
        n += 2;
    }
}
private void LancerTraitement()
{
    generationPremiers = ThreadPool.RunAsync(GenerePremiers);
}
private void ArreterTraitement()
{
    _generationPremiers.Cancel();
}
```

Appels asynchrones en JavaScript

JavaScript a deux caractéristiques importantes en termes de programmation asynchrone.

- Il n'est pas multitâche : il n'est donc pas possible de lancer un thread en arrière-plan pour exécuter un traitement long tout en conservant l'interface utilisateur réactive.
- Il intègre en revanche les appels asynchrones depuis longtemps et les développeurs JavaScript sont donc habitués à manipuler ce type d'appels : les requêtes XMLHTTP ont en effet toujours été exécutées de manière asynchrone.

Pour les applications Windows Store, Microsoft a choisi d'utiliser le modèle de programmation basé sur les promesses (*promise* en anglais) : une promesse est un objet qui sert à manipuler le résultat « futur » d'une opération.

Les méthodes proposées par l'objet `Promise` acceptent pour la plupart une ou plusieurs fonction(s) JavaScript à exécuter en fonction du résultat du traitement asynchrone.

La fonction principale est `then`, qui accepte trois paramètres :

- une fonction à appeler à la fin du traitement si tout s'est bien passé ;
- une fonction à appeler en cas d'erreur (facultative) ;
- une fonction à appeler pour signaler l'avancement (facultative).

Avec ce schéma, la lecture d'un fichier local en JavaScript peut s'implémenter de la manière suivante :

Copyright © 2013 Eyrolles.

Lecture d'un fichier disque

```
var documents = Windows.Storage.KnownFolders.documentsLibrary;
documents.getFileAsync("foo.txt").then(
    function (file) {
        Windows.Storage.FileIO.readTextAsync(file).then(function (text) {
            var res = document.getElementById("Result");
            res.innerText = text;
        });
    },
    function (error) {
        var dlg = new Windows.UI.Popups.MessageDialog(error.description,
"Erreur");
        dlg.showAsync();
    }
);
```

On peut noter au passage que le JavaScript étant monotâche, tout le code exécuté en asynchrone peut accéder sans problème aux divers composants de l'interface homme-machine.

Le code est plus difficile à lire que la version C# avec `async`/`await` et il faut aussi noter que les traitements d'erreurs doivent être spécifiés à chaque niveau de promesse (un `try`/`catch` global ne capte pas les exceptions levées dans les fonctions de rappel passées aux fonctions `then`) ; néanmoins, le fonctionnement global de l'application est celui attendu (réactivité) et certaines fonctions de la classe `Promise` permettent des schémas de programmation asynchrone assez riches :

- `any` : prend un tableau de promesses en paramètre et exécute la fonction de rappel dès lors que le traitement associé à l'une d'elles est terminé ;
- `join` : prend un tableau de promesses en paramètre et exécute la fonction de rappel quand tous les traitements associés sont terminés ;
- `theneach` : prend un tableau de promesses en paramètre et exécute la fonction de rappel pour chaque promesse lorsque son traitement est terminé ;
- `timeout` : associe un délai d'exécution à une promesse.

Appels asynchrones en C++

Dans une application C++, les appels asynchrones des API de WinRT s'appuient sur le *Concurrency Runtime*, qui se base essentiellement sur la classe générique `concurrency::Task`. La fonction `create_task` crée une tâche à partir de l'interface retournée par la fonction asynchrone appelée (`IAsyncAction`, `IAsyncOperation<T>`, etc) et il est ensuite possible de préciser une fonction qui sera appelée lorsque le traitement asynchrone sera terminé.

Copyright © 2013 Eyrolles.

Pour obtenir un bloc de code où les différentes parties du traitement sont rassemblées et partagent éventuellement un même ensemble de variables locales, il est nécessaire de recourir aux fonctions *lambda*. Il est à noter que la capture du contexte n'est pas automatique ; il faut donc préciser, pour chaque lambda, quels sont les éléments à capturer.

L'opération de lecture d'un fichier texte qui a déjà servi d'exemple dans les autres langages s'écrit en C++ de la manière suivante :

```
StorageFolder^ documents = KnownFolders::DocumentsLibrary;
create_task(documents->GetFileAsync("foo.txt"))
       .then([this](StorageFile^ file)->IAsyncOperation<Platform::String^>^ {
  return FileIO::ReadTextAsync(file);
}).then([this](Platform::String^ text) {
  this->ResultTextBlock->Text = text;
});
```

Comme en C# avec les méthodes asynchrones associées au mot-clé `await`, la classe `Task` capture le contexte d'appel. Si celui-ci est réalisé sur le thread responsable de l'interface utilisateur, les fonctions lambda sont elles aussi exécutées sur ce thread et l'accès direct aux composants de l'interface utilisateur est donc possible.

Bonnes pratiques

D'une certaine manière, il est plus facile de respecter les bonnes pratiques de programmation asynchrone dans une application Windows Store que dans une application Windows classique, puisque pratiquement toutes les méthodes susceptibles de rendre non réactive une application ont disparu ou été transformées en méthodes asynchrones.

La classe `Task<T>` constitue l'essentiel des risques pour une application Windows Store : en effet, certains appels bloquants subsistent, comme `Task.Wait`, et il faut donc impérativement éviter d'utiliser ces appels sur le thread responsable de l'interface utilisateur.

Enfin, il est préférable d'homogénéiser tout le code asynchrone des applications et bien évidemment de le standardiser selon l'approche WinRT basée sur `async` et `await` :

- lorsqu'on utilise des composants existants exposant plutôt les anciens modèles de programmation (`BeginXXX/EndXXX` et utilisation de `IAsyncResult`), moderniser leur utilisation à l'aide de la classe `TaskFactory` comme décrit précédemment ;
- lorsqu'on développe une bibliothèque de composants : respecter le standard WinRT en imposant un modèle asynchrone dès qu'un traitement risque de prendre un peu de temps (rappel : les API standards sont asynchrones dès lors qu'elles peuvent dépasser cinquante millisecondes) et en implémentant le modèle basé sur les `Task<T>` en .NET.

Copyright © 2013 Eyrolles.

5

Cycle de vie des applications

Sous les anciennes versions de Windows et aujourd'hui encore pour le bureau dit classique, les applications ont un cycle de vie simple : elles sont exécutées puis à un moment donné terminées, généralement par l'utilisateur.

Avec la nécessité de gérer de nouvelles plates-formes matérielles, comme les tablettes, ce n'est pas simplement le côté graphique qui a été revu, mais également le fonctionnement des applications et leur cycle de vie afin notamment de minimiser l'usage des ressources du système et maximiser la durée de la batterie. C'est ce nouveau cycle de vie et les changements qu'il impose qui seront étudiés en détail dans ce chapitre.

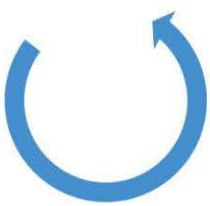


Figure 5–1
Comprendre et apprivoiser le nouveau cycle de vie des applications Windows Store

Chaque fois qu'une application est lancée, celle-ci consomme de nombreuses ressources : des cycles processeur pour effectuer des opérations, de la mémoire pour stocker des données, des ressources graphiques et enfin du temps et de l'espace disque pour stocker des données ou y accéder. Tout cela a une incidence sur les performances globales du système, mais surtout sur la tenue de la charge de la batterie. Avec l'avène-

Copyright © 2013 Eyrolles.

ment des périphériques mobiles (ordinateurs portables, smartphones et maintenant tablettes), il devient indispensable de minimiser la déperdition énergétique.

Lorsqu'une application de bureau est lancée, elle consomme des ressources jusqu'au moment où l'utilisateur décide de l'arrêter en fermant la fenêtre correspondante. C'est ainsi l'utilisateur qui contrôle le nombre d'applications lancées en parallèle sur son système et, par conséquent, la quantité grandissante de ressources utilisées qui en découle. Le système est bien sûr capable d'optimiser la façon dont les ressources sont allouées, mais si les applications sont trop nombreuses, la limite des ressources du système peut être atteinte, avec pour conséquence un ralentissement sensible de l'ordinateur et une expérience utilisateur dégradée.

Afin d'éviter à l'utilisateur la tâche fastidieuse de gérer finement l'allocation des ressources de sa machine, le système d'exploitation de Windows Phone a introduit un mécanisme qui « suspend » (met en pause) automatiquement une application. Ce même mécanisme se retrouve d'ailleurs dans d'autres systèmes d'exploitation pour mobiles.

La vraie nouveauté réside dans la généralisation de ce procédé à toutes les applications Windows Store, même celles tournant sur PC. Le système intervient lorsque le niveau des ressources devient critique et peut alors décider de terminer les applications en arrière-plan qui étaient déjà suspendues. Pour l'utilisateur, tout cela doit rester transparent et il doit pouvoir reprendre une application dans l'état où il l'a laissée, exactement comme une fenêtre laissée en arrière-plan puis reprise en avant-plan sur le bureau classique. Cela impose au développeur des contraintes nouvelles.

Le principe de cycle de vie

Avant d'aborder la suspension en détail, il est important de bien comprendre le cycle de vie d'une application Windows Store dans son ensemble, pourquoi et comment celle-ci va donner l'impression d'être toujours « en vie ». Le schéma suivant représente les trois états possibles d'une application, ainsi que les événements déclenchés lors du passage d'un état à un autre.

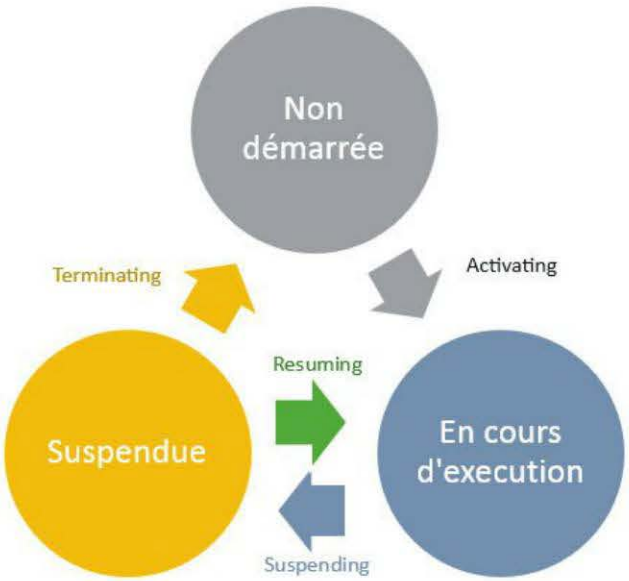
Au démarrage d'une application Windows Store, via une vignette de l'écran de démarrage par exemple, Windows affiche un écran d'accueil (*splash screen*, faisant partie du manifeste de l'application) pendant qu'il instancie l'application. Ensuite, la méthode `OnLaunched` est appelée pour que l'application s'initialise (ce qui doit être exécuté en moins d'une dizaine de secondes, sous peine d'être automatiquement terminée par le système). Une fois cette méthode exécutée, la première page de l'application est affichée et l'état de l'application devient `Running`.

Il faut noter que l'application peut être lancée de plusieurs autres façons :

- en utilisant le charme de recherche, ce qui déclenchera et appellera la méthode `OnSearchActivated()` (sauf pour les applications JavaScript) ;

Copyright © 2013 Eyrolles.

Figure 5–2
Cycle de vie et événements d'une application Windows 8



- en étant choisie comme application cible du charme de partage (méthode `OnShareTargetActivated()`) ;
- ou bien en utilisant un protocole d'action personnalisée (voir le chapitre 7), ce qui déclenchera l'appel de la méthode `OnActivated()`.

> **EXCEPTION Les applications JavaScript et la recherche ou le partage**
>
> Comme cela sera expliqué plus tard dans ce chapitre et en détail dans le chapitre 7, les applications JavaScript n'ont qu'un seul événement `Activated`, mais celui-ci possède un typage (`ActivationKind.search` ou `ActivationKind.shareTarget`) qui indique si l'activation est déclenchée par une recherche ou par un partage.

Une seule application Windows 8 est autorisée à occuper le premier plan (ou deux dans le mode ancré décrit dans le chapitre 2). Ainsi, lorsque l'utilisateur quitte l'application pour en lancer une autre, la première passe en arrière-plan et le système attend quelques secondes (pour voir si l'utilisateur va revenir sur l'application) avant de la suspendre. L'événement `Suspending` est donc ensuite reçu et l'application passe à l'état `Suspended`.

> **ATTENTION Délai de sauvegarde**
>
> Comme cela sera vu plus tard dans le chapitre, si la sauvegarde des données prend plus de 5 secondes, Windows annule la suspension et ferme l'application purement et simplement.

Copyright © 2013 Eyrolles.

L'état `Suspended` indique simplement au système de garder l'application en mémoire, mais fait en sorte que celle-ci ne puisse pas utiliser le processeur (sauf pour des tâches de fond). Lorsque l'utilisateur accède de nouveau à l'application, celle-ci est réveillée par Windows et la méthode associée à l'événement `Resuming` est appelée. Il faut d'ailleurs noter que, contrairement aux méthodes `OnLaunched` ou `OnActivated`, il est nécessaire d'enregistrer explicitement une méthode pour les événements `Suspending` et `Resuming` (généralement dans le constructeur de la classe `Application`) afin d'être notifié de ces changements d'état.

Il est visuellement possible de vérifier qu'une application est suspendue en ouvrant le gestionnaire de tâches et en affichant le statut des applications (menu *Affichage*>*Valeur d'état*>*afficher l'état Suspendu*). Une colonne apparaît alors, précisant si oui ou non l'application est suspendue.

Figure 5–3
Applications et leur statut au sein du gestionnaire de tâches

Comme indiqué précédemment, si plusieurs applications sont suspendues et si la mémoire devient insuffisante, le système décide de terminer certaines d'entre elles. Il le fait de façon brutale, en libérant les ressources et, surtout, ne déclenche aucun événement particulier qui signalerait que l'application va être arrêtée. C'est pourquoi l'événement `Suspending` reste le seul moment disponible pour éventuellement sauvegarder les données. Une fois terminée, l'application passe alors au statut `Not Running`.

Dans les arguments de lancement de l'application, que ce soit un lancement normal ou une activation par un charme ou autre, il est possible de connaître l'état précédent pour déclencher une action spécifique lorsque cela s'avère nécessaire : par exemple, la restauration au lancement du dernier état sauvegardé par l'application lors de son arrêt par Windows.

Copyright © 2013 Eyrolles.

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
      if(args.PreviousState == ApplicationExecutionState.Terminated)
        {
         // restauration des données
      }
}
```

Le principe de suspension

Ce principe est simple puisque c'est le système qui contrôle les applications une fois lancées. Il décide de leur mise en pause de manière totalement transparente. Lorsque l'utilisateur cherche à accéder de nouveau à son application, celle-ci se « réveille » sans que l'utilisateur soit au courant de sa mise en pause.

Les traitements en cours reprennent, les données ne sont jamais perdues et l'utilisation de l'application peut alors continuer normalement. Pour arriver à ce résultat, Windows fait en sorte que les threads des applications suspendues ne soient jamais pris en compte par le noyau système (*kernel*) lorsqu'il doit déterminer quel thread va s'exécuter sur quel processeur.

Responsabilités d'une application

Le cycle de vie appliqué par le système aux applications Windows Store est transparent et connu. Grâce aux nombreux événements accompagnant ce cycle de vie, l'application a les moyens de savoir exactement où elle en est, à quel moment l'utilisateur ou le système la met en veille ou bien de quelle façon ce dernier l'exécute.

Si le code produit par Visual Studio gère la majorité des cas sans aucun développement particulier de la part du programmeur, il est loin de résoudre les cas plus complexes : la mise en place de la persistance des données (pour que l'utilisateur retrouve les données qu'il avait saisies avant que l'application ne soit suspendue puis arrêtée) relève de la responsabilité du développeur. Celui-ci a tous les outils (événements et API) à sa disposition pour mettre en place un mécanisme donnant l'impression que l'application n'a jamais cessé de fonctionner.

Sauvegarde et restauration d'état

Contrairement à une application classique où il n'est utile de sauvegarder son état qu'au moment de son arrêt, une application Windows Store risque au cours de son cycle de vie particulier de passer plusieurs fois par le statut *Suspendue*, puis éventuel-

Copyright © 2013 Eyrolles.

lement par celui de *Terminée* avant d'être relancée de nouveau, et ce, même si l'utilisateur n'en a pas conscience. C'est là qu'intervient la première difficulté pour le développeur, car il a la responsabilité de gérer les mises en pause, les extinctions brusques et les réveils, de façon à ce que cela reste transparent pour l'utilisateur.

Un exemple typique des difficultés soulevées par ce fonctionnement est le cas d'un utilisateur travaillant avec une application Windows Store qui demande de remplir un formulaire. L'utilisateur décide alors d'ouvrir son client mail pour rechercher un courriel dans lequel se trouve l'information qu'il souhaite saisir ou copier.

Le fait d'ouvrir le client mail fait passer la première application Windows Store en arrière-plan ; celle-ci est suspendue par le système au bout de quelques secondes. Il est alors possible que Windows décide de la terminer sans prévenir avant que l'utilisateur ne décide d'y retourner une fois l'information récupérée dans le courriel. Quoi de plus frustrant alors si l'application a oublié son ancien contexte d'utilisation et s'ouvre sur sa page de démarrage, perdant au passage toutes les données déjà saisies par l'utilisateur ?

L'application a ainsi la responsabilité de sauvegarder à la fois sa navigation (l'endroit où l'on se trouve et d'où on vient), mais également les données créées ou mises à jour par l'utilisateur, même non validées (en cours de saisie). Le moment idéal est bien entendu quand l'application est notifiée qu'elle va être suspendue sous peu : elle doit utiliser ce court laps de temps pour sauvegarder son état.

Comme cela a été expliqué, Windows ne donne qu'un temps très court à l'application pour exécuter certaines actions comme l'activation ou la suspension. Pour de nombreuses applications, cela ne pose pas de problème. Cependant, ce temps est parfois trop court s'il y a beaucoup de données dans le contexte à sauvegarder. Il existe donc d'autres stratégies consistant par exemple à sauvegarder en permanence les données utilisateur (au niveau page, voire au niveau champ de saisie), afin que le travail à effectuer dans le gestionnaire de l'événement de suspension soit le plus léger possible et tienne à coup sûr dans le laps de temps très court imparti par le système. Bien entendu, ces sauvegardes plus nombreuses consommeront plus de ressources et chaque développeur, pour chaque cas particulier, devra faire une balance parfois délicate entre sauvegardes fréquentes et consommation de la batterie par son application. Les utilisateurs d'unités mobiles rejettent facilement une application si elle tire trop sur la batterie, l'autonomie étant souvent le problème majeur sur ce type de machines.

RAPPEL Pas d'événement Terminating

L'application ne reçoit pas d'événement lorsqu'elle est terminée par le système puisqu'elle est suspendue, sans possibilité d'accéder aux processeurs qui auraient dû exécuter le code. Il n'est donc pas possible de sauvegarder les données à ce moment-là. Il faut le faire au moment de la notification de l'événement `Suspending` ou de manière régulière pendant que l'application est utilisée.

Copyright © 2013 Eyrolles.

Sauvegarde de l'état applicatif en C#

S'il est possible de sauvegarder régulièrement les données de la page affichée, c'est donc principalement et de préférence l'événement `Suspending` qui doit être utilisé pour cette tâche, car il est le seul garantissant que cela puisse être réalisé avant que l'application ne soit terminée.

```
private async void OnSuspending(object sender, SuspendingEventArgs e)
{
    // Sauvegarde des données
}
```

C'est au sein de cette méthode que doit se trouver la logique de sauvegarde des données et son temps d'exécution ne doit en aucun cas dépasser cinq secondes (valeur indiquée dans la propriété `Deadline` de `SuspendingOperation` proposée par `SuspendingEventArgs`), sous peine de voir le traitement interrompu brutalement par le système qui terminera alors définitivement l'application. Comme les mécanismes de sauvegarde sont asynchrones avec WinRT, cela signifie que la méthode se terminera certainement avant que la sauvegarde ne soit complètement effectuée, indiquant à Windows qu'il peut suspendre tous les threads, y compris celui en charge de la sauvegarde. Pour attendre la terminaison de la gestion d'un événement par du code asynchrone, WinRT propose un mécanisme basé sur l'utilisation d'un objet `Deferral`. Il s'agit d'un objet (récupérable à partir de la méthode `GetDeferral` de la propriété `SuspendingOperation` de `SuspendingEventArgs`) qui est passé au code de sauvegarde asynchrone. Tant que sa méthode `Complete` n'est pas appelée, Windows sait que l'événement n'est pas complètement traité et les threads ne sont donc pas suspendus. Bien évidemment, le délai des cinq secondes reste fatidique et si cette durée est dépassée, Windows terminera le processus de façon abrupte.

```
private async void OnSuspending(object sender, SuspendingEventArgs e)
{
    // il faut indiquer au système qu'un peu de temps est nécessaire
    SuspendingDeferral deferral = e.SuspendingOperation.GetDeferral();

    // Sauvegarde asynchrone des données : la méthode retourne donc
immédiatement ici, mais Windows sait qu'il doit attendre que Complete soit
appelée
    await ...

    // On indique au système que la sauvegarde est terminée
    deferral.Complete();
}
```

Copyright © 2013 Eyrolles.

Les données peuvent être sauvegardées sous deux formes :

- en tant que paramètres d'application, c'est-à-dire des paires clé-valeur. C'est le moyen recommandé pour conserver des paramètres simples ;
- sous forme de fichier(s), dont le format et le contenu sont libres, mais dont le dossier de stockage est limité par le système et reste propre à l'application. Il existe trois emplacements possibles pour stocker ces fichiers :
  - le dossier local, pour stocker les données propres au périphérique sur lequel l'application s'exécute ;
  - le dossier itinérant, pour les données à synchroniser entre les différents périphériques utilisés par le compte Microsoft de l'utilisateur. Il a une taille limite dont la valeur peut être obtenue en récupérant `Windows.Storage.ApplicationData.RoamingStorageQuota` (qui est par défaut de 100 ko) ;
  - le dossier temporaire qui, comme son nom l'indique, stocke des fichiers qui peuvent à tout moment être nettoyés par l'utilisateur ou le système.

Ainsi, selon la nature des données à faire persister, il faut choisir le conteneur le plus adapté :

```
// Récupération des destinations de sauvegarde
Windows.Storage.ApplicationDataContainer parametresLocaux =
Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.ApplicationDataContainer parametresItinerants =
Windows.Storage.ApplicationData.Current.RoamingSettings;
Windows.Storage.StorageFolder dossierLocal =
Windows.Storage.ApplicationData.Current.LocalFolder;

// Sauvegarde des données
parametresLocaux.Values["donneSpecifique"] = maVariableSpecifique;
parametresItinerants.Values["donneeUtilisateur"] = maVariableUtilisateur;
await serialization.SaveDataAsync(dossierLocal, monObjetASerialiser);
```

Utilisation du SuspensionManager

Un point commun à la plupart des applications est la navigation. L'utilisateur est souvent amené à naviguer d'un écran à un autre et il serait désagréable qu'à chaque basculement entre deux applications l'utilisateur perde sa position et doive repartir depuis l'écran d'accueil. C'est donc également une information qu'il est nécessaire de faire persister. Heureusement pour le développeur, les projets Visual Studio Windows Store sont dorénavant fournis avec une classe utilitaire `SuspensionManager` (située dans le dossier `Common` du projet), dont la principale tâche est de simplifier la sauvegarde du contexte de navigation, c'est-à-dire la page où se trouve l'utilisateur ainsi que les données qui s'y trouvent (son état). Utilisé conjointement avec la classe de base `LayoutAwarePage`, classe dont héritent normalement les différentes pages

Copyright © 2013 Eyrolles.

d'une application, le SuspensionManager va, au fil de l'utilisation de l'application et à travers les méthodes OnNavigatedTo et OnNavigatedFrom sauvegarder ou restaurer l'état de la page. C'est cet état que la page déclare/recharge dans les méthodes LoadState et RestoreState.

Dans la pratique, une application doit donc initialiser le SuspensionManager en lui précisant l'objet Frame de l'application, cet objet englobant toute la navigation vers les différentes pages de l'application.

```
SuspensionManager.RegisterFrame(rootFrame, "appFrame");
```

Puis, chaque fois que l'utilisateur ouvre ou quitte une page, le SuspensionManager sauvegarde l'état de la page quittée et restaure celui de la page appelée. Comme il ne connaît pas les données importantes de l'état à conserver, il délègue à chaque page la responsabilité de déclarer elle-même son propre état, dans sa méthode SaveState. De la même façon, lors de la restauration, il fournit simplement l'état désérialisé à la page, et appelle la méthode virtuelle LoadState où la page implémente son propre traitement pour utiliser les données restaurées.

Méthode de LayoutAwarePage

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
            var frameState = SuspensionManager.SessionStateForFrame(this.Frame);
            var pageState = new Dictionary<String, Object>();
            this.SaveState(pageState);
            frameState[_pageKey] = pageState;
}
```

Surcharge de la page héritante

```
protected override void SaveState(Dictionary<String, Object> pageState)
{
      pageState["ObjetASauvegarder"] = "ma valeur";
}
```

D'une façon similaire (bien que pouvant se passer du SuspensionManager), il peut être intéressant de déclencher une action ou une sauvegarde des données dès que l'application n'est plus au premier plan. C'est par exemple le cas des jeux, qu'il est préférable de mettre en pause à la seconde même où ils passent en arrière-plan, pour ainsi sauvegarder l'état sans attendre plusieurs secondes que le système se décide à suspendre l'application.

Copyright © 2013 Eyrolles.

La méthode adéquate consiste à surveiller l'événement `VisibilityChanged` afin de savoir si l'application vient de perdre le focus ou si l'utilisateur vient de revenir dessus :

```
public MaPage()
{
    this.InitializeComponent();
    Window.Current.VisibilityChanged += VisibilityChanged;
}

private void VisibilityChanged(object sender,
Windows.UI.Core.VisibilityChangedEventArgs e)
{
    if (!e.Visible)
    {
        // mise en pause du jeu + sauvegarde des données
    }
    else
    {
        // reprise du jeu
    }
}
```

Sauvegarde de l'état applicatif en JavaScript

Très proche du fonctionnement des applications XAML, le cycle de vie d'une application HTML possède également des statuts *En cours d'exécution*, *Suspendue* et *Terminée*, mais en diffère par les événements déclenchés. Si l'application possède bien une méthode `onactivated`, la suspension passe par la méthode `oncheckpoint`.

Cette dernière est idéale pour sauvegarder les données avant que l'application ne soit terminée.

```
app.oncheckpoint = function (args) {
        // sauvegarde des données
};
```

Tout comme pour les applications XAML, plusieurs emplacements de stockage sont disponibles pour sauvegarder les différents types de données dans le respect de l'isolation imposée par le principe de bac à sable (*sandbox*).

```
app.oncheckpoint = function (args) {

// Récupération des destinations de sauvegarde
var parametresLocaux = Windows.Storage.ApplicationData.current.localSettings;
var parametresItinerants =
Windows.Storage.ApplicationData.current.roamingSettings;
var dossierLocal = Windows.Storage.ApplicationData.current.localFolder;
```

Copyright © 2013 Eyrolles.

```
// Sauvegarde des données
parametresLocaux.values["donneSpecifique"] = maVariableSpecifique;
parametresItinerants.values["donneeUtilisateur"] = maVariableUtilisateur;
serialization.SaveDataAsync(dossierLocal, monObjetASerialiser);
};
```

Ici aussi, le traitement de sauvegarde ne doit pas dépasser 5 secondes, sous peine de voir l'application terminée par le système avant que la sauvegarde n'ait pu aboutir. De plus, pour les méthodes asynchrones, afin d'éviter que le système ne considère de façon anticipée le traitement comme étant terminé, l'utilisation d'un objet `Deferral` est requise, soit en appelant la méthode `suspendingOperation.getDeferral()`, soit en appelant plus simplement la méthode `args.setPromise()` pour englober le traitement asynchrone.

```
app.oncheckpoint = function (args) {
        args.setPromise(sauvegardeAsync());
}
```

Sauvegarde de l'état applicatif en C++

Pour les applications C++, le comportement reste identique et tout se passe dans la méthode `OnSuspending`.

```
void App::OnSuspending(Object^ sender, SuspendingEventArgs^ e)
{
     (void) sender; // Unused parameter

     auto parametresLocaux = Windows::Storage::ApplicationData::Current-
>LocalSettings;
     auto parametresItinerants = Windows::Storage::ApplicationData::Current-
>RoamingSettings;
     auto dossierLocal = Windows::Storage::ApplicationData::Current-
>LocalFolder;

     auto deferral = e->SuspendingOperation->GetDeferral();
     serialization::SaveDataAsync().then([=]()
     {
          deferral->Complete();
     });
}
```

Copyright © 2013 Eyrolles.

Reprise des données

Lorsqu'une application est réveillée, elle repart de l'état dans lequel elle était au moment où elle a été suspendue : les valeurs dans les éléments graphiques et son modèle objet interne sont toujours là. Au moment de sa suspension, son état et ses données sont conservés en mémoire et le système met en pause l'exécution des threads en leur limitant l'accès au processeur. Il n'y a donc aucun traitement particulier à faire, y compris pour les données qui ont été sauvegardées « manuellement » au moment de la suspension.

Néanmoins, plusieurs cas d'utilisation nécessitent de récupérer ces données ou de réaliser un traitement particulier au moment du réveil de l'application. L'application est parfois restée longtemps suspendue et, si elle affiche des données externes (information, météo…) ou possède des connexions actives avec un serveur, cette durée peut impliquer l'obsolescence des données ou des connexions interrompues.

Pour offrir la meilleure expérience utilisateur possible, le développeur doit alors prendre en compte le temps d'inactivité de l'application et décider s'il faut recharger des données plus récentes ou s'assurer que les connexions sont de nouveau actives. Pour cela, il a à sa disposition l'événement `Resuming`, déclenché lorsque l'application repasse au premier plan après avoir été suspendue (mais non terminée), ou bien les événements liés au démarrage standard lorsque celle-ci a été complètement éteinte (`OnLaunched`, `FileActivated`, etc). Comme l'information du temps passé dans l'état suspendu n'est pas fournie, il faut penser à sauvegarder la date et l'heure du dernier rafraîchissement des données dans la gestion de l'événement `Suspending` (dans un paramètre clé-valeur par exemple), et les comparer avec l'heure courante lors du réveil.

La reprise en C#

La première chose à réaliser est de s'abonner à l'événement `Resuming` :

```
public App()
{
      this.InitializeComponent();
      this.Resuming+=App_Resuming;
}
```

Le délégué associé permet alors d'appliquer le traitement de son choix :

```
void App_Resuming(object sender, object e)
{
      // mise à jour des données
}
```

Copyright © 2013 Eyrolles.

La reprise en JavaScript

En JavaScript, comme en C#, il faut commencer par s'abonner à l'événement :

```
Windows.UI.WebUI.WebUIApplication.addEventListener("resuming", reprise, false);
```

Puis on réalise le traitement désiré dans la méthode associée :

```
function reprise() {
        // rafraîchissement des données
}
```

La reprise en C++

Le C++ ne déroge pas à la règle, en nécessitant de déclarer un délégué pour l'événement `Resuming`, puis en consommant ce délégué.

```
App::App()
{
      InitializeComponent();
      Resuming +=
       ref new EventHandler<Platform::Object^>(this, &App::OnResuming);
}

void App::OnResuming(
    _In_ Platform::Object^ /* sender */,
    _In_ Platform::Object^ /* args */
    )
{
   // restauration ou rafraîchissement des données
}
```

L'activation

La suspension, la reprise (`resume`) ou bien le lancement normal d'une application Windows Store via une vignette de l'écran d'accueil facilite la mise en place une stratégie pour la persistance ou le rafraîchissement des données, mais comme cela sera vu dans le chapitre sur les Charmes, cette granularité d'événements n'est pas suffisante lorsque le développeur ne souhaite agir qu'à certains moments précis.

Par exemple, comment faire la différence lorsque l'application est lancée manuellement (et qu'une reprise de données est nécessaire) et un lancement « automatique » via l'utilisation d'un charme de partage ? La solution se trouve dans l'activation, un événement supplémentaire qui n'est déclenché que dans certains cas précis, et différent du lancement standard d'une application.

Copyright © 2013 Eyrolles.

L'activation ne sera que survolée dans ce chapitre, car elle est traitée de façon approfondie dans le chapitre 6 sur l'utilisation des charmes. Toutefois, de façon générale, il est intéressant de savoir qu'une application peut être activée des manières suivantes (liste non exhaustive) :

- à l'ouverture d'un fichier dont l'extension est associée à l'application ;
- par un protocole d'action personnalisée ;
- par le charme de recherche ;
- par le charme de partage ;
- par une notification ;
- par le charme de périphérique (AutoPlay) ;
- lorsque l'utilisateur veut sauvegarder un fichier et choisit l'application comme destination ;
- lorsque l'utilisateur choisit des fichiers qui sont fournis par l'application ;
- lorsque l'application capture des images/vidéos qui proviennent d'une caméra ;
- lorsque l'application gère l'impression...

Figure 5–4
Les différents points d'entrée d'activation disponibles

Ces types de lancement listés dans l'énumération `ActivationKind` vont servir à fournir un comportement différent en fonction du type de lancement et à utiliser les paramètres qui seront automatiquement transmis à l'application.

L'activation en C#

L'activation d'une application via un charme, un fichier ou bien un protocole d'action personnalisée, déclenche l'événement `OnActivated` dont les arguments indiquent de quelle façon l'application est démarrée.

```
protected override void OnActivated(IActivatedEventArgs args)
{
      base.OnActivated(args);

      // Traitement à faire lors de l'activation
      ...
}
```

Copyright © 2013 Eyrolles.

Petite particularité des applications C#, l'activation via l'ouverture d'un fichier dont l'extension est associée à l'application Windows Store se fait à l'aide d'un événement particulier : `OnFileActivated`. Cela fournit une méthode précise pour éventuellement ne pas restaurer de données et effectuer un traitement particulier du fichier passé en paramètre.

```
protected override void OnFileActivated(FileActivatedEventArgs args)
{
      base.OnFileActivated(args);

      if (args.Files.Count > 0) // au moins un fichier
      {
         string nomDuFichier = args.Files[0].Name;
         string cheminDuFichier = args.Files[0].Path;

         // Traitement du fichier
         ...
      }
}
```

L'activation en JavaScript

En JavaScript, l'événement `onactivated` est utilisé aussi bien pour l'activation que pour la reprise, comme vu plus haut :

```
app.addEventListener("activated", function (args) {
      if (args.detail.kind === activation.ActivationKind.shareTarget)
      {
           // activation via le charme de partage
      }
      else {
      ...
      }
}
```

Dans le cas de l'activation d'un fichier, et contrairement aux applications C# qui ont un événement particulier, il suffit de vérifier que le mode d'activation est de type `File` :

```
if (args.detail.kind === activation.ActivationKind.file)
{
      // Activation via un fichier
}
```

Copyright © 2013 Eyrolles.

Il est important de penser à utiliser l'objet `sessionstate` qui a été automatiquement sérialisé lors de la suspension. Celui-ci peut contenir des informations qu'il serait alors intéressant d'utiliser.

```
if (app.sessionState.history) {
      nav.history = app.sessionState.history;
}
```

L'activation en C++

Au sein des applications C++, il est nécessaire de s'abonner à l'événement `OnActivated` pour ensuite effectuer le traitement désiré :

```
void
App::OnActivated(Windows::ApplicationModel::Activation::IActivatedEventArgs^
args)
{
   // Traitement à faire lors de l'activation
}
```

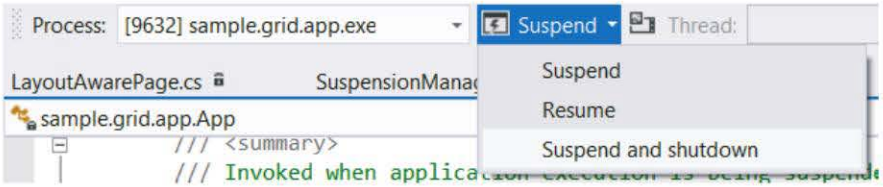
Tout comme les applications C# / VB.Net, les applications C++ possèdent un événement particulier nommé `OnFileActivated` pour gérer le lancement via un fichier.

```
void
App::OnFileActivated(Windows::ApplicationModel::Activation::FileActivatedEventA
rgs^ args)
{
      if (args->Files->Size > 0) // au moins un fichier
    {
      auto nomDuFichier = args->Files->GetAt(0)->Name;
      auto cheminDuFichier = args->Files->GetAt(0)->Path;

      // Traitement du fichier
        ...
   }
}
```

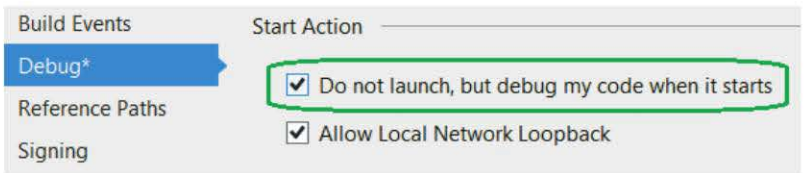
Copyright © 2013 Eyrolles.

ASTUCE Déboguer facilement le cycle de vie

Afin de déboguer les différents états d'une application, il est possible au sein de Visual Studio d'afficher la barre d'outils *Emplacement de débogage* puis de jouer avec les actions forçant la suspension ou le réveil de l'application.



Et pour aider au débogage des lancements via les charmes (recherche et partage par exemple), il est possible dans les propriétés du projet, dans l'onglet *Débogage*, d'indiquer à Visual Studio de déboguer le code sans lancer l'application.



Conclusion

Pour répondre aux exigences des nouveaux mécanismes du système pour la gestion des applications, ou bien de ses nouvelles fonctionnalités comme les charmes, les applications Windows Store possèdent un cycle de vie qui leur est propre. Ce dernier est enrichi de nombreux événements auxquels il est nécessaire de s'abonner, afin de garantir une expérience utilisateur idéale par l'impression que l'application est toujours « vivante », toujours disponible.

Copyright © 2013 Eyrolles.

6

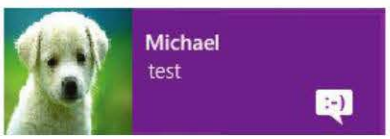
Les vignettes et notifications

Comprendre, créer et contrôler les vignettes et notifications aide à intégrer une application dans l'environnement Windows 8.

Points d'entrée des applications, mais également composants « vivants », les vignettes établissent le premier contact entre une application et l'utilisateur en lui fournissant des informations enrichies sans que celui-ci n'ait à lancer l'application.

Ce chapitre aborde en profondeur la gestion de ces vignettes et les différents moyens disponibles pour soumettre une information à l'utilisateur.

Figure 6–1



Vignettes dynamiques

Même si cela ne constitue qu'une petite partie du langage de design Modern UI, un écran d'accueil rempli de pavés regorgeant d'informations est l'emblème de cette nouvelle interface utilisateur que Microsoft généralise sur ses différentes plates-formes.

Copyright © 2013 Eyrolles.

Figure 6–2
Écrans d'accueil Windows 8 et Windows Phone, utilisant des vignettes

Ces vignettes (souvent appelées tuiles comme avec leur nom anglais *tiles*) constituent un véritable tableau de bord pour l'utilisateur qui peut, d'un seul regard, suivre toutes les informations qui lui tiennent à cœur, comme l'activité de ses proches ou les cours de la bourse. En effet, les vignettes sont bien plus que de classiques icônes : elles contiennent aussi bien du texte que des images, organisés de différentes manières, et surtout, elles peuvent être mises à jour quasiment en temps réel, sans que l'utilisateur n'ait à lancer les applications correspondantes.

Par ailleurs, l'écran d'accueil de Windows peut être personnalisé à l'aide d'une fonction appelée Zoom Sémantique, grâce à laquelle l'utilisateur crée des groupes de vignettes, déplace celles-ci ou en change la taille (largeur simple ou double). Il est aussi possible d'associer une vignette non pas à une application, mais à une fonction de celle-ci, voire à des données particulières. Par exemple, un utilisateur peut afficher sur son écran d'accueil les prévisions météorologiques de la ville où il habite et celles du lieu de ses prochaines vacances dans deux vignettes différentes mais gérées par la même application.

Pour le développeur, une vignette affichant des informations pertinentes concourt au succès de l'application de deux façons. D'abord sur le Windows Store où l'utilisateur trouvera généralement plusieurs applications aux fonctionnalités voisines, celles qui auront les vignettes les plus attrayantes seront généralement les premières téléchargées. Une application de prévisions météorologiques ou d'affichage des cours de la bourse aura certainement moins de chances de succès si elle ne propose pas une vignette mise à jour avec des données pertinentes. Ensuite, une vignette offrant des informations vivantes est une invitation à lancer l'application correspondante par un simple toucher, ce qui va contribuer à l'attachement de l'utilisateur à cette application. Les vignettes ne sont pas autorisées à afficher des publicités, mais elles constituent la meilleure publicité pour une application.

Copyright © 2013 Eyrolles.

Organisation d'une vignette

Une vignette Windows 8 suit un format bien défini, mais accepte des variantes pour s'adapter à l'information qui sera affichée. Il en existe deux tailles : l'une plutôt destinée à afficher un raccourci pour lancer l'application et la seconde, plus large, mieux adaptée à la présentation d'informations vivantes.

Figure 6–3
Les différentes tailles de vignettes disponibles

Une fois que l'application est lancée, elle peut modifier la disposition de sa vignette et afficher différents types d'éléments :

- du texte ;
- des images (téléchargées ou appartenant au package de l'application) ;
- le logo de l'application ;
- un badge (indicateur dynamique qui fournit une information sous forme minimaliste, par exemple le nombre de courriels non lus) ;
- une icône standard.

Ces éléments vont servir à afficher de manière non redondante, claire et minimaliste un maximum d'informations. Outre le titre et l'icône, qui affichent l'identité de l'application, le texte, mais également le badge, vont donner une plus-value à la vignette.

Figure 6–4
Vignette complète avec badge, icône et contenu dynamique

Si les éléments sont pour la plupart entièrement personnalisables, le badge, situé en bas à droite de la vignette (ou bas gauche pour les langages écrits de droite à gauche) est au choix un nombre compris entre 1 et 99 ou une icône choisie parmi une liste de badges prédéfinis et non personnalisables.

Copyright © 2013 Eyrolles.

Tableau 6–1 Liste des types de badges disponibles

Énumération	Rendu graphique	Énumération	Rendu graphique	Énumération	Rendu graphique
`Activity`		`Busy`		`Unavailable`	
`Alert`		`newMessage`		`Error`	
`Available`		`Paused`		`Attention`	
`Away`		`Playing`		`1-100`	

Tous ces éléments sont configurables par le développeur à l'aide de mécanismes de mise à jour détaillés un peu plus loin dans ce chapitre.

Lors de l'installation d'une application, sa vignette a nécessairement la présentation par défaut, c'est-à-dire le logo de l'application, accompagné ou non de son titre. Le logo doit impérativement être une image de 150 × 150 pixels, mais il est possible d'en fournir une version large de 310 × 150 pixels que l'utilisateur sélectionnera s'il la préfère. Une troisième image, de 30 × 30 pixels, est quant à elle utilisée dans les dispositions dynamiques de la vignette. Les trois images, ainsi que le nom de l'application, sont précisés dans le fichier `Package.appxmanifest`, qui peut être modifié via Visual Studio à l'aide d'une interface dédiée (ou manuellement puisqu'il s'agit d'un fichier XML).

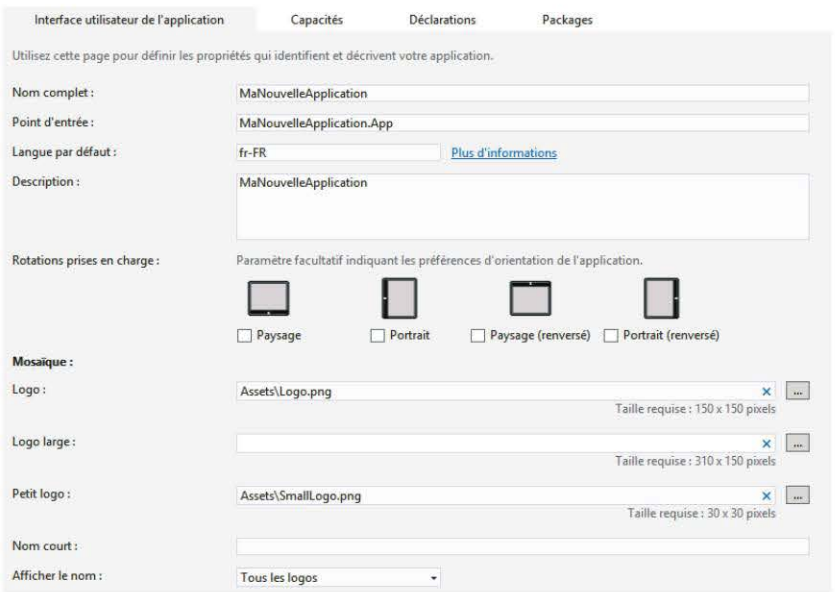


Figure 6–5
Interface de modification du fichier package.appxmanifest

Copyright © 2013 Eyrolles.

Le développeur ne fixe pas librement l'organisation de ces éléments, mais doit choisir parmi un certain nombre de dispositions standardisées. En voici quelques-unes :

TileWideText1 TileWideImageAndText TileWideSmallImageAndText02

Figure 6–6 Exemples de disposition de pavés

L'ensemble des dispositions standardisées est représenté par le type énuméré `Windows.UI.Notifications.TileTemplateType`. La vignette à afficher est ainsi décrite à l'aide d'un document XML qui contient à la fois le nom de la disposition retenue et son paramétrage. C'est ce document XML qui est envoyé au système pour qu'il crée la vignette correspondante. Ainsi, la vignette `TileWideText1` précédente est affichée à l'aide du document XML suivant :

```
<?xml version="1.0" encoding="utf-16"?><tile>
  <visual lang="en-US">
    <binding template="TileWideText01">
      <text id="1">DOW ▼ -89.23</text>
      <text id="2">Last Trade 12801.23</text>
      <text id="3">Last Trade Time Fri 01:03 PM PDT</text>
      <text id="4">Open 12889.55</text>
      <text id="5"></text>
    </binding>
  </visual>
</tile>
```

Le modèle à utiliser est précisé dans l'attribut `template` de l'élément `binding`. Celui-ci expose ensuite des éléments `text` ou `image` qui contiennent les messages ou les URL des images à afficher, ceux-ci étant identifiés à l'aide de leur attribut `id`. Il appartient au développeur de connaître l'organisation du modèle qu'il a choisie. Toujours dans l'exemple du `TileWideText1`, l'organisation du pavé est la suivante.

text id=1
text id=2
text id=3
text id=4
Logo

Copyright © 2013 Eyrolles.

BONNE PRATIQUE Choisir la bonne taille de vignette

Toute application possède une vignette carrée et il est de la responsabilité du développeur de décider si oui ou non, une vignette plus large sera proposée. Le choix des vignettes ne doit pas se faire sur un principe esthétique, mais sur des recommandations faites par Microsoft, pour éviter de voir l'application refusée par le Windows Store en cas de non-respect.

- Les vignettes larges ne doivent être utilisées que si elles sont vivantes et proposent un contenu dynamique. Dans le cas contraire, seule la vignette carrée doit être utilisée. La vignette large ne doit donc être utilisée que si l'information est régulièrement mise à jour (au moins une fois par semaine).
- Même dans le cas d'une vignette dynamique, la petite taille est recommandée si l'information est minimaliste et n'utilise qu'un badge, par exemple, pour afficher un nombre de courriels ou SMS non lus.

La petite vignette proposant moins de place pour afficher l'information, il devient alors nécessaire de bien sélectionner les détails affichés.

Mise à jour d'une vignette

Il revient au développeur de décider, au sein de son application, comment et quand la vignette associée à cette dernière est mise à jour pour former une vignette dynamique. L'affichage de cette vignette se réalise en deux temps.

Création du document XML décrivant son organisation

Le document XML est construit manuellement ou récupéré à l'aide de la méthode `Windows.UI.Notifications.TileUpdateManager.GetTemplateContent()` qui prend en paramètre le modèle dont le template est à récupérer :

```
XmlDocument doc = TileUpdateManager.GetTemplateContent(TileTemplateType.
TileWideText1);
```

Le document ainsi récupéré se manipule à l'aide de toutes les méthodes classiques des documents XML (ici du XPath) ; par exemple, pour modifier la source de l'image d'identifiant « 1 » :

```
var node = doc.DocumentElement.SelectSingleNode("//image[@id=1]/@src");
if (node != null)
{
    node.NodeValue = "http://www.monserveur.com/image1.png";
}
```

TECHNIQUE XPath

XPath est un langage qui sert à naviguer dans un document XML et à en extraire des portions particulières. À l'aide d'un axe de recherche, de filtres de nœuds et de prédicats complémentaires, il repère un ou plusieurs bout(s) de code XML de façon très précise. Il est le langage d'interrogation XML le plus simple et le plus utilisé.

Copyright © 2013 Eyrolles.

Création de la notification et mise à jour de la vignette

Dans un premier temps, il faut créer un objet `TileUpdater` qui sera chargé d'appliquer la modification désirée. Cet objet possède différentes méthodes :

- `Update` applique le document XML au pavé de l'application. C'est la principale méthode de cette classe :

```
var tileNotification = new TileNotification(doc);
TileUpdateManager.CreateTileUpdaterForApplication().Update(tileNotification);
```

- `Clear` supprime toute personnalisation du pavé principal de l'application ;
- `EnableNotificationQueue` active l'affichage en boucle des dernières notifications (utile par exemple dans le cas d'une application affichant un flux d'informations).

Mise à jour du badge

La personnalisation du badge est particulière, car elle n'utilise pas le `TileUpdateManager`, mais son propre schéma XML et également son propre objet de mise à jour : `BadgeUpdateManager`.

Son utilisation se base sur le même principe de création de XML puis de notification de la vignette :

```
XmlDocument badgeXml = new XmlDocument();
badgeXml.LoadXml("<badge value=\"attention\" />");
BadgeNotification badge = new BadgeNotification(badgeXml);
BadgeUpdateManager.CreateBadgeUpdaterForApplication().Update(badge);
```

Dans le cas plus courant de l'affichage d'un nombre précis, il suffit de créer une notification typée comme le montre l'exemple suivant :

```
BadgeNumericNotificationContent badgeContent =
                    new BadgeNumericNotificationContent((uint)number);
BadgeUpdateManager.CreateBadgeUpdaterForApplication().Update(badgeContent
                    .CreateNotification());
```

Enfin, petite précision : pour le cas particulier des vignettes secondaires (abordé plus loin dans ce chapitre), c'est la méthode `CreateBadgeUpdaterForSecondaryTile` qu'il est nécessaire d'appeler :

```
BadgeUpdateManager.CreateBadgeUpdaterForSecondaryTile(ID_DE_MA_VIGNETTE).Update
(badge);
```

Copyright © 2013 Eyrolles.

Le respect des bonnes pratiques

Toute application ayant ses propres fonctionnalités et besoins, les cas d'utilisation des vignettes sont quasi-infinis. Pour prévenir une mauvaise utilisation, Microsoft a pris soin de fournir une liste de recommandations et une liste de mauvais exemples, disponibles à l'emplacement suivant : http://msdn.microsoft.com/en-US/library/windows/apps/hh465403.

Parmi ces bonnes pratiques, on trouve notamment comment choisir correctement le logo, le titre ou les couleurs utilisées sur la vignette, ainsi que tous les points à prendre en compte pour afficher une image, tant pour la positionner que pour optimiser son apparence (redimensionnement, prise en compte des différentes résolutions d'écran).

Toutes ces recommandations sont vérifiées dans la phase de contrôle de l'application préalablement à sa publication sur le Windows Store. Toutefois, elles servent avant tout à fournir la meilleure expérience à l'utilisateur et à garantir la bonne appréciation générale de l'application.

Vignettes secondaires

Il est également possible d'enrichir la présentation d'une application dans le menu de Windows par l'utilisation de vignettes secondaires. Il s'agit d'offrir la possibilité d'épingler des vignettes supplémentaires sur l'écran d'accueil devenant autant de points d'entrée personnalisés dans l'application et agissant ainsi comme des raccourcis facilement identifiables.

Voici des exemples de scénarios d'utilisation de vignettes secondaires :

- informations météo d'une ville particulière ;
- fiche d'un contact et dernières actions de celui-ci ;
- informations sur un flux RSS particulier au sein d'une application servant d'agrégateur ;
- etc.

> **ATTENTION Certification de l'application**
>
> Il est important que la vignette secondaire soit plus qu'un simple raccourci vers une partie de l'application. Elle doit afficher des informations complémentaires et gérer des notifications pour offrir une réelle plus-value. Dans le cas contraire, l'application risque d'être refusée par le Windows Store.

La principale différence avec la vignette principale est qu'il n'est pas permis de créer par programmation une vignette secondaire. Il est simplement possible de proposer à l'utilisateur (via une pop-up) d'en ajouter une, comme le montre la capture d'écran ci-après.

Copyright © 2013 Eyrolles.

Figure 6–7
Formulaire pop-up de création de vignette secondaire

Création d'une vignette secondaire

La création d'une vignette se réalise avec l'objet `SecondaryTile`, puis en appelant la méthode `requestCreateAsync` (ou `requestCreateForSelectionAsync` pour placer la pop-up à un emplacement précis de l'écran). La création de l'objet `SecondaryTile` accepte de nombreux paramètres qui sont dans l'ordre :

- une chaîne représentant l'ID de la vignette ;
- une chaîne représentant le nom court de la vignette ;
- une chaîne représentant le nom d'affichage (`tooltip`) ;
- une chaîne pour les arguments d'activation (voir section suivante) ;
- les `TileOptions`, qui servent à définir le format d'affichage ;
- l'URI du logo pour la petite vignette ;
- l'URI du logo pour la vignette en mode large.

> **ATTENTION Limitations pour l'ID**
>
> Celui-ci doit être unique, ne pas comporter d'espaces ni de caractères spéciaux et être limité à 64 caractères. Si jamais la nouvelle vignette prend un ID déjà utilisé par une autre vignette, celle-ci est remplacée par la nouvelle.

On instancie la vignette :

```
Uri largeLogo = new Uri("ms-appx://" + image);
Uri smallLogo = new Uri("ms-appx://" + image);
```

Copyright © 2013 Eyrolles.

```
SecondaryTile secondaryTile = new SecondaryTile(id, tileName,
String.Format("Mon element - {0}", tileName), null,
TileOptions.ShowNameOnLogo,
smallLogo,
largeLogo);

secondaryTile.ForegroundText = ForegroundText.Dark;
```

Il ne reste plus qu'à appeler la méthode appropriée :

```
bool isPinned = await secondaryTile.RequestCreateAsync();
```

Le résultat indique si oui ou non, l'utilisateur a accepté de créer la vignette et si la création a bien réussi.

Concernant la suppression, il suffit d'appeler la méthode `RequestDeleteAsync` :

```
if (SecondaryTile.Exists(vignetteId))
{
    await new SecondaryTile(vignetteId).RequestDeleteAsync();
}
```

Transmission de contexte

Si les vignettes secondaires ont pour objectif d'afficher des informations complémentaires sur un élément précis de l'application, elles agissent *in fine* comme la vignette principale en lançant l'application lorsque l'utilisateur clique ou touche la vignette.

C'est justement ce lancement particulier qu'il est possible d'exploiter pour ouvrir l'application sur un élément particulier plutôt que sur sa page de démarrage. Le contexte de lancement est basé sur des arguments que transmet la vignette à l'application au moment où elle l'exécute.

Ces paramètres (arguments) sont à définir sous forme d'une chaîne de caractères au moment de la création de la vignette :

```
SecondaryTile secondaryTile = new SecondaryTile(id, tileName,
String.Format("Mon element - {0}", tileName),"Arguments",
TileOptions.ShowNameOnLogo,
smallLogo,
largeLogo);
```

Le paramètre est alors récupérable au sein de l'événement `OnLaunched` qu'il est possible de surcharger au sein du fichier `App.xaml.cs` :

Copyright © 2013 Eyrolles.

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
      if (string.IsNullOrWhiteSpace(args.Arguments) == false)
      {
         // utilisation des arguments
      }
}
```

Toute liberté est laissée au développeur d'utiliser l'argument ou non, de naviguer directement jusqu'à un élément ou bien de déclencher une action particulière.

Bonnes pratiques

Là encore, de nombreuses recommandations sont proposées par Microsoft sur le lien suivant : http://msdn.microsoft.com/en-us/library/windows/apps/hh465398.aspx.

Il y est conseillé d'ajouter un bouton dans l'*appbar* pour proposer le pin de cette seconde vignette. Il y est également indiqué que le bouton doit modifier son affichage suivant que la vignette existe déjà ou non, afin d'éviter à l'utilisateur d'avoir à ouvrir le menu d'accueil pour le vérifier.

De même, les recommandations précisent les nombreux cas de mauvaise utilisation et rappellent qu'il est important au démarrage de l'application de vérifier la présence des vignettes existantes pour s'assurer qu'elles pointent toujours vers des ressources de l'application. Enfin, il est important de noter qu'avec l'option `CopyOnDeployment` (qui copie la vignette entre deux PC synchronisés), certaines informations, comme les canaux de mise à jour (*push*) ou les notifications périodiques, ne sont pas transférées ; il est alors nécessaire de les recréer.

Les vignettes secondaires sont ainsi bien plus que des raccourcis supplémentaires vers l'application, mais bien des extensions de celle-ci, qui augmentent et enrichissent les informations fournies à l'utilisateur, mais également qui proposent des points d'entrée spécialisés dans l'application pour faciliter son utilisation.

Ainsi, en suivant quelques règles de base, le développeur tirera profit de ces vignettes et créera une application offrant à l'utilisateur une expérience enrichie.

Utilisation du Windows Push Notification Service

Les sections précédentes ont montré comment piloter les vignettes dynamiques depuis l'application cliente, mais ces vignettes prennent toute leur ampleur lorsque ces mises à jour sont réalisées sans que l'application ne soit lancée : c'est alors généralement un service internet qui les déclenche lorsque survient un événement concernant l'application

Copyright © 2013 Eyrolles.

ou plus précisément l'application et son utilisateur (mise à jour des horaires d'un vol, réponse d'un adversaire dans une partie d'échecs par correspondance...).

Néanmoins, il faut noter que les *Push Notifications* ne sont pas intrusives, puisque l'utilisateur ne voit les mises à jour des vignettes correspondantes que lorsqu'il affiche son écran d'accueil. Elles ne sont donc pas adaptées au temps réel. Pour ce type de situation il faut plutôt faire appel aux *Toast Notifications*. La différence d'utilisation de ces procédés selon les scénarios est bien illustrée par le comportement des applications standards de Windows Phone : la réception d'un courriel se traduit uniquement par une mise à jour de la vignette correspondante, alors que celle d'un SMS se traduit par une Toast Notification qui est affichée à l'utilisateur même si celui-ci est en train d'utiliser une autre application.

La mise en œuvre de Push Notifications met en jeu quatre composants distincts :

Sur la machine Windows elle-même, l'application ainsi que la couche client de notification :

- **L'application.** Pour recevoir des notifications, l'application doit créer un abonnement. Celui-ci a une durée de vie limitée à 30 jours, mais l'application peut le renouveler à tout moment. Ce mécanisme d'obsolescence sert à réduire le trafic engendré par les applications inutilisées et il suffit qu'une application renouvelle son abonnement à chaque lancement pour continuer à voir ses pavés mis à jour par son service Internet.
- **Le socle client de notification de Windows.** C'est lui qui va prendre en charge l'enregistrement de l'abonnement demandé par l'application auprès du Windows Notification Service, ainsi que la mise à jour du pavé de l'application lorsqu'une notification est reçue.

Sur Internet, le service émettant la notification (la partie serveur liée à l'application) et enfin le Windows Notification Service :

- **Le service émettant la notification.** Ce service contacte le Windows Notification Service pour transmettre les notifications à ses abonnés.
- **Windows Notification Service.** Ce service Internet opéré par Microsoft est chargé de la gestion des abonnements, ainsi que du routage des notifications.

Les échanges nécessaires à l'établissement d'un abonnement, puis à l'émission d'une notification sont présentés dans le diagramme suivant.

Étape 0 : enregistrement du service Internet auprès du Windows Notification Service

Cette première étape consiste à s'enregistrer sur le site https://manage.dev.live.com/build. En effet, le Windows Notification Service ne relayera pas de notifications anonymes pour éviter le spam des machines clientes. Il faut s'authentifier sur le site, puis fournir le nom de l'application et de son éditeur, ce qui permet de récupérer des identifiants. L'un

Copyright © 2013 Eyrolles.

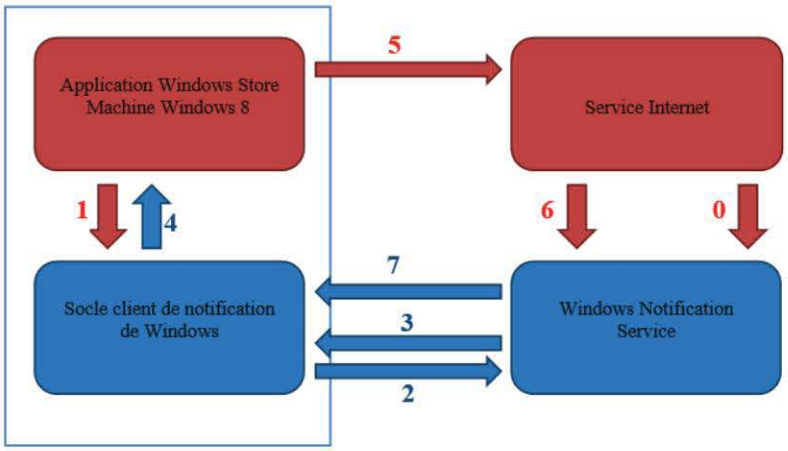


Figure 6–8
Mécanisme de notification Push

d'entre eux doit être ajouté au package de l'application, les autres vont servir à l'authentification du Service Internet de l'application auprès du WNS lors de l'envoi effectif des Push Notifications. Ces derniers identifiants sont assez sensibles puisqu'ils permettent l'authentification du Service Internet de l'application auprès de Windows Live.

Étapes 1 à 5 : création de l'abonnement

L'application doit demander la création de l'abonnement en utilisant la méthode `PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync` :

```
…
var operation = PushNotificationChannelManager
                              .CreatePushNotificationChannelForApplicationAsync();
operation.Completed = OnChannelCreationCompleted;
operation.Start();
…

void OnChannelCreationCompleted(IAsyncOperation<PushNotificationChannel>
operation)
{
   if (operation.Status == AsyncStatus.Completed)
   {
      var pushNotificationChannel = operation.GetResults();
…
```

La demande est transmise au WNS par Windows et la réponse suit le chemin inverse : cette réponse (stockée dans la variable `pushNotificationChannel` dans l'extrait de code précédent) contient à la fois un URI qui devra être utilisé par le Service Internet de l'application et une date de péremption. Cet URI doit donc être transmis à ce Service Internet, qui le conserve, et associé à des données propres à l'utilisateur (par exemple la liste des villes pour lesquelles il désire recevoir les prévisions météorologiques).

Copyright © 2013 Eyrolles.

Comme il s'agit d'une communication purement interne à l'application, elle peut être réalisée à l'aide de toute API de communication ; l'URI est une chaîne de quelques dizaines de caractères, il n'est donc pas encombrant.

Étapes 6 et 7 : émission de la notification

L'envoi d'une notification par le Service Internet de l'application s'effectue à l'aide d'une requête HTTP assez simple : il faut poster le code XML de la notification (décrivant la nouvelle organisation du pavé dynamique) sur l'URI retourné par WNS lors de la création de l'abonnement. Cette requête doit être authentifiée. Pour cela, le Service Internet doit fournir au WND un jeton de sécurité récupéré auprès du service Windows Live à l'aide des identifiants secrets récupérés lors de l'enregistrement du serviceLa requête de récupération du jeton de sécurité s'effectue de la manière suivante :

```
[DataContract]
public class OAuthToken
{
   [DataMember(Name = "access_token")]
   public string AccessToken { get; set; }
   [DataMember(Name = "token_type")]
   public string TokenType { get; set; }
}

public static class AuthenticationHelper
{
   static OAuthToken GetOAuthTokenFromJson(string jsonString)
   {
      using (var ms = new MemoryStream(Encoding.Unicode.GetBytes(jsonString)))
      {
         var ser = new DataContractJsonSerializer(typeof(OAuthToken));
         var oAuthToken = (OAuthToken)ser.ReadObject(ms);
         return oAuthToken;
      }
   }

   public static OAuthToken GetAccessToken(string sid, string secret)
   {
      var urlEncodedSid = HttpUtility.UrlEncode(String.Format("{0}", sid));
      var urlEncodedSecret = HttpUtility.UrlEncode(secret);
      var body =
String.Format("grant_type=client_credentials&client_id={0}&client_secret={1}&sc
ope=notify.windows.com", urlEncodedSid, urlEncodedSecret);
```

Copyright © 2013 Eyrolles.

```
        var client = new WebClient();
        client.Headers.Add("Content-Type", "application/x-www-form-urlencoded");
        string response = client.UploadString("https://login.live.com/
accesstoken.srf", body);
        var oAuthToken = GetOAuthTokenFromJson(response);
        return oAuthToken;
    }
}
```

Une fois cette authentification effectuée, le membre `AccessToken` de la structure `OAuthToken` peut être ajouté aux en-têtes HTTP de la requête d'envoi de la notification :

```
public static void SendNotification(string uri, string notificationXml,
OAuthToken token)
{
   WebRequest webRequest = HttpWebRequest.Create(uri);
   HttpWebRequest request = webRequest as HttpWebRequest;
   webRequest.Method = "POST";
   webRequest.Headers.Add("X-WNS-Type", "wns/tile");
   webRequest.Headers.Add("Authorization",
                          String.Format("Bearer {0}", token.AccessToken));
   byte[] contentInBytes = Encoding.UTF8.GetBytes(notificationXml);
   using (Stream requestStream = webRequest.GetRequestStream())
   {
      requestStream.Write(contentInBytes, 0, contentInBytes.Length);
   }
   HttpWebResponse webResponse = (HttpWebResponse)webRequest.GetResponse();
}
```

REMARQUE Appels synchrones

Ces deux exemples effectuent des requêtes HTTP synchrones pour améliorer leur lisibilité, mais il est préférable de les exécuter plutôt de manière asynchrone, afin d'optimiser l'utilisation du service Internet de l'application.

Conclusion

Ce chapitre a montré à quel point les vignettes sont versatiles, les différents cas dans lesquels les utiliser et les principales bonnes pratiques qui leur sont associées.

Copyright © 2013 Eyrolles.

7

Tirer profit de la puissance des charmes

Windows favorise les interactions entre les applications pour en démultiplier la puissance et l'attrait. Ces interactions sont rendues possibles grâce à des contrats qui permettent aux différentes applications de se mettre d'accord sur la façon de communiquer entre elles et sur le contenu de ces communications.

Ce chapitre présente ces contrats particuliers et la façon de les implémenter.

Figure 7–1

La notion de contrat

Les contrats sont l'un des concepts importants du développement Windows : c'est grâce à eux que des applications différentes interagissent entre elles et avec le système d'exploitation, et ce de façon structurée et organisée. Les ancêtres de ces contrats, comme le presse-papier ou les interfaces COM du Shell, sont des exemples classiques, mais Windows pousse aujourd'hui ces concepts plus loin. Maintenant, ces con-

Copyright © 2013 Eyrolles.

trats plongent au cœur des applications pour offrir à l'utilisateur des options de recherche ou de partage uniformisées et formalisées. Les applications se fondent encore plus dans le système d'exploitation et son interface visuelle ; elles participent dynamiquement à un tout, à une expérience utilisateur homogène.

Un contrat est un accord que l'application « signe » en indiquant qu'elle est capable de répondre à l'interaction ciblée par celui-ci en appliquant les méthodologies, les prérequis et les contraintes attendus.

Ces contrats prennent différentes formes. Les principaux sont utilisés dans le cadre des « charmes », ces actions situées dans la barre droite de l'écran qui apparait lorsque la souris atteint un coin à droite de l'écran ou que le côté droit de l'écran tactile est touché d'un doigt. Ces charmes sont l'une des principales nouveautés ergonomiques introduites avec Windows 8.

On distingue plusieurs types de contrat.

- Le contrat de *partage* sert à envoyer ou recevoir des données sans connaître le destinataire ou la source.
- Le contrat de *recherche* met à disposition l'interface de recherche du système d'exploitation pour chercher parmi les données d'une application.
- Le contrat de *paramètres* affiche un écran standardisé et rapide d'accès pour configurer les paramètres d'une application.
- Le contrat *PlayTo* permet à l'application de partager des données avec des périphériques comme la Xbox.
- Le contrat *Sélection de fichier* utilise les fenêtres du système d'exploitation pour parcourir les dossiers.

La mise en place de ces contrats au sein d'une application est souvent simplifiée au maximum, mais apporte surtout une plus-value en termes de fonctionnalités et d'intégration avec le système ou les autres applications. Néanmoins, chaque contrat a sa propre implémentation et c'est ce qui est étudié en détail dans la suite de ce chapitre.

Le charme de partage

Le partage est aujourd'hui la force même de l'informatique. Les applications partagent entre elles via des messages ou des canaux de transmission et ont toutes un point commun : le communiquant connaît le destinataire. Cela signifie que les deux applications se connaissent et « parlent le même langage » pour comprendre les données qui sont transmises. Dans le cas de Windows 8, où de nombreuses applications différentes peuvent être téléchargées et installées, il était nécessaire de trouver un moyen de passer outre ce prérequis.

Copyright © 2013 Eyrolles.

Pour cela a été mis en place le contrat de partage. Ce mécanisme consiste à préciser les formats qu'une application sait gérer ; ainsi, le système est en mesure de lister automatiquement les applications sachant gérer ce format et pouvant répondre à la demande de partage. De cette façon, une application d'affichage de nouvelles (news) peut proposer une option de partage de l'URL de l'information affichée. En se reposant sur le charme de partage et son contrat, toute application sachant gérer une URL pourra alors automatiquement y avoir accès et le système d'exploitation saura proposer à l'utilisateur cette sélection d'applications compatibles. Cette liste contiendrait par exemple :

- Internet Explorer pour ouvrir la page web ;
- Twitter pour l'enregistrer sur son compte après l'avoir transformée en URL courte ;
- Facebook pour la partager sur son profil auprès de ses amis ;
- etc.

L'application source, qui n'a qu'à partager son URL sans se soucier de la façon dont sera utilisé le lien, accroît ainsi ses fonctionnalités en tirant parti d'autres applications installées dont elle n'a pas même connaissance. Ainsi, plus les applications installées sont nombreuses, plus les possibilités de partage sont importantes. Cela aboutit à disposer d'une galaxie d'applications sachant communiquer entre elles à l'aide d'un langage unique : le contrat de partage.

Une application peut jouer le rôle de source (elle partage ses données) ou de cible (elle accepte des données externes), voire les deux rôles à la fois, et pour plusieurs formats différents en même temps. L'application désirant participer au charme de partage (comme cela est précisé plus loin) doit enregistrer auprès du système les formats de données qu'elle est prête à recevoir. La figure suivante présente une vue globale du fonctionnement du partage au sein des applications Windows Store.

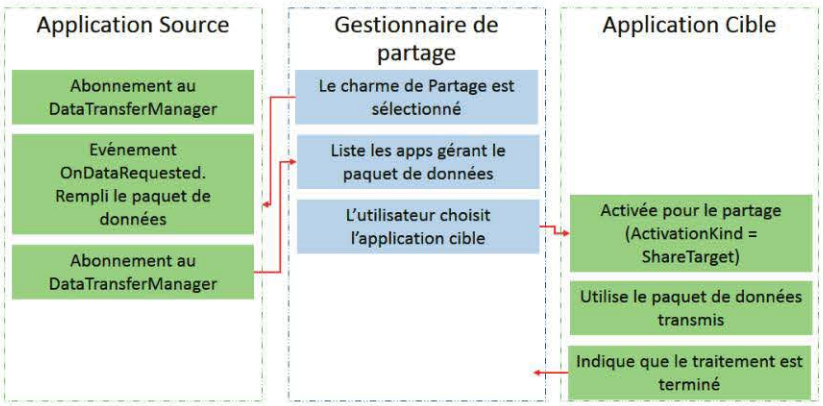


Figure 7–2
Mécanisme de l'utilisation du charme de Partage

Copyright © 2013 Eyrolles.

Source de partage

Une application informatique sert avant tout à brasser des informations, voire à en produire de nouvelles. Quelle que soit l'application que le développeur crée, il est probable qu'à un moment donné, elle nécessite de traiter une information quelconque. Or, ces informations intéressent probablement d'autres applications et méritent d'être partagées et réutilisées afin de proposer une fonctionnalité supplémentaire à l'utilisateur. Qu'il s'agisse d'une application de gestion de photos qui partage une image pour qu'elle soit publiée sur Internet ou retouchée, d'une application de traitement de texte qui partage un paragraphe pour qu'il soit vérifié par une application de correction orthographique ou envoyé par courriel, il existe potentiellement pour chaque donnée un besoin auquel une application externe est en mesure de répondre si l'information est partagée. C'est pourquoi il est nécessaire et important d'avoir le réflexe de partager les données de ses applications, même si le besoin ne semble pas immédiat. C'est à l'utilisateur de décider si oui ou non la donnée lui sera utile via une autre application.

Dans la pratique, les échanges passent par un conteneur de données (`DataPackage`) qui contient toutes les données à transférer d'une application à une autre. Ce conteneur peut gérer les formats suivants :

- texte clair ;
- HTML ;
- texte formaté (RTF) ;
- images au format bitmap ;
- URI (Uniform Resource Identifier) de fichier ;
- format personnalisé défini par le développeur.

Un conteneur contient souvent plusieurs de ces formats à la fois, y compris pour la même donnée, et ce, afin de partager l'information avec un maximum d'applications compatibles.

La première étape consiste à récupérer une instance du gestionnaire de partage que le système fournit à l'application, puis à s'abonner à l'événement `DataRequested` déclenché lorsque l'utilisateur active le charme de partage.

Exemple en C#

```
datatransferManager = DataTransferManager.GetForCurrentView();
datatransferManager.DataRequested += new TypedEventHandler<DataTransferManager,
                                        DataRequestedEventArgs>(this.DataRequested);

...
```

Copyright © 2013 Eyrolles.

```
private void DataRequested(DataTransferManager sender, DataRequestedEventArgs e)
{
   // partage des données vers l'extérieur
}
```

Exemple en JavaScript

```
var dataTransferManager =
Windows.ApplicationModel.DataTransfer.DataTransferManager.getForCurrentView();
dataTransferManager.addEventListener("datarequested", function (e) {
    // partage des données
});
```

Exemple en C++

```
DataTransferManager^ dataTransferManager =
DataTransferManager::GetForCurrentView();
auto dataRequestedToken = dataTransferManager->DataRequested
                        += ref new TypedEventHandler<DataTransferManager^,
    DataRequestedEventArgs^>(this, &MainPage::Partage);
…
void MainPage::Partage(DataTransferManager^ sender, DataRequestedEventArgs^ e)
{
    // partage des données
}
```

Contrairement aux anciennes applications, qui poussaient leurs données vers un contenant (presse-papier, *provider* de recherche, etc.), les applications Windows Store s'enregistrent auprès du système et attendent que celui-ci réclame une donnée pour la fournir. Une fois l'application enregistrée pour le partage, elle doit s'attendre à être sollicitée et à préparer le paquet qu'elle doit mettre à disposition du gestionnaire de partage. Pour cela, il suffit d'utiliser la demande de données transmise à l'événement `OnDataRequested` et de remplir les champs que l'application sait reconnaître. L'exemple le plus simple est sûrement le partage de texte, pour lequel il suffit d'appeler la méthode `SetText()` après avoir défini un titre et une description optionnelle.

Exemple en C#

```
DataRequest request = args.Request;
request.Data.Properties.Title = "titre informatif";
request.Data.Properties.Description = "description de la donnée";
request.Data.SetText("le texte à partager");
```

Le résultat est affiché sur la figure suivante avec l'application Courrier prête à recevoir les données partagées.

Copyright © 2013 Eyrolles.

Figure 7–3
Barre de partage et exemple d'application cible

OPTIMISATION Traitement asynchrone et partage

La préparation du paquet de données peut requérir l'appel à des méthodes asynchrones. Tout comme dans le cas de la suspension (chapitre 5), un objet `Deferral` doit être demandé pour indiquer au système que le traitement asynchrone n'est pas encore terminé. Sans cela, l'application risque de partager un paquet vide, car pas encore rempli au moment de sa transmission.

Partager autre chose que du texte

Bien évidemment, il est possible de partager de nombreux autres formats de données que du texte brut. Voici des exemples montrant les lignes de code particulières à mettre en place pour chaque type de données. Il est possible de partager plusieurs types de données à la fois lors d'une unique opération de partage. Par exemple, un document texte RTF sera ainsi partagé tel quel ou alors sous forme de texte brut ou de code HTML, ce qui élargit considérablement le spectre des scénarios d'utilisations possibles de cette donnée (ainsi que le nombre d'applications à même de l'utiliser).

Partage en C#

```
DataRequest request = args.Request;
request.Data.Properties.Title = "titre informatif";
request.Data.Properties.Description = "description de la donnée";

// partage de texte
request.Data.SetText("le texte à partager");

// partage de HTML
string html = "<b>le texte à <u>partager</u></b>" ;
string texteBienFormatte = HtmlFormatHelper.CreateHtmlFormat(html);
request.Data.SetHtmlFormat(texteBienFormatte);

// partage de lien
request.Data.SetUri(new Uri("http://www.monlien.com"));

// partage d'image (une miniature de l'image peut être associée)
StorageFile fichierImage = await
Package.Current.InstalledLocation.GetFileAsync("monimage.png");
```

Copyright © 2013 Eyrolles.

```
request.Data.Properties.Thumbnail =
RandomAccessStreamReference.CreateFromFile(fichierImage);
request.Data.SetBitmap(RandomAccessStreamReference
                                              .CreateFromFile(fichierImage));

// partage de fichiers
StorageFile fichier =
await Package.Current.InstalledLocation.GetFileAsync("monfichier.txt");
List<IStorageItem> storageItems = new List<IStorageItem>();
storageItems.Add(fichier);
request.Data.SetStorageItems(storageItems);
```

Partage en JavaScript

```
var request = args.Request;
request.data.properties.Title = "titre informatif";
request.data.properties.Description = "description de la donnée";

// partage de texte
request.data.setText("le texte à partager");

// partage de HTML
var texteHtml = "<b>le texte à <u>partager</u></b>";
    // appel de createHtmlFormat pour ajouter les en-tetes HTML dont a besoin le
système
    var texteBienFormatté = Windows.ApplicationModel
                    .DataTransfer.HtmlFormatHelper.createHtmlFormat(texteHtml);
    request.data.setHtmlFormat(texteBienFormatté);

// partage de lien
    request.data.SetUri(new Windows.Foundation.Uri("http://www.monlien.com"));

// partage d'image (une miniature de l'image peut être associée)
    Windows.ApplicationModel.Package.current.installedLocation
                    .getFileAsync("fichierImage.png").then(function (file) {
        request.data.properties.thumbnail = Windows.Storage.Streams
                    .RandomAccessStreamReference.createFromFile(file);
        request.data.setBitmap(Windows.Storage.Streams
                    .RandomAccessStreamReference.createFromFile(file));
    });

// partage de fichiers
    Windows.ApplicationModel.Package.current.installedLocation
                    .getFileAsync("monfichier.txt").then(function (file) {
        request.data.setStorageItems([file]);
    });
```

Copyright © 2013 Eyrolles.

Partage en C++

```
DataRequest^ request = e->Request;
request->Data->Properties->Title = "mon titre informatif";
request->Data->Properties->Description = "description de la donnée";

// partage de texte
request->Data->SetText("le texte à partager");

// partage de HTML
String^ html = "<b>le texte à <u>partager</u></b>";
String^ texteBienFormatte = HtmlFormatHelper::CreateHtmlFormat(html);
request->Data->SetHtmlFormat(texteBienFormatte);

// partage de lien
request->Data->SetUri(ref new Uri("http://www.monlien.com"));

// partage d'image (une miniature de l'image peut être associée)
create_task(Package::Current->InstalledLocation-
>GetFileAsync("monimage.png")).then(
        [this, request](task<StorageFile^> file)
{
      request->Data-
>SetBitmap(RandomAccessStreamReference::CreateFromFile(file.get()));
});

// partage de fichiers
create_task(Package::Current->InstalledLocation-
>GetFileAsync("monfichier.txt")).then(
        [this, request](task<StorageFile^> fileTask)
{
         auto fichier = fileTask.get();
        auto storageItems = ref new Vector<IStorageItem^>();
        storageItems->Append(fichier);
        request->Data->SetStorageItems(storageItems);
});
```

Déclenchement du partage

Du fait de la nouveauté de ces mécanismes, il est fort probable que l'utilisateur n'ait pas le réflexe d'ouvrir régulièrement le charme de Partage pour voir si quelque chose est partageable. Il est alors intéressant d'inclure au sein de son application un bouton ou un menu activant le partage.

Dans le code gérant le clic de ce bouton, il suffit d'appeler la méthode `ShowShareUI()` pour simuler le fait que l'utilisateur a cliqué sur le charme de Partage.

Copyright © 2013 Eyrolles.

Exemple en C#

```
protected void ShowUIButton_Click(object sender, RoutedEventArgs e)
{
       Windows.ApplicationModel.DataTransferManager.ShowShareUI();
}
```

Exemple en JavaScript

```
function showShareUI() {
        Windows.ApplicationModel.DataTransfer.DataTransferManager
                                    .showShareUI();
}
```

Exemple en C++

```
void SharePage::ShowUIButton_Click(Platform::Object^ sender,
                                    Windows::UI::Xaml::RoutedEventArgs^ e)
{
    Windows::ApplicationModel::DataTransfer::DataTransferManager::ShowShareUI();
}
```

Cible de partage

Pour que le partage de données fonctionne, il faut bien évidemment des applications émettrices, mais aussi des applications réceptrices. À première vue, leurs rôles semblent relativement symétriques, mais en réalité il n'en est rien, ni en termes d'implémentation, ni surtout en termes de visibilité. En effet, une application capable de recevoir plusieurs formats de données sera proposée comme potentiel réceptacle à chaque fois que le charme de partage sera appelé pour un de ces formats.

Si partager des données est simple, accepter des données externes est toujours plus délicat. L'application qui émet envoie ce qu'elle possède et c'est tout. L'application qui reçoit doit valider avec attention la « qualité » des données reçues et doit savoir les intégrer correctement à son fonctionnement. C'est pour cela que l'on trouve beaucoup d'applications sources de partage et beaucoup moins offrant d'être cibles. Si Windows simplifie les échanges, il faut être conscient des limites de ceux-ci. Bien taillé pour partager un Tweet, une photo ou le texte d'un SMS, le charme de partage n'apporte pas de solution miracle aux applications qui sortent de ce cadre standard (réseaux sociaux, affichage d'images...). Par exemple, une application gérant des recettes de cuisine pourra très facilement être source de partage, car transformer une fiche en texte brut ou en HTML ne pose aucun problème ; mais qu'en est-il si elle désire être cible de partage ? Comment recréer proprement une fiche recette si les données reçues

Copyright © 2013 Eyrolles.

ne sont pas segmentées en champs identifiables (titre, difficulté de la recette, prix moyen par personne…) ? Il s'agit d'autant de challenges à relever par le développeur afin de créer une application attrayante en termes de puissance et de fonctionnalités.

Du point de vue purement technique, pour être une cible de partage, une application doit simplement déclarer au système d'exploitation qu'elle accepte le partage et quels formats elle reconnaît. Elle doit en outre s'abonner à un événement pour réagir au moment où elle sera sollicitée. Le développeur doit ainsi commencer par ouvrir le manifeste de l'application dans Visual Studio, se rendre dans l'onglet *Déclaration* et ajouter une déclaration *Cible du partage*. Il peut alors déclarer les types et les formats de fichiers qu'il est prêt à recevoir. C'est sur la base de ces filtres que l'application sera proposée comme cible au moment du partage.

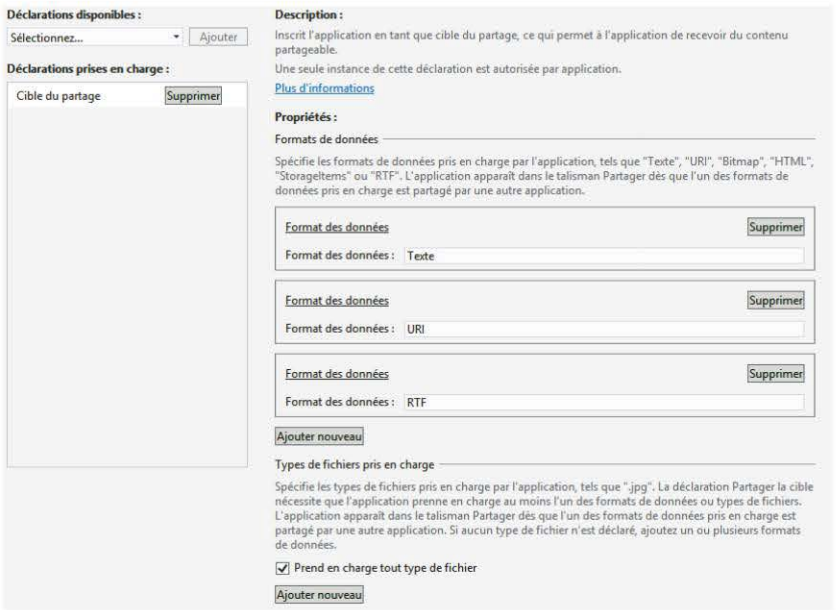


Figure 7–4
Manifeste et déclaration d'utilisation du charme de partage

Une fois l'application déclarée comme cible de partage, il est nécessaire d'écouter l'événement `ShareTarget` pour positionner la réception et le traitement de la donnée enregistrée.

Traitement de la donnée en C#

```
protected override async void
OnShareTargetActivated(Windows.ApplicationModel.Activation
                              .ShareTargetActivatedEventArgs args)
        {
                ShareOperation shareOperation = args.ShareOperation;
```

Copyright © 2013 Eyrolles.

```
            if (shareOperation.Data.Contains(StandardDataFormats.Text))
            {
                string textePartage =
                            await shareOperation.Data.GetTextAsync();

                if (!String.IsNullOrWhiteSpace(textePartage))
                {
                   // utilisation de la donnée reçue
                }
            }

            shareOperation.ReportCompleted(); // le traitement est terminé
        }
```

BONNE PRATIQUE **Blocage de l'application source**

Le traitement des données peut prendre un certain temps. Or, le paquet de données partagées reste « vivant » dans l'application source jusqu'à ce que le partage soit terminé (`ReportCompleted`). Si l'intégration des données reçues prend du temps, l'application source reste active potentiellement pour rien tout en consommant des ressources qui ne peuvent que ralentir encore plus le traitement de réception. Il est conseillé d'appeler la méthode `ReportDataReceived` dès que les données sont reçues pour libérer la source au plus vite. Si un traitement doit être effectué sur ces données, il est préférable de le faire après en avoir accusé réception.

Traitement de la donnée en JavaScript

```
app.onactivated = function (args) {
    if (args.detail.kind === Windows.ApplicationModel.Activation
                                    .ActivationKind.shareTarget) {
        var shareOperation = args.detail.shareOperation;
        if (shareOperation.data.contains(Windows.ApplicationModel
                                    .DataTransfer.StandardDataFormats.text)) {
            shareOperation.data.getTextAsync().done(function (text) {
                // traitement
            });
        }

        shareOperation.reportCompleted();
    }
}
```

Copyright © 2013 Eyrolles.

Traitement de la donnée en C++

```
void App::OnShareTargetActivated(Windows::ApplicationModel::
                        Activation::ShareTargetActivatedEventArgs^ args)
{
      auto shareOperation = args->ShareOperation;

      if (shareOperation->Data->Contains(StandardDataFormats::Text))
      {
          create_task(shareOperation->Data->GetTextAsync())
                                                    .then([this](String^ text)
         {
             // traitement
         });
      }

      shareOperation->ReportCompleted();
}
```

ATTENTION Données dangereuses

Comme expliqué en introduction, une cible de partage peut recevoir des données provenant de n'importe quelle source, données dont la « qualité » n'est pas garantie. Une cible de partage doit absolument considérer ces données comme potentiellement dangereuses et porter une attention particulière à leur validation avant de les intégrer à son propre fonctionnement. Ainsi, une URL reçue n'est pas exécutée directement, du HTML partagé n'est pas affiché sans être filtré de tout code et un texte partagé n'est pas enregistré en base de données sans avoir été préalablement contrôlé, notamment contre une éventuelle tentative d'injection SQL.

Le charme de recherche

Aujourd'hui, les applications qui gèrent du contenu sont pour la plupart capables de traiter un grand nombre d'informations et il est bien souvent difficile pour l'utilisateur de retrouver rapidement ce qu'il cherche. C'est pour cela que nombre d'entre elles, tout comme le système d'exploitation, implémentent un module de recherche qui leur est propre. C'est justement ce module propre à chaque application que le charme de recherche permet d'exposer au sein d'un mécanisme de recherche global et centralisé. L'application reste maîtresse des données qu'elle gère et de la façon dont elle fait la recherche ; le charme a pour simple objectif de fournir une interface unique pour que l'utilisateur cherche simultanément dans toutes les applications ce qu'il souhaite trouver.

Copyright © 2013 Eyrolles.

C'est une avancée gigantesque par rapport à la recherche intégrée aux versions plus anciennes de Windows, qui se bornait à chercher des noms de fichier ou du contenu indexé pour des types de fichiers bien connus (Word, Excel, images possédant des Exif...), laissant de côté le potentiel incroyable que représente le contenu géré par les applications. S'il y a un charme que toute application se doit de prendre en charge, c'est bien le charme de recherche !

Ergonomiquement, le module de recherche apparaît sur la droite de l'écran lorsque l'utilisateur a cliqué sur le charme correspondant.

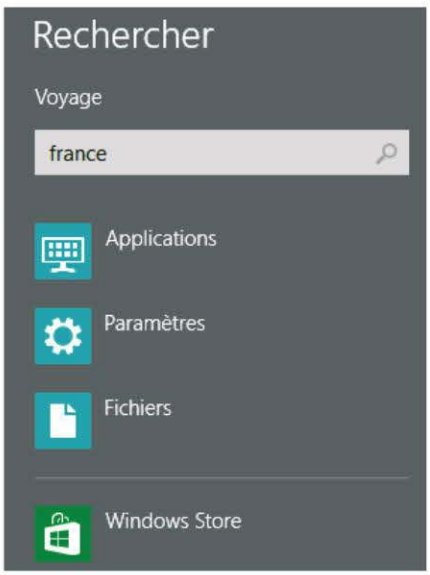


Figure 7–5
Module de recherche

Du point de vue du code, trois opérations sont nécessaires :

- préciser que l'application est capable d'utiliser le charme de recherche ;
- récupérer une référence vers la barre de recherche que le système fournit à l'application ;
- s'abonner à l'événement déclenché lorsque l'utilisateur clique sur le bouton *Rechercher*.

La première étape est relativement simple, puisqu'il suffit d'ouvrir le manifeste de l'application dans Visual Studio, d'ouvrir l'onglet *Déclarations* et d'ajouter une déclaration de recherche.

Une omission de cette déclaration déclenche une exception au moment de l'exécution et peut faire échouer la validation de l'application lors de sa publication sur le Windows Store.

Copyright © 2013 Eyrolles.

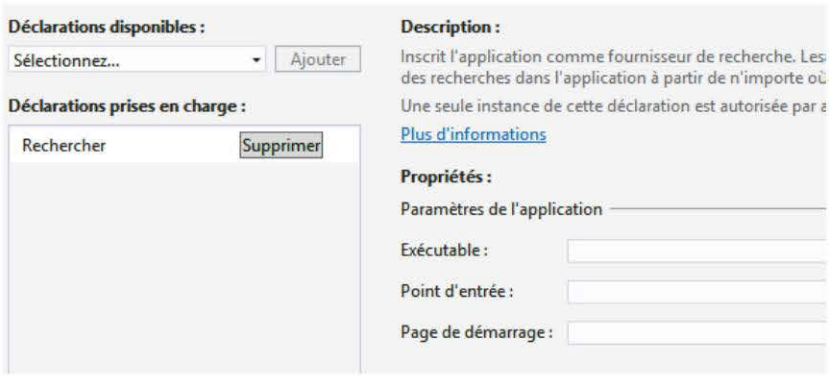


Figure 7–6
Déclarer que l'application utilise le charme de recherche

La seconde étape consiste à appeler la méthode `GetForCurrentView()` de l'objet `SearchPane`. La dernière étape consiste à s'abonner à l'événement `OnQuerySubmitted` de l'objet retourné par la méthode. Lorsque ce dernier est déclenché, il suffit alors d'effectuer une recherche interne à l'application à l'aide du terme saisi par l'utilisateur et d'afficher les résultats sous la forme de son choix, au sein de l'application.

Exemple en C#

```
privateSearchPane searchPane;

public MonConstructeur()
{
    this.searchPane = SearchPane.GetForCurrentView();
    this.searchPane.QuerySubmitted += new TypedEventHandler<SearchPane,
                    SearchPaneQuerySubmittedEventArgs>(this.OnQuerySubmitted);
}
public void OnQuerySubmitted(object sender, SearchPaneQuerySubmittedEventArgs
args)
{
    // recherche et affichage des résultats
}
```

Exemple en JavaScript

```
Windows.ApplicationModel.Search.SearchPane.getForCurrentView().onquerysubmitted
= function (eventObject) {

    var queryText = eventObject.queryText; // terme recherché

    // recherche et affichage des résultats
};
```

Copyright © 2013 Eyrolles.

Exemple en C++

```
void App::OnWindowCreated(WindowCreatedEventArgs^ pArgs)
{
    SearchPane^ searchPane = SearchPane::GetForCurrentView();
    searchPane->QuerySubmitted += ref new TypedEventHandler<SearchPane^,
            SearchPaneQuerySubmittedEventArgs^>(this, &App::OnQuerySubmitted);
}

void App::OnQuerySubmitted(SearchPane^ sender,
                                SearchPaneQuerySubmittedEventArgs^ pArgs)
{
    auto queryText = pArgs->QueryText;// terme recherché
      // recherche et affichage des résultats
}
```

La suggestion de résultat

Depuis de nombreuses années maintenant, les moteurs de recherche offrent pour la plupart une suggestion de résultat ou une aide à la saisie, afin d'aider à trouver des données dont l'orthographe est incertaine. Cette aide à la saisie, basée sur des données réelles, augmente la probabilité pour l'utilisateur de trouver ce qu'il recherche.

L'aide à la saisie consiste à réaliser une recherche anticipée pendant la saisie de l'utilisateur. Cela sert à lui proposer des termes de recherche correspondant à des résultats existants. Il peut alors cliquer sur ces propositions pour effectuer la recherche correspondante avec la garantie d'obtenir au moins un résultat correspondant à son critère de recherche. Pour ce faire, il est nécessaire d'intercepter le moment où l'utilisateur tape sa recherche (avant qu'il ne clique sur le bouton *Rechercher*) en s'abonnant à l'événement `SuggestionsRequested`, puis de remplir des suggestions de recherche qui sont automatiquement affichées par le système sous la boîte de recherche.

Suggestion de résultat en C#

```
this.searchPane.SuggestionsRequested += searchPane_SuggestionsRequested;

...

void searchPane_SuggestionsRequested(SearchPane sender,
SearchPaneSuggestionsRequestedEventArgs args)
{
    int searchPaneMaxSuggestions = 5; // le panneau de recherche affiche au
maximum 5 résultats

    foreach (MonObject suggestion in this.ListItems)
    {
```

Copyright © 2013 Eyrolles.

```
        if (suggestion.Title.StartsWith(args.QueryText,
                                    StringComparison.CurrentCultureIgnoreCase))
        {
            // ajoute la suggestion à la collection
            args.Request.SearchSuggestionCollection
                                    .AppendQuerySuggestion(suggestion.Title);
        }

        // s'arrête si plus de 5 éléments car ils ne seront pas affichés
        if (args.Request.SearchSuggestionCollection.Size
                                    >= searchPaneMaxSuggestions)
        {
            break;
        }
    }
}
```

Suggestion de résultat en JavaScript

```
Windows.ApplicationModel.Search.SearchPane.getForCurrentView()
                            .onsuggestionsrequested = function (eventObject) {
        var queryText = eventObject.queryText;

        var searchPaneMaxSuggestions = 5;

        for (var i = 0; i < mesObjets.length; i++) {
            if (mesObjets[i].toLowerCase() === queryText)
            {
                suggestionRequest.searchSuggestionCollection
                            .appendQuerySuggestion(mesObjets[i]);
            }

            if (suggestionRequest.searchSuggestionCollection.size
                            === searchPaneMaxSuggestions ) {
                break;
            }
        }
    }
```

Suggestion de résultat en C++

```
 searchPane->QuerySubmitted += ref new TypedEventHandler<SearchPane^,
           SearchPaneQuerySubmittedEventArgs^>(this, &App::OnQuerySubmitted);
…
void App::OnQuerySubmitted(SearchPane^ sender,
                           SearchPaneQuerySubmittedEventArgs^ pArgs)
{
      auto queryText = pArgs->QueryText;
```

Copyright © 2013 Eyrolles.

```
        int searchPaneMaxSuggestions = 5;

        for each(wstring monItem in mesObjets)
    {
        if (monItem.find(queryText) == 0)
        {
            request->SearchSuggestionCollection->AppendQuerySuggestion(
                        ref new String(const_cast<wchar_t*>(monItem.c_str())));

            if (request->SearchSuggestionCollection->Size
                                >= searchPaneMaxSuggestions)
            {
                break;
            }
        }
    }
}
```

Une fois le traitement effectué, les résultats sont affichés à l'utilisateur.

Figure 7–7
Suggestion de recherche

Il est également possible de mettre en page les suggestions en utilisant des séparateurs (`AppendSearchSeparator`) ou bien en utilisant la méthode `AppendResultSuggestion` qui affiche des résultats accompagnés d'images.

Les templates de contrat de recherche

L'abonnement au charme de recherche est l'étape la plus délicate de l'implémentation. Cette tâche consiste à afficher les résultats d'une recherche dans une fenêtre adjacente. Idéalement, le fonctionnement d'une telle fenêtre doit être anticipé de manière à afficher les résultats et gérer les cas où il n'y en a aucun, tout en tenant compte de la suspension (voir chapitre 5), le tout en proposant un retour en arrière sur la page précédente.

Tout ceci peut prendre un peu de temps à coder et à vérifier. C'est pourquoi il est possible de s'aider d'un template de contrat de recherche, qui produira automatiquement le squelette de code nécessaire. Dans Visual Studio, il suffit de cliquer sur son projet et sur le menu *Ajouter>Nouvel élément*, puis de choisir *Contrat de Recherche*.

Copyright © 2013 Eyrolles.

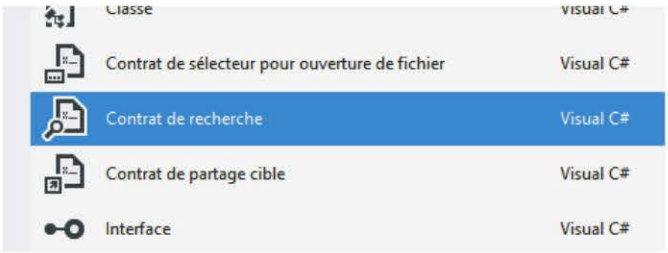


Figure 7–8
Ajouter un contrat de recherche

Cela a pour effet de modifier la classe de démarrage pour s'abonner à la recherche et de créer une fenêtre pré-designée, prête à afficher des résultats. Elle ne demande qu'à être personnalisée pour afficher les résultats comme souhaité. À l'instar des projets templates d'application, cet élément a le mérite de faire gagner du temps pour laisser le développeur se concentrer sur les tâches importantes et spécifiques à l'application.

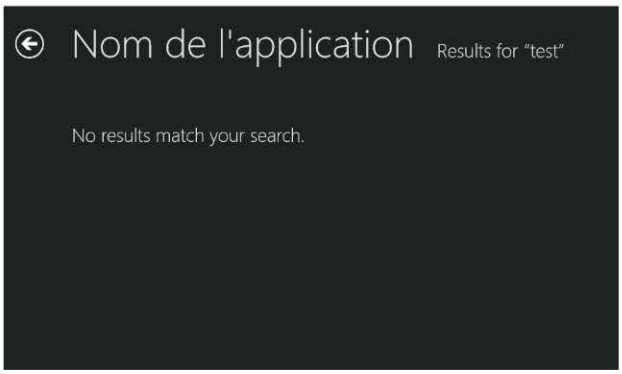


Figure 7–9
Fenêtre d'affichage des résultats avant personnalisation

Le charme de paramétrage

Le plus souvent, les applications proposent à leurs utilisateurs un écran destiné à paramétrer les options de fonctionnement, ainsi qu'un écran d'à-propos. Ces écrans prennent souvent plusieurs formes et surtout sont accessibles de manières différentes pour chaque application, au risque de dérouter l'utilisateur. Windows apporte une solution à ce problème en incluant dans chaque application Windows Store une barre latérale servant de panneau de configuration applicatif et contenant automatiquement certaines informations, comme le nom de l'éditeur de l'application et la liste des fonctionnalités techniques internes de l'application (accès à la caméra, envoi d'information sur le Web...).

Mieux encore, le système donne la possibilité à l'application d'ajouter des éléments à cet écran, notamment lorsque l'application ne possède que quelques paramètres ou des liens généraux (site web, politique de confidentialité...).

Copyright © 2013 Eyrolles.

Personnalisation de l'écran Paramètres

La personnalisation de cet écran Paramètres est très limitée puisqu'il n'est possible d'ajouter que des liens cliquables appelant d'autres actions. Pour des paramètres plus riches, comme des cases à cocher, des listes déroulantes ou autres, il faudra créer son propre écran de paramétrage comme expliqué dans la section suivante.

Techniquement, la personnalisation de l'écran revient à s'abonner à l'événement `CommandsRequested`, déclenché par l'utilisateur lorsqu'il ouvre le charme des paramètres. Le système envoie alors une requête à l'application lui demandant de lui retourner la liste des commandes qu'elle veut voir ajouter.

Ajout de commande en C#

```
// abonnement
SettingsPane.GetForCurrentView().CommandsRequested += App_CommandsRequested;
...
private void App_CommandsRequested(SettingsPane sender,
                    SettingsPaneCommandsRequestedEventArgs args)
{
            SettingsCommand maCommand = new SettingsCommand("nomCommande",
                                "ma commande", new Windows.UI.Popups
                                 .UICommandInvokedHandler(maCommandeMethod));
            if (!args.Request.ApplicationCommands.Contains(maCommand))
                args.Request.ApplicationCommands.Add(maCommand);
}
```

Ajout de commande en JavaScript

```
var settingsPane = Windows.UI.ApplicationSettings
                                .SettingsPane.getForCurrentView();
settingsPane.addEventListener("commandsrequested", onCommandsRequested);
...
function onCommandsRequested(eventArgs)
{
    var maCommand = new Windows.UI.ApplicationSettings
              .SettingsCommand("nomCommande", "Ma commande", maCommandeMethod);
    eventArgs.request.applicationCommands.append(maCommand);
}
```

Ajout de commande en C++

```
SettingsPane::GetForCurrentView()->CommandsRequested += ref new
TypedEventHandler<SettingsPane^, SettingsPaneCommandsRequestedEventArgs^>(this,
&App::onCommandsRequested);
...
```

Copyright © 2013 Eyrolles.

```
void App::onCommandsRequested(Windows::UI::ApplicationSettings::SettingsPane^
settingsPane,
Windows::UI::ApplicationSettings::SettingsPaneCommandsRequestedEventArgs^
eventArgs)
{
    UICommandInvokedHandler^ handler = ref new UICommandInvokedHandler(this,
&App::maCommandeMethode);

    SettingsCommand^ maCommande = ref new SettingsCommand("nomCommande", "ma
commande", handler);
    eventArgs->Request->ApplicationCommands->Append(maCommande);
}
```

La figure suivante montre le résultat final après l'ajout de la commande personnalisée.

Figure 7–10
Ajout de la commande personnalisée à l'écran Paramètres

Affichage d'un écran personnalisé

Si l'écran standard des commandes peut répondre à certains besoins, il a surtout pour objectif d'être le point d'entrée que l'utilisateur aura le réflexe d'atteindre lorsqu'il cherchera à configurer l'application. Cela n'empêche pas le développeur d'y insérer une commande ouvrant une seconde barre latérale, proposant plus de choix de configuration et contenant des contrôles (listes déroulantes, zones de textes...) plus appropriés.

Il faut pour cela créer une commande dans l'écran de paramètres (voir exemple précédent) et, dans la méthode appelée par la commande, créer un objet `Flyout` ou `Popup` (c'est-à-dire une fenêtre qui s'affiche par-dessus la fenêtre actuelle et qui se comporte comme un simple conteneur), y injecter la page, puis afficher la pop-up à l'écran.

Tout cela ne prend que quelques lignes, comme le montre l'exemple suivant en C# :

```
Popup settingsPopup = new Popup(); // sert de conteneur pour la page
settingsPopup.Width = 200;
settingsPopup.Height = Window.Current.Bounds.Height;
```

Copyright © 2013 Eyrolles.

```
ConfigFlyout mypane = new ConfigFlyout(); // ConfigFlyout est une page Xaml que
le développeur doit créer
mypane.Width = 200;
mypane.Height = windowBounds.Height; // hauteur de l'écran

settingsPopup.Child = mypane;

// le flyout est positionné à droite et en haut
settingsPopup.SetValue(Canvas.LeftProperty, Window.Current.Bounds.Width - 200);
settingsPopup.SetValue(Canvas.TopProperty, 0);
settingsPopup.IsOpen = true;
```

En JavaScript, c'est quelque peu différent. En effet, WinJS, le framework fourni par le système, contient un objet `SettingsFlyout` qui simplifie la tâche. Il faut commencer par s'abonner à l'événement `OnSettings`, qui correspond à l'ouverture du charme de paramétrage, puis y ajouter les commandes. Ces dernières ne contiennent pas forcément de délégué à exécuter, mais un identifiant, un nom d'affichage et l'URL de la page qui sera ouverte dans le `SettingsFlyout`. Une fois la commande ajoutée, il suffit d'utiliser l'ensemble de la requête du panneau de paramétrage (qui contient donc la liste des commandes) et de le passer en paramètre à la méthode `PopulateSettings`. Le système se charge alors de créer la bonne pop-up et d'y charger la page spécifiée dans la commande lorsque celle-ci est cliquée par l'utilisateur. Avec ce comportement presque entièrement géré par WinJS, le code ne nécessite que quelques lignes :

```
WinJS.Application.onsettings = function (e) {
      e.detail.applicationcommands = { "maCommande": { title: "ma commande",
href: "/pagedeconfiguration.html" } };
      WinJS.UI.SettingsFlyout.populateSettings(e);
};
```

Le résultat final, après avoir créé et personnalisé une page de configuration, ressemble à la capture de la figure suivante.

Figure 7–11
Personnalisation d'une page de configuration

Copyright © 2013 Eyrolles.

Les associations applicatives

Que ce soit pour ouvrir un format de fichier existant (une image, un fichier Zip, une vidéo, ou tout autre fichier) ou un format propre à l'application, il est important de proposer à l'utilisateur une association avec l'application lorsque celle-ci sait le gérer. En outre, l'application peut se proposer d'être le gestionnaire par défaut pour ce type de fichier. Il s'agit ici d'un mécanisme très semblable à celui déjà en vigueur dans les versions précédentes de Windows. Ainsi, si l'application est un logiciel de retouche de photos, il peut être intéressant de l'associer à l'extension `.jpg` pour qu'elle s'ouvre automatiquement lorsque l'utilisateur accède à une image de ce format.

De plus, de la même façon que le navigateur web s'ouvre lorsque l'utilisateur clique sur un lien qui commence par http://, ou que l'application Windows Store fait de même avec les liens ms-store://, il est tout à fait possible de créer un protocole dédié, dit protocole d'action personnalisée, pour ouvrir l'application avec certains paramètres.

Ces associations fournissent une meilleure expérience à l'utilisateur, car il se verra automatiquement proposer à l'ouverture d'un fichier ou d'un protocole la liste des applications sachant le gérer. Cela peut également servir à lancer une application depuis une autre application (voir plus loin).

L'association de fichier

Au niveau système, l'association d'une extension de fichier avec une application se fait au sein du registre et relève du parcours du combattant si cela est réalisé manuellement. Généralement, cette association se fait au moment où l'utilisateur choisit un fichier, puis sélectionne *Ouvrir avec...* avant de choisir l'application appropriée. Dans le cas d'une application Windows Store, le développeur a la possibilité de déclarer cette association de façon assistée, au sein de son projet Visual Studio, et celle-ci est alors automatiquement enregistrée dans le registre au moment de l'installation de l'application.

Pour ce faire, il est nécessaire d'ouvrir le manifeste de l'application, au sein du projet Visual Studio, et d'aller dans l'onglet *Déclarations*, puis d'ajouter une *Association de type de fichier*.

De là, il est alors au minimum nécessaire de remplir le nom et le type de fichier (extension), mais de nombreux autres champs sont disponibles.

Copyright © 2013 Eyrolles.

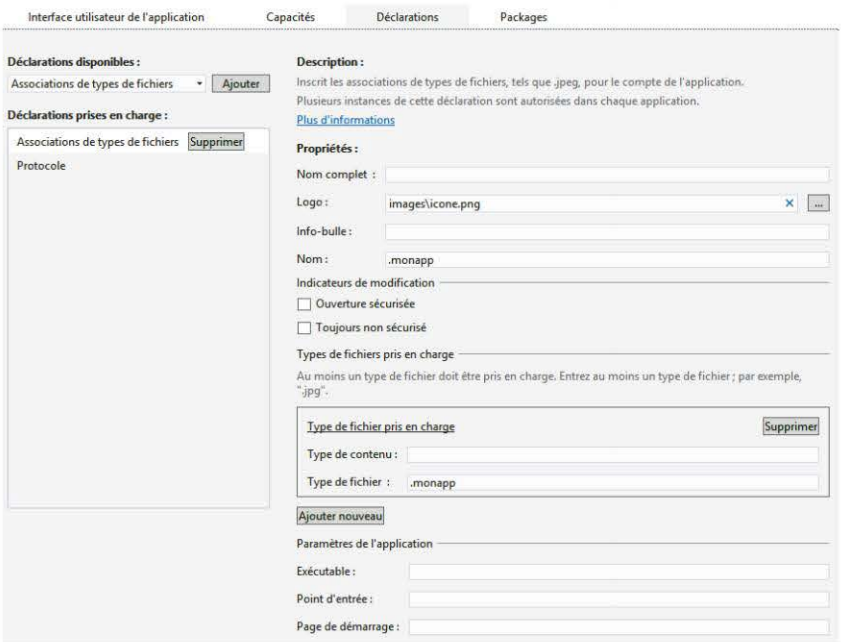


Figure 7–12
Association d'un type de fichier à l'application

Tableau 7–1 Liste des champs disponibles pour l'association de fichier

Champ	Description
Nom complet	Nom complet du type de fichier, tel qu'il apparaîtra dans la fenêtre système *Définir les programmes par défaut*.
Logo	Définir une icône pour ce type de fichier. Si omis, le logo de l'application est alors utilisé.
Info-bulle	Texte affiché lorsque l'utilisateur pointe sa souris sur l'icône d'un fichier de ce type.
Nom	Le nom de l'association (le nom doit être en minuscules).
Ouverture sécurisée	Spécifie le type de message de sécurité affiché quand l'utilisateur lance un fichier de type associé. « Ouverture sécurisée » (*Open is safe*) affichera un dialogue d'avertissement standard.
Toujours non sécurisé	Spécifie le type de message de sécurité affiché quand l'utilisateur lance un fichier de type associé. « Toujours non sécurisé » (*Always unsafe*) affichera un dialogue d'avertissement de sécurité.
Type de contenu	Sert à préciser le type de contenu MIME du fichier, par exemple `image/jpeg`. Certains types sont réservés et interdits d'usage : `application/force-download`, `application/octet-stream`, `application/unknown`, `application/x-msdownload`
Type de fichier	Représente l'extension du fichier. Dans le cas où il faut gérer plusieurs extensions, il est possible dans le formulaire, de cliquer sur *Ajouter Nouveau* pour avoir un nouveau type de fichier pris en charge.

Copyright © 2013 Eyrolles.

Lorsque l'application est installée, l'association est faite automatiquement. Lorsque l'utilisateur tente d'ouvrir un fichier d'un des types reconnus par l'application, celle-ci est lancée directement si elle est enregistrée comme seul gestionnaire de ce type de fichier ; dans le cas contraire, une fenêtre système propose de choisir parmi toutes les applications concernées.

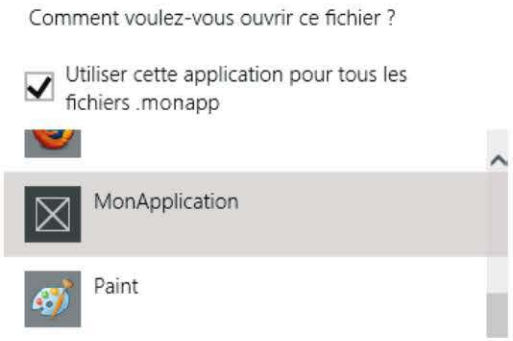


Figure 7–13
Choix parmi les applications associées au format de fichier

ATTENTION Extensions interdites

Certaines extensions de fichiers sont réservées ou interdites et ne peuvent être utilisées dans le manifeste de l'application : `Accountpicture-ms`, `Appx`, `application`, `Appref-ms`, `Bat`, `Cer`, `Chm`, `Cmd`, `Com`, `Cpl`, `crt`, `dll`, `drv`, `Exe`, `fon`, `gadget`, `Hlp`, `Hta`, `Inf`, `Ins`, `jse`, `lnk`, `Msi`, `Msp`, `ocx`, `pif`, `Ps1`, `Reg`, `Scf`, `Scr`, `Shb`, `Shs`, `Sys`, `ttf`, `url`, `Vbe`, `Vbs`, `Ws`, `Wsc`, `Wsf`, `Wsh`.

Chargement du fichier

Associer une application à un type de fichier n'est pas suffisant pour qu'elle puisse ouvrir et traiter réellement ce dernier. En effet, l'association ne fait qu'une chose : autoriser l'application à ouvrir un type de fichier donné, rien de plus. Quand un document de ce type est choisi par l'utilisateur, l'application est lancée par le système, le nom du fichier étant simplement passé en paramètre : si ce dernier n'est pas géré par du code, il ne se passera rien.

Pour intercepter le fichier et effectuer son chargement puis son traitement, il est ainsi nécessaire de vérifier le mode de lancement dans lequel se trouve l'application (`ActivationKind`) pour ensuite traiter le ou les fichier(s) éventuellement passé(s) en paramètre.

Traitement du fichier en C#

En C#, il existe une méthode particulière, `OnFileActivated`, appelée uniquement dans le cas de l'ouverture par un fichier associé. C'est ici le moyen le plus simple d'implémenter le traitement souhaité du fichier.

Copyright © 2013 Eyrolles.

```
protected override void OnFileActivated(FileActivatedEventArgs args)
{
          base.OnFileActivated(args);

      if (args.Files.Count > 0) // au moins un fichier
      {
         string nomDuFichier = args.Files[0].Name;
         string cheminDuFichier = args.Files[0].Path;

         // Traitement du fichier
         ...
      }
}
```

Traitement du fichier en JavaScript

En JavaScript, il n'y a pas d'événement dédié au chargement de fichier. Il suffit de passer par l'événement `OnActivated` et de vérifier le mode de lancement.

```
app.onactivated = function (args) {
        if (args.detail.kind === activation.ActivationKind.file) {
            var nomDuFichier = eventArgs.detail.files[0].name;
            // traitement du fichier
        }
    }
```

Traitement du fichier en C++

Les applications C++ profitent, tout comme les applications .Net, d'une méthode `OnFileActivated` permettant de placer facilement le code de traitement.

```
void App::OnFileActivated(Windows::ApplicationModel::Activation
        ::FileActivatedEventArgs^ args)
{
       if (args->Files->Size > 0) // au moins un fichier
    {
       auto nomDuFichier = args->Files->GetAt(0)->Name;
        auto cheminDuFichier = args->Files->GetAt(0)->Path;

        // Traitement du fichier
        ...
   }
}
```

Une fois le fichier récupéré, il revient au développeur de le charger et d'en utiliser le contenu. On appliquera ici les mêmes précautions que celles préconisées pour être cible du charme de partage : toute donnée externe doit être considérée comme suspecte et doit être validée avant d'être utilisée par l'application.

Copyright © 2013 Eyrolles.

L'association de protocole

Un protocole est un préfixe de lien (URI) qui sert à associer la partie dynamique de l'URI en tant que paramètre pour une application particulière. Le protocole le plus connu est le protocole HTTP, associé à une URL, qui permet d'agir avec un élément situé sur le réseau à un emplacement précis. Bien souvent, l'URL sert à afficher une page web, mais elle peut également être utilisée pour télécharger un fichier.

Dans le cas d'une application Windows Store, un protocole est un moyen supplémentaire de la lancer, et ce, avec des paramètres contextuels. Cette option doit être considérée avec attention, car elle ouvre des possibilités nouvelles et intéressantes.

Imaginons le cas d'une application de partage d'albums de photos en ligne dont il existe une version en mode déconnecté (une application Windows Store) proposant de télécharger et visualiser les albums localement. Il serait intéressant de pouvoir envoyer par courriel un lien du type monapp://download/3123123 qui, une fois cliqué par le destinataire, ouvre l'application en question et télécharge automatiquement l'album.

Un autre exemple assez différent se trouve dans l'application, bien réelle cette fois-ci, Bing Maps, qui gère le protocole `locx`. Ce protocole gère des liens comme locx://aerial/47.639747~-122.129731 (qui ouvre l'application, affiche la carte en mode vue aérienne et se positionne directement au bon endroit en utilisant la latitude et la longitude passées en paramètre).

L'ajout de la gestion d'un protocole à son application est une idée neuve offrant de nouvelles opportunités d'interagir avec l'utilisateur et de lui offrir plus qu'un simple logiciel autonome sous Win32. C'est une voie qui mérite certainement d'être étudiée de près dans de nombreuses applications Windows Store.

Techniquement, il n'y a rien de bien compliqué. Tout comme pour l'association de type de fichier, il suffit d'ouvrir le manifeste de l'application, de se rendre dans l'onglet *Déclarations* et d'y ajouter une déclaration de *Protocole*.

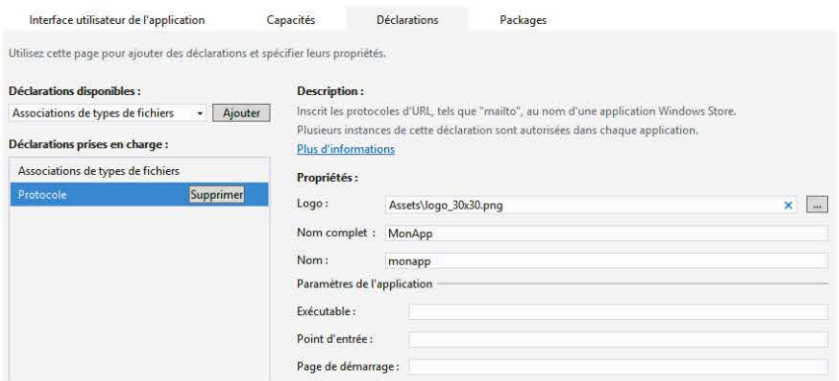


Figure 7–14
Ajout d'une association de protocole

Copyright © 2013 Eyrolles.

> **ATTENTION Bien choisir le nom du protocole**
>
> Les protocoles sont gérés de manière assez similaire aux extensions de fichiers : une application peut se déclarer capable de traiter un protocole standard (`mailto` par exemple) ou créer son propre protocole pour inviter les autres applications à l'invoquer.
> Dans ce dernier cas, il faut veiller à bien choisir le nom du protocole afin d'éviter les risques de collisions, par exemple en reprenant le nom de son entreprise ou de l'application qui va le gérer.

Utilisation du protocole

Une fois le protocole associé à l'application, celle-ci peut être lancée chaque fois qu'un URI implémentant ce protocole est utilisé par l'utilisateur. Il est alors nécessaire d'intercepter le démarrage de l'application pour déterminer si l'activation provient d'un protocole afin de proposer le traitement ad hoc.

Cela se passe au niveau de l'événement `OnActivated`, déjà étudié au chapitre traitant du cycle de vie des applications. Au sein de celui-ci, il est possible de vérifier que le mode de lancement est lié au protocole, pour ensuite appeler le code conçu pour le traiter.

En reprenant l'exemple d'une application de partage d'albums qui s'ouvrirait à l'aide du protocole monapp://download/123123, le code de traitement ressemblerait à ce qui suit.

Le protocole en *C#*

```
protected override void OnActivated(IActivatedEventArgs args)
{
    if (args.Kind == ActivationKind.Protocol)
    {
        ProtocolActivatedEventArgs protocolArgs =
                                    args as ProtocolActivatedEventArgs;

        if (protocolArgs.Uri != null)
        {
            string action = protocolArgs.Uri.Host;
            string randomKey = protocolArgs.Uri.LocalPath.Substring(1);

            if (action == "download" && !String.IsNullOrWhiteSpace(randomKey))
            {
                TelechargerFichier(randomKey);
            }
        }
    }
}
```

Copyright © 2013 Eyrolles.

Le protocole en JavaScript

```
app.onactivated = function (args) {

        if (args.detail.kind === activation.ActivationKind.protocol) {
            var action = args.detail.uri.host;
            var randomKey = args.detail.uri.LocalPath;

            if (action === "download") {
                // téléchargement du fichier
            }
        }
    };
```

Le protocole en C++

```
void
App::OnActivated(Windows::ApplicationModel::Activation::IActivatedEventArgs^
args)
{

    if (args->Kind == Windows::ApplicationModel::Activation
                                                ::ActivationKind::Protocol)
    {
        auto arguments = safe_cast<Windows::ApplicationModel::Activation
                                            ::ProtocolActivatedEventArgs^>(args);
         if(arguments->Uri->Host == "download")
         {
             //téléchargement du fichier
         }
      }
}
```

Le protocole n'est que trop rarement utilisé par les applications. Pourtant, ce mécanisme étend les capacités de l'application tout en offrant de nouvelles possibilités d'interaction depuis l'extérieur. Cela ouvre par exemple des scénarios en entreprise où il est possible, au sein de pages web des intranets, d'insérer des balises qui lanceront directement des applications Windows Store installées sur le poste utilisateur. C'est un point à ne pas sous-estimer lors de la réalisation d'une application.

Les API de lancement

Si l'association de fichier ou de protocole permet d'utiliser son application comme gestionnaire dudit fichier ou protocole, l'application peut se trouver dans un cas inverse : celui où elle ne sait pas gérer le format d'un fichier et souhaite renvoyer l'utilisateur vers une application sachant le prendre en compte. Il est possible d'imaginer

Copyright © 2013 Eyrolles.

une application gérant des pièces jointes à un dossier client : elle sait stocker les fichiers de tout type dans le dossier, mais elle ne sait pas forcément comment afficher tous les contenus de tous les types de fichiers. Si l'utilisateur veut en visualiser un, elle devra appeler une autre application capable de le faire.

Bien entendu, l'application ne connaissant pas la liste de toutes les applications installées dans la machine, le seul moyen de lancer la bonne application reste de faire confiance au système d'exploitation. Celui-ci propose justement deux API pour réaliser ce type d'appel externe, de façon simple et propre vis-à-vis de l'utilisateur : `LaunchFileAsync` et `LaunchUriAsync` ouvrent l'application par défaut associée respectivement à un type de fichier et au protocole utilisé.

Ouverture de l'application, par défaut ou non, d'un fichier

Afin d'ouvrir l'application enregistrée par défaut pour traiter un type de fichier, il suffit d'appeler la méthode asynchrone `LaunchFileAsync` et de lui passer en paramètre le nom du fichier à traiter (`IStorageFile`), qui doit se trouver dans l'espace de stockage local de l'application (c'est une limitation sérieuse imposée par la sécurité très stricte des applications Windows Store).

Exemple en C#

```
string imageFile = @"images\monfichier.jpg";
var file = await Windows.ApplicationModel.Package.Current.InstalledLocation
                                          .GetFileAsync(imageFile);

if (file != null)
{
    var resultat = await Windows.System.Launcher.LaunchFileAsync(file);
}
```

Exemple en JavaScript

```
var imageFile = "images\\monfichier.jpg";

Windows.ApplicationModel.Package.current.installedLocation
                                          .getFileAsync(imageFile).then(
    function (file) {
        Windows.System.Launcher.launchFileAsync(file);
    });
```

Il est également possible de passer des options en tant que paramètres secondaires de la méthode `LaunchFileAsync`, pour personnaliser la façon dont le fichier est lancé par l'application externe et comment l'utilisateur peut agir sur ce lancement.

Copyright © 2013 Eyrolles.

Voici la liste des options possibles :

- L'option `ContentType`, utilisée uniquement dans le cas de l'ouverture d'URI (voir section suivante), précise le format de fichier correspondant à l'URI transmise. C'est utile dans le cas où l'URI ne contient pas d'information quant à la nature de son contenu. Par exemple, il est possible de préciser le type MIME `application/pdf` lors de l'ouverture de l'URL http://www.monsite.com/download.aspx?id=document&mode=pdf.
- l'option `displayApplicationPicker`, quand elle est définie à `true`, affiche une fenêtre avec la liste complète des applications sachant gérer ce format de fichier. Il ne s'agit donc plus de choisir l'application par défaut, mais de proposer à l'utilisateur de choisir celle qu'il veut utiliser dans ce cas précis.

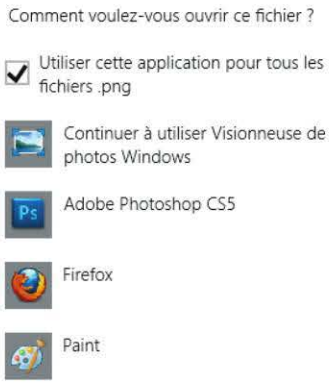


Figure 7–15
Choix de l'application qui va ouvrir le fichier

- L'option `fallbackUri` définit un URI vers lequel est renvoyé l'utilisateur dans le cas où aucune application sachant gérer ce format de fichier n'a été trouvée sur le système ni sur le Windows Store.
- Les options `preferredApplicationDisplayName` et `preferredApplicationPackageFamilyName` précisent l'application à télécharger depuis le Windows Store si aucune application ne pouvant gérer le fichier n'a été trouvée sur le système.

Figure 7–16
Choix d'une application sur le Windows Store pour ouvrir le fichier

Copyright © 2013 Eyrolles.

- L'option `treatAsTrusted` active un avertissement à l'utilisateur pour le prévenir qu'en continuant son action, il va changer d'application. C'est un moyen d'être totalement transparent avec lui et d'éviter les cas d'hameçonnage (*phishing*) avec un format de pièce jointe qui pourrait l'envoyer vers une autre application mal-intentionnée.

Figure 7–17
Avertir l'utilisateur qu'il est renvoyé vers une autre application

- Enfin, l'option `UI`, qui utilise un objet `LauncherUIOptions`, sert uniquement à préciser l'emplacement où se trouve le menu de choix d'applications dans le cas où le paramètre `displayApplicationPicker` est à `true`. Cela permet notamment d'afficher ce menu juste à côté du bouton qui a déclenché l'action.

> **INFORMATION Valeur de retour**
>
> Le résultat de la méthode `launchAsync` est un booléen qui vaut `true` en cas de succès et `false` en cas d'échec. Normalement, l'opération n'a pas de raison d'échouer, mais elle peut retourner une erreur si le fichier ouvert est lui-même un exécutable ou s'il est bloqué par le système pour des raisons de sécurité.

L'utilisation de ces options ne demande qu'une instanciation puis une configuration des paramètres souhaités.

```
// définition des options
var options = new LauncherOptions();
options.DisplayApplicationPicker = true;

var resultat = await Windows.System.Launcher.LaunchFileAsync(file, options);
```

Ouverture de l'application par défaut d'un protocole

Le comportement est quasi identique pour l'ouverture d'un protocole, à la différence de la méthode appelée, qui est `LaunchUriAsync`. Cette dernière prend un URI en paramètre, ainsi que des options de lancement, les mêmes que celles décrites dans la section précédente.

Copyright © 2013 Eyrolles.

Exemple en C#

```
Uri uri = // uri de mon choix
if (uri != null)
{
    var resultat = await Windows.System.Launcher.LaunchUriAsync(uri);

}
```

Conclusion

Les contrats sont la force même des applications Windows Store. Ils permettent à l'aide des charmes d'uniformiser l'expérience utilisateur au sein des différentes applications. Ils offrent surtout aux applications des moyens simples d'interagir entre elles, de façon structurée et organisée, et ce sans avoir besoin de se connaître au préalable.

De plus, les contrats d'associations de fichiers ou de protocoles enrichissent les fonctionnalités d'une application et lui permettent de passer la main à d'autres applications pour élargir le champ de ses capacités du point de vue de l'utilisateur, ce qui est essentiel.

Toute bonne application Windows Store se doit d'utiliser au maximum le mécanisme des contrats. La qualité et l'expérience utilisateur que Windows autorise prennent alors tout leur sens.

Copyright © 2013 Eyrolles.

8

Techniques d'animation

L'utilisation élégante des animations est une des constituantes maîtresses d'une expérience utilisateur réussie. L'interface moderne de Windows implémente des animations omniprésentes mais sobres, de l'écran d'accueil à l'affichage progressif des icônes à la droite de l'écran. Savoir tirer parti des mécanismes d'animation en s'inspirant du minimalisme de Windows est l'une des compétences indispensables pour développer des applications Windows Store abouties.

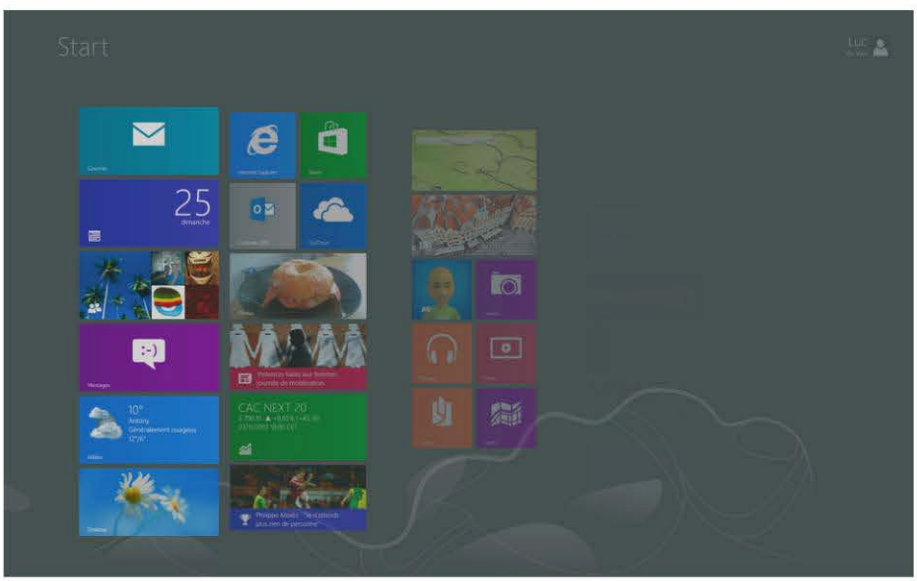


Figure 8–1
L'écran d'accueil Windows s'affiche en animant les vignettes.

Copyright © 2013 Eyrolles.

Le Windows Runtime fournit aux développeurs toutes les fonctionnalités nécessaires à l'implémentation des animations au sein des applications Windows Store. Ce chapitre a pour objectif de décrire l'usage des animations ainsi que les scénarios où leur utilisation est la plus pertinente.

La dernière partie de ce chapitre est dédiée aux spécificités des animations pour les applications C# et C++ utilisant le langage XAML.

Rôle et importance des animations

De nombreux développeurs considèrent les animations comme de simples décorations accessoires et sans grand intérêt vis-à-vis du fonctionnement global de leurs applications. Cet état d'esprit évolue désormais avec la popularisation d'interfaces fluides, telles que celles disponibles par le biais des smartphones et des tablettes tactiles. Utilisées adroitement, les animations améliorent bien plus que l'esthétique de l'application en enrichissant l'expérience utilisateur.

Guider l'utilisateur

Les animations, alliées à une disposition efficace des données, guident l'utilisateur dans son usage des fonctionnalités de l'application. Il s'agit d'un point essentiel pour l'ergonomie des applications Windows Store, où l'affichage d'informations superflues est à proscrire, renforçant l'importance du guidage de l'utilisateur (*discoverability*).

Par exemple, on peut imaginer des éléments qui s'affichent progressivement au fur et à mesure du remplissage d'un formulaire, des informations additionnelles qui apparaissent en temps réel, ou des actions qui s'ajoutent dynamiquement en fonction du contexte. En d'autres mots, l'utilisation d'animations aide à construire l'interface de manière compréhensible et fluide.

Renforcer la présentation

Les animations sont primordiales pour fournir aux utilisateurs une interface moderne et dynamique en accord avec les principes fondamentaux de la philosophie de design Microsoft. Les animations types sont pour la plupart disponibles par le biais des interfaces de programmation, individuellement ou directement au sein des contrôles, et le développeur dynamise facilement l'interface graphique en les utilisant. Cet usage standardisé des animations renforce la cohésion avec le système d'exploitation et les autres applications.

Copyright © 2013 Eyrolles.

Animations types

Les animations s'appliquent le plus souvent à trois types d'éléments différents : les conteneurs, les listes et les panneaux. La bibliothèque d'animations Windows implémente et met à la disposition du développeur des animations liées à ces éléments, facilitant leur intégration aux applications Windows Store.

L'implémentation des animations pour les applications JavaScript se base sur les transformations CSS3, tandis que l'implémentation XAML se base sur un moteur dédié similaire à celui que l'on peut retrouver sur les plates-formes WPF et Silverlight.

Transitions de thème (XAML)

Tous les éléments de l'interface peuvent être animés, en particulier lors de leur apparition à l'écran. Par exemple, la barre d'icônes Windows apparaît à la droite de l'écran avec une légère décélération, tandis que les icônes la composant apparaissent progressivement par la suite. Les menus contextuels ou les boîtes de dialogues sont d'autres exemples d'éléments fréquemment animés.

Figure 8–2
Barre d'icônes Windows

Copyright © 2013 Eyrolles.

L'animation des éléments visuels peut s'effectuer par le biais de la propriété `UIElement.Transitions`. Cette dernière définit un ensemble de transitions par le biais de la classe `TransitionCollection`. L'API XAML définit huit transitions par défaut, appelées transitions de thème, correspondant à celles utilisées par le système d'exploitation. Les transitions de thème sont déclarées au niveau des éléments et sont appelées automatiquement par le runtime lorsque cela s'avère nécessaire, par exemple lors de l'ajout de l'élément à l'arbre visuel ou du déplacement de l'élément dans une liste.

XAML

```
<TextBlock Text="Animez-moi!">
    <TextBlock.Transitions>
        <TransitionCollection>
            <PopupThemeTransition/>
        </TransitionCollection>
    </TextBlock.Transitions>
</TextBlock>
```

C#

```
var txt = newTextBlock();
var transcol = newTransitionCollection();
transcol.Add(newPopupThemeTransition());
txt.Transitions = transcol;
```

C++

```
TextBlock^ txt = ref new TextBlock();
TransitionCollection^ transcol = ref new TransitionCollection();
transcol->Append(ref new PopupThemeTransition());
txt->Transitions = transcol;
```

En fonction du type de l'opération effectuée sur l'élément cible, le moteur de rendu appliquera certains types de transitions, si celles-ci sont définies sur l'élément en question.

Tableau 8–1 Liste des transitions disponibles

Transition	Cas d'utilisation
`AddDeleteThemeTransition`	Appliquée lorsque l'élément est ajouté ou supprimé de l'arbre visuel.
`ContentThemeTransition`	Appliquée lorsque le contenu de l'élément parent change.
`EdgeUIThemeTransition`	Appliquée lorsque l'élément est ajouté ou supprimé de l'arbre visuel et qu'il est situé sur un des bords de l'écran.
`EntranceThemeTransition`	Appliquée lorsque l'élément est ajouté à l'arbre visuel (apparition).
`PaneThemeTransition`	Appliquée lorsqu'un élément de grande taille est ajouté à l'arbre visuel (apparition).

Copyright © 2013 Eyrolles.

Tableau 8–1 Liste des transitions disponibles (suite)

Transition	Cas d'utilisation
`PopupThemeTransition`	Appliquée lorsqu'un élément en surimpression (menu contextuel, boîte de dialogue) est ajouté à l'arbre visuel.
`ReorderThemeTransition`	Appliquée lorsqu'un élément change de position au sein d'une collection.
`RepositionThemeTransition`	Appliquée lorsqu'un élément change de position à l'écran, par exemple lorsque son conteneur parent ou sa propriété `Grid.Row` change.

Certains éléments, comme les conteneurs et les listes décrits ci-après, disposent de plusieurs propriétés définissant des transitions. Il faudra donc leur porter une attention particulière pour utiliser la bonne propriété, afin que les transitions définies le soient dans le bon contexte.

Dans le cas d'une liste, par exemple, la propriété `Transitions` définit l'animation utilisée lorsque la liste elle-même s'affiche à l'écran, tandis que la propriété `ItemContainerTransitions` définit l'animation des éléments appartenant à cette liste.

BONNE PRATIQUE Animations libres ou Windows ?

Les développeurs provenant du monde WPF/Silverlight noteront que Windows propose désormais des animations prêtes à l'emploi en harmonie avec le reste du système d'exploitation. Bien qu'il soit toujours possible de définir ses propres animations par le biais de `storyboard`, il convient de préférer les animations et transitions standards lorsqu'elles font sens.

Transition de contenu

Les conteneurs s'animent lors du changement de leur contenu. Le contrôle `FlipView`, par exemple, exécute une translation horizontale et un fondu entre les images affichées.

Figure 8–3
Le contrôle FlipView anime le changement de son contenu (des images) par une translation horizontale et un fondu.

Copyright © 2013 Eyrolles.

Implémentation XAML

Les contrôles utilisateur de type conteneurs dérivant de la classe `ContentControl` implémentent des animations par défaut, ce qui permet aux applications les utilisant d'en bénéficier sans ajout de code supplémentaire. D'autres contrôles, comme `FlipView`, implémentent également des effets de transition par défaut. Les contrôles dérivant de `ContentControl` disposent de la propriété `ContentTransition`, qui définit l'animation à utiliser lorsque le contenu, c'est-à-dire la valeur de la propriété `Content`, change.

Dans l'exemple suivant, la transition `EntranceThemeTransition` est appliquée lorsque le contenu du bouton (qui est un `ContentControl`) est modifié.

```
<Button>
    <Button.ContentTransitions>
        <TransitionCollection>
            <EntranceThemeTransition/>
        </TransitionCollection>
    </Button.ContentTransitions>
</Button>
```

Implémentation JavaScript

WinJS met à la disposition du développeur deux méthodes `enterContent` et `exitContent` pour implémenter des effets de transition sur le contenu d'un calque. Les deux fonctions prennent comme premier paramètre le calque parent et comme deuxième paramètre la position de départ de l'animation de transition. Il est recommandé de garder cette valeur à `null` pour respecter les valeurs par défaut de l'interface *Modern UI* de Windows.

La méthode `enterContent` déclenche la transition d'entrée d'un contenu dans un calque. Il est recommandé de donner à l'opacité du nouveau contenu la valeur 0 avant de démarrer la transition :

```
<div id="contentArea">
    <p>Lorem ipsum dolor sit amet, consectetuer</p>
</div>

// Lors du chargement de la page, cacher contentArea
var page = WinJS.UI.Pages.define("/html/enterContent.html", {
    ready: function (element, options) {
        contentArea.style.opacity = "0";
    }
});
…
// Déclenche l'effet de transition faisant apparaître le calque contentArea
WinJS.UI.Animation.enterContent(contentArea, null)
```

Copyright © 2013 Eyrolles.

La méthode `exitContent` fonctionne de la même manière et fait disparaître le calque spécifié en paramètre :

```
// Lors du chargement de la page, afficher contentArea (pour la démonstration)
var page = WinJS.UI.Pages.define("/html/enterContent.html", {
    ready: function (element, options) {
        contentArea.style.opacity = "1";
    }
});
…
// Déclenche l'effet de transition faisant disparaître contentArea
WinJS.UI.Animation.enterContent(contentArea, null)
```

L'enchaînement de ces deux méthodes donne un effet de transition entre deux contenus. Dans l'exemple suivant, le calque `c1` laisse sa place au calque `c2` :

```
<div id="mesCalques"">
    <div id="c1">
        <h3>Premier calque</h3>
        <p>Lorem ipsum dolor sit amet, consectetuer</p>
    </div>
    <div id="c2">
        <h3>Deuxième calque</h3>
        <p>Lorem ipsum dolor sit amet, consectetuer</p>
    </div>
</div>

// Au chargement, affiche c1 et cache c2
var page = WinJS.UI.Pages.define("/html/transitionContents.html", {
    ready: function (element, options) {
        runAnimation.addEventListener("click", transitionBetweenContent, false);
        c1.style.display = "block";
        c2.style.display = "none";
    }
});

…
// Enchaîne l'appel de la fonction exitContent avec enterContent
// Entre les deux, définit les attributs display afin de spécifier
// les états ciblés par enterContent (c1 caché et c2 affiché)
WinJS.UI.Animation.exitContent(mesCalques, null).done( function () {
    c1.style.display = "none";
    c2.style.display = "block";
    return WinJS.UI.Animation.enterContent(mesCalques, null);
```

Copyright © 2013 Eyrolles.

Figure 8–4
Le premier calque disparaît et le deuxième apparaît avec un effet de translation et de fondu.

Listes

Les listes animent leurs éléments enfants lorsque ces derniers sont ajoutés, supprimés ou réordonnés. Par exemple, l'application Courrier anime l'arrivée d'un nouveau message en faisant défiler verticalement les anciens messages vers le bas tout en faisant apparaître progressivement le nouveau message en haut de la liste. Windows propose des animations par défaut pour l'ajout, la suppression et le déplacement d'éléments au sein d'une liste.

Implémentation XAML

En XAML, l'affichage des listes est géré par les contrôles dérivant de la classe `ItemsControl`. Par défaut, ces listes ne sont pas animées ; le développeur devra donc déclarer explicitement quelles animations le runtime devra appliquer sur la liste et ses éléments.

Les listes disposent d'une propriété `ItemContainerTransitions` définissant une collection de transitions qui sera appliquée individuellement à chaque élément enfant de la liste lorsque ces derniers seront ajoutés/supprimés (`AddDeleteThemeTransition`) ou déplacés (`ReorderThemeTransition`).

```
<ItemsControl ItemsSource="{Binding MaCollection}" >
    <ItemsControl.ItemContainerTransitions>
        <TransitionCollection>
            <AddDeleteThemeTransition/>
        </TransitionCollection>
    </ItemsControl.ItemContainerTransitions>
</ItemsControl>
```

Implémentation JavaScript

Les animations d'ajout et de suppression d'éléments au sein des listes sont accessibles par les fonctions `createAddToListAnimation` et `createDeleteFromListAnimation` respectivement.

```
.listItem
{
    background: #336699;
    height: 50px;
```

Copyright © 2013 Eyrolles.

```
    width: 120px;
    margin: 0px 20px 20px 20px;
}

<div id="maListe">
    // Un premier item
    <div class="listItem"></div>
</div>

var page = WinJS.UI.Pages.define("/html/ajoutSuppressionListe.html", {
    ready: function (element, options) {
        maListe = element.querySelector("#maListe");
    }
});
```

L'utilisation de `createAddToListAnimation` est la suivante.

1. Appeler `createAddToListAnimation` avec comme paramètres l'élément à ajouter et la liste des éléments déjà présents.
2. Sauvegarder le résultat dans une variable.
3. Créer un élément représentant le nouvel item dans le DOM, par exemple en l'ajoutant au même parent que les autres éléments de la liste.
4. Appeler `execute()` sur la variable sauvegardée à la première étape pour exécuter la transition.

```
// Récupération des éléments de la liste (ceux de la classe listItem)
var affectedItems = document.querySelectorAll(".listItem");

// Création du nouvel élément
var nouvelItem = document.createElement("div");
nouvelItem.className = "listItem";

// Enregistrement de l'animation du nouvel élément parmi les éléments existants
var addAnimation = WinJS.UI.Animation.createAddToListAnimation(nouvelItem,
affectedItems);

// Insertion du nouvel élément en première position
// (pour voir les autres éléments se déplacer)
maListe.insertBefore(nouvelItem, maListe.firstChild);

// Exécution de la transition
addAnimation.execute();
```

L'utilisation de `createDeleteFromListAnimation` est la suivante.

Copyright © 2013 Eyrolles.

1. Appeler `createAddToListAnimation` avec comme paramètres l'élément du DOM à supprimer et la liste des autres éléments de la liste (i.e. ne pas inclure l'élément à supprimer dans cette liste).
2. Sauvegarder le résultat dans une variable.
3. Donner les valeurs finales aux attributs de l'élément à supprimer (`opacity = 0`, `position = fixed`).
4. Appeler `execute()` sur la variable sauvegardée à la première étape pour exécuter la transition.
5. Supprimer l'élément du DOM une fois l'animation terminée.

```
// Sélection du premier item et définition d'un attribut deleting
var itemASupprimer = listItems[0];
itemASupprimer.setAttribute("deleting", true);

// Obtention de la liste sans l'item à supprimer
var listeSansItemASupprimer =
document.querySelectorAll(".listItem:not([deleting])");

// Enregistrement de l'animation de suppression
var deleteAnimation = WinJS.UI.Animation.createDeleteFromListAnimation(
                                        itemASupprimer,listeSansItemASupprimer);

// Définition de l'état final de l'item à supprimer (cible de la transition CSS)
itemASupprimer.style.position = "fixed";
itemASupprimer.style.opacity = "0";

// Exécution de la transition
deleteAnimation.execute().done(
    // Suite à la transition, suppression de l'élément à enlever du DOM
    function () { maListe.removeChild(itemASupprimer); });
```

Animations libres

Le développeur peut bien entendu définir lui-même ses animations appelées alors *animations libres*. Plus complexes à mettre en œuvre, elles ne sont à utiliser que pour des scénarios non couverts par les animations standards. En effet, elles sont propres à l'application dans laquelle elles sont définies et ne bénéficieront pas des évolutions futures des animations du système d'exploitation.

L'implémentation d'animations libres en XAML s'effectue par le biais des `storyboard` et des états visuels. Les applications JavaScript quant à elles, peuvent définir des animations libres en utilisant les animations standards CSS3 ou par l'utilisation de bibliothèques tierces telles que jQuery UI.

Copyright © 2013 Eyrolles.

Animations indépendantes et dépendantes

Les animations sont divisées en deux catégories : celles qui sont indépendantes et celles qui sont dépendantes. Contrairement aux autres, les animations indépendantes utilisent pleinement l'accélération matérielle, garantissant la rapidité et la fluidité des applications. Toutes les animations mises à disposition dans la bibliothèque d'animations Windows sont indépendantes.

Animations indépendantes

XAML

Les animations indépendantes n'affectent pas la disposition d'autres éléments dans l'arbre visuel et sont accélérées matériellement par le processeur de la carte graphique. Elles sont donc fluides, efficaces et n'ont que très peu d'effet sur les performances de l'application.

Les animations indépendantes sont les suivantes :

- animations de durée nulle ;
- animations des propriétés `Canvas.Left` et `Canvas.Top` ;
- animations de `UIElement.Opacity` ;
- animations ciblant un objet graphique de type `SolidColorBrush` (`ColorAnimation`) ;
- animations de type `RenderTransform` (`Scale`, `Rotate`, `Translate`, `Composite`) ;
- animations de projection et de *clipping* (`Clip`).

HTML/CSS

Le moteur de rendu d'Internet Explorer utilisé par les applications Windows Store HTML/JavaScript accélère matériellement certains rendus, dont certaines animations CSS3. Les animations pouvant tirer parti de cette accélération sont celles n'exécutant pas de code Javascript ou qui n'interagissent pas avec le DOM.

Les animations CSS3 des propriétés `transform` (translation, rotation, *skew* 2D/3D) ou `opacity` sont indépendantes. Elles sont donc à privilégier par rapport aux animations de propriétés CSS standards comme `left`, `top` ou `visibility` qui déclenchent une recomposition de la page.

Copyright © 2013 Eyrolles.

Animations dépendantes

XAML

Les animations dépendantes sont celles qui affectent la mise en page. En d'autres mots, une animation est considérée comme dépendante si elle nécessite des informations en provenance du thread UI responsable de la mise en page des éléments les uns par rapport aux autres. Par exemple, l'animation des propriétés `Height` ou `Width` est dépendante. De par leur dépendance envers le thread UI géré par le processeur principal et non par la carte graphique, les animations de ce type ne sont pas accélérées matériellement, ce qui les rend moins performantes et susceptibles de ralentir l'exécution de l'application.

Afin de garantir des performances optimales et de prévenir l'usage non intentionnel des animations dépendantes, ce type d'animation est désactivé par défaut et doit être explicitement activé par le biais de la propriété `EnableDependentAnimation`.

```
<DoubleAnimation EnableDependentAnimation="True" ...
```

HTML/CSS

Les animations dépendantes en HTML sont exécutées par le thread UI, donc par le processeur principal. Elles sont affectées par l'exécution de code, les calculs de remise en page ou encore la gestion des événements du DOM.

Bien que leur usage ne soit pas recommandé, les animations indépendantes devant toujours être privilégiées, leur effet négatif sur les performances peut être réduit par les moyens suivants :

- en isolant les éléments animés indépendamment ou en les plaçant devant les autres éléments (en leur donnant un z-index supérieur) ;
- en réduisant la surface (nombre de pixels) affectée.

Quelle que soit la technologie utilisée, XAML ou HTML/CSS, il est fortement recommandé d'utiliser les animations indépendantes dont le rendu est effectué par la carte graphique. Ces dernières donnent des animations d'une fluidité irréprochable n'affectant pas ou peu les performances de l'application.

Storyboard (XAML)

Les `storyboard` sont des objets regroupant plusieurs animations, définis en tant que ressources XAML ou directement par code. Ils peuvent être utilisés indépendamment depuis le code, ou lors de la définition d'états visuels.

Copyright © 2013 Eyrolles.

Définition

Un `storyboard` se compose d'une ou plusieurs animation(s). Ces dernières utilisent les propriétés attachées `Storyboard.TargetName` et `Storyboard.TargetProperty` pour préciser respectivement l'élément et la propriété à animer. L'attribut `To` précise la valeur cible, par exemple une valeur numérique pour une `DoubleAnimation`, ou une couleur pour une `ColorAnimation`. Plusieurs éléments différents peuvent être animés au sein d'un même `Storyboard`.

Storyboard XAML

```
<Page.Resources>
    <Storyboard x:Key="monStoryboard">
        <DoubleAnimation Storyboard.TargetName="btn1"
                         Storyboard.TargetProperty="Opacity" To="0"/>
        <ColorAnimation Storyboard.TargetName="monBackground"
                        Storyboard.TargetProperty="Color" To="#FF2222"/>
    </Storyboard>
</Page.Resources>
```

Utilisation

Si le Storyboard est défini en XAML, il doit d'abord être récupéré depuis les ressources :

C#

```
Storyboard sb = this.Resources["monStoryboard"] as Storyboard;
```

C++

```
Storyboard^ sb = safe_cast<Storyboard^>(this->Resources->Lookup(
                                                    "monStoryboard"));
```

Une fois récupéré, il suffit d'appeler sa méthode `Begin` pour l'exécuter :

C#

```
sb.Begin();
```

C++

```
Sb->Begin();
```

Copyright © 2013 Eyrolles.

États visuels (XAML)

Les états visuels servent à définir différentes combinaisons de disposition d'éléments au sein d'un écran, en fonction de l'espace disponible et du contexte. Les projets Visual Studio par défaut pour XAML C# et XAML C++ implémentent les états visuels en fonction de l'état d'affichage en cours (plein écran, ancré…). Un état visuel se différencie d'un autre, non pas par l'ajout d'un ou plusieurs élément(s), mais par la modification des valeurs de certaines propriétés. Par exemple, la propriété `ItemTemplate` d'une liste peut utiliser un template différent en fonction de l'état visuel en cours, ou bien des éléments peuvent être affichés ou cachés par une modification de leur opacité ou de leur propriété `Visibility`. En d'autres termes, tous les éléments utilisés dans les états visuels doivent être définis dès le départ dans le fichier XAML, car seules les propriétés d'éléments existants sont modifiées d'un état visuel à un autre.

Souvent utilisés pour définir des mises en page d'écran en fonction de l'état d'affichage (plein écran, ancré ou remplissage), les états visuels permettent également aux développeurs de gérer plusieurs états d'un même écran, sans avoir à créer de nouvelles pages ou contrôles utilisateurs.

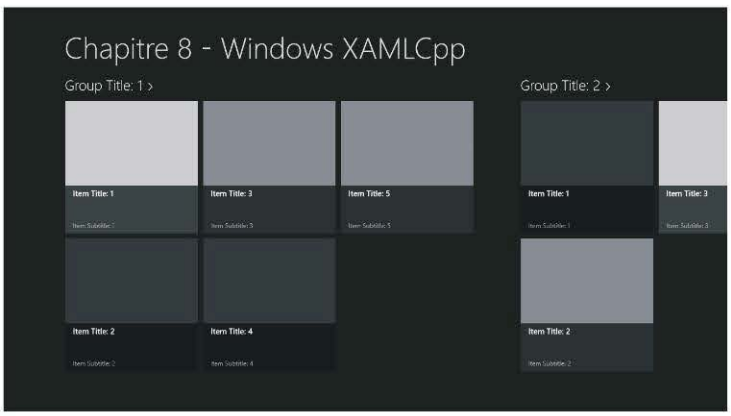


Figure 8–5
Application par défaut : état visuel FullScreenPortrait

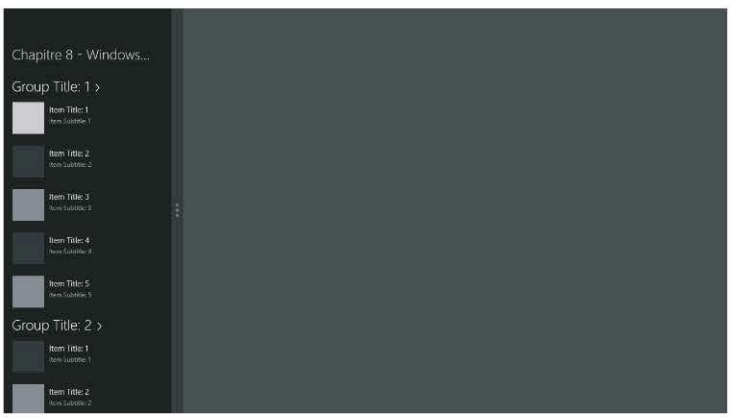


Figure 8–6
Application par défaut : état visuel Snapped

Copyright © 2013 Eyrolles.

Le moteur de rendu XAML anime automatiquement le passage d'un état visuel à un autre en prenant en charge l'interpolation des valeurs des différentes propriétés entre deux états visuels. Il est ainsi possible de définir une durée de transition d'un état visuel à un autre, et d'appliquer des fonctions de lissage aux interpolations.

Définition des états visuels

Les états visuels sont définis en XAML par le biais de l'objet `VisualStateManager`. Ce dernier est généralement associé à l'objet `Panel` racine d'une vue, par exemple un élément `Grid` :

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="mesEtatsVisuels">
            <VisualState x:Name="EtatVisuel1">
                <Storyboard>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="EtatVisuel2">
                <Storyboard>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>

    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Button x:Name="btn1" Content="Confirmer" HorizontalAlignment="Center"/>
    <Button x:Name="btn2" Content="Annuler" HorizontalAlignment="Center"
Opacity="0"/>

    <StackPanel Orientation="Horizontal" Grid.Row="1">
        <Button Content="Vers l'état visuel 1" Click="Button_Click1"/>
        <Button Content="Vers l'état visuel 2" Click="Button_Click2"/>
    </StackPanel>
</Grid>
```

Dans le `VisualStateManager` sont définis des groupes, les `VisualStateGroups`, et chacun de ces groupes définit un ensemble d'états visuels liés. Afin de permettre les combinaisons, les propriétés définies dans les états visuels d'un groupe ne doivent pas être reprises dans les états visuels d'un autre groupe. Par exemple, les boutons peuvent à la fois avoir le focus clavier (état `Focused` du groupe `FocusStates`) et être enfoncés (état `Pressed` du groupe `CommonStates`).

Copyright © 2013 Eyrolles.

BONNE PRATIQUE Bien séparer les propriétés

Il faut veiller à ne pas animer la même propriété dans deux groupes différents : les états des groupes étant indépendants et pouvant se superposer, XAML ne pourrait plus savoir quelle valeur attribuer à la propriété. Sous Blend, cette erreur est signalée par une icône d'alerte à côté du nom des états concernés (l'infobulle indiquant le nom de la propriété concernée et les raisons du conflit de valeurs).

Chaque état visuel définit ensuite un `Storyboard` décrivant les animations nécessaires pour atteindre l'état en question. Dans l'exemple, `EtatVisuel1` et `EtatVisuel2` afficheront respectivement les boutons `Btn1` et `Btn2` en animant leur propriété `Opacity`.

```
<VisualState x:Name="EtatVisuel1">
    <Storyboard>
        <DoubleAnimation Storyboard.TargetName="btn1"
                         Storyboard.TargetProperty="Opacity" To="1"/>
        <DoubleAnimation Storyboard.TargetName="btn2"
                         Storyboard.TargetProperty="Opacity"
                         To="0"/>
    </Storyboard>
</VisualState>
<VisualState x:Name="EtatVisuel2">
    <Storyboard>
        <DoubleAnimation Storyboard.TargetName="btn1"
                         Storyboard.TargetProperty="Opacity"
                         To="0"/>
        <DoubleAnimation Storyboard.TargetName="btn2"
                         Storyboard.TargetProperty="Opacity"
                         To="1"/>
    </Storyboard>
</VisualState>
```

Dans le cas de l'état visuel 1, l'opacité du bouton `Btn1` vaudra 1 (visible), alors que celle du bouton `Btn2` vaudra 0 (invisible). L'état visuel 2 définit l'inverse. Aller d'un état visuel à un autre montrera l'un ou l'autre des deux boutons.

Figure 8–7

Copyright © 2013 Eyrolles.

Appliquer un état visuel

L'application d'un état visuel s'effectue très simplement : il suffit d'appeler la méthode `VisualStateManager.GoToState` et de préciser l'élément sur lequel les états visuels sont définis, le nom de l'état visuel cible et une valeur indiquant si la transition doit être animée. Si elle ne l'est pas, l'état visuel cible sera appliqué immédiatement sans respect du délai défini par le storyboard (une seconde par défaut).

Dans l'exemple suivant, cet appel est effectué sur le clic des boutons *État Visuel 1* ou *État Visuel 2* :

C++

```
VisualStateManager::GoToState(this, "EtatVisuel1", true);
```

C#

```
VisualStateManager.GoToState(this, "EtatVisuel1", true);
```

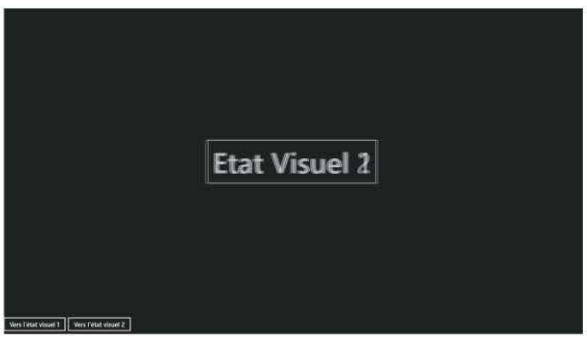


Figure 8–8
Le passage d'un état à un autre modifie automatiquement les propriétés. Ici, la variation de l'opacité des deux boutons donne un effet de fondu.

La notion d'état visuel étant basée sur les `storyboard`s, les transitions profitent pleinement du moteur d'animation XAML. Dans l'exemple précédent, basculer rapidement entre les deux états visuels définis donnera toujours un résultat déterministe avec une animation fluide et logique.

Édition des états visuels avec Blend

Bien qu'il soit possible d'éditer les états visuels manuellement, Blend pour Visual Studio fournit au développeur l'outillage nécessaire pour éditer ces derniers visuellement et efficacement.

L'onglet *États* (ou *States*) montre les groupes et états visuels définis dans un fichier XAML. Lorsqu'un état est sélectionné, Blend passe en mode enregistrement : à partir de cet instant, toutes les modifications des propriétés sont enregistrées et associées à l'état visuel préalablement choisi.

Copyright © 2013 Eyrolles.

ATTENTION

Comme dit précédemment, les états visuels définissent des valeurs pour les propriétés d'éléments existants. L'ajout ou la suppression d'éléments en mode enregistrement provoquera un changement de l'arbre visuel global et ne sera donc pas lié à l'état visuel en cours d'édition !

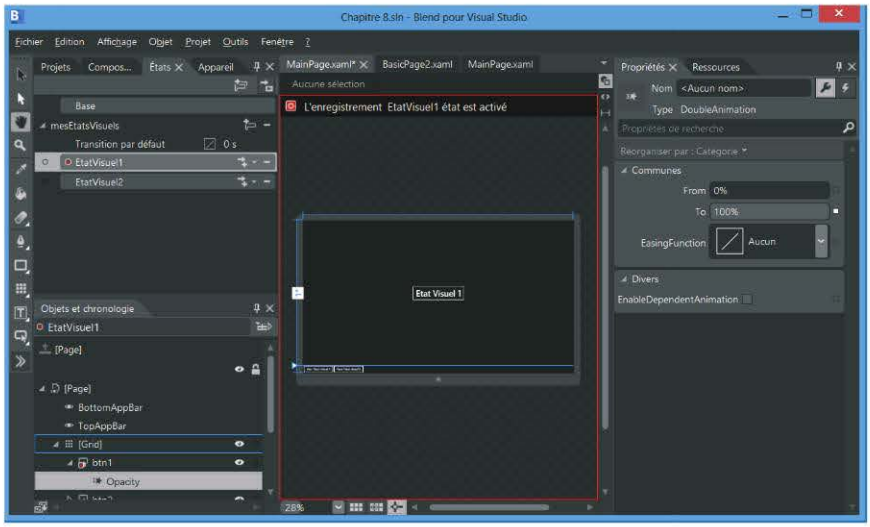


Figure 8–9
Blend édite et visualise les états et les propriétés modifiées.

Blend aide à créer des animations complexes par le biais d'un éditeur visuel. Ainsi, pour reprendre l'exemple précédent.

1. Sélectionner `EtatVisuel1` et le bouton `btn1`.
2. Sélectionner le curseur noir (*Sélection*) dans le menu à gauche.
   Déplacer le bouton sur la surface de conception. Blend ajoute automatiquement une animation sur la propriété `RenderTransform` de `btn1`.

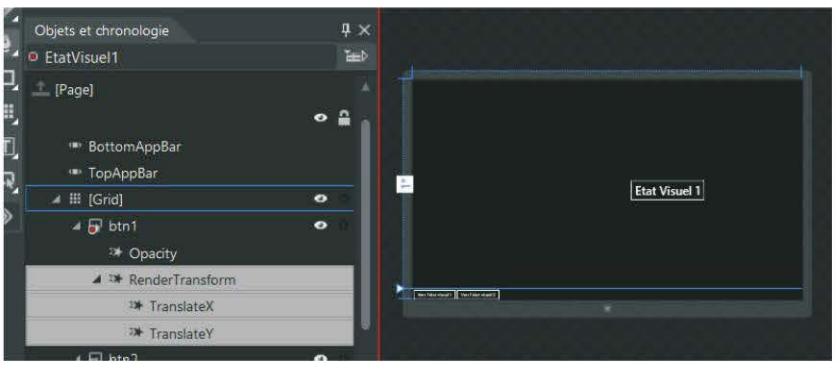


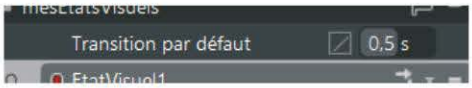
Figure 8–10

Basculer entre `EtatVisuel1` et `EtatVisuel2` : Blend montre très facilement le rendu des différents états visuel définis.

Copyright © 2013 Eyrolles.

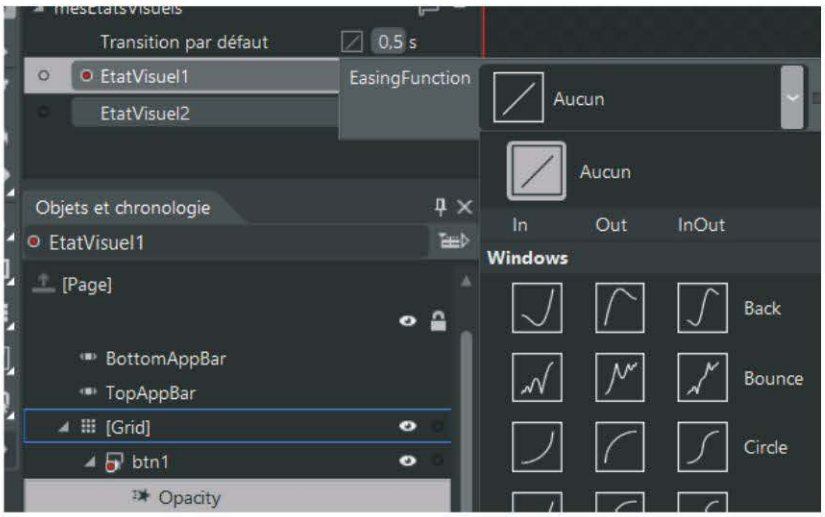
3 Ajuster le délai de transition par défaut à 0,5 seconde.

Figure 8–11



En cliquant sur l'élément situé à gauche du délai, il est également possible de sélectionner des fonctions de lissage (aussi appelées fonctions organiques, car elles rendent les animations plus naturelles) pour personnaliser l'animation.

Figure 8–12



4 Exécuter l'application pour observer la manière dont le moteur XAML anime automatiquement et simultanément le déplacement et l'opacité des différents éléments.

Figure 8–13

Copyright © 2013 Eyrolles.

Transitions CSS

Les développeurs d'applications Windows Store en JavaScript peuvent mettre à profit les transitions du standard CSS3 pour implémenter des animations dans leurs écrans. L'utilisation de bibliothèques tierces reste également possible, mais il faudra veiller à ce que ces dernières utilisent des animations indépendantes, donc des transitions CSS, afin de garantir une performance optimale via l'accélération matérielle.

Le principe des transitions CSS est simple : faire évoluer progressivement (interpoler) la valeur d'une propriété sur une durée déterminée donnant ainsi un effet d'animation. Les transitions sont déclenchées par l'application d'une pseudo-classe. Deux exemples de transitions très communes sont le fondu (animation de la propriété `opacity`) et le déplacement d'un élément (animation d'une ou de plusieurs propriétés `padding`).

Les transitions CSS se basent sur deux propriétés : `transition-duration` (durée de la transition) et `transition-property` (propriétés à animer, qui doivent être de type couleur, entier, flottant, rectangle, shadow, gradient, visibility, SVG ou transformation).

L'exemple suivant définit une classe CSS pour un bouton et anime trois de ses propriétés lorsque le curseur de la souris le survole. La propriété `transition-duration` vaut `0` par défaut et le simple fait de la définir à une valeur de temps non nulle permet d'animer simultanément les trois propriétés `opacity`, `padding-left` et `font-size`.

CSS

```
.boutonAnime:hover {
      opacity: 0.2;
      padding-left: 100px;
      font-size: 16pt;
      transition-duration: 0.3s;
}
```

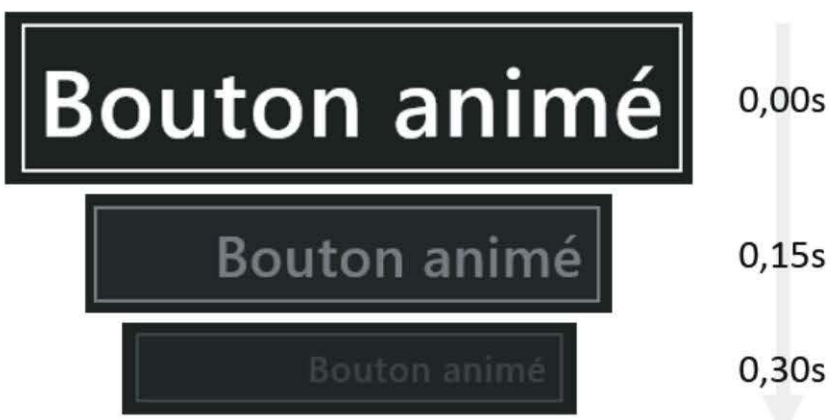


Figure 8–14
Animation des propriétés opacity, padding-left et font-size d'un bouton

Copyright © 2013 Eyrolles.

La propriété `transition-property` vaut `all` par défaut, ce qui a pour effet d'animer toutes les propriétés compatibles lorsque `transition-duration` est supérieure à `0`, ce qui est le cas de l'exemple précédent. En définissant sa valeur, il est possible de préciser indépendamment quelles propriétés seront animées et pendant quelle durée. Dans l'exemple suivant, la propriété `opacity` est animée pendant 0,6 s, la propriété `padding-left` pendant 0,3 s, tandis que la propriété `font-size`, exclue de `transition-property`, a pour valeur `16pt` dès que le curseur de souris survole le bouton (elle n'est donc pas animée).

```
.boutonAnime:hover {
      opacity: 0.2;
      padding-left: 100px;
      font-size: 16pt;
      transition-property: opacity,padding-left;
      transition-duration: 0.6s,0.3s;
}
```

Deux autres propriétés sont utilisables pour préciser d'avantage le déroulement des transitions :

- `transition-delay` définit le délai après lequel la transition démarrera. Elle vaut `0` par défaut ;
- `transition-timing-function` définit une fonction de lissage qui s'appliquera lors de l'interpolation, en donnant par exemple un effet d'accélération ou de décélération. Sa valeur par défaut est `ease` et les valeurs possibles sont : `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, `step-start`, `step-end`, `steps`, `cubic-bezier`.

Événement DOM transitionend

L'événement DOM `transitionend` est levé lorsqu'une transition se termine et permet ainsi au développeur d'exécuter du code suite à une animation. Cet événement est attaché à deux attributs :

- `propertyName` : le nom de la propriété dont la transition vient de s'achever ;
- `elapsedTime` : le temps qu'a duré la transition, sans prendre en compte le délai (`transition-delay`).

Utilisation conjointe avec l'extension ms-view-state

L'extension Microsoft aux media queries CSS, `ms-view-state` sert à gérer les états d'affichage (plein écran, ancré, ou remplissage) des applications Windows Store en JavaScript. Elle se combine parfaitement avec les transitions CSS3 : les éléments sur lesquels une transition est définie s'animeront automatiquement lors du changement d'état d'affichage de l'application, fluidifiant élégamment ce dernier.

Copyright © 2013 Eyrolles.

9

Gestion de l'interface tactile

Depuis sa toute première version, Windows a été conçu pour être utilisé avec un clavier et une souris. Windows XP Tablet Edition a ajouté l'usage du stylet et Windows 7 une première gestion native des interfaces tactiles. C'est dans Windows 8 que Microsoft intègre pleinement les usages tactiles, aussi bien au sein de son interface utilisateur que de ses interfaces de programmation.

Windows et le Windows Runtime intègrent ainsi des fonctionnalités utiles au développeur d'applications Windows Store pour gérer clavier, souris, stylet et événements tactiles multipoints de manière simple et intuitive. Ce chapitre a pour objectif de décrire les différents concepts liés à l'utilisation des interfaces tactiles, ainsi que leur usage conjoint à d'autres types d'interfaces utilisateur.

Interactions utilisateur sous Windows

Windows est un système d'exploitation capable de gérer une très grande variété de périphériques d'entrée. Plus encore, il est utilisable quel que soit le type d'interface utilisée : clavier, souris, stylet ou encore écran tactile. Cette diversité rend le système d'exploitation accessible et utilisable dans des environnements très variés.

Bien que l'implémentation de la gestion des différents périphériques puisse apparaître comme une lourde tâche, Windows facilite leur intégration par le biais de mul-

Copyright © 2013 Eyrolles.

tiples mécanismes. Par exemple, le zoom sémantique est effectué de diverses façons en fonction des périphériques disponibles, et ce sans code supplémentaire :

- en écartant ou pinçant deux doigts sur un écran tactile ;
- en tenant la touche *CTRL* enfoncée pendant qu'on manipule la roulette d'une souris ;
- en tenant la touche *CTRL* enfoncée pendant qu'on appuie sur les touches + ou – du clavier.

À l'heure où cet ouvrage est écrit, les utilisateurs ne disposant pas d'écrans tactiles restent largement majoritaires. Il est donc judicieux de concevoir des applications qui fonctionnent aussi bien avec un clavier et une souris qu'avec des périphériques tactiles. Ces précautions de conception garantissent aux applications Windows Store ainsi développées une audience la plus large possible.

Considérations pour les interfaces tactiles

Les applications Windows Store doivent être conçues pour être utilisées aussi bien de manière tactile qu'à la souris et au clavier. Elles doivent respecter certains principes pour garantir un usage optimal et une cohérence avec le reste du système.

La manipulation directe des données est une des attentes importantes des utilisateurs d'interfaces tactiles. Les retours visuels immédiats comme les effets de pressions et de manipulations directes avec les doigts, ainsi que l'utilisation de vignettes au contenu riche, sont des facteurs contribuant à une expérience tactile réussie.

Les éléments interactifs doivent être suffisamment accessibles pour être manipulés avec confiance, la taille minimale recommandée étant de 7 × 7 millimètres. Les éléments responsables d'actions critiques (suppression, fermeture) doivent avoir une taille minimale de 9 × 9 millimètres afin de réduire les risques d'erreur de manipulation. Le graphique suivant indique les taux d'erreurs enregistrés par les laboratoires Microsoft en fonction de la taille des éléments à atteindre.

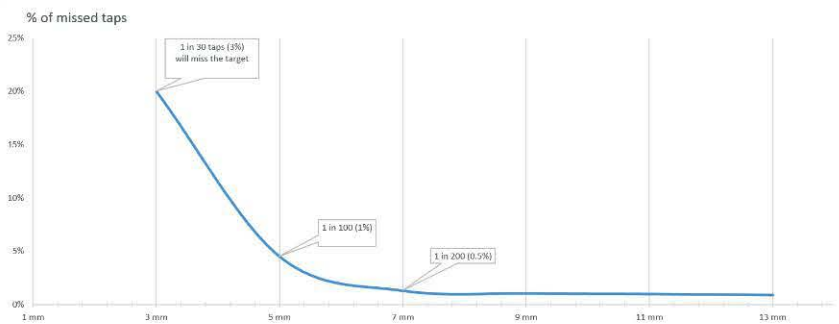


Figure 9–1
Taux d'erreurs de sélection en fonction de la taille de la cible

Copyright © 2013 Eyrolles.

Certaines applications Windows Store visent à être utilisées sur des appareils au format tablette et/ou par des utilisateurs se tenant debout. Pour cette raison, il est conseillé de placer les éléments interactifs principaux près des points chauds *(hot spots),* les plus facilement accessibles par les doigts des utilisateurs, aux bords de l'écran. Ces zones sont identifiées dans la figure publiée par Microsoft (voir figure suivante : les zones de chaque côté sont les plus accessibles, la zone blanche est la plus attrayante visuellement).

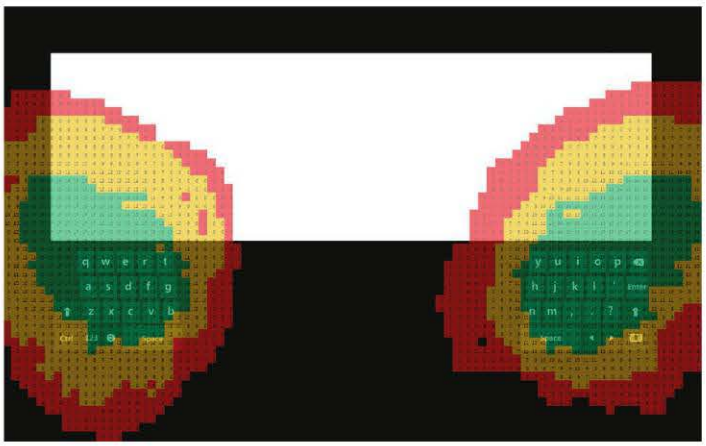


Figure 9–2
Zones d'atteintes sur les tablettes en mode paysage

Les pointeurs

Au fil des versions, la variété des dispositifs de pointage reconnus par Windows a multiplié le nombre d'interfaces de programmation, chaque nouveau périphérique de pointage se construisant sur les interfaces de programmation de ses prédécesseurs en y ajoutant ses spécificités.

Pour la première fois, le Windows Runtime implémente des interfaces de programmation extensibles qui ne sont plus centrées autour des interactions propres à la souris. Le concept de pointeur abstrait a été introduit dans les API, ainsi que les concepts sous-jacents (souris, stylet, tactile monopoint, tactile multipoint…).

Un pointeur est créé par le système à chaque fois qu'un contact est effectué (un clic, un appui ou un contact tactile), puis supprimé lorsque le contact disparaît physiquement (le bouton de la souris est relâché, le stylet quitte la surface d'écriture ou le doigt est levé). Chaque pointeur est donc unique dans l'espace et dans le temps et est représenté par une instance de la classe `PointerPoint`.

Les `PointerPoint` contiennent des informations communes à tous les dispositifs de pointage, comme la position (X, Y) du point de contact ainsi que le type d'interface déclarant le contact (souris, stylet ou interface tactile).

Copyright © 2013 Eyrolles.

Les propriétés de la classe `PointerPoint` les plus communément utilisées sont listées dans le tableau suivant.

Propriété	Usage
`FrameId`	Identifiant du point de contact dont la valeur est gardée dans le temps pour un même contact physique. Il permet par exemple de différencier un doigt de l'autre dans une manipulation multipoint.
`IsInContact`	Indique que le contact associé est toujours actif (appuyé).
`PointerDevice`	Obtient des informations sur le dispositif matériel utilisé pour l'interaction.
`Position`	Renvoie la position (X, Y) du contact.
`TimeStamp`	Renvoie le moment auquel le contact a eu lieu.

Les pointeurs disposent de plusieurs événements permettant au développeur de réagir à une interaction. Ces événements sont valables pour tous les types de dispositifs de pointage reconnus. En XAML, ils sont définis au niveau de la classe `System.Windows.UIElement`, tandis qu'en JavaScript, ils sont ajoutés en écoute (`addListener`) aux éléments du DOM.

Tableau 9–1 Liste des événements des pointeurs

Événement	Usage
`PointerPressed`	Souris : clic Stylet : contact Tactile : contact
`PointerReleased`	Souris : relâchement du bouton Stylet : stylet relevé Tactile : doigt relevé
`PointerCanceled`	Le pointeur a été perdu pour des raisons inconnues.
`PointerCaptureLost`	La capture des contacts en dehors de la zone d'origine a été perdue (prise par un autre élément).
`PointerEntered`	Un pointeur est entré dans la surface d'un élément.
`PointerExited`	Un pointeur est sorti de la surface d'un élément.
`PointerMoved`	Un pointeur s'est déplacé dans la surface d'un élément.
`PointerWheelChanged` (présent pour des raisons de simplicité de développement)	Souris : roulement de la roulette Stylet : N/A Tactile : N/A

Des informations supplémentaires sont également disponibles afin que le développeur puisse, s'il le souhaite, s'adapter aux spécificités liées à l'interface utilisée.

Copyright © 2013 Eyrolles.

Détection des capacités tactiles

Bien que le comportement par défaut des applications Windows Store s'adapte à tout type d'interaction, Windows laisse au développeur la possibilité d'adapter finement son interface aux capacités de la machine cliente. Il est notamment possible de déterminer les capacités tactiles de la machine sur laquelle s'exécute le programme.

Par exemple, la classe `Windows.Devices.Input.TouchCapabilities` détermine les capacités tactiles de la machine sur laquelle s'exécute l'application Windows Store. Le développeur peut ensuite adapter l'expérience utilisateur en fonction du résultat. Cette classe très simple détermine si la machine de l'utilisateur dispose d'une interface tactile (`TouchCapabilities.TouchPresent`) et, le cas échéant, le nombre maximum de contacts (`TouchCapabilities.Contacts`).

C#

```
TouchCapabilities touchCapabilities = new
Windows.Devices.Input.TouchCapabilities();
bool estTactile = touchCapabilities.TouchPresent;
uint nombreContacts = touchCapabilities.Contacts;
```

C++

```
Windows::Devices::Input::TouchCapabilities touchCapabilities;
bool estTactile = touchCapabilities.TouchPresent;
unsigned int nombreContacts = touchCapabilities.Contacts;
```

JavaScript

```
var touchCapabilities = new Windows.Devices.Input.TouchCapabilities();
  var estTactile = touchCapabilities.touchPresent;
  var nombreContacts = touchCapabilities.contacts;
```

En JavaScript, il est aussi possible de vérifier que la valeur de la propriété `window.navigator.maxTouchPoints` est supérieure à zéro.

Il faut noter que ces fonctionnalités de détection existent afin d'optimiser l'utilisation de la machine cliente. Toute application Windows Store se doit cependant de rester pleinement utilisable, même si l'utilisateur ne dispose pas de fonctionnalités tactiles.

Copyright © 2013 Eyrolles.

Mouvements et manipulations

Les mouvements sont le propre des interfaces tactiles multipoints. Le chapitre 2 a décrit les manipulations standards du langage d'interaction implémenté directement au sein des contrôles fournis par défaut avec le kit de développement Windows 8. Les éléments de ce langage, les mouvements et manipulations, peuvent également être utilisés indépendamment, par exemple dans des contrôles utilisateur personnalisés.

GestureRecognizer

Le Windows Runtime expose l'objet `Windows.UI.Input.GestureRecognizer` grâce auquel les applications Windows Store profitent directement du moteur Windows de reconnaissance de manipulations tactiles. Ce dernier gère le paramétrage, la détection des mouvements tactiles et un moteur d'inertie rendant les interactions plus naturelles.

Le moteur XAML des applications Windows Store intègre l'utilisation du `GestureRecognizer` directement au sein des éléments visuels, ce qui le rend par défaut invisible au développeur. Les développeurs d'applications Windows Store écrites en HTML et JavaScript, quant à eux, doivent manipuler directement l'objet `GestureRecognizer` pour, par exemple, appliquer les manipulations détectées à des éléments du DOM.

Utilisation du GestureRecognizer (JavaScript)

L'objet `GestureRecognizer` est instancié une seule fois par page. Une fois configuré, il lève des événements, appelés événements de manipulation, renvoyant des informations sur les manipulations tactiles en cours.

JavaScript

```
// Instanciation du GestureRecognizer
gstRec = new Windows.UI.Input.GestureRecognizer();

// Configuration pour reconnaître la rotation et le pincement
gstRec.gestureSettings =
    Windows.UI.Input.GestureSettings.manipulationRotate |
    Windows.UI.Input.GestureSettings.manipulationScale;
```

L'abonnement aux événements de manipulation permettra d'associer du code aux manipulations ainsi détectées :

Copyright © 2013 Eyrolles.

JavaScript

```
gstRec.addEventListener('manipulationstarted',
that.manipulationStartedHandler);
gstRec.addEventListener('manipulationupdated', that.manipulationDeltaHandler);
gstRec.addEventListener('manipulationcompleted', that.manipulationEndHandler);
gstRec.addEventListener('tapped', that.tappedHandler);
```

Le développeur invoque les fonctions d'alimentation `ProcessDownEvent`, `ProcessUpEvent`, `ProcessMoveEvent` et `ProcessWheelEvent` pour fournir au `GestureRecognizer` les informations de pointeurs (notamment leur position) dont les algorithmes de détection ont besoin. Les événements de manipulations sont émis au fur et à mesure des appels aux méthodes d'alimentation. Ainsi, les événements de pointeurs `MSPointerDown`/`Move`/`Up`/`Cancel` et `wheel` en provenance de l'élément DOM source sont transmis aux fonctions d'alimentation du `GestureRecognizer`. Les gestionnaires d'événements de pointeurs appellent donc les fonctions d'alimentation leur correspondant.

Abonnement aux événements pointeur

```
monElement.addEventListener('MSPointerDown', that.processDown, false);
monElement.addEventListener('MSPointerMove', that.processMove, false);
monElement.addEventListener('MSPointerUp', that.processUp, false);
monElement.addEventListener('MSPointerCancel', that.processUp, false);
monElement.addEventListener('wheel', that.processMouse, false);

…

that.processDown = function (evt) {
    var pp = evt.getCurrentPoint(monElement);
    gstRec.processDownEvent(pp);
};

that.processMove = function (evt) {
    var pps = evt.getIntermediatePoints(monElement);
    gstRec.processMoveEvents(pps);
};

that.processUp = function (evt) {
    var pp = evt.getCurrentPoint(monElement);
    gstRec.processUpEvent(pp);

};

that.processMouse = function (evt) {
    var pp = evt.getCurrentPoint(monElement);
    gstRec.processMouseWheelEvent(pp, evt.shiftKey, evt.ctrlKey);
};
```

Copyright © 2013 Eyrolles.

Les gestionnaires d'événements des manipulations donnent accès à des informations détaillées concernant les manipulations en cours. L'événement `manipulationupdated`, notamment, sert à suivre les manipulations au fur et à mesure, au travers de la classe `manipulationUpdatedEventArgs`.

Propriété	Signification
`Cumulative`	Indique la valeur totale de la manipulation depuis sa détection.
`Delta`	(Uniquement pour `ManipulationDelta`) Indique la différence de valeur entre cet événement et le précédent.
`IsInertial`	Indique s'il s'agit d'une manipulation d'inertie (engendrée par le moteur d'inertie).
`PointerDeviceType`	Indique le type de matériel utilisé pour effectuer la manipulation.
`Position`	Point d'origine de la manipulation.
`Velocities`	Indique les vitesses des différents éléments de la manipulation : 1. angulaire en degrés / millisecondes ; 2. réduction / agrandissement en unité indépendante par milliseconde ; 3. linéaire en unité indépendante par milliseconde. La valeur par défaut de l'unité indépendante est de vingt pixels.

Les transformations CSS3 seront ensuite utilisées, par exemple, pour appliquer des transformations géométriques aux éléments du DOM en fonction des valeurs des manipulations.

Événements de manipulation (XAML)

Les éléments visuels (`UIElement`) des interfaces conçues en XAML disposent par défaut d'un ensemble de propriétés et d'événements pour gérer directement les manipulations. Bien que l'implémentation se base sur l'objet `GestureRecognizer` du Windows Runtime, le développeur XAML n'a pas à l'utiliser directement.

Sur le ou les élément(s) devant recevoir les manipulations, il faut définir la propriété `ManipulationMode` à une valeur différente de la valeur par défaut (`ManipulationModes.None`). Les valeurs de cette énumération sont combinables et permettent aux éléments de ne s'abonner qu'à la notification de certaines manipulations. Un élément recevra par exemple les événements liés à la rotation et à l'agrandissement, mais pas ceux liés aux translations. Une fois la propriété `ManipulationMode` définie, l'élément déclenche les événements de manipulation lorsque ceux-ci sont détectés par Windows.

Certaines valeurs de l'énumération, comme `ManipulationModes.RotateInertia`, activent les effets d'inertie sur les manipulations, rendant les interactions plus naturelles et plus attractives. Les mouvements dus à l'inertie sont envoyés de manière quasi-identique aux manipulations tactiles, à l'exception près qu'elles ne sont pas directement liées à des con-

Copyright © 2013 Eyrolles.

tacts physiques, mais sont créées automatiquement par le moteur de manipulation de Windows. Les manipulations d'inertie sont différenciables car elles sont déclenchées après l'événement `UIElement.ManipulationInertiaStarting` ou lorsque la valeur du paramètre des événements de manipulation a sa propriété `IsInertial` à `true`.

L'utilisation des événements de manipulation est donc la suivante :

1 La propriété `ManipulationMode` est définie.

XAML

```
<Rectangle x:Name="monRectangle" ManipulationMode="Rotate,Scale" … />
```

C#

```
monRectangle.ManipulationMode = ManipulationModes.Rotate |
ManipulationModes.Scale;
```

C++

```
monRectangle->ManipulationMode = ManipulationModes::Rotate |
ManipulationModes::Scale;
```

2 S'abonner à l'un des événements qui sont levés lorsqu'une manipulation couverte par le `ManipulationMode` défini précédemment est détectée.

Événement	Condition de levée
`ManipulationStarting`	Le processeur de manipulation est initialisé.
`ManipulationStarted`	Une manipulation est détectée.
`ManipulationDelta`	Une propriété de la manipulation en cours a changé. Cet événement, souvent le plus important, permet de suivre l'évolution d'une manipulation pour fournir un retour visuel immédiat à l'utilisateur.
`ManipulationInertiaStarting`	L'effet d'inertie commence à être appliqué, généralement juste après que l'utilisateur a cessé d'interagir avec l'élément.
`ManipulationCompleted`	La manipulation sur l'élément est terminée (inertie comprise).

XAML

```
<Rectangle x:Name="monRectangle" ManipulationDelta=" monRectangle
_ManipulationDelta" … />
```

C#

```
monRectangle.ManipulationDelta += monRectangle_ManipulationDelta;
```

Copyright © 2013 Eyrolles.

C++

```
monRectangle.ManipulationDelta += ref new ManipulationDeltaEventHandler(this,
&MyControl::OnManipulationDelta);
```

Les événements `ManipulationDelta` et `ManipulationCompleted` contiennent des informations très précises sur les manipulations effectuées. Les propriétés les plus utiles des paramètres de ces événements (`ManipulationDeltaEventArgs` ou `ManipulationCompletedEventArgs`) sont listées dans le tableau de la section précédente (sur `GestureRecognizer`).

Propriété	Signification
`Cumulative`	Indique la valeur totale de la manipulation depuis sa détection
`Delta`	(uniquement pour `ManipulationDelta`) Indique la différence de valeur entre cet événement et le précédent
`IsInertial`	Indique s'il s'agit d'une manipulation d'inertie (générée par le moteur d'inertie)
`PointerDeviceType`	Indique le type de matériel utilisé pour effectuer la manipulation
`Position`	Point d'origine de la manipulation
`Velocities`	Indique les vitesses auxquelles ont lieu les différents éléments de la manipulation : • angulaire en degrés / millisecondes • réduction/agrandissement en unité indépendante par milliseconde • linéaire en unité indépendante par milliseconde La valeur par défaut de l'unité indépendante est de vingt pixels.

Les propriétés `Cumulative` et `Delta` sont de type `ManipulationDelta`, une structure indiquant les mesures de la manipulation en cours.

Propriété	Signification
`Rotation`	Angle de la rotation en degrés.
`Translation`	Déplacement horizontal ou vertical.
`Expansion`	Mesure des réductions/agrandissements en pixels.
`Scale`	Mesure des réductions/agrandissements en pourcentage.

L'application de ces valeurs peut se faire de plusieurs manières. La plus courante est d'appliquer les transformations liées aux manipulations directement aux éléments visuels.

Copyright © 2013 Eyrolles.

Par exemple, si un utilisateur effectue une rotation de 45° sur un rectangle, il faut que ce rectangle tourne sur lui-même de 45°. Techniquement cela revient à :

- sur le rectangle, définir la propriété `ManipulationMode` à au moins `ManipulationModes.Rotate` ou `ManipulationModes.RotateInertia` ;
- s'abonner à l'événement `ManipulationDelta` du rectangle afin de récupérer les informations de la manipulation de rotation au fur et à mesure qu'elle est effectuée ;
- dans le code appelé lors d'un événement `ManipulationDelta`, définir le centre de la rotation et appliquer une transformation `RotateTransform` via la propriété `RenderTransform` du rectangle, en donnant pour angle la valeur `Delta.Rotation` du paramètre de l'événement `ManipulationDelta`.

D'autres utilisations plus originales peuvent être faites des manipulations tactiles, comme le zoom sémantique qui, depuis une manipulation de pincement, effectue une modification importante dans la représentation des données. Dans cet exemple précis, les actions d'agrandissement ou de rétrécissement s'appliquent à partir d'un certain seuil, disponible par exemple via la propriété `Cumulative`.

Conclusion

Que ce soit en XAML ou en JavaScript, le Windows Runtime permet aux développeurs d'applications Windows Store de gérer élégamment les interfaces tactiles. Il fournit les API nécessaires pour s'adapter aux différents dispositifs de pointage, détecter efficacement les manipulations tactiles et même appliquer des effets d'inertie pour rendre les interactions plus naturelles et plus attrayantes.

Copyright © 2013 Eyrolles.

10

Interfaces avec les capteurs et périphériques

Les PC et tablettes contiennent de plus en plus de capteurs, introduits initialement dans les téléphones, qui multiplient les possibilités d'intégration dans l'environnement de l'utilisateur et augmentent la richesse des applications. Windows propose des API qui facilitent la manipulation de ces nouveaux périphériques au sein des applications Windows Store.

Figure 10–1

La géolocalisation

Si l'intérêt sur un PC de bureau peut sembler limité (mais fonctionne néanmoins pour peu qu'on l'équipe d'une puce ad hoc), sur les nouveaux ordinateurs portables en revanche, et spécialement sur les tablettes tactiles, l'utilisation de la localisation apporte souvent une réelle plus-value à une application. Par exemple, une application utilisant une carte se positionnera automatiquement à l'endroit où l'utilisateur se trouve ou bien une application de services listera les restaurants et magasins se trouvant dans un environnement proche.

Copyright © 2013 Eyrolles.

HISTOIRE Autoroute Express

Depuis plusieurs années déjà, le logiciel Autoroute Express de Microsoft communiquait avec un GPS. Malheureusement, les ordinateurs disponibles à cette époque étaient trop volumineux pour être facilement transportables et tirer parti de cette fonctionnalité. De plus, le GPS devait être acquis à part et connecté via USB ou RS232, ce qui n'était ni économique ni pratique.

Intégrer la géolocalisation dans une application est très simple. Cela commence par la déclaration de capacité, c'est-à-dire que l'application annonce au système d'exploitation et à l'utilisateur qu'elle désire accéder à des informations privées, notamment la géolocalisation de l'utilisateur. Pour réaliser cette déclaration, il faut ouvrir le manifeste de l'application dans Visual Studio et cocher la case *Emplacement* dans l'onglet *Capacités*.

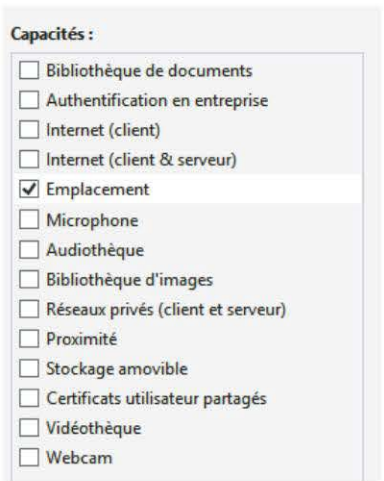


Figure 10–2
Déclaration de capacité pour la géolocalisation

Dès lors, les données de localisation sont accessibles. Elles sont fournies par le système via l'API `Windows.Devices.Geolocation`, qui se charge d'interagir avec le système pour servir de fournisseur de données de position.

La géolocalisation de Windows recourt à différents moyens pour fournir la position la plus précise possible. Elle utilise par défaut la résolution d'adresse IP et la triangulation Wi-Fi, qui sont peu consommatrices de ressources et gratuites, mais qui ont le défaut d'être peu précises (plusieurs centaines de mètres d'erreur). Il est aussi possible d'utiliser la triangulation via les cellules de communication des réseaux mobiles ou bien le positionnement GPS (*Global Position System*), dont la précision est de 5 mètres environ. Cette précision dépend à la fois des capacités matérielles de l'ordinateur/tablette utilisé, mais également du développeur, qui peut choisir le mode de précision via la propriété `DesiredAccuracy` détaillée dans les exemples suivants.

Copyright © 2013 Eyrolles.

Selon le besoin de l'application, il est possible de recueillir l'information sur l'emplacement de deux manières :

- ponctuellement, pour connaître à un instant T la position courante, utile pour faire une recherche d'informations aux alentours ou pour enregistrer la position sur une photo ou un réseau social ;
- en continu, pour suivre la trajectoire de l'utilisateur (de la machine plus exactement).

De plus, malgré l'ajout au manifeste, lors de la première tentative de récupération de données d'emplacement, une demande de confirmation est automatiquement lancée par Windows auprès de l'utilisateur pour valider son accord d'être géolocalisé par l'application.

Figure 10–3
Avertissement automatique lors du premier lancement de l'application

On se rappellera que l'utilisateur peut toujours refuser une telle demande et que l'application doit être à même de le gérer : message d'avertissement, fonctionnement en mode « dégradé », passage en grisé de certaines options des menus… mais à aucun moment le refus de l'utilisateur ne doit faire « planter » l'application.

Accès ponctuel

L'accès ponctuel aux données de localisation est extrêmement simple. Il faut en premier lieu instancier un fournisseur à l'aide de la classe `Geolocator`. Puis on fait appel à sa méthode `GetPositionAsync()` ; elle renvoie une latitude, une longitude ainsi qu'une adresse civique qui, à défaut d'être précise, pourra donner le nom de la ville ou du pays où se trouve l'utilisateur.

Exemple en C#

```
try
{
    Geolocator monLocator = new Geolocator();

    monLocator.DesiredAccuracy = PositionAccuracy.Default;
    Geoposition position = await monLocator.GetGeopositionAsync();

    double latitude = position.Coordinate.Latitude;
    double longitude = position.Coordinate.Longitude;
    double acc = position.Coordinate.Accuracy;
}
```

Copyright © 2013 Eyrolles.

```
catch (System.UnauthorizedAccessException)
{
    // pour les cas où le manifeste n'est pas bien rempli ou si
l'utilisateurrefuse la fonctionnalité
}
catch (Exception ex)
{
    // pour les timeout ou toute autre erreur
}
```

Exemple en JavaScript

```
var monLocator = new Windows.Devices.Geolocation.Geolocator();
monLocator.positionAccuracy =
Windows.Devices.Geolocation.PositionAccuracy.high;
monLocator.getGeopositionAsync().done(gestionPosition, gestionErreur);

function gestionPosition(pos) {
    var lat = pos.coordinate.latitude;
    var long = pos.coordinate.longitude;
    var acc = pos.coordinate.accuracy;
}

function gestionErreur(err) {
    // gestion des erreurs
}
```

Exemple en C++

```
Windows::Devices::Geolocation::Geolocator^ geolocator = ref new Geolocator();
    task<Geoposition^> geopositionTask(geolocator->GetGeopositionAsync());
    geopositionTask.then([this](task<Geoposition^> getPosTask)
    {
        try
        {
           Geoposition^ pos = getPosTask.get();

           auto latitude= pos->Coordinate->Latitude.ToString();
           auto longitude = pos->Coordinate->Longitude.ToString();
           auto position = pos->Coordinate->Accuracy.ToString();
        }
        catch (Exception^ ex)
        {

        }
    });
```

Copyright © 2013 Eyrolles.

En complément de la précision configurable, il est possible de transmettre deux paramètres à la méthode `GetGeopositionAsync` :

- le paramètre `Timeout` indique la durée maximum que peut prendre la méthode pour rapporter la localisation la plus précise possible. Une fois le temps limite atteint, la méthode retourne la position récupérée, même si elle n'est pas précise, ou bien retourne une erreur si aucune position n'a pu être détectée ;
- le paramètre `MaximumAge` indique à la requête de retourner la dernière position mise en cache si celle-ci est moins ancienne que le paramètre transmis. Ainsi, si jamais l'application n'a besoin que d'une position approximative, cela fournit cette information de façon quasi instantanée sous réserve qu'une position ait déjà été mise en cache.

Accès continu

L'accès continu n'est pas plus complexe et consiste à s'abonner au fournisseur de localisation et à écouter chaque message de modification d'emplacement.

Exemple en C#

```
private Geolocator monLocator = new Geolocator();

private void DemarrageEcoute(object sender, RoutedEventArgs e)
{
    monLocator.DesiredAccuracy = PositionAccuracy.High;
    monLocator.PositionChanged += new TypedEventHandler<Geolocator,
PositionChangedEventArgs>(OnPositionChanged);
}

private void OnPositionChanged(Geolocator sender, PositionChangedEventArgs e)
{
    Geoposition position = e.Position;

    double latitude = position.Coordinate.Latitude;
    double longitude = position.Coordinate.Longitude;
    double acc = position.Coordinate.Accuracy;
}
```

Exemple en JavaScript

```
var monLocator = new Windows.Devices.Geolocation.Geolocator();
monLocator.desiredAccuracy = Windows.Devices.Geolocation.PositionAccuracy.high;
monLocator.addEventListener("positionchanged", onPositionChanged);
```

Copyright © 2013 Eyrolles.

```
function onPositionChanged(e) {
    var lat = e.position.coordinate.latitude;
    var long = e.position.coordinate.longitude;
    var acc = e.position.accuracy;
}
```

Exemple en C++

```
Windows::Devices::Geolocation::Geolocator^ geolocator;

void App::DemarrageEcoute(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
{
    geolocator->PositionChanged::add(ref new TypedEventHandler<Geolocator^,
PositionChangedEventArgs^>(this, &Scenario1::OnPositionChanged));
}
void App::OnPositionChanged(Geolocator^ sender, PositionChangedEventArgs^ e)
{
    auto latitude= e->Position->Coordinate->Latitude;
    auto longitude = e->Position->Coordinate->Longitude;
    auto position = e->Position->Coordinate->Accuracy;
}
```

Bonnes pratiques de la géolocalisation

Si la géolocalisation est simple à utiliser, il faut néanmoins rester conscient de certaines de ses spécificités.

La première chose à garder en tête est que son utilisation consomme de la batterie et rien n'est plus déplaisant que d'avoir un périphérique qui ne tient pas la charge le temps d'un voyage ou d'une ballade. Par défaut, la valeur de `Windows.Devices.Geolocation.PositionAccuracy.default` est zéro, c'est-à-dire une optimisation des performances et de l'énergie consommée. Le plus souvent, une application sera tentée de demander une précision plus grande, ce qui consomme alors beaucoup plus d'énergie.

De plus, il faut admettre et comprendre que les données ne sont pas précises au centimètre près, même en mode GPS. Ce dernier peut aussi, par nature, indiquer un emplacement sensiblement différent sans pour autant que l'utilisateur ne se soit déplacé (phénomènes relativistes réclamant des corrections sur le retard des horloges en mouvement à bord des satelittes et autres considérations techniques de même ordre). Ainsi, dans le cadre de l'accès continu aux données de localisation, il est nécessaire de bien prendre en compte et de régler les propriétés `movementThreshold` et `reportInterval`, qui limitent les notifications du fournisseur de géolocalisation en définissant un déplacement minimum et un intervalle de temps minimum entre deux envois d'informations pour, là encore, économiser la batterie.

Copyright © 2013 Eyrolles.

L'autre point important reste l'expérience utilisateur. Bien que les latitudes et longitudes servent à certains scénarios, il est préférable de localiser l'utilisateur à l'aide de l'adresse civique ou bien d'une carte qui est centrée sur l'emplacement actuel. L'utilisateur doit d'ailleurs pouvoir actualiser manuellement les données d'emplacement.

Enfin, l'utilisateur a la possibilité de désactiver la fonctionnalité de géolocalisation. De ce fait, le `Geolocator` n'est donc pas toujours en mesure de fournir des données. Dans tous les cas, des exceptions peuvent être levées et des messages clairs et appropriés doivent être affichés à l'utilisateur.

Les capteurs

Si tous les capteurs traités au sein de ce chapitre ne se trouvent pas nécessairement sur toutes les machines, Windows 8 offre un accès simple aux données qu'ils peuvent fournir. Les capteurs sont d'une grande diversité et sont utilisés de mille façons, notamment pour tenir compte de l'environnement de l'utilisateur (luminosité, orientation...). Si ces données (mouvements principalement) sont plus facilement utilisables dans un jeu vidéo que dans une application LOB (*line of business*), cela ne doit surtout pas décourager le développeur inventif. Utilisées de façon créative, les données des capteurs ouvrent la voie à des applications d'un type nouveau.

Parmi la liste des capteurs standards gérés par le système (et ses API), il est possible de trouver :

- un accéléromètre 3D, pour calculer l'accélération ;
- un capteur gyroscopique, pour évaluer la vitesse de rotation de la tablette ;
- un capteur de luminosité ambiante ;
- une boussole, pour connaître le nord géographique ou le nord magnétique ;
- un capteur d'orientation, pour savoir si l'unité mobile est en mode portrait/paysage et dans quel sens (à l'endroit ou à l'envers) ;
- un capteur composite, pour fournir l'orientation dans un espace complexe à 9 axes très précis intégrant les données de plusieurs capteurs physiques.

Ces capteurs sont pour la plupart liés à un composant matériel précis. Ils fournissent des données brutes au système d'exploitation, qui sont alors exposées au travers des API WinRT afin de les rendre accessibles et consommables par les applications. Il n'est pas nécessaire d'ajouter des déclarations dans le manifeste, les périphériques présents étant automatiquement disponibles pour l'application (exception faite de la géolocalisation, car elle peut constituer une atteinte à la vie privée).

Techniquement, il suffit de récupérer une instance du capteur désiré `Windows.Devices.Sensors.<moncapteur>.GetDefault()`, puis d'appeler sa méthode `GetReading()` qui

Copyright © 2013 Eyrolles.

retourne des données standardisées et prêtes à l'usage, comme le montre l'exemple suivant avec le capteur de luminosité.

Récupérer la luminosité en C#

```
private void ObtientLuminositeAmbiante()
{
    var sensor = Windows.Devices.Sensors.LightSensor.GetDefault();
    if (sensor != null)
    {
        var reading = sensor.GetCurrentReading();
        if (reading != null)
        {
            float lumi = reading.IlluminanceInLux;
            if (lumi < 1000)
            {
                // il fait sombre
            }
            else if (lumi <= 3000)
            {
                // il fait clair à l'intérieur
            }
            else if (lumi <= 30000)
            {
                // il fait sombre dehors
            }
            else
            {
                // il fait plutot clair dehors
            }
        }
    }
    else
    {
        // pas de capteur de ce type disponible
    }
}
```

> **NOTE**
>
> Les codes C++ et JavaScript étant quasi identiques au code C#, le lecteur n'aura aucune difficulté à transposer ce dernier.

Suivant le même fonctionnement qu'avec les données de géolocalisation, il est possible d'accéder aux données des capteurs :

- ponctuellement, avec l'appel à la méthode `GetCurrentReading` (exemple précédent),

Copyright © 2013 Eyrolles.

- ou bien de façon continue en s'abonnant à l'événement `ReadingChanged`, qui retourne les informations du capteur toutes les *X* secondes (valeur définie par la propriété `Interval`) :

```
var sensor = Windows.Devices.Sensors.LightSensor.GetDefault();
    sensor.ReadingChanged += sensor_ReadingChanged;
```

Ces capteurs sont donc une source d'informations supplémentaires pour enrichir les fonctionnalités des applications, leur permettant d'adapter leur style visuel en fonction de la luminosité ou de l'orientation, mais également de fournir une expérience immersive, voire de la réalité augmentée, en combinant géolocalisation et orientation. Il est ainsi possible d'imaginer un jeu de course de voitures où l'utilisateur tiendrait la tablette comme un volant et la voiture réagirait aux inclinaisons effectuées par l'utilisateur.

Utilisation de la technologie NFC

La technologie NFC *(Near Field Communication)* est une communication sans fil à très courte portée (moins de 10 cm) qui permet l'échange d'informations entre deux unités mobiles (généralement). L'objectif de ces périphériques est de transmettre des informations simplement et de façon sécurisée, en posant simplement le téléphone contre une borne de paiement par exemple. L'échange d'information entre deux smartphones peut aussi s'effectuer par contact des deux appareils grâce à cette technologie.

IDÉE

On peut imaginer une application qui utiliserait la technologie NFC pour récupérer des informations d'un autre périphérique, comme une fiche détaillée de la pièce en cours dans un musée, fiche dont les informations seraient stockées dans une « étiquette » (un tag) collée sous un tableau d'art par exemple.

Il existe aussi des étiquettes NFC dites passives, car ne nécessitant pas d'énergie ; elles utilisent le champ d'induction engendré par l'antenne du lecteur (périphérique ou unité mobile) au moment de la lecture des informations. Certaines permettent également l'écriture d'informations par un autre périphérique, servant ainsi d'étiquettes de « stockage ».

Windows fournit au travers de son API de proximité la possibilité d'utiliser la communication NFC si le matériel du périphérique le permet. Bien que d'utilisation plus complexe que pour les capteurs de base, les API de Proximité de Windows simplifient l'utilisation du capteur NFC grâce notamment à l'encapsulation des données NDEF (*NFC Data Exchange Format* – standard pour le transfert d'information via NFC).

Copyright © 2013 Eyrolles.

Les API gèrent les formats NDEF standards, mais fournissent également des messages Windows compatibles NDEF.

Tableau 10–1 Liste des messages NFC les plus courants

Message	Envoi d'info	Lecture d'info	Description
Windows	X	X	Transmission de données binaires.
WindowsUri	X	X	Transmission d'un URI. Le protocole de l'URI est libre et si une application est associée au protocole, alors celle-ci se lance automatiquement.
WindowsMime	X	X	Envoie un fichier dans un format particulier et en précise le format MIME. Le système Windows proposera de lancer les applications associées à ce format.
Windows:WriteTag	X		Identique au protocole Windows mais pour écrire sur un tag statique.
WindowsUri:WriteTag	X		Identique au protocole WindowsUri mais pour écrire sur un tag statique.
WindowsMime:WriteTag	X		Identique au protocole WindowsMime mais pour écrire sur un tag statique.
LaunchApp:WriteTag	X		Lance une application en lui transmettant des paramètres. Le format du message est : parametres[tab] plateforme [tab] package_app!nom_application Par exemple : param1=Param1\tWindows\tMonApplication_jg9nn7vn05wdy!MonApplication
WriteableTag		X	Message reçu lorsqu'un tag statique est à proximité. Le message contient la taille maximum d'information que peut recevoir le tag.
NDEF	X		Publie des données formatées en tant qu'enregistrement NDEF.
NDEF:ext		X	Format utilisé pour la lecture de messages NDEF.
NDEF:MIME		X	Format utilisé pour la lecture de messages MIME au format NDEF.
NDEF:URI		X	Format utilisé pour la lecture de messages URI au format NDEF.
NDEF:WriteTag	X		Utilisé pour écrire des données NDEF sur un tag statique.
NDEF:Unknown		X	Utilisé pour les messages NDEF non typés, ayant probablement un format personnalisé.

Copyright © 2013 Eyrolles.

Envoyer ou recevoir des informations via NFC

S'il est équipé du matériel adéquat, un périphérique peut, à la demande d'une application, transmettre des informations à un autre périphérique. Cependant, du fait que le canal de communication est disponible uniquement lorsque les deux périphériques sont à proximité, l'utilisation de la technologie NFC requiert de prêter attention à certains points.

Pour utiliser la technologie NFC dans une application, il est nécessaire de le déclarer dans le manifeste de celle-ci en cochant la case *Proximité*.

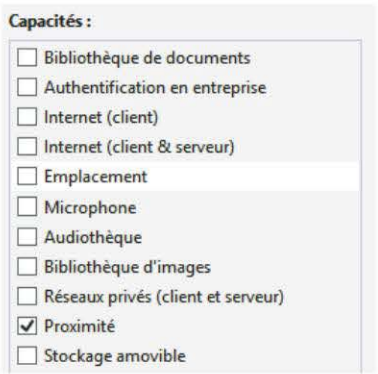


Figure 10–4
Déclaration de capacité d'utilisation du capteur de proximité

ATTENTION Choix de l'utilisateur

Ne pas oublier que l'utilisateur est notifié qu'une tentative de connexion NFC est en cours et qu'il peut la refuser. Il est important que l'application sache le gérer.

L'application doit toujours vérifier qu'elle a accès au capteur demandé, qu'il s'agisse du capteur de proximité ou des autres. Pour déterminer la présence d'un capteur NFC, il faut utiliser la classe `PeerFinder`, qui fournit les modes NFC utilisables, ou plus simplement demander le périphérique NFC par défaut (les rares périphériques qui ont une puce NFC n'en ont souvent qu'une).

```
using Windows.Networking.Proximity;
private void Publication()
{
    ProximityDevice device = ProximityDevice.GetDefault();
    if(device!=null)
    {
        long messageId = device.PublishUriMessage(new Uri(monUri));
    }
}
```

Copyright © 2013 Eyrolles.

OUTIL Bibliothèque NDEF

Afin de faciliter les traitements du format NDEF, une bibliothèque gratuite est à disposition sur CodePlex :

- http://ndef.codeplex.com/

Elle simplifie les traitements dans les scénarios les plus courants, tels que l'usage des tags *SMS*, *Social* ou *Geo*, et gère l'écriture et la lecture de paquets NDEF correctement formatés à l'aide de méthodes génériques.

À l'inverse, le périphérique peut aussi servir pour la lecture et la réception des informations, par exemple dans le cadre d'une application qui s'abonne au canal d'écoute NFC en précisant le type de message souhaité. Ce filtrage s'effectue en appelant la méthode `SubscribeForMessage`. Il suffit alors de déclarer un délégué dont le rôle sera de traiter les messages reçus, comme le montre l'exemple suivant :

```
private void ecoute()
{
    ProximityDevice peripheriqueNFC = ProximityDevice.GetDefault();
    if (peripheriqueNFC != null)
    {
        messageId = peripheriqueNFC.SubscribeForMessage("WindowsUri",
            new MessageReceivedHandler((ProximityDevice sender,
                                        ProximityMessage message) =>
            {
                    String uri = CryptographicBuffer.ConvertBinaryToString(
                         BinaryStringEncoding.Utf16LE, message.Data).TrimEnd();
                    // utilisation de l'URI
                    ...

                    peripheriqueNFC.StopSubscribingForMessage(messageId);
            })
        );
    }
}
```

Utiliser NFC sans NFC

S'il est possible d'utiliser la technologie NFC directement depuis une application avec les API qui ont été présentées précédemment, il est aussi envisageable de laisser le système d'exploitation le faire et de ne s'occuper que des données à transmettre ou recevoir.

En effet, NFC est inclus au sein de Windows 8 et, depuis les applications Windows Store, il est possible de faire appel au charme Périphériques pour transmettre des données NFC. Lorsque l'utilisateur dispose d'un matériel compatible, il lui suffit d'utiliser le charme Périphériques, qui offre l'option *Toucher et Partager* pour initier un échange NFC.

Copyright © 2013 Eyrolles.

Du côté de l'application (voir chapitre 7), le développeur doit utiliser un `DataPackage` et le préparer au moment de l'événement `DataRequested` du `DataTransferManager`. Ce mode de transfert est plus simple que l'utilisation directe des API NFC, mais il ne transfère que des URI et des fichiers (`StorageItem`).

Quant à la lecture d'informations NFC de façon automatisée, l'application doit déclarer dans son manifeste les protocoles et les formats de fichiers MIME qu'elle sait gérer (voir l'utilisation des protocoles d'action au chapitre 7). En cas de réception d'un message NFC, Windows propose à l'utilisateur d'ouvrir le message transmis à l'aide des applications compatibles. Le mode de transfert NFC est ainsi totalement transparent pour l'utilisateur et ne demande que peu d'efforts de la part du développeur.

Bien qu'existant depuis de nombreuses années, la technologie NFC commence tout juste à faire son apparition auprès du grand public. Avec son intégration croissante dans les unités mobiles, il semble intéressant de savoir exploiter cette technologie, qui permet notamment au périphérique Windows d'échanger des données avec d'autres plates-formes, comme Android ou bien de simples tags statiques.

La gestion des périphériques

Comme expliqué dans les premiers chapitres de ce livre, les applications Windows Store sont exécutées dans un bac à sable (*sandbox*) pour garantir une sécurité maximale à l'utilisateur. Cela a pour conséquence de bloquer tout accès des ressources extérieures à ce bac à sable, tels les périphériques connectés à l'ordinateur. Cette restriction pouvant apparaître trop drastique Windows fournit des API listant les périphériques connectés et donnant accès à certains d'entre eux.

Lister les périphériques disponibles

Au sein d'une application Bureau, lister les périphériques peut se faire à l'aide de l'API WMI (*Windows Management Instrumentation*). Pour obtenir des informations sur le système au sein des applications Windows Store, il est nécessaire d'utiliser les API de l'espace de noms `Windows.Devices`.

Dans cet espace de noms se trouve la méthode `DeviceInformation.FindAllAsync()` qui liste tous les périphériques répondant au filtre de recherche transmis. Ce dernier est rédigé en utilisant la syntaxe AQS *(Advanced Query Syntax)*, une version simplifiée du langage SQL utilisée par les produits Microsoft. Dans ce cas précis, le filtre réduit le nombre de périphériques retournés par la méthode `FindAllAsync` (actifs/non actifs) et, surtout, cibler un type de périphérique particulier (imprimante, caméra, disques amovibles...).

Copyright © 2013 Eyrolles.

L'exemple qui suit utilise un sélecteur AQS qui filtre à la fois les périphériques activés et possédant un GUID d'interface particulier correspondant ici aux périphériques d'acquisition d'images. Le GUID représente l'interface « générique » du périphérique, c'est-à-dire l'interface sous laquelle le périphérique s'enregistre auprès du système.

Lister les périphériques d'acquisition d'images en C#

```
string filtre = "System.Devices.InterfaceClassGuid:=\"{
  E5323777-F976-4F5B-9B55-B94699C46E44
}\" AND System.Devices.InterfaceEnabled:=System.StructuredQueryType
                                                    .Boolean#True";
    var periphs = await DeviceInformation.FindAllAsync(filtre);

    foreach (DeviceInformation deviceInterface in periphs)
    {
        string nomPeriph = deviceInterface.Name;
       DeviceThumbnail iconePeriph = await deviceInterface.GetThumbnailAsync();
    }
```

Il est à noter qu'il s'agit ici d'illustrer une API particulière et qu'il existe des moyens plus simples pour détecter la présence d'une caméra et s'en servir au sein d'une application Windows Store.

Interagir avec les périphériques

Connaître la liste des périphériques disponibles est utile, les utiliser l'est encore plus. On peut par exemple supposer une application offrant un moyen de sauvegarder ses données sur un disque amovible. Ainsi, il est facile d'imaginer un cas d'utilisation qui exploite l'export de données sur un périphérique de stockage amovible pour réaliser une sauvegarde. L'application proposerait alors uniquement la liste des espaces de stockages différents du disque dur principal pour dupliquer la donnée en prévision d'une possible panne matérielle.

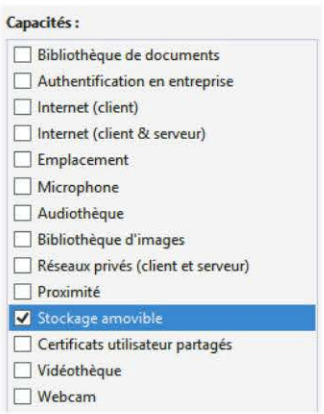


Figure 10–5
Déclaration de capacité d'utilisation du stockable amovible

Copyright © 2013 Eyrolles.

Dans un tel cas, l'application va tenter d'accéder au périphérique pour y stocker des données. Cette action, qui implique une ressource extérieure au bac à sable, réclame d'être signalée à l'utilisateur. Il est ainsi nécessaire de déclarer l'utilisation des stockages amovibles dans le manifeste du projet.

Cette déclaration effectuée, il faut utiliser la méthode `FromId()` pour récupérer un objet `StorageFolder` considéré par l'application comme un dossier de stockage classique. Il est alors possible d'y écrire ou d'y lire des fichiers.

```
StorageFile fichierAEcrire = FichierASauvegarder();
StorageFolder storage = StorageDevice.FromId(peripheriqueID);
await sourceFile.CopyAsync(storage, fichierAEcrire.Name,
NameCollisionOption.ReplaceExisting);
```

La ressource obtenue se comporte comme n'importe quel dossier et il est possible de parcourir le `StorageFolder` pour récupérer ses dossiers et fichiers.

On peut ainsi imaginer une application chargée de traiter toutes les photos situées sur la carte mémoire d'un appareil branché à l'ordinateur/tablette. Il faut alors utiliser les méthodes `GetFilesAsync` et `GetFoldersAsync`.

```
var photos = await storage.GetFilesAsync();
```

Grâce aux contrats de Windows (voir chapitre 7), l'application sera associée à certains périphériques. Dans le cas de l'application fictive de traitement de photos, une simple modification du fichier manifeste conduit à exécuter celle-ci chaque fois qu'un appareil photo est branché à l'ordinateur via un câble USB ou qu'une carte mémoire est insérée dans un lecteur de l'ordinateur.

Il faut pour cela se rendre dans l'onglet *Déclarations* du fichier manifeste de l'application et ajouter une déclaration *Périphérique de lecture automatique*. Trois champs sont alors à préciser :

- le verbe correspond à un mot-clé qui sera utilisé par l'application pour connaître le mode de lancement. Ce nom est au choix du développeur ;
- le nom complet de l'action contient le texte affiché à l'utilisateur lors des choix disponibles pour l'exécution automatique ;
- l'événement de périphérique est déclenché par le système lors de l'exécution automatique de certains périphériques (voir aparté pour la liste des événements disponibles).

Concernant l'exemple d'application de traitement de photographies, le paramètre le plus important serait ici l'événement `WPD\ImageSource` qui correspond à la technologie Windows Portable Devices et donne accès à une multitude d'appareils multimédia comme les appareils-photos, les caméscopes ou bien les téléphones mobiles.

Copyright © 2013 Eyrolles.

INFORMATION Les événements de périphériques

On obtient la liste complète des événements des périphériques en ouvrant le registre Windows à la clé `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\AutoplayHandlers`, puis en regardant les clés disponibles dans la partie `EventHandlers`. Les valeurs ne sont malheureusement pas documentées, mais parlent souvent d'elles-mêmes. Il est d'ailleurs possible d'enregistrer l'application pour plusieurs événements de périphériques. On se méfiera toutefois de ce genre de « hack » s'appuyant sur des fonctionnalités non documentées.

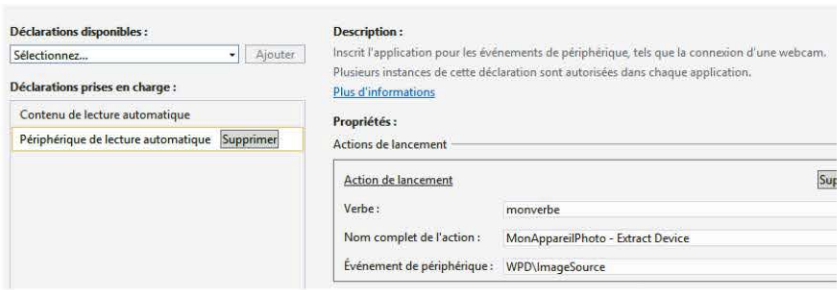


Figure 10–6
Enregistrement pour la lecture automatique

Une fois l'application déployée, elle sera proposée pour l'exécution automatique chaque fois qu'un appareil multimédia contenant des photos sera branché à l'ordinateur.

Figure 10–7
Lecture automatique au branchement de l'appareil photo

Il est également possible de s'associer au contenu d'un périphérique de stockage. Il faut pour cela déclarer le *Contenu de lecture automatique*. La différence vient de la valeur de l'événement de contenu. Dans l'exemple, c'est la valeur `ShowPictureOnArrival` qui est utilisée car elle correspond aux cartes mémoire d'appareils-photos.

Copyright © 2013 Eyrolles.

INFORMATION Les événements de contenu

Les événements comme `ShowPictureOnArrival` sont définis par le système. Celui-ci analyse les fichiers d'un périphérique et, selon leur type, définit l'événement à lancer. Par exemple, les événements courants du tableau suivant sont définis :

Contenu	Événement
Images	`ShowPicturesOnArrival`
Musiques	`PlayMusicFilesOnArrival`
Vidéos	`PlayVideoFilesOnArrival`
Contenu varié	`MixedContentOnArrival`
CD/DVD vierge	`HandleCDBurningOnArrival`
CD audio	`PlayCDAudioOnArrival`
DVD video	`PlayDVDMovieOnArrival`

Ceci implique qu'une clé USB standard peut être considérée comme carte mémoire photo si elle contient une majorité de fichiers images et, inversement, une carte mémoire d'appareil-photo pourrait déclencher un événement `PlayVideoFilesOnArrival`.

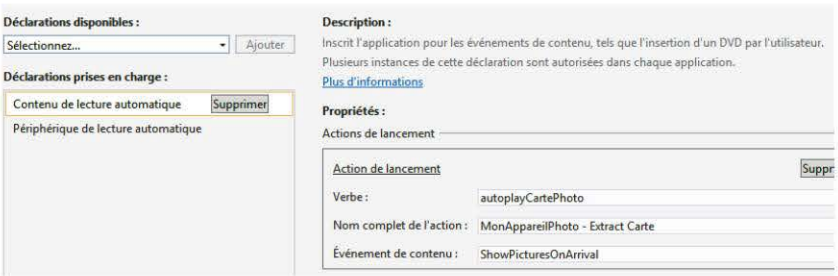


Figure 10–8
Déclaration des actions de lancement

L'application est alors proposée à l'utilisateur lorsqu'une carte mémoire, dont le contenu se trouve être majoritairement des images, est branchée sur l'ordinateur. Le typage par contenu est intéressant, car il sert à catégoriser un périphérique externe en fonction de ce qu'il véhicule pour aiguiller l'utilisateur vers la ou les application(s) appropriée(s), cela au prix d'un court temps d'analyse des fichiers se trouvant sur le support.

Les deux associations ainsi déclarées (périphérique et contenu) sont enregistrées auprès du système et deviennent paramétrables par l'utilisateur dans le panneau de configuration, partie *Exécution Automatique*.

Rarement l'intégration d'une application dans un système n'aura été aussi simple. Et au sein des applications Windows Store, le développeur a, grâce aux manifestes et aux API, de nombreuses possibilités pour proposer des fonctionnalités enrichies et fournir aux utilisateurs des applications de qualité répondant à de nombreux besoins.

Copyright © 2013 Eyrolles.

Figure 10–9
Paramétrage de l'Exécution Automatique de Windows

Les API de Windows offrent aujourd'hui aux développeurs les moyens de produire des logiciels plus attractifs et plus connectés aux besoins des utilisateurs, voire d'en susciter de nouveaux.

Conclusion

Grâce à ses API enrichies et grâce à ses fichiers manifestes qui réalisent l'enregistrement automatique des fonctionnalités auprès du système, Windows 8 propose au développeur un environnement quasi entièrement accessible doté d'une interface de pilotage simplifiée, qui permet l'utilisation aussi bien des capteurs que des nombreux périphériques susceptibles d'être connectés à l'ordinateur. Les applications Windows disposent de tout le nécessaire pour être vivantes, originales et adaptées à leur environnement (*environment-aware*).

Copyright © 2013 Eyrolles.

11

Le Windows Store

L'application est terminée et le moment est venu de la publier sur le Windows Store afin de la rendre disponible pour le plus grand nombre. Ce chapitre présente les détails d'une étape délicate, dont les choix définiront le succès ou non de l'application.

Figure 11–1

Le Windows Store

Le Windows Store, à l'instar de nombreux marchés d'applications (pour iPhone, Android ou même Windows Phone), est un portail intégré au sein de Windows qui, d'un simple clic, donne accès à des milliers d'applications catégorisées, ordonnées et notées. Chacune dispose d'une fiche descriptive donnant à l'utilisateur potentiel un aperçu du visuel, mais aussi des fonctionnalités proposées. Il est alors possible de télécharger et d'installer simplement l'application sur son système, où elle devient immédiatement utilisable.

Pour l'utilisateur, ce nouveau marché numérique accroît de façon illimité les fonctionnalités de son système en lui ajoutant de nouvelles applications, gratuites ou

Copyright © 2013 Eyrolles.

payantes (de 1,49 $ à 999 $). Il est également possible d'y référencer des applications dites « Bureau » mais l'intégration se limite alors à une fiche descriptive sans achat ni téléchargement possible. L'utilisateur est redirigé vers le site de l'éditeur depuis lequel il peut télécharger l'application.

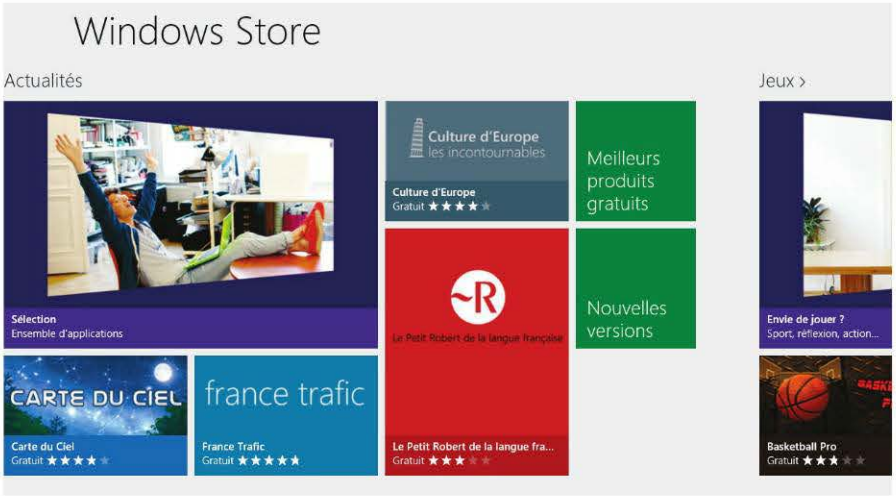


Figure 11–2
Vue générale du Windows Store

Grâce à son compte Microsoft, chaque utilisateur peut télécharger des applications qui seront disponibles sur tous les ordinateurs Windows 8 sur lesquels il s'authentifiera. Ainsi, pour la première fois, et à l'instar du Windows Phone Store, Microsoft fournit à son système d'exploitation le plus célèbre une plate-forme dématérialisée accessible au plus grand nombre.

Une découverte facilitée

Le souci d'Internet, et des sites de téléchargement de façon générale, est de ne pas noyer le visiteur sous un flot d'informations trop important pour qu'il puisse les utiliser convenablement. C'est pourquoi le Windows Store a été pensé pour faciliter son exploration : à la fois pour que l'utilisateur trouve exactement ce qu'il cherche, mais également pour qu'il découvre ce qu'il n'avait pas forcément pensé à chercher.

Recherche, parcours par catégorie, listes triées, promotions ou notation des utilisateurs, tout est mis en œuvre pour que chacun trouve l'application qui lui convient.

Présentant des fiches détaillées sur chaque application, le Windows Store est le point d'entrée unique pour la publication d'applications pour Windows. Le développeur a donc la responsabilité de bien comprendre son concept et de s'aligner sur les recommandations Microsoft pour tirer parti des fonctionnalités qu'il propose. Il lui incombe également de s'assurer de la promotion de son application pour qu'elle sorte du lot, un « lot » souvent pourvu de nombreux concurrents. Savoir vendre son application est aussi important que de savoir la programmer !

Copyright © 2013 Eyrolles.

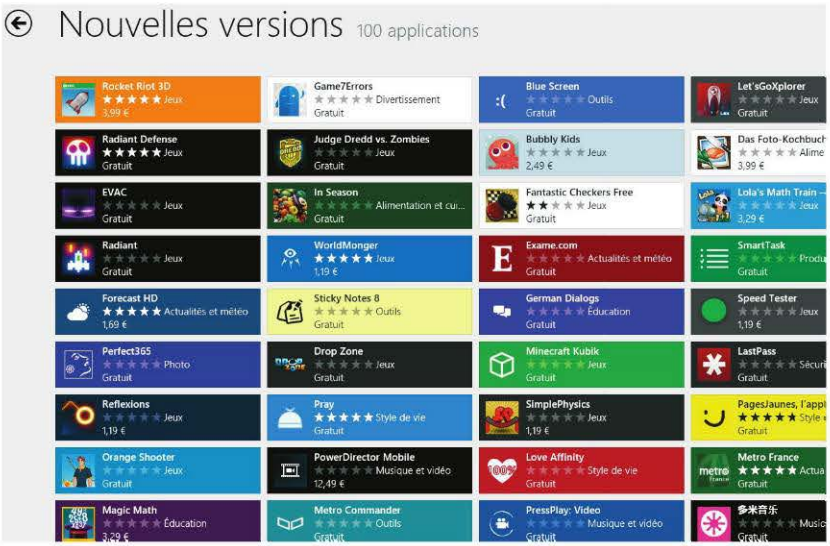


Figure 11–3
Mise en avant
des applications récentes

Chaque application possède sa propre fiche contenant une description générale, à laquelle s'ajoutent plusieurs captures d'écrans donnant au visiteur du magasin une idée de l'aspect graphique et des fonctionnalités proposées. Des informations importantes comme les langues prises en charge, les plates-formes matérielles ciblées (x86, x64 ou ARM), ainsi que les fonctionnalités particulières de l'application (accès à Internet, accès aux fichiers de l'utilisateur, utilisation de la Webcam...) sont aussi précisées. Attachez la plus grande importance à cette fiche car elle sera votre premier contact avec l'utilisateur potentiel, votre unique chance peut-être de le convaincre de télécharger et de tester votre application.

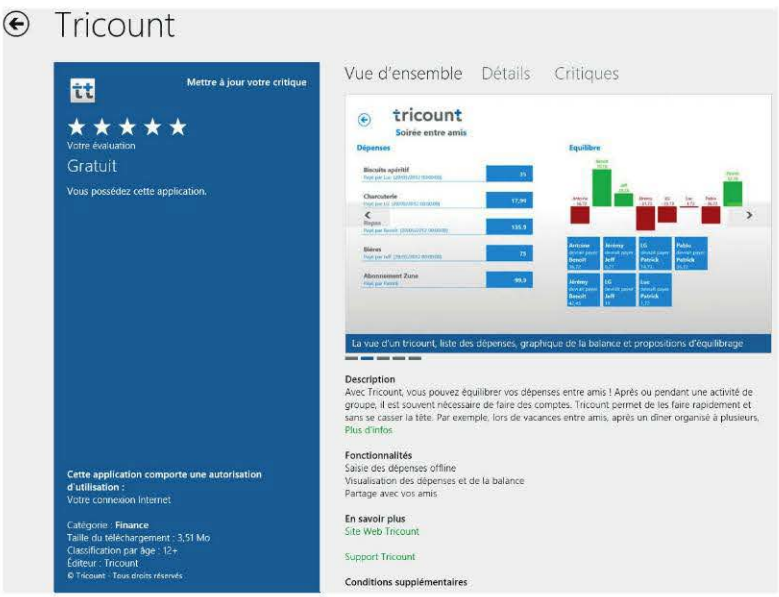


Figure 11–4
Fiche détaillée d'une
application Windows 8

Copyright © 2013 Eyrolles.

Publier son application sur le Store

Ce magasin en ligne est un moyen simple pour un développeur de promouvoir son application. Il n'y a plus besoin de créer son propre site web pour la distribuer, plus besoin d'aller la référencer sur chaque site de téléchargement ; il suffit simplement de l'enregistrer auprès de Microsoft pour que l'application s'intègre au sein de Windows et soit plus facilement découverte par l'utilisateur final.

De plus, à l'aide d'un système de paiement intégré ou bien en implémentant un module de publicité au sein de l'application, chaque développeur est à même de tirer un revenu de son application.

Licence développeur

La publication sur le Windows Store déclenche automatiquement une analyse de l'application par Microsoft, afin de s'assurer qu'elle répond bien aux critères de sélection et ne possède pas de partie malveillante (malware, virus). Cela entraîne un coût. C'est pour cela que le kit de développement comporte un module de « prévalidation » de votre application pour ne la soumettre qu'une fois que vous serez certain (ou presque) qu'elle passera les tests finaux réalisés par Microsoft. Ce dernier s'assure également de l'identité de la personne publiant des applications. Ainsi, Microsoft rend obligatoire la création d'un compte auquel on associe une licence développeur valable une année.

> **IMPORTANT Expiration de la licence**
>
> Il est important de comprendre que la licence est un contrat entre le développeur et Microsoft. C'est pourquoi, lorsque la licence expire, toutes les applications du développeur qui ne renouvelle pas son contrat peuvent tout simplement être supprimées du Windows Store, mais également des machines les ayant installées. De plus, le nom de l'éditeur ne serait plus réservé et deviendrait utilisable par n'importe qui.

Cette licence est nécessaire pour publier une application, qu'elle soit gratuite ou non. En contrepartie, le développeur a le contrôle total pour décider du prix, ainsi que des pays dans lesquels l'application est disponible. Il est en revanche important de savoir que Microsoft prélève une commission de 30 % des revenus engendrés par la vente des applications (ou 20 % si les revenus dépassent 25 000 $).

Pour créer son compte, il est nécessaire de se rendre sur le site dédié et de remplir le formulaire d'inscription : https://appdev.microsoft.com/StorePortals/fr-fr/Account/Signup/Start

Une fois le compte développeur configuré, la soumission d'applications au Windows Store peut commencer.

Copyright © 2013 Eyrolles.

Procédure d'enregistrement et de validation

Pour chaque publication ou mise à jour d'une application, il est nécessaire de faire une déclaration sur le portail de développement. Chaque déclaration d'application nécessite des informations textuelles, des captures d'écran, mais surtout le package de l'application, c'est-à-dire son module d'installation, celui qui est déployé sur l'ordinateur des utilisateurs.

Ce package, sous la forme d'un fichier à l'extension `appxmanifest`, est en réalité un fichier ZIP qui contient le code compilé de l'application, ses ressources telles que les images, ainsi qu'un manifeste d'installation. C'est ce manifeste qui indique au Windows Store, mais également à l'utilisateur, les prérequis de l'application ; de plus, il sert à Windows pour décider des paramètres généraux à appliquer l'application. Il est ainsi possible de décider si l'application est capable de gérer le mode paysage/portrait ou encore si elle est autorisée à envoyer des notifications (toasts) à l'utilisateur.

Ce manifeste, un fichier XML, peut être configuré au sein du projet Visual Studio comme nous l'avons vu à plusieurs reprises dans les chapitres précédents

Manifeste et capacités

Lorsque l'on publie une application, il est important que l'utilisateur puisse lui faire confiance, que ce soit pour l'accès à ses fichiers, à ses périphériques ou aux éléments de sa vie privée qui pourraient être collectés ou divulgués. L'utilisateur est en droit de savoir ce que fait l'application. C'est justement l'un des objectifs du manifeste d'application.

Il s'agit du fichier `Package.appxmanifest` déjà rencontré au cours des chapitres précédents de ce livre. Il contient notamment une partie *Capacités* qui sera affichée au niveau du Windows Store, dans les détails de l'application. Il est nécessaire de déclarer toutes les fonctionnalités spéciales de l'application. Ne pas le faire déclenchera le rejet de l'application au moment de la soumission sur le Windows Store et tout cocher au hasard éveillera les soupçons des utilisateurs, notamment si certaines autorisations ne sont pas en phase avec les fonctionnalités principales de l'application.

Tableau 11–1 Liste des capacités disponibles

Bibliothèque de documents	Accède aux fichiers situés dans la bibliothèque de documents.
Authentification en entreprise	Utilise des identifiants pour se connecter à des ressources distantes de son entreprise.
Internet (Client)	Accède à Internet en ouvrant des pages web par exemple.
Internet (Client et serveur)	Accède à Internet (connexions sortantes et/ou entrantes).
Emplacement	Informe sur la position actuelle de l'utilisateur si celle-ci est disponible.
Microphone	Enregistre le flux audio des micros branchés à l'ordinateur.

Copyright © 2013 Eyrolles.

Tableau 11–1 Liste des capacités disponibles (suite)

Audiothèque	Ajoute, modifie ou supprime des musiques situées dans la bibliothèque audio de l'ordinateur.
Bibliothèque d'images	Ajoute, modifie ou supprime des images situées dans la bibliothèque images de l'ordinateur.
Réseau privés (Client et serveur)	Accède aux réseaux authentifiés comme les réseaux locaux. Utiles pour les jeux en réseau (LAN).
Proximité	Se connecte aux périphériques de proximité connectés à l'ordinateur (Bluetooth ou Xbox via PlayTo par exemple).
Stockage amovible	Accède à certains fichiers situés sur des périphériques amovibles (clés USB par exemple). Les fichiers accessibles sont ceux dont l'extension a été déclarée dans le manifeste de l'application en tant qu'association.
Certificats utilisateurs partagés	Dans le cas d'applications d'entreprises, utilise les certificats de l'utilisateur pour accéder à des ressources de l'entreprise.
Vidéothèque	Ajoute, modifie ou supprime des vidéos situées dans la bibliothèque vidéo de l'ordinateur.
Webcam	Permet de capturer des images et des vidéos des caméras branchées sur l'ordinateur

Il est important de connaître son application et ses fonctionnalités, car lors de la soumission sur le Windows Store, une procédure automatisée analyse son code pour le comparer aux déclarations de capacités ; la détection d'un code particulier (par exemple un accès au Web) alors que la capacité correspondante n'est pas déclarée lèvera une alerte et le rejet de l'application.

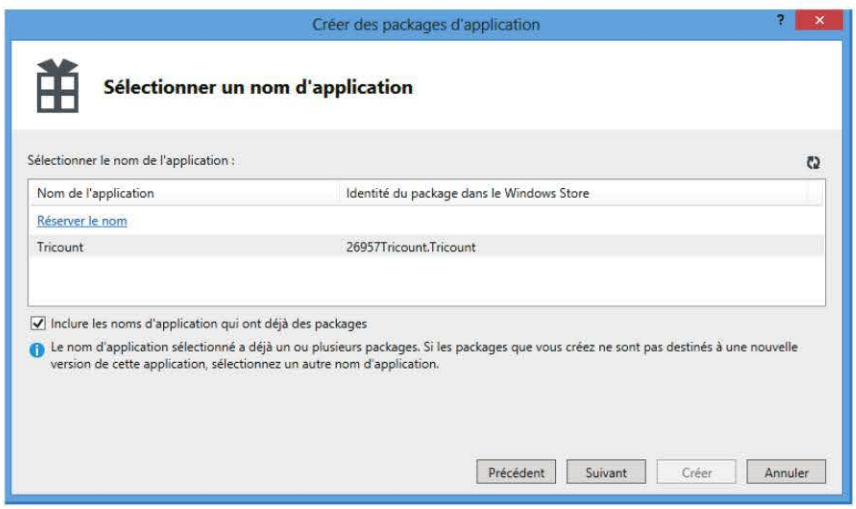


Figure 11–5
Assistant de création de package

Copyright © 2013 Eyrolles.

Lorsque les capacités sont convenablement déclarées, il est temps de créer le package. Il suffit pour cela de cliquer-droit sur le projet principal de la solution et de choisir le menu *Windows Store>Créer des packages d'application*… Un assistant se lance pour s'authentifier à l'aide du compte développeur. Il est alors possible de préciser les plates-formes ciblées (x86, x64 et ARM) et le répertoire d'export.

> **ATTENTION Le mode release**
>
> Il est important de passer son projet en mode *release* afin de créer le package correspondant. Un package compilé en mode *debug* sera refusé par le processus de certification.

Une fois le package créé, il est recommandé d'en stocker une copie comme archive. L'original va servir pour déclarer l'application auprès de Microsoft, mais avant même de démarrer le processus de validation, il est possible de simuler une partie de celui-ci à l'aide d'un outil de validation local : le kit de certification. Celui se trouve en ouvrant l'écran d'accueil (touche *Windows*) puis en saisissant « kit cert ».

Cet outil teste plusieurs critères de validation comme la performance, la gestion de la mémoire ou le cycle de vie. Il teste également la validité du manifeste, la gestion des pannes et certains points de sécurité. Une fois le test terminé, un rapport est délivré.

Bien entendu, seul un test de certification réussi permet de prétendre à une validation sur le portail en ligne. En cas d'échec, il est nécessaire d'analyser le rapport détaillé pour voir les points qui ont été mal gérés.

Windows App Certification Kit - Test Results

Nom de l'application :	**26957Tricount.Tricount**	
Version de l'application :	**1.0.1.1**	
Éditeur de l'application :	**Tricount**	
Version du système d'exploitation :	**Microsoft Windows 8 Professionnel (6.2.9200.0)**	
Heure du rapport :	**16/10/2012 21:43:52**	

Score général : RÉUSSITE

Figure 11–6
Rapport délivré par le kit de certification

Soumission de l'application

Une fois le package testé, il faut se rendre sur le portail de centre de développement (https://appdev.microsoft.com/StorePortals) pour soumettre l'application. Après s'être authentifié sur ce dernier, il est possible, via le tableau de bord, d'accéder au formulaire de soumission.

Copyright © 2013 Eyrolles.

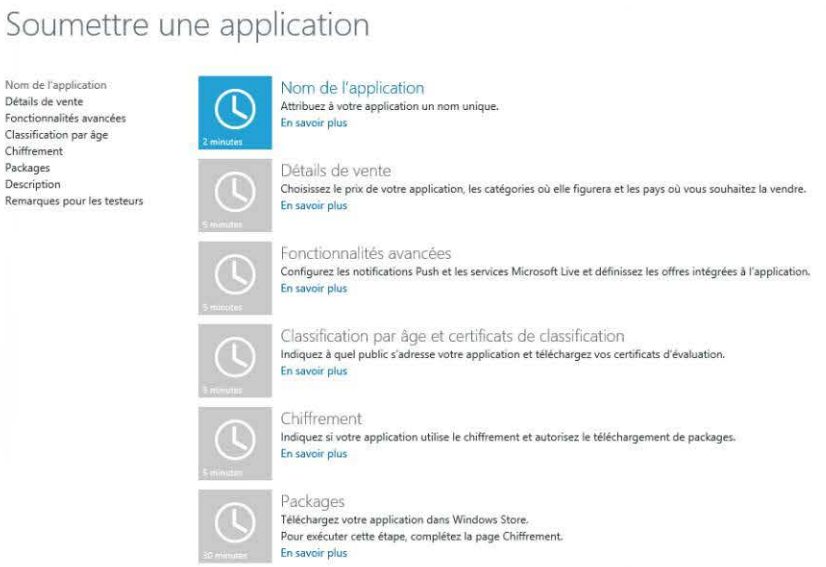


Figure 11–7
Formulaire de soumission

La première étape concerne le nom de l'application. Il doit être suffisamment ciblé pour être à la fois un signe bien distinctif de l'application, mais également un mot-clé que les utilisateurs utiliseront pour leur recherche. Ce nom, limité à 256 caractères, ne peut contenir de marque déposée faisant l'objet d'un copyright (sauf si l'on en est le propriétaire officiel).

Viennent ensuite les détails de la vente précisant les marchés concernés par l'application, le modèle financier de l'application (gratuit ou payant), son prix mais également sa période d'essai si désirée. C'est également cette rubrique qui définit la catégorie de l'application, choix déterminant du fait de l'affichage sous forme de catégories du Windows Store.

L'étape suivante concerne les fonctionnalités avancées de l'application, par exemple les modules complémentaires payables à la demande ou l'utilisation de services web comme Live Connect.

Dans l'étape *classification*, il faut préciser l'âge minimal requis pour utiliser l'application. Comme la notation des jeux vidéo, elle permet au contrôle parental de limiter l'accès à certaines applications. Ainsi, toute application faisant le moindre accès à Internet pour ouvrir un lien ou faire un transfert, doit être catégorisé comme *12+* sous peine de se voir automatiquement refuser par le processus de certification.

L'étape *chiffrement* sert, pour des questions de droit et de sécurité (principalement aux États-Unis, mais également en France), à déclarer les sécurités mises en place au sein de l'application ainsi que le type des données qui seront chiffrées.

Vient enfin l'étape du *package*. Il est simplement demandé de publier le fichier `appxupload` qui a été créé dans la section précédente.

Copyright © 2013 Eyrolles.

Pour finir, il est nécessaire de remplir le long formulaire de description. Celui-ci doit être rempli autant de fois qu'il existe de langues gérées par l'application. Il demandera un grand nombre d'informations extrêmement importantes pour le succès marketing de l'application. Cela concerne en effet le choix des mots-clés, des captures d'écrans et la description, qui tous ensemble aideront les utilisateurs à trouver l'application et leur donneront envie de la télécharger.

> **ATTENTION Captures d'écrans localisées**
>
> Il existe un formulaire de description pour chaque langue prise en charge (c'est le formulaire américain qui est affiché dans les pays pour lesquels la langue n'est pas gérée) et il est obligatoire de fournir des captures d'écrans localisées pour chaque langue. Ainsi, si l'on affiche des captures d'écrans en anglais sur le formulaire de description français, l'application est rejetée.

Il ne reste qu'à cliquer sur le bouton *Soumettre pour certification*. Un courriel vous est alors envoyé et un écran d'avancement indique les étapes de validation effectuées jusqu'à ce que l'application apparaisse sur le Windows Store.

Figure 11–8
Rapport d'avancement de la certification

Cet écran ne propose aucune action, mais il affiche l'avancement de la publication de l'application. Il faut noter que les premières étapes sont automatisées et ne prennent que quelques heures, tandis que la conformité du contenu est réalisée par du personnel Microsoft et peut prendre bien plus de temps.

Copyright © 2013 Eyrolles.

À la fin du parcours de validation, un courriel récapitulatif vous est envoyé pour indiquer si l'application a été publiée sur le Windows Store. En cas d'échec, un rapport détaillé est alors fourni.

Figure 11–9
Exemple de rapport de certification

Les principales règles à respecter

La première cause de déception d'un développeur Windows est sa certitude de pouvoir faire valider son application du premier coup. En effet, et bien que basé sur une liste de prérequis connus, le processus de certification est intransigeant au point que la moindre erreur cause le rejet de l'application. Bien entendu, il est possible de soumettre l'application plusieurs fois, mais le processus peut alors prendre plusieurs heures, voire jours à chaque essai, d'où l'intérêt de minimiser ce risque d'échec.

Pour bien se préparer, Microsoft a réalisé une liste de tous les points qui sont évalués, soit de façon automatique, soit de façon humaine, pour valider l'application. La liste complète (et régulièrement mise à jour) se trouve sur le site http://msdn.microsoft.com/fr-fr/library/windows/apps/hh694083.aspx

La bonne pratique consiste à se servir de cette liste et à l'inclure dans le processus de test mis en place pour l'application. Chaque point sans exception doit être vérifié avant même de tenter la validation.

Lors d'un refus de la part du portail de soumission, un rapport est délivré. Il identifie le ou les point(s) en échec, mais n'apporte pas forcément de solution quant au remède à appliquer. Il convient alors de se reporter à la liste des critères (dont le lien est fourni plus haut) ou bien d'utiliser la page des scénarios d'échec les plus courants associés à leur solution : http://msdn.microsoft.com/library/windows/apps/hh921583.aspx

Copyright © 2013 Eyrolles.

Augmenter la visibilité de votre application

Il est souvent reproché aux différents marchés d'applications de regorger de centaines de milliers d'applications inutiles ou peu visibles. C'est généralement l'effet opposé qui est souhaité lors de la publication d'une application, à savoir lui donner un maximum de visibilité et de crédibilité.

Outre la campagne marketing onéreuse et le bouche à oreille bien souvent trop peu fiable, c'est en utilisant le Windows Store comme levier que le développeur peut augmenter nettement le nombre de téléchargements.

Noter l'application

Pour bien distribuer son application, il est nécessaire de se mettre à la place de l'utilisateur, de ses attentes et de son comportement. Ainsi, une application esthétique mais remplie de bogues ne sera pas téléchargée ; l'inverse sera probablement vrai également. De même, une application esthétique mais dont les commentaires d'utilisateurs sont négatifs se verra non seulement rejetée par les visiteurs du Windows Store, mais sera surtout déclassée des meilleures applications par Microsoft.

Pour améliorer son classement, il convient donc d'avoir de bonnes notes et également un grand nombre de notations. Par conséquent, on incitera l'utilisateur à donner son avis en le redirigeant directement vers la page de critique à l'aide d'un lien ou d'un bouton dans l'application.

Ce n'est pas bien complexe, puisqu'il s'agit d'ouvrir ce que l'on nomme un protocole : une partie d'une URL qui va déclencher une application particulière, exactement comme les protocoles `http://` et `ftp://`. Dans ce cas précis, le protocole est celui du Windows Store (`ms-windows-store`) et il suffit d'ajouter le code suivant, qui ouvre la fiche détaillée de l'application souhaitée :

Ouverture d'un lien vers le Windows Store

```
string url = "ms-windows-store:REVIEW?PFN={NOM_PACKAGE}";
await Windows.System.Launcher.LaunchUriAsync(new Uri(url));
```

Le nom du package est celui qui se trouve dans le manifeste de l'application.

Figure 11–10

Copyright © 2013 Eyrolles.

Faire télécharger l'application depuis un site web

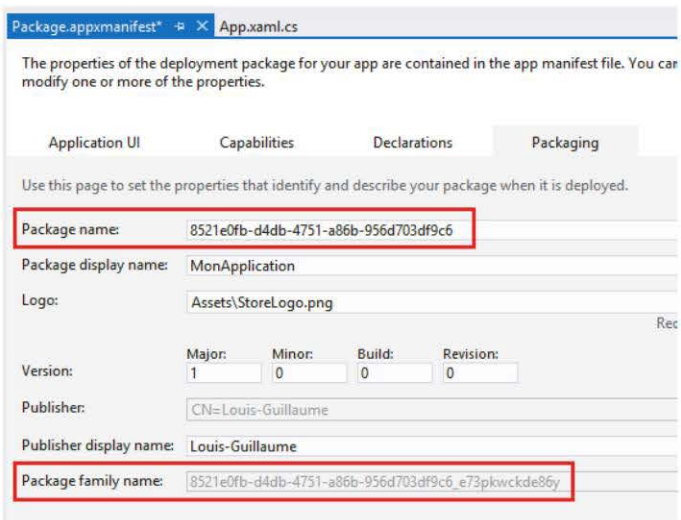
Une solution complémentaire pour augmenter le nombre d'utilisateurs de l'application est de proposer le téléchargement de l'application depuis un site web associé. Par exemple, un site web d'actualités qui possèderait une application Windows pourrait automatiquement proposer de télécharger cette dernière à tous les utilisateurs visitant le site web depuis un ordinateur sous Windows. Pour cela, il faut simplement ajouter deux lignes de code, sous forme de balise `meta`, dans la balise `<head>` des pages du site web. Elles consistent à préciser le nom de l'application et l'identifiant unique de son package.

En-tête HTML pour la reconnaissance de l'application Windows

```
<head>
...
<meta name="msApplication-ID" content="{ID_APPLICATION}" />
<meta name="msApplication-PackageFamilyName" content="{NOM_PACKAGE}" />
...
</head>
```

Il faut alors remplacer respectivement les champs `{ID_APPLICATION}` et `{NOM_PACKAGE}` par les informations *Nom du package* et *Nom de famille du package* qui sont stockées dans le projet de l'application au sein de son manifeste, comme le montre la capture suivante.

Figure 11–11
Onglet Packages du manifeste



Une fois ces lignes ajoutées, l'utilisateur qui navigue sur le site web avec une machine Windows à jour se verra notifié de la disponibilité de l'application, via une icône dans la barre d'actions, et pourra alors se rendre directement sur la page de l'application au sein du Windows Store pour en découvrir les détails et la télécharger.

Copyright © 2013 Eyrolles.

Figure 11–12
Proposition d'application au sein d'Internet Explorer

À SAVOIR Restriction

Étant donnée la façon dont elle est intégrée dans la barre d'applications, la fonctionnalité de téléchargement ne sera présente que si l'utilisateur surfe depuis une machine équipée de Windows 8 au minimum et s'il utilise Internet Explorer dans sa version Modern UI et non la version bureau classique.
Ce cas d'utilisation étant très restrictif, il existe alors une alternative consistant à créer un hyperlien HTML qui a la faculté d'être compatible avec tous les navigateurs installés sur une machine Windows. Pour cela, dans la ligne de protocole décrite pour le lien vers la notation, il suffit de remplacer le mot-clé `REVIEW` par `PDP`. Cela donne une URL de la forme `ms-windows-store:PDP?PFN={NOM_PACKAGE}`
Cliquer sur ce lien a pour effet d'ouvrir la fiche détaillée de l'application dont le nom de package a été précisé.

Gestion des applications publiées

La publication de l'application est l'étape la plus importante, mais vient ensuite le suivi de sa nouvelle vie sur le Windows Store… Cette veille est cruciale et se réalise à l'aide du portail de développement, qui fournit, pour chaque application publiée, un panneau de contrôle visualisant un certain nombre d'informations sur l'application.

Figure 11–13
Tableau de bord d'une application publiée

Copyright © 2013 Eyrolles.

Ce panneau sert notamment à surveiller le nombre de téléchargements et de ventes si l'application est payante, mais surtout à visualiser les détails dans le temps et par pays. Il est possible de visualiser les notes attribuées par les utilisateurs, critère important car il permet non seulement de jauger leur satisfaction, mais également de récolter des retours intéressants pouvant donner lieu à des améliorations futures de l'application.

Enfin, le tableau de bord donne des informations sur les erreurs et exceptions produites sur les postes des utilisateurs, ce qui en fait un outil de débogage précieux et totalement automatisé.

Tout ceci vous aidera à préparer la future version de l'application, encore plus stable et répondant au mieux aux attentes des utilisateurs !

Copyright © 2013 Eyrolles.

12

Accéder à Internet

Windows 8 peut être considéré comme un système d'exploitation mobile de par son orientation vers l'utilisation des services en ligne : ouverture de sessions à l'aide de comptes Microsoft (auparavant appelés Windows Live), sauvegarde des paramètres personnels sur SkyDrive... Les API proposées aux développeurs reflètent cette orientation.

Dans ce chapitre, nous répondrons aux questions suivantes :

- Comment sauvegarder les données d'une application sur Internet ?
- Comment accéder à des services Internet personnalisés ?
- Comment accéder à des services Internet standards comme SkyDrive ?
- Comment exploiter le kit de développement Azure pour Windows 8 qui permet d'accélérer le développement d'applications connectées ?

Données applicatives

Presque toutes les applications ont besoin d'enregistrer des données dans des fichiers. Bien entendu, ces données sont de natures différentes, par exemple :

- une application de traitement d'images doit bien évidemment permettre à l'utilisateur de sauvegarder les images retouchées ;
- un navigateur télécharge ses pages depuis Internet, mais constitue un cache local afin d'accélérer les requêtes ultérieures ;

Copyright © 2013 Eyrolles.

- quasi toutes les applications autorisent l'utilisateur à paramétrer leur fonctionnement et enregistrent alors ces paramétrages.

Ces divers types de données ont évidemment des contraintes différentes quant à la sécurité, l'itinérance ou la durée de vie.

Ainsi, des images sauvegardées deviennent indépendantes de l'application qui les a créées : si celle-ci est désinstallée, les images ne doivent pas être supprimées. Selon les cas, l'utilisateur souhaitera enregistrer ce type de données localement ou dans un stockage en ligne afin d'y accéder depuis un autre ordinateur, voire depuis son téléphone.

Le cache d'un navigateur n'est qu'un moyen d'accélérer le fonctionnement de celui-ci. Si le cache est perdu, la seule conséquence sera que les pages auparavant enregistrées dans celui-ci devront être téléchargées à nouveau, mais le fonctionnement de l'application ne sera pas remis en question. Par ailleurs, sauvegarder ce cache en ligne n'a pas de sens car le téléchargement des pages ainsi sauvegardées ne serait pas nécessairement plus rapide que leur téléchargement depuis leur site d'origine.

La configuration d'une application par un utilisateur a des besoins différents en termes d'itinérance et de durée de vie : si elle n'a pas de raison de survivre à une désinstallation du programme, il est en revanche intéressant pour l'utilisateur de la retrouver sur tous les ordinateurs sur lesquels il utilise la même application.

Le premier de ces exemples concerne la sauvegarde de fichiers qui ne sont pas nécessairement spécifiques à l'application les manipulant : par exemple, plusieurs applications proposent la modification ou la visualisation d'images. Être capable d'accéder à des espaces de stockage en ligne comme SkyDrive est en revanche définitivement un atout pour une application et sera donc décrit plus loin dans ce chapitre. Les deux exemples suivants concernent quant à eux des données fortement liées à l'application elle-même.

En pratique, Windows offre à chaque application des espaces de stockage correspondant à différents scénarios et, en particulier, des mécanismes permettant à une application de retrouver certaines données sur tout ordinateur sur lequel l'utilisateur a installé l'application. Cela met en avant le lien fort entre une application et l'utilisateur qui l'a installée, à mettre en perspective avec l'approche plus traditionnelle liant une application à l'ordinateur sur lequel elle a été installée.

Données itinérantes d'une application

Des paramètres simples peuvent être stockés sur Internet à l'aide de la propriété `ApplicationData.Current.RoamingSettings`. Cette dernière retourne un conteneur de données de type `ApplicationDataContainer` qui expose :

- un dictionnaire de paires clé/valeur dont les clés sont des chaînes de caractères et les valeurs des types WinRT simples (chaînes, nombres, dates...) ;

Copyright © 2013 Eyrolles.

- des méthodes pour créer et lire des sous-conteneurs qui sont aussi du type `ApplicationDataContainer` et organisent ainsi hiérarchiquement des valeurs simples à stocker dans le conteneur `RoamingSetting` (cette hiérarchie peut contenir jusqu'à trente-deux niveaux de profondeur).

L'écriture d'une valeur simple à la racine du conteneur s'écrit de la façon suivante :

```
ApplicationData.Current.RoamingSettings.Values["userName"] = userName;
```

La lecture se fait de manière symétrique :

```
string userName = ApplicationData.Current.RoamingSettings.Values["userName"] as
string;
```

Comme son nom l'indique, le conteneur `RoamingSettings` est itinérant et son contenu peut être lu à partir de n'importe quel ordinateur sur lequel l'utilisateur a installé l'application. Néanmoins, il faut noter que cette synchronisation des paramètres est asynchrone, comme beaucoup d'autres mécanismes de Windows. Il ne faut donc pas s'attendre à faire communiquer ainsi deux instances de la même application s'exécutant simultanément. L'objectif est uniquement de sauvegarder la configuration d'une application sur Internet et, comme souvent, il convient d'utiliser les API décrites ici dans leur cadre d'utilisation standard.

De la même manière, ce mécanisme est à réserver à de petites quantités de données (une centaine de Ko). Pour des paramètres consommant plus d'espace, il est nécessaire d'adopter d'autres stratégies mieux adaptées (un stockage partagé sur Internet avec SkyDrive par exemple), en créant des fichiers propres à l'application mais qui seront malgré tout disponibles sur tous les ordinateurs sur lesquels l'utilisateur exécute celle-ci.

EXEMPLE

Un scénario d'utilisation de ce mécanisme serait par exemple un jeu d'échecs avec lequel l'utilisateur poursuivrait une même partie sur les différents périphériques sur lesquels il se connecte. Pour le développeur d'applications, ces fonctions de Windows 8 sont très utiles, car elles autorisent à implémenter de tels scénarios sans avoir à installer (et financer) un composant serveur de l'application.

Les fichiers itinérants sont situés dans une arborescence de répertoires dont la racine s'obtient d'une manière assez similaire à celle présentée précédemment pour les paramètres :

- d'abord, il faut obtenir une référence sur le répertoire racine à l'aide de la propriété `RoamingFolder` de la classe `ApplicationData` :

```
StorageFolder storageRoot = ApplicationData.Current.RoamingFolder;
```

Copyright © 2013 Eyrolles.

- ensuite, l'objet `StorageFolder` obtenu se manipule exactement de la même façon qu'un répertoire local simple : la synchronisation est effectuée en arrière-plan par Windows et l'accès à Internet est donc invisible pour le développeur de l'application. L'écriture d'une chaîne de caractères dans un tel fichier peut s'écrire de la façon suivante :

```
StorageFolder storageRoot = ApplicationData.Current.RoamingFolder;
StorageFile gameFile = await storageRoot.CreateFileAsync("CurrentGame.pgn");
using (Stream str = await gameFile.OpenStreamForWriteAsync())
using (StreamWriter writer = new StreamWriter(str))
{
    writer.WriteLine(gameString);
}
```

Deux points importants sont à noter lors de l'utilisation de ce stockage en ligne « gratuit » :

- l'espace disque ainsi synchronisé est de taille limitée : celle-ci s'obtient en appelant l'API `ApplicationData.Current.RoamingStorageQuota` qui la retourne exprimée en kilo-octets ;
- cet espace est disponible sur Internet tant que l'application est exécutée régulièrement : si ce n'est pas le cas, l'espace est potentiellement libéré après 30 jours sans lancement.

On en conclut que les véritables paramètres fonctionnels d'une application ne doivent absolument pas être stockés dans cet espace, car l'utilisateur peut légitimement rester plus d'un mois sans se servir d'une application et ne doit pas être pénalisé par la perte de tout le paramétrage. Cet espace limité en taille et dans le temps trouve en revanche toute son utilité dans d'autres scénarios, où l'information peut se permettre d'être détruite sans remettre en cause ni la stabilité de l'application ni les données de l'utilisateur.

Sécurité

Pour accéder à des services Internet, une application du Windows Store doit déclarer les capacités correspondantes dans son manifeste, sinon les requêtes HTTP échouent avec différents types d'exceptions. Il faut noter que ces exceptions apparaissent de manière asynchrone, comme si le service appelé n'était pas accessible, et non de manière synchrone lors de l'émission de la requête. La cause de l'échec n'est donc pas immédiatement visible :

- l'exception reçue est une `HttpRequestException` contenant le message « *An error occurred while sending the request.* » ;
- les exceptions incorporées (propriété `InnerException`) dans celle-ci varient selon le type de requête :

Copyright © 2013 Eyrolles.

- `GetStringAsync("http://www.microsoft.com")` déclenche une `WebException` de message « *The remote name could not be resolved : 'www.microsoft.com'* » ;
- `GetStringAsync(("http://64.4.11.42")` déclenche une `WebException` de message « *Unable to connect to the remote server* » contenant elle-même une `SocketException` ayant enfin un message explicite « *An attempt was made to access a socket in a way forbidden by its access permissions 64.4.11.42:80* ».

Deux capacités servent à déclarer l'utilisation d'Internet par une application du Windows Store :

Internet (Client)	L'application effectue des connexions sortantes.
Internet (Client & Server)	L'application effectue des connexions sortantes, mais accepte aussi des connexions entrantes. C'est nécessaire pour les applications « peer to peer » comme les jeux en réseau ou les applications de communication.

Dans un certain nombre de scénarios, il est tentant de connecter une application du Windows Store à une application Bureau classique ou à un service Windows exécuté localement : les possibilités que cette approche procure sont très nombreuses mais constituent une menace en termes de sécurité, car elles ouvrent l'accès à des ressources par ailleurs contrôlées.

Pour cette raison, une application Windows Store ne peut par défaut pas accéder à l'ordinateur local : ceci n'est pas configurable via le manifeste de l'application et la déclaration de capacités. Il faut passer par un outil en ligne de commande appelé `CheckNetIsolation`, ce qui garantit qu'une telle autorisation ne sera pas donnée de manière involontaire par l'utilisateur.

Il faut noter que pour faciliter le travail des développeurs, ces restrictions n'apparaissent pas quand l'application du Windows Store est lancée depuis Visual Studio ; cela conduit parfois à quelques surprises au moment du déploiement, puisque ces appels échoueront si `CheckNetIsolation` n'a pas été correctement appelé au préalable.

Accès à des services Internet personnalisés

Fondamentalement, un service Internet est une application qui reste à l'écoute sur un port réseau IP et répond aux connexions et requêtes émises sur ce port. Plusieurs protocoles ont été élaborés au-dessus de cette définition un peu brute, pour faciliter l'interconnexion des applications clientes et serveurs. Sans aller jusqu'à la description du modèle en couches OSI (Open Systems Interconnection) de l'ISO, voici les principaux protocoles utilisés de nos jours sur Internet et dont l'utilisation est décrite dans la suite de ce chapitre :

- **TCP/IP** : c'est le protocole de base d'Internet. Il propose la notion de connexion fiable (tous les octets écrits à une extrémité de la connexion sont reçus, et ce dans

Copyright © 2013 Eyrolles.

le bon ordre, de l'autre côté de celle-ci) et est accessible à l'aide des sockets. Les autres protocoles décrits par la suite s'appuient sur TCP/IP ;

- **HTTP :** c'est le protocole du Web, qui sert généralement à faire transiter les pages HTML affichées dans les navigateurs. Son approche Requête/Réponse est particulièrement adaptée au développement de services de grande capacité, car elle s'adapte très bien aux fermes de serveurs. Des classes dans le framework .Net facilitent l'accès à de tels services pour les applications XAML. Les applications JavaScript utilisent de manière classique les objets XML HTTP comme le font la plupart des applications web ;
- **REST/JSON, SOAP :** il s'agit de protocoles basés sur HTTP et qui décrivent en outre le contenu des messages échangés. Ils permettent le développement d'applications distribuées en offrant au développeur une vision des requêtes échangées assez proche de simples appels de méthodes locales.

Utilisation de sockets

Plusieurs types de sockets sont accessibles aux applications du Windows Store : il y a d'une part les sockets classiques orientées flux de données (utilisant généralement TCP/IP) ou datagrammes (utilisant UDP/IP), mais aussi les websockets apparues plus récemment et dont le rôle initial est de simplifier la communication entre une page et un serveur web.

Toutes les classes correspondant à ces divers types de sockets appartiennent à l'espace de noms `Windows.Networking.Sockets` et sont ainsi accessibles à la fois aux applications XAML/.NET, XAML/C++ et HTML 5/JavaScript.

Voici comment connecter une application cliente à un serveur Internet à l'aide de sockets TCP/IP. On suppose que le serveur, parfaitement classique, existe et est à l'écoute des messages. Le schéma habituel n'est pas modifié par le fait que l'application cliente vienne du Windows Store : le serveur doit être préalablement à l'écoute sur un port TCP/IP donné et connu à l'avance de l'application cliente (les services standards sont pratiquement toujours sur les mêmes ports, typiquement 80 pour HTTP, 25 pour SMTP...).

L'application cliente crée une socket de type flux sans l'associer à un numéro de port particulier (un port sera attribué automatiquement, mais il peut être ignoré), puis elle le connecte au serveur en précisant, d'une part, le nom ou l'adresse IP du serveur et, d'autre part, le numéro de port ou le nom du service (s'il s'agit d'un service connu).

Comme toujours avec une application du Windows Store et une opération potentiellement longue, la connexion au serveur est asynchrone :

Copyright © 2013 Eyrolles.

```
async private Task<StreamSocket> Connect(string serverName, string portNumber)
{
    StreamSocket ret = new StreamSocket();
    await ret.ConnectAsync(new HostName(serverName), portNumber);
    return ret;
}
```

Si le serveur n'est pas disponible, une exception est levée dont la nature dépend de l'erreur rencontrée :

- « *No connection could be made because the target machine actively refused it* » si la machine distante est accessible mais aucun processus n'est à l'écoute sur le port demandé ;
- « *No such host is known* » si la résolution DNS échoue…

À NOTER

Si le nom de serveur fourni correspond à une adresse du réseau local, une exception *Access Denied* est levée si l'application n'a pas déclaré la capacité *Home or Work Networking*.

Une fois la socket connectée, l'application peut commencer à envoyer des données ou à en lire. Pour envoyer des données, il faut utiliser la propriété `OutputStream` du socket. Il s'agit d'une interface de type `IOutputStream` utilisable de trois manières :

- en passant directement un objet implémentant l'interface `IBuffer` à sa méthode `WriteAsync` ;
- en créant un objet de type flux (de classe `Stream`) via la méthode d'extension `WindowsRuntimeStreamExtensions.AsStreamForWrite` ;
- en instanciant un objet de type `DataWriter` auquel est passé le `OutputStream` du socket en paramètre. Cette classe facilite l'envoi de données structurées sur le flux :
  - `WriteBoolean`, `WriteInt32`, `WriteDouble`, `WriteString`... écrivent des types de données simples ;
  - `StoreAsync` force le vidage du tampon et l'émission des donnés écrites.

Attention, les données écrites via `DataWriter` sont stockées de manière brute sans séparateur ou métadonnée associée. Il faut donc gérer manuellement – par programmation – les données de longueur variable. Voici un extrait de code exemple puis le contenu effectif des données écrites sur le flux :

```
class Vin
{
    public string Nom { get; set; }
    public string Appellation { get; set; }
```

Copyright © 2013 Eyrolles.

```
    public string Region { get; set; }
    public int Millesime { get; set; }
}

async void SendWine(DataWriter writer)
{
    Vin v = new Vin { Nom = "Château Sociando-Mallet",
                      Appellation = "Haut-Médoc",
                      Region = "Bordeaux",
                      Millesime = 1998 };
    writer.WriteInt32(v.Millesime);
    writer.WriteString(v.Nom);
    writer.WriteString(v.Appellation);
    writer.WriteString(v.Region);
    await _writer.StoreAsync();
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	7	206	67	104	195	162	116	101	97	117	32	83	111
				C	h	Ã	¢	t	e	a	u		S	o
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
99	105	97	110	100	111	45	77	95	108	108	101	116	72	97
c	i	a	n	d	o	-	M	a	l	l	e	t	H	a
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
117	116	45	77	195	169	100	111	99	66	111	99	66	101	97
u	t	-	M	Ã	©	d	o	c	B	o	r	d	e	a
45	46													
117	120													
u	x													

L'analyse de ce tableau montre effectivement que :

- les entiers sont écrits en Big Endian (cette représentation est donc incompatible sur plate-forme Intel avec un décodage via `BitConverter.ToInt32`). Il s'agit ici d'une spécificité de la classe `DataWriter` (et sa réciproque `DataReader`), qui encode par défaut en Big Endian car les autres plates-formes utilisent généralement ce format. On modifie ce comportement via la propriété `ByteOrder` ;
- les chaînes de caractères n'ont pas de séparateur et sont enregistrées en UTF-8 (ce qui optimise la taille occupée et représente correctement toutes les chaînes).

Le récepteur pourra lire le millésime (car l'API de lecture saura qu'un entier occupe 4 octets) mais pas les chaînes de caractères. Pour résoudre cette difficulté, il faut précéder généralement l'écriture des chaînes par celle de leur longueur en UTF-8, qui

Copyright © 2013 Eyrolles.

sera obtenue via `DataWriter.MeasureString()`. Le code d'écriture de la classe `Vin` décrite plus haut devient donc :

```
async void SendWine(DataWriter writer)
{
    Vin v = new Vin { Nom = "Château Sociando-Mallet",
                      Appellation = "Haut-Médoc",
                      Region = "Bordeaux",
                      Millesime = 1998 };
    writer.WriteInt32(v.Millesime);
    writer.WriteUInt32(_writer.MeasureString(v.Nom));
    writer.WriteString(v.Nom);
    writer.WriteUInt32(_writer.MeasureString(v.Appellation));
    writer.WriteString(v.Appellation);
    writer.WriteUInt32(_writer.MeasureString(v.Region));
    writer.WriteString(v.Region);
    await _writer.StoreAsync();
}
```

La lecture d'une socket s'effectue d'une manière symétrique à l'écriture, en utilisant la propriété `InputStream` et une instance de la classe `DataReader`. Contrairement à l'écriture, qui est déterministe au sens où l'application sait quand elle doit envoyer des données (même si cet envoi est lui-même asynchrone), la lecture ne l'est pas nécessairement. Il faut s'appuyer plus explicitement sur le caractère asynchrone des opérations sur les sockets en appelant `DataReader.LoadAsync` pour déclencher la lecture et en enregistrant une fonction de rappel (callback) qui sera exécutée lors de la réception effective de données :

```
_reader = new DataReader(_socket.InputStream);
_reader.InputStreamOptions = InputStreamOptions.Partial;
DataReaderLoadOperation op = _reader.LoadAsync(1024);
op.Completed = ReadCompleted;
…
async private void ReadCompleted(IAsyncOperation<uint> asyncInfo, AsyncStatus
asyncStatus)
{
    if (asyncInfo.Status == AsyncStatus.Completed)
    {
        Vin v = new Vin();
        v.Millesime = _reader.ReadInt32();
        uint len = _reader.ReadUInt32();
        v.Nom = _reader.ReadString(len);
        len = _reader.ReadUInt32();
        v.Appellation = _reader.ReadString(len);
        len = _reader.ReadUInt32();
        v.Region = _reader.ReadString(len);
```

Copyright © 2013 Eyrolles.

```
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
() =>
        {
            DisplayWine(v);
        });
        DataReaderLoadOperation op = _reader.LoadAsync(1024);
        op.Completed = ReadCompleted;
    }
    else
    // Traitement d'erreur
}
```

Il faut noter au passage que la méthode `DataReader.ReadString()` attend une longueur pour les raisons évoquées précédemment.

En conclusion, on remarquera que le programmeur doit s'attacher à de nombreux détails techniques avec les sockets. Il est donc préférable de privilégier des protocoles de plus haut niveau et les sockets seront réservées à des utilisations spécifiques :

- lorsque l'application se connecte à un serveur existant et implémentant un protocole autre que HTTP et bien documenté (serveur de streaming vidéo par exemple) ;
- lorsque la latence et le volume des échanges réseau doivent être particulièrement optimisés (jeux, communication audio ou vidéo...).

Service HTTP « brut »

Fondamentalement, un service Internet est une URL HTTP qui propose d'accéder à des données ou des traitements indépendamment de la mise en forme proposée par une page HTML classique.

En effet, même si un site Internet comme un site d'informations météorologiques propose d'une certaine manière un service, celui-ci est enrobé dans une interface adaptée à une lecture par un utilisateur humain et non par une machine. Un service Internet est en revanche conçu pour des clients de type ordinateur et l'enrobage lié à la présentation est alors supprimé : il peut subsister des informations supplémentaires qui accompagnent l'information brute, mais il s'agit alors de données techniques, par exemple pour gérer l'autorisation d'accès au service.

L'exploitation d'un tel service consiste donc à appeler une URL HTTP en transmettant éventuellement des données supplémentaires dans le corps de la requête (les URL sont limitées en taille) et à lire le résultat de l'exécution de cette requête dans la réponse HTTP.

Une application du Windows Store effectue de telles requêtes HTTP à l'aide de la classe `HttpClient` et des classes associées `HttpRequestMessage` et `HttpResponseMessage`. Les méthodes de la classe `HttpClient` se décomposent en trois groupes principaux :

Copyright © 2013 Eyrolles.

- les différentes surcharges de `SendAsync` contrôlent finement la requête HTTP en acceptant un `HttpRequestMessage` en entrée et en retournant un `HttpResponseMessage` ;
- les surcharges de `GetAsync`, `PostAsync`, `PutAsync` et `DeleteAsync` implémentent directement le verbe du protocole HTTP que leur nom indique ;
- enfin `GetStringAsync`, `GetByteArrayAsync` et `GetStreamAsync` permettent l'exécution de requêtes HTTP de la manière la plus simple, c'est-à-dire en un seul appel de méthode.

ATTENTION Asynchronisme obligatoire

Toutes les méthodes évoquées se terminent par `Async`, ce qui est attendu compte tenu du caractère asynchrone de l'API WinRT et du temps d'exécution, à la fois potentiellement long et non déterministe, des requêtes Internet.

La forme la plus simple de récupération de données situées sur un serveur Internet est donc basée sur `GetStringAsync` :

```
private async void Telecharger(string url)
{
    try
    {
        HttpClient client = new HttpClient();
        string content = await client.GetStringAsync(url);
        // Faire quelque chose avec 'content'
    }
    catch (Exception ex)
    {
        HandleException(ex);
    }
}
```

Pour une requête `POST`, qui demande un corps, la méthode `PostAsync` accepte en plus de l'URL un paramètre de type `HttpContent`. Il s'agit d'une classe abstraite dont certaines classes dérivées sont fournies dans l'API WinRT :

- `StreamContent` : qui renseigne le corps de la requête à l'aide d'un objet `Stream` ;
- `MultiPartContent` : un contenu à parties multiples utilisé typiquement pour le téléchargement de plusieurs fichiers ;
- `ByteArrayContent` : contenu précisé sous forme de tableau d'octets. Cette classe a elle-même deux sous-classes :
  - `FormUrlEncodedContent` : contenu similaire à un formulaire HTML posté par l'utilisateur (typiquement des paires clé-valeur) ;
  - `StringContent` : contenu renseigné sous la forme d'une chaîne de caractères.

Copyright © 2013 Eyrolles.

L'exemple de code qui suit télécharge via une requête de type `POST` un contenu XML fourni sous forme de chaîne de caractères et traite ensuite la réponse du serveur :

```
private async void TelechargerAvecPost(string url)
{
    try
    {
        HttpClient client = new HttpClient();
        StringContent body = new StringContent(
@"<wine name=""Amiral de Beychevelle"">
    <appellation type=""AOC"" name=""Saint Julien""/>
  </wine>"
        );
        HttpResponseMessage response =
            await client.PostAsync("http://localhost:57845/SaveWine.ashx",
body);
        if (response.StatusCode == HttpStatusCode.OK)
        {
            // Traiter le cas où la requête s'est bien passée
        }
        else
        {
            // Traiter les erreurs
        }
    }
    catch (Exception ex)
    {
        HandleException(ex);
    }
}
```

Les méthodes de la classe `HttpClient` possèdent pour la plupart une surcharge acceptant un paramètre de type `CancellationToken`. Ce dernier permet une annulation explicite de la requête, par exemple si l'utilisateur de l'application lance une nouvelle action rendant caduque l'action précédente (dans le cas d'un navigateur web, il s'agit typiquement d'une navigation vers une autre page).

L'exemple précédent devient alors :

```
…
CancellationTokenSource tokSource = new CancellationTokenSource();
var response = await client.PostAsync("http://localhost:57845/SaveWine.ashx",
                                      body,
                                       tokSource.Token);
…
```

Copyright © 2013 Eyrolles.

Il est possible d'utiliser le `CancellationTokenSource` pour imposer un délai maximum d'attente du résultat de la requête HTTP. Cependant, il y a plus simple car la classe `HttpClient` expose justement une propriété appelée `TimeOut` : si le délai est dépassé, c'est une exception de type `TaskCanceledException` qui est levée (ce serait aussi le cas si le délai était précisé lors de l'appel du constructeur du `CancellationTokenSource`) :

```
try
{
    HttpClient client = new HttpClient();
    client.Timeout = TimeSpan.FromMilliseconds(5000.0); // 5 secondes
    StringContent body = new StringContent(
@"<wine name=""Amiral de Beychevelle"">
    <appellation type=""AOC"" name=""Saint Julien""/>
  </wine>");
    var response = await client.PostAsync(url, body);
    …
}
catch (TaskCanceledException tcEx)
{
    ResultTextBlock.Text = "Délai de téléchargement dépassé";
}
catch (Exception ex)
{
    HandleException(ex);
}
```

Pour préciser des options complémentaires, par exemple l'utilisation d'un proxy, il faut passer un objet de type `HttpClientHandler` au constructeur de l'objet `HttpClient`. La classe `HttpClientHandler` sert à configurer le traitement des requêtes HTTP transmises via l'instance correspondante de l'objet `HttpClient` :

- configuration des redirections (automatiques ou non, combien...) ;
- configuration des cookies ;
- configuration du proxy ;
- fourniture d'identité pour des requêtes authentifiées ;
- configuration de l'utilisation des certificats clients.

Service REST et sérialisation JSON

De plus en plus de services Internet sont exposés via une interface de type REST, consistant à échanger des objets sérialisés au format JSON (*JavaScript Object Notation*) via des requêtes HTTP simples.

Si le service Internet auquel se connecte une application est basé sur WCF, voire de type WCF Data Service, la communication entre l'application et ce service est très

Copyright © 2013 Eyrolles.

simple à établir : il suffit d'ajouter la référence de service sous Visual Studio.NET et celui-ci saura reconnaître les métadonnées publiées par le service.

S'il s'agit d'un service Internet développé dans une autre technologie (et s'exécutant éventuellement sur une autre plate-forme que Windows), il n'est pas certain qu'il expose des métadonnées le décrivant sous un format compréhensible par Visual Studio. Cela ne signifie pas pour autant que le développeur doive prendre en charge toutes les opérations de bas niveau, et en particulier le codage des entités en notation JavaScript ou leur lecture à partir d'une telle notation.

En effet, la classe utilisée par le runtime .NET pour écrire ou lire des objets au format JSON est accessible indépendamment de la création de proxies par Visual Studio : combiné à la classe `HttpClient` présentée précédemment, le `DataContractJsonSerializer` extrait ou écrit des instances d'objets dans le corps de requêtes HTTP. Il suffit de lui fournir une description des classes correspondantes, annotées comme des DataContracts WCF car, même si le service effectivement appelé n'est pas implémenté avec cette technologie, le format de sérialisation des objets reste standard.

Par exemple, le téléchargement et le décodage d'un tableau d'objets disponibles via une requête GET peut se faire de la manière suivante :

```
[DataContract]
public class Appellation
{
    [DataMember]
    public int Id { get; set; }

    [DataMember]
    public string Nom { get; set; }

    [DataMember]
    public Region Region { get; set; }

…
}
…
private async Task<Appellation[]> GetAppellations(string serviceUrl)
{
    HttpClient client = new HttpClient();
    DataContractJsonSerializer ser =
              new DataContractJsonSerializer(typeof(LibEntites.Appellation[]));
    using (var stream = await client.GetStreamAsync(serviceUrl))
    {
        return ser.ReadObject(stream) as Appellation[];
    }
}
```

Copyright © 2013 Eyrolles.

Service WCF

Comme il a été dit précédemment, les services WCF sont pour la plupart utilisables de manière classique en ajoutant la référence du service au projet de l'application cliente. Visual Studio .NET prend en charge la création des proxies en n'exposant bien évidemment que des méthodes asynchrones pour invoquer le service distant.

Figure 12–1
La création des opérations asynchrones est obligatoire.

L'appel d'un tel service est assez naturel (il faut noter au passage que le proxy retourne une `ObservableCollection` qui est la collection la plus naturelle à lier aux données en XAML) :

```
ServiceReference1.Service1Client proxy = new
ServiceReference1.Service1Client();
ObservableCollection<Appellation> ret = await proxy.GetAppellationsAsync();
```

Accéder à des services standards - SkyDrive

Aujourd'hui, les services Internet standards sont accessibles via des requêtes HTTP qui peuvent être constituées manuellement par le développeur ou automatiquement grâce à des bibliothèques à incorporer dans les projets.

C'est typiquement le cas des services Windows Live, auxquels les développeurs accèdent en s'appuyant sur un kit de développement téléchargeable sur le site de Microsoft. Ces services présentent un avantage supplémentaire pour le développeur et l'utilisateur : les comptes Microsoft (appelés auparavant comptes Windows Live) étant fortement intégrés à Windows, il est possible d'y accéder sans demander à l'utilisateur de saisir à nouveau son mot de passe :

- l'utilisateur est rassuré en termes de sécurité, puisqu'il ne confie pas son mot de passe à une application tierce ;
- le développeur est déchargé de la responsabilité de sécuriser des mots de passe importants et peut aussi toucher un public plus large ;
- néanmoins, cette authentification sans saisie de mot de passe (en anglais, on parle de SSO pour *Single Sign On*) ne se fait bien évidemment pas sans le consentement

Copyright © 2013 Eyrolles.

de l'utilisateur, car cela serait contraire à toutes les règles de sécurité et à la philosophie de Windows 8.

Décrire l'ensemble des API du kit de développement Windows Live déborderait du cadre de cet ouvrage. La suite de cette section se concentre donc sur l'intégration avec Windows 8 en décrivant essentiellement les étapes pour mettre en place cette authentification intégrée.

La première étape consiste à télécharger et installer le kit de développement accessible via l'adresse http://msdn.microsoft.com/en-US/live et à le référencer dans le projet Visual Studio. Il se trouve alors dans la rubrique *Windows/Extensions* de la boîte de dialogue d'ajout de références.

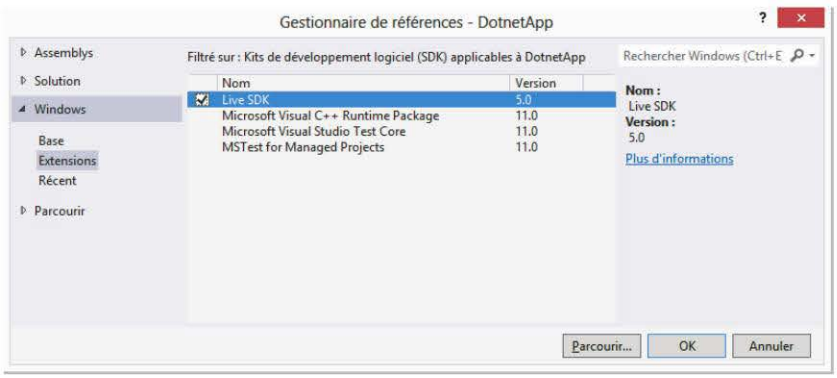


Figure 12–2
Ajout d'une référence sur un kit de développement

Ensuite, il faut déclarer l'application auprès de Microsoft, de la même manière qu'un service émettant des notifications.

- Aller sur le site Live Connect Developer Center (http://go.microsoft.com/fwlink/p/?LinkId=227628).
- Renseigner les champs *Package Display Name* et *Publisher*, puis cliquer sur le bouton *I accept* (il suffit de recopier les éléments affichés par Visual Studio).
- Un nom unique de package est alors créé, qu'il faut copier dans le champ correspondant sous Visual Studio.

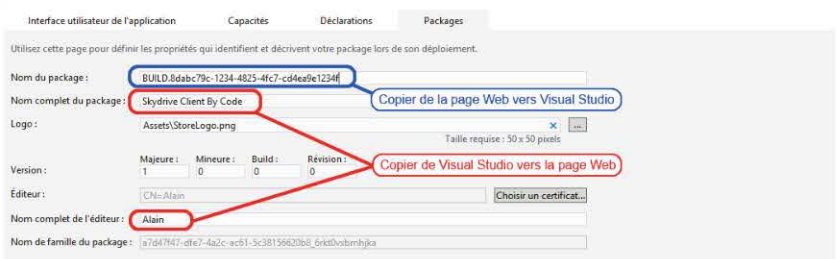


Figure 12–3
Configuration Live Connect

Copyright © 2013 Eyrolles.

Une fois le projet ainsi configuré, il est possible de lancer la connexion à Windows Live. Ceci se fait par code ou en insérant un contrôle de type `SignInButton` dans la page XAML. Au moment de la connexion, l'application Windows Store doit déclarer les éléments du compte Microsoft de l'utilisateur auxquels elle désire accéder, ce qui est assez similaire à la déclaration des capacités de Windows déclarées dans le fichier `.appxmanifest`, même si cette déclaration se fait ici de manière explicite et non à l'aide d'un fichier externe.

La documentation sur le site MSDN donne la liste exhaustive des autorisations à demander (appelées « étendues »). Pour authentifier l'utilisateur et accéder à ses documents stockés sur SkyDrive, les autorisations à demander sont :

- `wl.signin` : authentification automatique de l'utilisateur ;
- `wl.basic` : accès aux propriétés de base de l'utilisateur comme son nom ;
- `wl.SkyDrive` : accès en lecture à son stockage SkyDrive ;
- `wl.SkyDrive_update` : accès en lecture et en écriture à son stockage SkyDrive.

Le code d'authentification pour un accès en lecture au stockage SkyDrive est alors :

```
async private Task DoConnectLive()
{
    var client = new LiveAuthClient();
    LiveLoginResult loginResult =
        await client.LoginAsync(new List<string>() { "wl.signin",
                                                      "wl.basic",
                                                      "wl.SkyDrive" });
    if (loginResult.Status == LiveConnectSessionStatus.Connected)
    {
        _session = client.Session;
        _client = new LiveConnectClient(_session);
…
```

Lorsque ce code est exécuté, Windows présente l'affichage suivant pour demander à l'utilisateur s'il accorde à l'application les droits demandés.

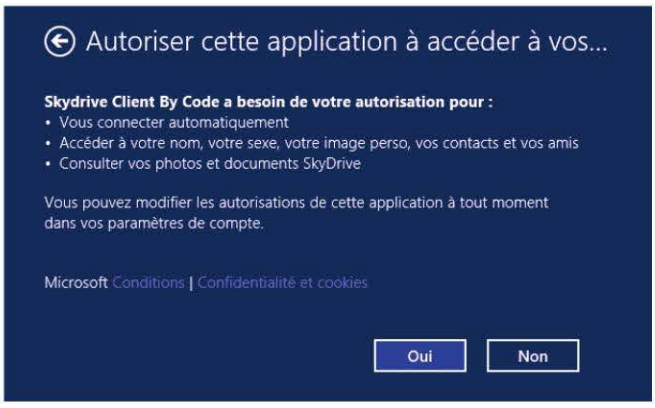


Figure 12–4
Demande de consentement de l'utilisateur pour l'accès à Windows Live

Copyright © 2013 Eyrolles.

Une fois l'autorisation accordée, l'application peut accéder aux informations de l'utilisateur (grâce à `wl.basic`) :

```
LiveOperationResult result = await _client.GetAsync("me");
dynamic meResult = result.Result;
string fullName = (meResult.first_name ?? "") + " " +
                  (meResult.last_name);
```

La liste des fichiers se récupère de manière similaire :

```
result = await _client.GetAsync("me/SkyDrive/files");
dynamic filesResult = result.Result;
var items = new ObservableCollection<SkyDriveItem>();
var vdata = filesResult.data;
dynamic ddata = filesResult.data;
foreach (dynamic item in filesResult.data)
{
  // récupération des propriétés d'un fichier ou répertoire
…
}
```

Il faut remarquer l'utilisation du mot-clé C# `dynamic` pour accéder au résultat des requêtes : il permet la récupération des informations de l'utilisateur par l'invocation dynamique des propriétés `first_name` et `last_name` et celles des fichiers par celle de propriétés comme `name` et `id` (par exemple pour parcourir les sous-répertoires) ou `type`.

Développer plus vite grâce aux services mobiles Windows Azure

L'accès à SkyDrive est idéal pour des applications orientées fichiers et dont le contenu est propre à chaque utilisateur. En revanche, pour des applications s'appuyant sur des données structurées, éventuellement partagées entre plusieurs utilisateurs, une base de données est nécessaire. Pour faciliter le développement de telles applications, Microsoft propose depuis l'été 2012 le kit de développement services mobiles pour Windows Azure.

Celui-ci est constitué, d'une part, d'une infrastructure serveur qui s'appuie bien évidemment sur Azure et qui héberge une base de données et un service d'accès à celle-ci via le protocole OData et, d'autre part, d'un kit de développement pour les applications clientes accédant à cette base.

Cet ensemble met à disposition du développeur d'applications Windows trois fonctionnalités complémentaires :

- l'accès à des données structurées stockées sur un serveur (méthodes d'insertion, de mise à jour, de suppression et de lecture) ;

Copyright © 2013 Eyrolles.

- l'authentification des utilisateurs à partir de leur compte Microsoft (en s'appuyant sur les classes présentées à la section précédente) ;
- l'envoi de notifications.

Par ailleurs, le portail de gestion d'un service mobile Azure sert au développeur pour suivre l'utilisation effective de son service.

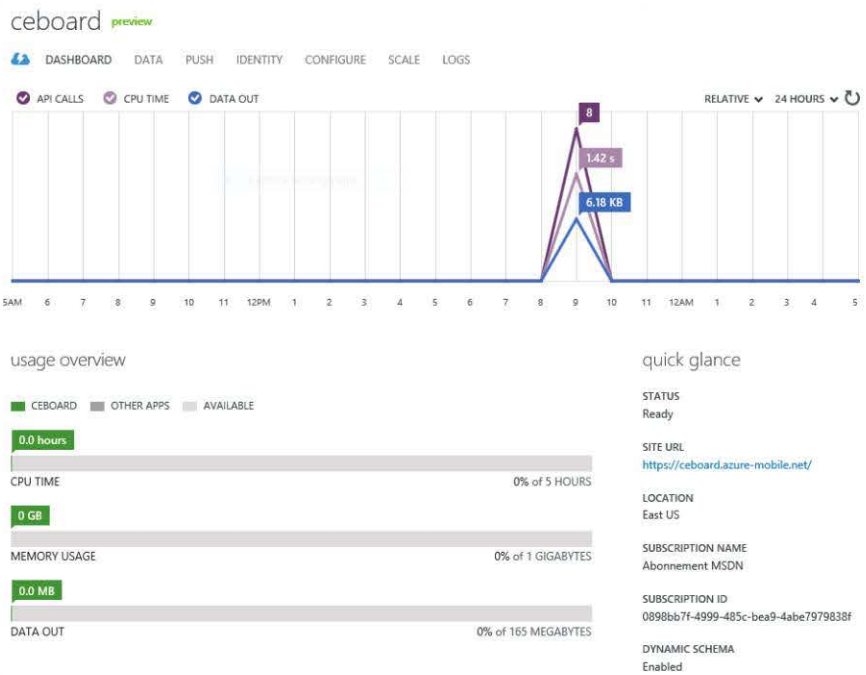


Figure 12–5
Tableau de bord d'un service mobile Azure

L'hébergement de 10 services mobiles dans un environnement partagé Azure est gratuit : ainsi, aucun investissement particulier n'est nécessaire pour démarrer une application. Si le succès est au rendez-vous et nécessite une plus grande capacité de calcul ou de stockage, il faut opter pour un environnement plus conséquent. Les tarifs ainsi qu'un estimateur de coûts sont disponibles sur le site Azure.

Création d'un service mobile

La première étape de l'utilisation des services mobiles Windows Azure consiste à créer le service et la base de données associée sur le portail de gestion Azure. Pour cela, il faut aller dans l'onglet *MOBILE SERVICES* de ce portail et cliquer sur *CREATE A NEW APP*. Trois champs doivent être renseignés :

- le *nom du service* : celui-ci conditionne l'URL à laquelle le service est accessible. Cette URL doit être unique, mais il faut néanmoins noter que l'utilisateur de l'application Windows 8 ne la voit à aucun moment ;

Copyright © 2013 Eyrolles.

- la *base de données* : il est possible de créer une nouvelle base SQL Azure dédiée au service ou de se connecter à une base existante ;
- la *région géographique* dans laquelle le service doit être hébergé (rapprocher celui-ci de ses utilisateurs améliore les temps de réponse).

L'écran suivant de l'assistant sert à préciser les informations de la base de données (il est souhaitable que la base soit dans la même région que le service).

Au niveau de la structure de la base de données elle-même, il y a deux options, accessibles via l'onglet *Configure* du tableau de bord :

- *Utilisation d'un schéma dynamique* : les colonnes de chaque table sont créées dynamiquement par le service en fonction des objets sauvegardés par le développeur (voir plus loin) ;
- *Définition du schéma de la table* : les propriétés des objets sont sauvegardées dans les colonnes définies à l'avance par l'administrateur du service.

Si l'application Windows n'utilise que la fonctionnalité base de données du service mobile, la configuration s'arrête ici. Si elle exploite les autres services proposés, ceux-ci nécessitent une configuration complémentaire et cohérente avec cette création du service, typiquement au niveau de l'URL d'accès à celui-ci.

Accès aux données stockées sur Azure

Avec ce kit, le développeur n'écrit pas de requêtes SQL explicites mais définit des entités qui sont « mappées » directement sur les tables et lues à l'aide de requêtes LINQ.

Une entité à sauvegarder doit posséder des propriétés annotées avec l'attribut `DataMember` en précisant le nom de la colonne correspondante. Elle doit aussi contenir une propriété appelée `Id` de type entier, par exemple :

```
public class Wine {
    public int Id { get; set; }
    [DataMember(Name = "NAME")]
    public string Name { get; set; }
    [DataMember(Name = "APPELLATION")]
    public string Appellation { get; set; }
    [DataMember(Name = "YEAR")]
    public int Year { get; set; }
}
```

Pour accéder à la base Azure, il faut instancier un objet de type `MobileServiceClient` en lui fournissant l'URL du service ainsi que le « secret » (chaîne de caractères servant en quelque sorte de mot de passe) récupéré sur le portail d'administration du service. Ensuite, les données sont manipulées via l'interface générique `IMobileServiceTable<T>` :

Copyright © 2013 Eyrolles.

- Récupération de l'objet table associé à la classe `Wine` :

```
private MobileServiceClient _mobileService;
private IMobileServiceTable<Wine> _wineTable;

public async Task Initialize() {
    _mobileService = new MobileServiceClient("https://ceboard.azure-mobile.net/
", clientSecret);
    _wineTable = _mobileService.GetTable<Wine>();
}
```

- Requête de sélection :

```
public MobileServiceCollectionView<Wine> WinesOlderThan(int year) {
    MobileServiceTableQuery<Wine> query = _wines.Where(wine => wine.Year <=
year);
    return query.ToCollectionView();
}
```

- Insertion d'un nouveau vin :

```
await _wineTable.InsertAsync(newWine);
```

Il est possible d'implémenter un certain nombre de règles de gestion côté serveur : cela se fait en écrivant de petites fonctions JavaScript qui sont invoquées au moment de l'exécution des requêtes (onglet *Script* de l'administration de la table sur le portail d'administration). Typiquement, on procédera ainsi pour valider certaines plages de valeurs ou introduire l'identité du client connecté afin de séparer les données de chaque utilisateur de l'application, par exemple :

- Vérification de plage + insertion de l'identité utilisateur :

```
function insert(item, user, request) {
    var year = parseInt(item.YEAR, 10);
    if (isNaN(year)) {
       request.respond(statusCodes.BAD_REQUEST, 'Year must be an integer : |' +
item.YEAR + '|');
    } else if ((year < 1900) || (year > 2012)) {
        request.respond(statusCodes.BAD_REQUEST, 'Year should be between 1900
and 2012');
    } else {
        item.userId = user.userId;
        request.execute();
    }
}
```

Copyright © 2013 Eyrolles.

- Lecture des données propres à l'utilisateur :

```
function read(query, user, request) {
   query.where({ userId: user.userId });
   request.execute();
}
```

Pour que le paramètre `User` soit renseigné dans ces requêtes, il faut préciser les permissions associées à la table dans le portail de management et réaliser l'authentification de l'utilisateur (voir section consacrée à SkyDrive), en fournissant à l'objet `MobileServiceClient` le jeton d'authentification Live récupéré lors de l'ouverture de la session :

```
…
LiveConnectClient client = new LiveConnectClient(result.Session);
MobileServiceUser loginResult = await
_mobileService.LoginAsync(result.Session.AuthenticationToken);
```

Bien évidemment, il est aussi possible d'implémenter un mécanisme d'authentification propre à l'application via des tables dédiées. Néanmoins, l'authentification à l'aide du compte Microsoft évite à l'utilisateur de créer et de mémoriser un nouveau mot de passe.

Copyright © 2013 Eyrolles.

13

Windows Runtime avancé

Les précédents chapitres de cet ouvrage ont présenté les différents aspects de la programmation d'une application Windows Store en illustrant ceux-ci à partir des principaux langages utilisés par les développeurs Windows. Voyons maintenant les rouages de Windows Runtime.

Ce chapitre détaille un certain nombre de mécanismes sous-jacents du Windows Runtime et explique :

- ce qu'est un composant WinRT ;
- ce qu'est un processus WinRT ;
- comment un processus WinRT est-il lancé ;
- comment des API WinRT identiques sont présentées de manière naturelle dans des langages a priori différents.

Processus WinRT

Lors des premières présentations de Windows 8 en général et de WinRT en particulier, la forte distinction faite entre les applications s'exécutant sur le bureau de manière classique et les applications provenant du Windows Store a donné à penser que ces dernières s'exécutaient dans un contexte particulier.

Copyright © 2013 Eyrolles.

Il n'en est rien : il s'agit de processus Win32 classiques qui s'exécutent dans la session de l'utilisateur courant. Ils fonctionnent de la même manière que les processus habituels, c'est-à-dire :

- ils sont démarrés via un exécutable Win32 ;
- et ils chargent un certain nombre de DLL.

> **VOCABULAIRE**
>
> Bien que WinRT signifie Windows Runtime, il n'y a pas de moteur d'exécution particulier comme le CLR de .NET : celui-ci sera certes présent dans les applications WinRT développées en C# ou avec Visual Basic .NET, mais les applications développées en C++ seront de vraies applications natives, exploitant au maximum les performances de la machine sur laquelle elles s'exécutent.

Dans le cas des applications natives écrites en C++ ou des applications .NET, cet exécutable correspond directement au code compilé de l'application. Dans le cas des applications développées en HTML5 et JavaScript, l'exécutable est en fait le processus `wwahost.exe`. Avec une version 64 bits de Windows, il s'agit de :

- `C:\Windows\system32\wwahost.exe` pour une application 64 bits (répertoire habituel),
- `C:\Windows\syswow64\wwahost.exe` pour une application 32 bits (répertoire contenant les versions 32 bits des utilitaires habituels),

Si pour HTML5 et JavaScript, il n'y a pas de réelle différence entre 32 et 64 bits, le choix entre les deux cibles peut avoir une importance lorsque des composants natifs sont utilisés : en effet, comme pour les autres applications Windows, une application WinRT ne peut pas mélanger au sein d'un même processus des composants 32 bits et des composants 64 bits.

Ces processus sont lancés par l'explorateur Windows via un service habituellement responsable de démarrer les composants DCOM (`svchost.exe`) quand l'utilisateur appuie sur une vignette de l'écran de démarrage. Ce lancement passe par l'intermédiaire d'un contrat similaire aux contrats de partage ou de recherche, le `Windows.Launch`, et les composants WinRT qui l'implémentent sont inscrits dans la base de registre sous la clé `HKEY_CURRENT_USER\Software\Classes\Extensions\ContractId\Windows.Launch`.

Les applications du Windows Store sont effectivement liées à l'utilisateur et non à la machine (elles peuvent d'ailleurs le suivre sur différentes machines) et il est donc normal que cet enregistrement soit sous `HKEY_CURRENT_USER` et non `HKEY_LOCAL_MACHINE` ou `HKEY_CLASSES_ROOT`. Les différents éléments du manifeste de l'application y sont consignés.

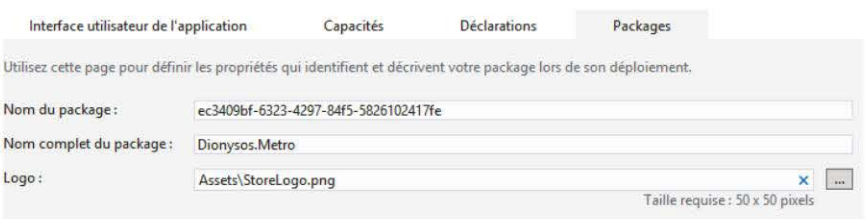


Figure 13–1
Identité du package vue sous Visual Studio

Copyright © 2013 Eyrolles.

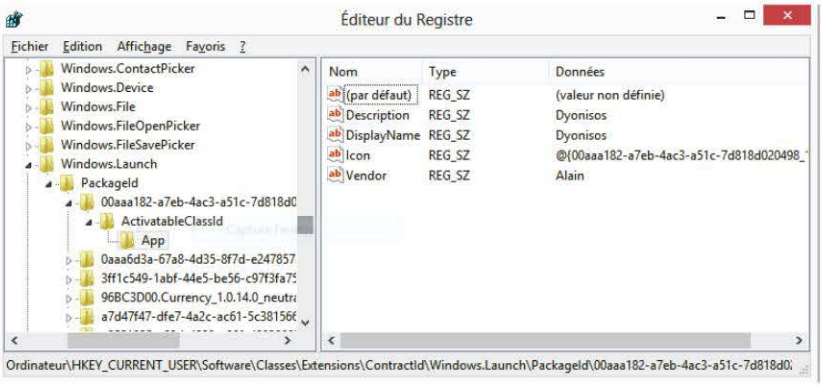


Figure 13–2
Identité du package retrouvée dans la base de registre.

En poursuivant les investigations dans la base de registre, l'identifiant du package ainsi que le chemin de l'exécutable apparaissent dans la clé `HKEY_CURRENT_USER\Software\Classes\ActivatableClasses\Package` : le lancement des processus WinRT s'apparente donc fortement à l'instanciation d'objets COM « *out of process* », par exemple comme pour le lancement de Word en Visual Basic par `CreateObject(« Word.Application »)`.

Il est maintenant temps de regarder plus précisément l'implémentation des composants eux-mêmes.

Composants WinRT

Évolution des composants logiciels

Quand les premières versions de Windows sont apparues, la programmation orientée objet n'était pas encore aussi répandue qu'aujourd'hui et, si ses concepts étaient déjà connus, les micro-ordinateurs se programmaient avec des langages moins élaborés, plus compatibles avec le manque de puissance de l'époque. Pour cette raison, l'interface de programmation de Windows était faite pour le langage C ou le Pascal et était constituée de collections de fonctions simples regroupées par famille : un composant était alors une DLL exposant un certain nombre de fonctions.

Encapsulation et langages non objet

Une interface sous forme de collection de fonctions n'est pas nécessairement synonyme de mauvaise programmation et d'absence de séparation entre interface et implémentation. En effet, beaucoup de fonctions de l'API Windows prennent pour paramètres des *handles* qui sont opaques pour le programmeur qui s'appuie sur ces API. Les notions de contrat et d'encapsulation sont donc bien respectées.

Copyright © 2013 Eyrolles.

Ce modèle de composition avait un certain nombre de limites :

- Tout d'abord, il se limitait aux appels intra-processus (les appels inter-processus étant alors réalisés via DDE qui était un modèle complètement différent, mais pas satisfaisant non plus).
- Ensuite, il ne proposait aucun moyen de découvrir de nouveaux composants ni de découvrir les fonctions proposées par ceux-ci. En effet, les DLL simples ne contiennent aucune métadonnée interrogeable par un programme.

Lorsque Microsoft a voulu proposer un modèle d'intégration des différentes applications Windows (appliqué avec succès dans la suite Office), il a été défini un nouveau modèle de composants compensant les lacunes du précédent : COM, pour *Component Object Model*. Il offre les fonctions suivantes :

- formalisation de l'interface binaire d'appel entre les composants permettant l'écriture de ceux-ci dans différents langages ;
- possibilité de réaliser des appels intra-processus ou inter-processus ;
- forte séparation entre interface et implémentation, autorisant différents composants à implémenter la même interface ;
- possibilité de découvrir les composants disponibles sur un système à partir d'une base de composants centralisée (partie de la base de registre) ;
- découverte des capacités d'un composant par interrogation dynamique du support des interfaces ;
- gestion des versions par immuabilité des interfaces existantes et enrichissement de celles-ci par héritage.

L'évolution suivante du modèle de composants de Windows est liée à l'apparition du framework .NET : la portée de celui-ci est cependant bien plus large, car .NET constitue un environnement de développement et d'exécution de tout type d'applications, allant du mobile au serveur hébergé dans le *cloud*. Certains concepts de .NET s'appliquent particulièrement bien aux composants et offrent de réelles avancées par rapport à COM :

- Un système de métadonnées détaillées incorporées dans chaque composant aide à découvrir l'ensemble des caractéristiques de celui-ci, y compris sans avoir à le charger (particulièrement important pour les outils de développement, mais aussi pour des raisons de sécurité).
- Les composants peuvent être découverts et chargés sans nécessiter le recours à la base de registre. Cela facilite de plus un certain nombre de scénarios comme la cohabitation sur une même machine de plusieurs versions d'un composant ou même d'une application (le *side-by-side*).

Il faut cependant noter que si .NET a remplacé Visual Basic pour le développement des applications, il n'a jamais remplacé COM au niveau du système d'exploitation et

Copyright © 2013 Eyrolles.

même les versions récentes de Windows (Vista ou Windows 7) apportaient toujours un lot de nouvelles interfaces COM, en particulier pour enrichir les possibilités d'interaction avec l'explorateur de Windows.

Il faut aussi ajouter que la taille du runtime .NET fait qu'il ne peut pas servir d'architecture de composants universelle pour Windows : une application écrite en C++ ne doit pas se voir imposer le chargement d'un environnement d'exécution conséquent pour charger un simple composant.

WinRT capitalise donc sur l'historique riche des modèles de composants de Windows pour proposer un modèle abouti qui va avoir une portée plus large que les précédents. L'API WinRT offre un modèle de programmation naturel, quels que soient les outils et les langages de développement utilisés, et pourra donc couvrir l'ensemble du spectre des applications Windows. Dans Windows 8, cette couverture se limite aux applications dites « modernes », mais l'API WinRT s'enrichira au fur et à mesure des futures mises à jour de Windows (il est possible d'appeler certaines API WinRT à partir d'une application Bureau, mais il n'est pas possible d'écrire une telle application uniquement avec ces API).

Au cœur de WinRT, se trouvent une évolution de COM et des métadonnées de .NET. De manière simplifiée :

- COM apporte son interface binaire permettant de relier des composants provenant d'environnements de développement différents, le schéma d'instanciation via une fabrique (*factory*), la gestion du cycle de vie par comptage de références[1] ;
- les métadonnées de .NET simplifient l'analyse des services rendus par chaque composant (ainsi que de leurs prérequis) et autorisent le développement des outils et interfaces qui rendent alors naturels les appels hétérogènes mentionnés précédemment.

  Dans les présentations et la documentation de WinRT, le terme de « projection » est utilisé pour décrire la traduction de l'API WinRT en un ensemble de classes et méthodes s'intégrant de manière naturelle dans chaque langage de programmation. Ce mécanisme est décrit plus en détails à la fin de ce chapitre ;
- enfin, WinRT n'impose pas l'enregistrement des composants : à part ceux qui font partie du système d'exploitation, les composants sont distribués et installés avec les applications qui les utilisent. Il n'y a donc pas de risque d'incompatibilité découverte au moment de l'exécution à cause du chargement d'un composant dans une version différente de celle avec laquelle l'application a été testée.

En termes de performances, un appel WinRT est identique à un appel COM. Dans le cas d'un appel au sein d'un processus, il s'agit essentiellement d'un appel de méthode virtuelle, ce qui est très efficace.

1. L'ennemi du comptage de références est la création de cycles qui engendrent des fuites mémoire. Pour en réduire le risque, WinRT propose la notion de WeakReference qui permet de créer ces cycles (par exemple dans le cas d'une liaison maître-détail bidirectionnelle) sans conséquence sur la durée de vie des objets.

Copyright © 2013 Eyrolles.

Anatomie d'un composant WinRT

Classe et interfaces

Quel que soit le langage dans lequel il est implémenté, un composant WinRT est donc un objet COM. À ce titre, il respecte les règles de base de ce modèle de composants et implémente la vénérable interface `IUnknown` et ses trois méthodes bien connues :

- `QueryInterface` pour obtenir les autres interfaces implémentées par le composant ;
- `AddRef` et `Release` pour en gérer la durée de vie.

La principale limitation de `IUnknown` et `QueryInterface` est que l'appelant doit demander une à une les interfaces qu'il est capable d'exploiter et qu'il n'a pas moyen de connaître à l'avance les interfaces prises en charge. Il faut les acquérir une à une et gérer les succès et les échecs. Ceci convient à un certain nombre de scénarios (surtout en mode natif), mais certainement pas aux scénarios plus dynamiques. C'est pour répondre à ce problème que les composants WinRT implémentent tous une deuxième interface obligatoire, `IInspectable`, définie ainsi :

```
MIDL_INTERFACE("AF86E2E0-B12D-4c6a-9C5A-D7AA65101E90")
IInspectable : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE GetIids(
        /* [out] */ ULONG *iidCount,
        /* [size_is][size_is][out] */ IID **iids) = 0;

    virtual HRESULT STDMETHODCALLTYPE GetRuntimeClassName(
        /* [out] */ HSTRING *className) = 0;

    virtual HRESULT STDMETHODCALLTYPE GetTrustLevel(
        /* [out] */ TrustLevel *trustLevel) = 0;
};
```

Le tableau suivant expose le rôle de chacune de ces méthodes.

`GetIids`	Retourne un tableau contenant les identifiants des interfaces implémentées par le composant WinRT. `IUnknown` et `IInspectable` ne sont pas retournées puisque systématiquement implémentées. L'appelant doit libérer via `CoTaskMemFree` le pointeur retourné.
`GetRuntimeClassName`	Retourne le nom complet du composant.
`GetTrustLevel`	Indique le type de ressources auxquelles le composant va chercher à accéder : 1. `BaseTrust` : accès aux ressources non sécurisées ; 2. `PartialTrust` : accès aux ressources déclarées dans le manifeste de l'application ; 3. `FullTrust` : accès complet aux ressources de l'utilisateur.

Copyright © 2013 Eyrolles.

> **ANECDOTE IKnown ?**
>
> La petite histoire veut que cette interface ait failli s'appeler `IKnown` pour mieux illustrer son rôle. Néanmoins, si à l'écrit les termes `IUnknown` et `IKnown` sont bien différenciés, il n'en est pas de même à l'oral. Microsoft a préféré éviter les risques de confusion et a donc choisi `IInspectable`.

La méthode `GetRuntimeClassName` illustre une différence importante entre COM et WinRT : COM s'attache uniquement aux interfaces alors que WinRT s'appuie certes sur les interfaces en termes de dialogue entre composants, mais **met en avant les classes elles-mêmes**. Ceci est en fait indispensable pour fournir aux applications écrites en JavaScript un accès complet et naturel à l'API WinRT, car JavaScript ne possède pas la notion d'interface et ne peut manipuler que des objets « entiers ».

De fait, l'implémentation et l'utilisation de ces interfaces techniques sont prises en charge par le compilateur et le développeur n'a donc à se soucier que des aspects fonctionnels de ses composants.

Voici le code d'un composant WinRT simple qui va servir à illustrer un certain nombre de points dans la suite de ce chapitre :

```
namespace CalculatorComponent
{
    public ref class IntegerCalculator sealed
    {
    public:
        IntegerCalculator();

        int Add(int left, int right);
        int Mult(int left, int right);
        int Sub(int left, int right);
        int Div(int left, int right, int *pRemainder);

        static IntegerCalculator^ CreateIntegerCalculator();

        Windows::Foundation::Collections::IIterable<int>^ Factors(int number);
    };
}
```

Dans cet exemple :

- c'est le `ref` devant le mot-clé `class` qui indique au compilateur C++ qu'il s'agit d'un composant WinRT et non d'une simple classe C++ (qui ne serait pas exploitable dans d'autres langages) ;
- une interface masquée aux développeurs (à la fois celui qui implémente le composant et ceux qui vont l'utiliser) définit toutes les méthodes publiques du composant : tous les appels externes à celui-ci passeront par cette interface.

Copyright © 2013 Eyrolles.

En revanche, lorsque les appels ont lieu dans le même environnement, ils s'appuient sur les mécanismes natifs du langage : ce sera par exemple le cas si la solution Visual Studio contient à la fois le projet du composant et celui de l'application qui l'utilise et si le fichier en-tête du composant est correctement inclus dans le code qui l'utilise ;

- une deuxième interface masquée aux développeurs contient les méthodes statiques de la classe. Celles-ci n'ont pas besoin de la création préalable d'une instance pour être appelées.

> .NET permettait déjà l'appel de méthodes statiques de composants, mais ce n'était pas le cas de COM qui imposait systématiquement l'instanciation de composants.

Modèle de threading

Les différents modèles de threading COM ont souvent été un cauchemar pour les développeurs, car il fallait combiner le modèle du composant et celui de l'appartement dans lequel il était instancié pour comprendre où le composant s'exécutait. WinRT simplifie ces modèles :

- l'interface utilisateur s'exécute classiquement sur un environnement mono-tâche qui n'est pas réentrant : les appels entrants vers cet environnement sont acceptés uniquement s'ils s'effectuent en réponse à un appel sortant ;

> Un composant COM STA effectuant un appel sortant rendait son appartement capable de traiter n'importe quel appel entrant, ce qui pouvait poser des problèmes… puisqu'il pouvait alors ne plus être disponible pour traiter les vraies réponses de son appel sortant.

- les composants de l'interface utilisateur doivent être invoqués uniquement sur le thread de celle-ci ;
- les composants non graphiques ne sont pour la plupart pas liés à un thread donné ; ils peuvent donc être appelés de partout. Cela signifie qu'il faut mettre en place des mécanismes de synchronisation si des appels concurrents sont possibles, car ces composants ne gèrent pas cet aspect eux-mêmes ;
- certains composants s'exécutent en dehors du processus appelant au sein d'un processus particulier, le *Runtime Broker*, qui gère ainsi les accès à certaines ressources ne pouvant pas être partagées en temps réel. C'est par exemple le cas des caméras intégrées (si deux applications reçoivent un même flux vidéo, il y aura conflit dès que l'une d'entre elles cherchera à modifier les réglages de la caméra).

Copyright © 2013 Eyrolles.

Métadonnées

WinRT associe des métadonnées à chaque composant. Cela facilite la projection des API, ainsi que l'écriture des outils de développement, et permet l'implémentation d'Intellisense quel que soit le langage de développement d'un composant WinRT ou de l'application qui l'utilise.

Le format des métadonnées de WinRT est directement issu de celui de .NET. Il est donc possible d'analyser un composant WinRT en utilisant la version d'ILDASM qui accompagne Visual Studio 2012. Ces métadonnées sont enregistrées dans des fichiers au suffixe `.winmd` (pour Windows Meta Data) :

- dans le cas d'un composant WinRT développé en .NET, le `.winmd` contient à la fois les métadonnées et le code IL : il remplace donc les `.DLL` et `.EXE` (en fait, il apparaît que le `.winmd` contient aussi les en-têtes classiques des fichiers exécutables Windows et est donc très similaire aux bibliothèques .NET) ;
- dans le cas d'un composant WinRT natif, le code exécutable n'est pas dans le `.winmd` mais reste dans le fichier `.DLL` ou `.EXE` habituel. Le fichier `.winmd` ne contient alors que les métadonnées utiles au runtime WinRT et aux outils associés. Il faut noter que ILDASM ne peut pas désassembler le code natif.

> **OUTILS JavaScript et composants**
>
> JavaScript et HTML 5 sont souvent mis en avant pour le développement d'applications Windows Store et sont des « citoyens de première classe ». En revanche, ils ne permettent pas de développer des composants WinRT *packagés*. Ils peuvent néanmoins en appeler facilement, ce qui fait du triplet HTML 5/JavaScript/C++ une excellente solution pour un grand nombre de scénarios.

Les composants WinRT intégrés à Windows sont gérés de la même manière que les autres : leurs métadonnées sont disponibles et regroupées dans le répertoire `C:\Windows\System32\WinMetadata` qui contient une quinzaine de fichiers. Les DLL correspondantes sont quant à elles dans le répertoire habituel `C:\Window\System32`.

ILDASM appliqué au composant présenté précédemment montre la dualité entre la classe écrite par le développeur et les interfaces créées automatiquement par le compilateur :

- `__IIntegerCalculatorPublicNonVirtuals` regroupe toutes les méthodes d'instance publiques du composant ;
- `__IIntegerCalculatorStatics` regroupe les méthodes statiques de la classe.

Copyright © 2013 Eyrolles.

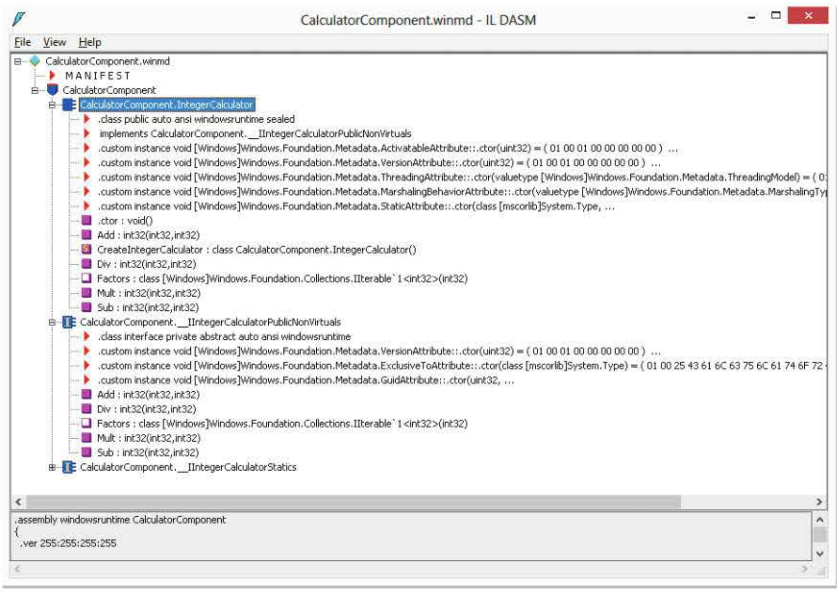


Figure 13–3
Analyse des interfaces avec ILDASM

Même si `ILDASM.EXE` sait lire les fichiers de métadonnées `.winmd`, les classes .NET de l'espace de nommage `System.Reflection` ne le peuvent pas. En effet, le format est commun, mais les classes WinRT ne sont pas nécessairement des classes .NET et leurs systèmes de types sont différents.

Pour lire les fichiers de métadonnées, il est donc nécessaire de s'appuyer sur des interfaces COM… et d'écrire un code assez proche de celui qui analysait l'interface `ITypeInfo` d'Automation. La première étape est d'instancier un `MetaDataDispenser`, ce qui se fait directement à l'aide d'une fonction contenue dans `mscoree.dll` :

```
hr = MetaDataGetDispenser(CLSID_CorMetaDataDispenser,
                          IID_IMetaDataDispenserEx,
                          (LPVOID*)&pMetadataDispenser);
```

Ensuite, il faut ouvrir le fichier `.winmd` pour récupérer une interface `IMetaDataImport2` :

```
LPCWSTR szDllPath = L"c:\\...\\CalculatorComponent.winmd";
hr = pMetadataDispenser->OpenScope(szDllPath,
                                   0,
                                   IID_IMetaDataImport2,
                                   (IUnknown**)&pMetaDataImport2);
```

Cette interface sera utilisée pour parcourir l'ensemble des métadonnées contenues dans le fichier (types, méthodes, propriétés, paramètres). Il n'y a pas d'interface

Copyright © 2013 Eyrolles.

« élémentaire » pour chaque type de métadonnée. Le code qui suit parcourt les types décrits dans le fichier et affiche le nom des méthodes de chacun :

- les types sont manipulés via des identifiants opaques (représentés par des entiers non signés) ;
- les énumérations se font en appelant en boucle des méthodes auxquelles il faut passer un *handle* d'énumération (à libérer par un appel ultérieur à `CloseEnum`) ;
- les noms des objets sont uniquement accessibles en Unicode.

Énumération des types contenus dans le fichier winmd

```
void DumpTypes(IMetaDataImport2 *pMetaDataImport2)
{
      HCORENUM hEnum = nullptr;
      mdTypeDef rTypeDefs[10];
      ULONG cTypeDefs;
      HRESULT hr;
      while (S_OK == (hr = pMetaDataImport2->EnumTypeDefs(&hEnum, rTypeDefs,
                                          ARRAYSIZE(rTypeDefs), &cTypeDefs))) {
         for (ULONG i=0;i<cTypeDefs;++i) {
            DumpType(pMetaDataImport2, rTypeDefs[i]);
         }
      }
      if (hr != S_FALSE) {
         Error(hr, L"EnumTypeDefs");
      }
      if (hEnum != nullptr)) {
         pMetaDataImport2->CloseEnum(hEnum);
      }
}
```

Énumération des méthodes de chaque type

```
void DumpType(IMetaDataImport2 *pMetaDataImport2, mdTypeDef tdef)
{
      WCHAR szTypeName[512];
      ULONG cbChars = ARRAYSIZE(szTypeName);
      DWORD dwFlags = 0;
      mdTypeDef parentTdef = 0;
      HRESULT hr = pMetaDataImport2->GetTypeDefProps(tdef,
                                szTypeName, ARRAYSIZE(szTypeName), &cbChars,
                                &dwFlags, &parentTdef);
      if (FAILED(hr)) {
         Error(hr, L"GetTypeDefProps");
      } else {
         wcout << L"Type: " << szTypeName << endl;
         HCORENUM hMethods = nullptr;
```

Copyright © 2013 Eyrolles.

```
         mdMethodDef methods[1];
         ULONG cMethods;
         wcout << L" Methods:" << endl;
         while (S_OK == (hr = pMetaDataImport2->EnumMethods(&hMethods, tdef,
                                                methods, ARRAYSIZE(methods),
                                                &cMethods))) {
            for (ULONG i=0;i<cMethods;++i) {
              DumpMethod(pMetaDataImport2, methods[i]);
            }
         }
         if (hr != S_FALSE) {
         Error(hr, L"EnumMethods");
         }
         if (hMethods != nullptr) pMetaDataImport2->CloseEnum(hMethods);
      }
}
```

Affichage du nom de la méthode (d'autres informations peuvent être obtenues par le même appel)

```
void DumpMethod(IMetaDataImport2 *pMetaDataImport2, mdMethodDef mdef)
{
      WCHAR szMethodName[512];
      ULONG cbChars = ARRAYSIZE(szMethodName);
      DWORD dwFlags = 0;
     HRESULT hr = pMetaDataImport2->GetMethodProps(mdef, nullptr, szMethodName,
                                                   cbChars, &cbChars, &dwFlags,
                                                    nullptr, nullptr, nullptr,
nullptr);
      if (FAILED(hr)) {
         Error(hr, L"GetMethodProps");
      } else {
         wcout << L"    " << szMethodName << endl;

      }
}
```

Exécuté sur le composant C++ WinRT présenté précédemment, ce code donne bien le résultat attendu.

Activation et fabriques

Comme pour un composant COM, l'instanciation d'un composant WinRT passe par une fabrique (*factory*) ; néanmoins, le couple `DllGetClassObject` (méthode exportée par toutes les DLL COM pour donner accès aux fabriques des objets implémentés par la DLL) et `IClassFactory` (interface implémentée par les fabriques) est remplacé par `DllGetActivationFactory` et `IActivationFactory`.

Copyright © 2013 Eyrolles.

Figure 13–4
Affichage des interfaces

Il faut donc parler ici d'activation de composant plutôt que d'instanciation. L'interface `IActivationFactory` est très simple puisqu'elle n'expose qu'une seule méthode :

```
MIDL_INTERFACE("00000035-0000-0000-C000-000000000046")
IActivationFactory : public IInspectable
{
public:
    virtual HRESULT STDMETHODCALLTYPE ActivateInstance(
        /* [out] */ IInspectable **instance) = 0;
};
```

Le développeur WinRT n'appelle pas explicitement `DllGetActivationFactory` ni l'interface `IActivationFactory`, car le code correspondant est créé par son compilateur à partir de l'instruction `new` des langages de programmation (`ref new` dans le cas du C++). Si un développeur C++ veut développer un composant sans utiliser la moindre bibliothèque ni les extensions du langage pour WinRT, il doit en revanche implémenter ces méthodes et interfaces.

Une fonction instancie directement un composant WinRT à partir du nom de sa classe. Il s'agit de `RoActivateInstance`, qui accepte en paramètres le nom complet d'une classe WinRT (par exemple `Windows.Storage.Pickers.FileOpenPicker`) et l'adresse d'un pointeur sur l'interface `IInspectable` qui sera retournée :

```
HSTRING hClassName = nullptr;
IInspectable *pFilePicker = nullptr;
LPCWSTR szClassName= L"Windows.Storage.Pickers.FileOpenPicker";
HRESULT hr = WindowsCreateString(szClassName, wcslen(szClassName),
                                                 &hClassName);
```

Copyright © 2013 Eyrolles.

```
if (FAILED(hr)) {
      Error(hr, L"WindowsCreateString");
      goto Cleanup;
}
hr = RoActivateInstance(hClassName, &pFilePicker);
if (FAILED(hr)) {
      Error(hr, L"RoActivateInstance");
      goto Cleanup;
}
…
Cleanup:
      if (pFilePicker != nullptr) pFilePicker->Release();
      if (hClassName != nullptr) WindowsDeleteString(hClassName);
```

`RoActivateInstance` est utilisée dans les cas où le développeur C++ préfère ne pas utiliser les extensions WinRT du compilateur (ce qui en pratique n'est guère justifié)… mais elle pourrait l'être aussi lorsqu'un programme Win32 a besoin d'instancier une classe WinRT ! Ceci est en effet possible dans une certaine mesure, même si c'est peu ou mal documenté et si la liste des composants compatibles avec Win32 n'est pas disponible.

Seuls les composants standards, i.e. ceux fournis par WinRT, sont instanciables par une application Win32 : il n'est donc pas envisageable d'écrire une bibliothèque de composants WinRT et de les utiliser dans deux versions d'une même application, une application Windows classique s'exécutant sur le bureau et une application pour le Windows Store lancée à partir de l'écran d'accueil. Si un développeur a un tel besoin, il doit développer le cœur du composant partagé de manière classique (c'est-à-dire avec une classe C++ utilisant des types standards) puis l'enrober dans un composant WinRT pour permettre son utilisation dans cet environnement.

Le code d'activation d'un composant WinRT à partir d'une application bureau classique est identique à celui de l'exemple précédent ; il faut néanmoins appeler `RoInitialize` avant d'appeler `RoActivateInstance`.

Sous WinRT, les fabriques de classes ne servent pas uniquement à créer les instances de celles-ci. Les classes WinRT peuvent définir des méthodes statiques. Ces méthodes sont regroupées dans une interface dont le nom se termine par `Statics` : la fabrique d'une classe WinRT implémente en fait elle-même cette interface. Ceci permet d'appeler une méthode statique sans avoir à créer d'instance de la classe correspondante.

La méthode `RoGetActivationFactory` récupère la fabrique d'une classe à partir du nom complet de celle-ci. Cette méthode accepte un paramètre indiquant l'interface à récupérer sur cette fabrique : pour instancier une classe, il faut demander l'interface `IActivationFactory`, mais il est possible de demander à `RoGetActivationFactory` l'interface contenant les méthodes statiques, ce qui évite d'appeler `QueryInterface` et d'avoir deux pointeurs à libérer.

Copyright © 2013 Eyrolles.

L'exemple de code qui suit utilise la méthode statique `GetDefault` de la classe `Windows.Devices.Sensors.LightSensor` pour obtenir une instance du capteur par défaut, puis lit la valeur courante de celui-ci.

```
LPCWSTR szLightSensor = L"Windows.Devices.Sensors.LightSensor";
HSTRING hClassName = nullptr;
ILightSensorStatics *pLightSensorStatics = nullptr;
ILightSensor *pLightSensor = nullptr;
ILightSensorReading *pReading = nullptr;

HRESULT hr = RoInitialize(RO_INIT_SINGLETHREADED);
if (FAILED(hr)) {…}

hr = WindowsCreateString(szLightSensor, wcslen(szLightSensor), &hClassName);
if (FAILED(hr)) {…}

hr = RoGetActivationFactory(hClassName, IID_ILightSensorStatics,
(void**)&pLightSensorStatics);
if (FAILED(hr)) {…}

hr = pLightSensorStatics->GetDefault(&pLightSensor);
if (FAILED(hr)) {…}

if (pLightSensor == nullptr) {
   wcout << "Pas de capteur de lumière"<< endl;
} else {
      hr = pLightSensor->GetCurrentReading(&pReading);
       if (FAILED(hr)) {…}

      FLOAT fLuminance;
      hr = pReading->get_IlluminanceInLux(&fLuminance);
       if (FAILED(hr)) {…}
      wcout << "Luminance " << fLuminance << endl;
}
…
```

Système de types WinRT et projections

L'API WinRT est constituée d'un ensemble de classes et d'interfaces couvrant l'essentiel des besoins des applications Windows Store : interface utilisateur, entrées-sorties, réseau, capteurs... Cette interface de programmation est accessible depuis différents langages et il est possible de mélanger au sein d'une même application des composants WinRT écrits dans ces différents langages.

Ceci n'est possible que grâce à la définition précise d'un certain nombre de mécanismes fondamentaux sur lesquels toutes les classes constituant WinRT sont construites.

Copyright © 2013 Eyrolles.

Le système de types tout d'abord. C'est le point de départ : il faut bien évidemment que les différents environnements de programmation partagent les mêmes concepts et les implémentent de manière similaire ou suffisamment proche pour que les passages d'un environnement à un autre ne soient pas pénalisants en termes de performances. Les types de base doivent donc être identiques et aussi représentés de manière identique : si un langage ne manipulait que des entiers signés et un autre uniquement des entiers non signés, de nombreuses valeurs ne pourraient pas être passées entre composants écrits dans ces langages. Les langages de programmation définissent aussi leur système de types et, dans certains cas, cette définition est incompatible avec celle de WinRT. C'est typiquement le cas du C++ pour les chaînes de caractères :

- Une chaîne de caractères WinRT est immuable : cela signifie qu'elle ne peut pas être modifiée une fois créée. Toutes les opérations comme le changement de casse, la concaténation ou encore la troncature se traduisent en réalité par la création de nouvelles chaînes.
- Une chaîne de caractères C/C++ est simplement un pointeur vers une zone de mémoire contenant une suite de caractères terminée par 0. Elle peut être modifiée par des accès directs en écriture et allongée ou raccourcie par déplacement du caractère nul (ce qui au passage est la cause d'innombrables failles de sécurité dans tous les systèmes d'exploitation).

Pour permettre l'interopérabilité entre les composants WinRT sans remettre en cause le fonctionnement du C++ et la compatibilité de Visual C++ avec la norme ISO du langage, WinRT définit l'espace de noms `Platform`, contenant tous les types de base qui ne sont pas naturels dans ses langages de programmation. Ainsi, la classe `Plaform::String` correspond à la notion de chaîne de caractères WinRT, mais est accessible en C++ et offre par ailleurs les fonctions pour convertir une chaîne C++ en `Platform::String` et vice-versa.

PERFORMANCES

On évitera néanmoins de multiplier les conversions de chaînes ou d'autres types un peu coûteux. Cette recommandation se traduit par une séparation assez forte entre le code lié au langage et le code proche de WinRT. L'intégration d'un composant écrit en C++ ISO se traduira généralement par l'écriture d'une couche d'interopérabilité fine qui ne sera traversée que pour certains appels un peu conséquents.

Les types de base ne sont pas suffisants pour définir une interface de programmation aussi riche que WinRT et il est nécessaire de les composer pour créer :

- des structures, des classes et interfaces ;
- des tableaux ;
- des types génériques.

Copyright © 2013 Eyrolles.

La convention d'appel des méthodes d'instances et statiques. Quasi tous les programmeurs cherchant à appeler une fonction Win32 ont un jour été confrontés à des erreurs ou exceptions liées aux conventions d'appel différentes entre le C et le Pascal que suivait Windows pour des raisons historiques. Ces conventions régissent l'ordre dans lequel les paramètres sont déposés sur la pile lors d'un appel et qui supprime ces paramètres lors du retour.

Le format des fichiers binaires exécutables et le moyen d'accéder aux objets qu'il implémente.

Tous ces éléments constituent ce qui s'appelle l'ABI pour *Application Binary Interface* et qui peut être rapprochée de la CLI *(Common Language Infrastructure)* de .NET.

La transposition des types et conventions WinRT dans un langage de programmation donné est quant à elle appelée **projection**. Cette opération consiste non seulement à rendre accessible à chaque langage de programmation l'ensemble de l'interface WinRT, mais aussi à rendre cette interface naturelle dans chacun de ces langages. Par exemple, les langages .NET utilisent la convention des majuscules en début de chaque mot (*Pascal casing*) pour l'écriture des noms des classes et des méthodes, alors que JavaScript utilise pour sa part la convention d'une minuscule au premier mot et des majuscules au début des mots suivants (*Camel casing*) pour les méthodes. Une même classe WinRT va apparaître différemment selon le langage pour respecter les conventions usuelles de celui-ci ; par exemple :

Lecture de la valeur d'un capteur de lumière en C#

```
LightSensor capteur = LightSensor.GetDefault();
LightSensorReading valeur = capteur.GetCurrentReading();
```

En JavaScript

```
var capteur = Windows.Devices.Sensors.LightSensor.getDefault();
var valeur = capteur.getCurrentReading();
```

Les projections de l'API WinRT vont au-delà de la casse des identifiants et plusieurs concepts sont soit implémentés ou utilisés de manière différente selon les langages, soit totalement absents.

Propriétés et méthodes

Le C++ standard ne connaît pas la notion de propriétés : celles-ci se traduisent donc par des méthodes `get_` et `put_` comme dans l'exemple qui suit. Ce dernier précise le mode de visualisation d'une boîte de sélection de fichiers :

```
hr = pFilePicker->put_ViewMode(PickerViewMode::PickerViewMode_List);
```

Copyright © 2013 Eyrolles.

Les extensions du C++/CX offrent cette notion et c'est alors le compilateur qui effectue la transposition (qui n'a aucune conséquence sur les performances).

CULTURE C++/CX

Le C++/CX est une version du C++ proposée par Microsoft pour le développement de composants et applications Windows Store, qui simplifie l'écriture du code en quelques extensions. Même si celles-ci ressemblent au C++/CLI de .NET, la ressemblance n'est que superficielle et le code créé est bien du code natif sans aucune concession sur les performances et totalement compatible avec des composants écrits dans un C++ « pur ».

Collections

De nombreuses méthodes de WinRT retournent des collections d'objets qui sont ensuite parcourues manuellement ou via les mécanismes de liaison de données. Par exemple, `StorageFolder.GetFilesAsync()` retourne, via une opération asynchrone, une collection de fichiers sous la forme d'une interface `IVectorView` (qui va en fait être utilisée pour tous les parcours en lecture seule).

Ainsi, en C++, `GetFilesAsync` renvoie un objet du type `IAsyncOperation<IVectorView<StorageFile^>^>^` et les en-têtes standards de WinRT contiennent les méthodes pour parcourir ce résultat d'une manière conforme au standard du langage (en C++ 11 dans l'exemple suivant) :

```
LogItems->Items->Clear();
concurrency::create_task(KnownFolders::PicturesLibrary->GetFilesAsync())
   .then([this](IVectorView<StorageFile^>^ files) {
      for (StorageFile^ file : files) {
        LogItems->Items->Append(file->DisplayName);
      }
});
```

En C#, l'interface `IVectorView` est « projetée » sur l'interface `IReadOnlyList` plus familière aux programmeurs .NET et compatible avec l'instruction `foreach` puisque dérivant de `IEnumerable`. `GetFilesAsync` retourne donc un objet du type : `IAsyncOperation<IReadOnlyList<StorageFile>>`. L'exemple précédent devient donc :

```
LogItems.Items.Clear();
foreach (var file in await KnownFolders.PicturesLibrary.GetFilesAsync())
{
    LogItems.Items.Add(file.DisplayName);
}
```

Copyright © 2013 Eyrolles.

En JavaScript, la notion de type générique n'existe pas, mais le langage est suffisamment dynamique pour compenser. Comme la collection de fichiers est renvoyée par un appel à WinRT, il est effectivement possible pour le runtime de créer dynamiquement un objet qui possède les propriétés attendues du type annoncé par Intellisense, c'est-à-dire `IAsyncOperation<IVectorView<StorageFile>>`. Le parcours de ce résultat est, comme dans les autres cas, une construction naturelle du langage :

```
var resultDiv = document.getElementById("result");
resultDiv.innerHTML = "";
Windows.Storage.KnownFolders.picturesLibrary.getFilesAsync()
    .then(function (files) {
        for (var i = 0; i < files.length; i++) {
            resultDiv.innerHTML += files[i].displayName + "<br/>";
        }
    }, function () {
        resultDiv.innerHTML = "Error...";
    });
```

Cette définition dynamique des objets JavaScript pour compenser l'absence des types génériques tout en exposant les méthodes et propriétés attendues est effectuée à l'exécution en se basant sur les métadonnées des méthodes invoquées.

Le tableau suivant présente les principales interfaces que les composants WinRT doivent exposer pour la gestion des collections.

Type de collection	Utilisation	Interface
Liste simple	Lecture seule	`IVectorView<T>`
	Lecture / écriture	`IVector<T>`
	Lecture / écriture avec notification	`IObservableVector<T>`
Collection énumérable	Démarrage de parcours	`IIterable<T>`
Énumération de collection	Parcours simple	`IIterator<T>`
Liste associative	Lecture seule	`IMapView<K,V>`
	Lecture / écriture	`IMap<K,V>`
	Lecture / écriture avec notification	`IObservableMap<K,V>`
Paire clé-valeur	Élément de liste associative	`IKeyValuePair<K,V>`

Lorsqu'un développeur écrit un composant WinRT en .NET, il utilise les interfaces .NET classiques qui répondent à ses besoins et c'est la projection qui effectue le renommage. Ainsi, une collection d'entiers qui supporte une énumération doit implémenter `IEnumerable<int>` : cette interface est alors vue par les clients de ce composant comme `IIterable<int>` (sauf bien sûr dans le cas d'un composant .NET qui voit bien l'interface `IEnumerable<int>`).

Copyright © 2013 Eyrolles.

Si un composant WinRT est écrit en C++, il existe un ensemble de méthodes ou de constructeurs qui exposent les interfaces attendues par WinRT à partir des conteneurs STL classiques de C++. En revenant à l'exemple du composant de calcul présenté au début de ce chapitre, il expose la méthode suivante :

```
Windows::Foundation::Collections::IIterable<int>^ Factors(int number);
```

Une implémentation naturelle de cette méthode en C++ consiste à ajouter les nombres trouvés à un objet de type `std::vector<int>`. Une fois les calculs effectués, il est possible de retourner un objet de type `VectorView<int>` qui implémente (entre autres) l'interface souhaitée.

Cet exemple illustre à nouveau la volonté de capitaliser sur le C++ standard, tout en respectant des interfaces interopérables entre composants écrits dans des langages différents.

Copyright © 2013 Eyrolles.

Conclusion

Windows crée sa petite révolution, notamment avec son nouvel écosystème applicatif qui permet d'installer une galaxie d'applications et de les faire interagir entre elles pour apporter de nouvelles fonctionnalités et ainsi enrichir l'expérience utilisateur.

Pour le développeur, les opportunités sont énormes. La double interface de Windows va immédiatement assurer la présence de cet environnement pourtant nouveau sur un nombre impressionnant d'ordinateurs et, grâce au Windows Store, le développeur proposera ses réalisations à tous ces utilisateurs de manière très simple.

Dans ce livre, nous avons abordé la programmation d'applications pour le Windows Store en combinant différents objectifs.

Tout d'abord, l'API WinRT est très riche et il est important d'en connaître les principaux concepts et leur mise en œuvre. Nous avons vu comment construire une application en gérant ses différents états visuels, puis présenté la gestion de ses données (à la fois en termes d'affichage et de persistance) et comment intégrer une application dans un environnement plus large (interfaces avec les autres applications, avec les périphériques et enfin avec les principaux types de services Internet).

Comme un tel ouvrage ne peut être exhaustif, nous avons le plus souvent possible pris un peu de recul et expliqué la logique qui se cache derrière les concepts présentés, afin que le développeur soit en mesure d'appréhender aussi facilement les API qui n'ont pas été abordées.

L'avantage d'une refonte complète d'une plate-forme de programmation est qu'elle gagne ainsi en cohérence. Contrairement à la vénérable API Win32, dont chaque sous-ensemble est le reflet de l'époque à laquelle il a été ajouté, l'API WinRT est homogène et le lecteur de cet ouvrage ne se sentira jamais perdu lorsqu'il abordera un nouveau concept.

Copyright © 2013 Eyrolles.

Enfin, nous avons insisté sur le respect des bonnes pratiques. Ceci s'applique à la fois à la vision externe des applications, mais aussi à leur implémentation. L'aspect externe est fondamental, car il est au contact de l'utilisateur qui abandonnera sans pitié toute application trop compliquée à utiliser ou peu réactive car, dans de nombreux cas, le Windows Store proposera d'autres applications aux fonctionnalités assez proches… Les évaluations laissées par des utilisateurs mécontents risquent d'hypothéquer fortement l'avenir d'une application. L'aspect interne ne doit pas être négligé pour autant, car le développeur y gagnera au final en termes de productivité, surtout lorsqu'il devra mettre son application à jour pour ajouter une fonction rendue indispensable par l'apparition de nouveaux services externes ou de nouvelles applications concurrentes. Rien n'est plus difficile que de se replonger dans un code mal ficelé après quelques mois d'éloignement.

Bon développement !

Alain, Louis-Guillaume et Luc

Copyright © 2013 Eyrolles.

Index

.NET 15

A

accès aux données 73
accès aux données distantes 81
activation d'une application suspendue 117
ADO.NET 73, 81
animation
  dépendante 182
  état visuel (XAML) 184, 185, 187
  indépendante 181
  libre 180
  liste animée 178
  rôle 172
  storyboard (XAML) 182
  transition CSS 190, 191
  transition de contenu 175
  transition de thème (XAML) 173
  type 173
API de lancement d'application 166
application Bureau 6, 17, 30
approche orientée objet 52
association application
  charger le fichier 162
association applicative 160, 219
  de fichier 160
  de protocole 164
  utiliser le protocole 165
async 96
asynchronisme 89

B

badge
  mise à jour 129
bibliothèque d'interface 82
Blend 19

C

C#
  sauvegarde de l'état 111
C++ 16
  liaison de données 64
  sauvegarde de l'état 115
capteur 211
charme
  paramètres 156, 157, 158
  partage 140, 144, 146, 147
  périphériques 216
  recherche 150, 153, 155
composant COM 12, 262
composant tiers 20
concept de navigation 33
  hiérarchique 34
  hybride 35
  linéaire 34
contexte local 14
contrat 139, 219
  paramètres 140
  partage 140
  PlayTo 140
  recherche 140
  sélection de fichier 140
cycle de vie 6, 105, 106

D

DataBinding 52
DataTemplate 62
délai de sauvegarde 107
déploiement interne 8
design language 27
déverrouillage développeur 9
données applicatives 237
  itinérantes 238

Copyright © 2013 Eyrolles.

E

écran d'accueil 106
écran type
  page 37
  panoramique 39
élément d'interface 40
  barre d'applications 44
  icône 44
  vignette dynamique 40
ergonomie 25
état d'affichage 31
  ancré 32
  plein écran 31
  remplissage 32
expérience utilisateur 6

G

géolocalisation 210
  accès continu aux coordonnées 209
  accès ponctuel aux coordonnées 207

H

HTML5 13

I

IAsyncOperation 93
interface avec les capteurs 205
  géolocalisation 205
interface tactile 193, 194
  détecter les capacités machine 197
  événement de manipulation (XAML) 200
  GestureRecognizer 198
  mouvement 198
  pointeur 195
interface utilisateur
  réactive 89
Internet 237
  accéder aux données d'un service mobile 256
  créer un service mobile 255
  service HTTP 246
  service REST 249
  service standard SkyDrive 251
  service WCF 251
  services mobiles Windows Azure 254

J

JavaScript 22
  sauvegarde de l'état 114

L

langage d'interaction tactile 35
  manipulation 36
  mouvement 35
langage visuel 47
lecture/écriture dans un fichier 74
  fichier texte 76
  fichier XML 78
  FileIO 78
liaison de données 52, 96
  HTML5 68
  liaison de listes 61
  liaison simple 55
LINQ 86

M

Media Center 27
méthode longue
  durée 93
modèle EDMX 86
Modern UI 6, 26, 123
MSDN 19
ms-view-state 191
MVVM 54

N

NDEF (NFC Data Exchange Format) 213, 216
NFC (Near Field Communication) 216
NFC (Near Field Communication) 213
  envoyer/recevoir des informations 215
notification 42
  vignette 43

O

OData 86

P

performance 10
périphériques
  événement de 220
  gérer 217
  interagir 218

Copyright © 2013 Eyrolles.

lister les périphériques disponibles 217
philosophie de design 26, 27
plate-forme de développement 11
pool de threads 101

R
reprise des données 116
requête asynchrone
C++ 103
callback 87
JavaScript 102
ressources machine 14
REST 86

S
sandbox 17
sauvegarde d'état 109
scénario d'utilisation 30
SDK Windows 8 18
sécurité 9, 150, 240
sérialisation 242
JSON 79
XML 79
service Internet personnalisé 241
service WCF simple 81
socket 242
suspendu 7, 108, 109
SuspensionManager 112

T
tâche 99

V
vignette dynamique 123
décrire en XML 128
mettre à jour 128
organisation 125
taille 128
transmettre le contexte 132
vignette secondaire 130
Visual Studio 19

W
WCF Data Services 86
Windows Phone 27
Windows Push Notification Service 133
Windows Runtime 12, 92, 259
composant WinRT 261, 264
fabrique de composants WinRT 270
métadonnées 267
modèle de threading 266
types WinRT 273
Windows Store 8, 223
augmenter la visibilité 233
découvrir une application 224
enregistrement et validation 227
gérer les applications publiées 235
licence développeur 226
localiser les captures d'écrans 231
manifeste d'une application 227
mode release 229
noter l'application 233
publier une application 226
règles à respecter 232
soumettre une application 229
télécharger 234, 235
Windows Store vs Bureau 30
Windows 8 10
Windows RT 11
WWAHost 13

X
XAML 20
liaison de données 55

Z
zoom sémantique 36, 46
Zune 27

Copyright © 2013 Eyrolles.