

Silverlight 2

G é r a r d L e b l a n c

Préface de Christophe Lauer



EYROLLES

Silverlight

2

G. LEBLANC. – **C# et .NET. Version 2.**

N°11778, 2006, 854 pages.

E. SLOÏM. – **Mémento Sites web. Les bonnes pratiques.**

N°12101, 2007, 14 pages.

S. POWERS. – **Débuter en JavaScript.**

N°12093, 2007, 386 pages.

J.-M. DEFRAÏCE. – **Premières applications Web 2.0 avec Ajax et PHP.**

N°12090, 2008, 450 pages.

R. GOETTER. – **CSS2. Pratique du design web.**

N°11976, 2007, 310 pages.

T. TEMPLIER, A. GOUGEON. – **JavaScript pour le Web 2.0.**

N°12009, 2007, 492 pages.

C. PORTENEUVE. – **Bien développer pour le Web 2.0. Bonnes pratiques Ajax.**

N°12028, 2007, 580 pages. *Nouvelle édition à paraître.*

M. PLASSE. – **Développez en Ajax.**

N°11965, 2006, 314 pages.

M. NEBRA. **Réussir son site web avec XHTML et CSS.**

N°11948, 2007, 306 pages.

F. DRAILLARD. – **Premiers pas en CSS et HTML. Guide pour les débutants.**

N°12011, 2006, 232 pages.

R. GOETTER. – **Mémento CSS.**

N°11726, 2006, 14 pages.

R. GOETTER. – **Mémento XHTML.**

N°11955, 2006, 14 pages.

J. ZELDMAN. – **Design web : utiliser les standards. CSS et XHTML.**

N°12026, 2^e édition 2006, 444 pages.

H. WITTENBRIK. – **RSS et Atom. Fils et syndications.**

N°11934, 2006, 216 pages.

T. ZIADÉ. – **Programmation Python. Syntaxe, conception et optimisation.**

N°11677, 2006, 530 pages.

J. PROTZENKO, B. PICAUD. – **XUL.**

N°11675, 2005, 320 pages.

E. DASPET et C. PIERRE de GEYER. – **PHP 5 avancé.**

N°12004, 3^e édition 2006, 804 pages.

G. PONÇON. – **Best practices PHP 5. Les meilleures pratiques de développement en PHP.**

N°11676, 2005, 480 pages.

R. RIMELÉ. – **Mémento MySQL.**

N°12012, 2007, 14 pages.

Silverlight 2

G é r a r d L e b l a n c

Préface de Christophe Lauer

EYROLLES



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



Le code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée notamment dans les établissements d'enseignement, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

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 autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2008, ISBN : 978-2-212-12375-3

Préface

Cela fait maintenant une bonne quinzaine d'années que je travaille dans le domaine de l'informatique, et donc une bonne quinzaine d'années que je suis avec passion les évolutions de l'industrie du logiciel.

Avec quinze à vingt ans de recul, on constate une alternance assez régulière de modèles d'architectures applicatives destinées à s'exécuter tantôt sur un serveur, tantôt sur le client.

Historiquement, les grandes applications métier étaient écrites pour fonctionner sur de gros serveurs d'entreprise – les *mainframes* – auxquels étaient raccordés plusieurs terminaux passifs via un réseau local. Au début des années 1990, est survenue une première transformation du paradigme à travers le modèle nommé client-serveur. Capitalisant sur la puissance de traitement encore inexploitée des PC, cette architecture applicative rompait avec la logique existante qui limitait alors l'usage des PC à de simples postes dédiés à la bureautique. Cette évolution était également marquée par une amélioration de l'ergonomie et de la réactivité des interfaces utilisateurs, avec notamment l'utilisation de la souris et des contrôles visuels qui sont devenus notre quotidien.

À la fin des années 1990, a lieu la seconde transformation du paradigme avec la découverte d'Internet et de ses techniques et langages associés. Il devient rapidement évident qu'il faut aller au-delà de l'Internet proposant uniquement des contenus via des sites Web pour envisager le réseau des réseaux comme une plate-forme d'exécution pour de véritables applications en mode Web. Des technologies telles que ASP, PHP, le modèle J2EE et ASP.NET viennent répondre à ce nouveau défi.

Après, l'histoire s'accélère. Le réseau s'élargit, sort du cadre de l'entreprise ou d'un département, et s'étend rapidement à la planète entière en passant de débits modestes de 10 Mbits/s à des capacités de l'ordre du gigabit. En parallèle, les utilisateurs se familiarisent à une vitesse prodigieuse avec les outils et les interfaces issus de l'Internet jusqu'à trouver naturelle l'idée selon laquelle une application d'entreprise n'est pas obligée de posséder une interface utilisateur terne et austère pour être sérieuse et efficace pour ses usagers.

Les technologies évoluent en offrant des expériences utilisateurs toujours plus riches sur des interfaces Web : l'utilisation de code JavaScript exécuté sur le client, au sein du navigateur, marque en 2006 le début de l'épopée du Web 2.0 dans le grand public, dont Ajax est l'une des dimensions techniques indissociables. Les entreprises commencent alors à

s'intéresser à ces technologies RIA, non pas pour leur attrait visuel, mais parce que les améliorations en matière d'ergonomie et d'usabilité qu'elles apportent relèvent des derniers domaines qui recèlent des gisements de productivité non encore totalement exploités.

Cela étant, bien que très couramment utilisées sur le Web grand public, les interfaces d'applications réalisées intégralement en technologie Adobe Flash restent tout de même l'exception en entreprise. En effet, les développeurs et les architectes applicatifs demeurent généralement frileux face à cette technologie qu'ils estiment caricaturalement uniquement capable de réaliser des bannières de publicité animées pour les sites de e-commerce.

C'est plus récemment avec l'arrivée sur le marché de Adobe Flex que le sujet des applications RIA en entreprise a été relancé. C'est dans ce contexte que Microsoft a développée la technologie Silverlight, sa nouvelle plate-forme qui puise ses racines dans son aînée WPF et dans .NET.

Silverlight est une technologie qui capitalise sur le meilleur des deux mondes et qui permet de réaliser des interfaces clients d'applications de type « Software + Services », qui proposent un accès « sans couture » depuis les données publiées sur le réseau local de l'entreprise jusqu'aux services distants au travers de l'Internet, matérialisant le concept du « Web en tant que plate-forme » et du *Cloud Computing*.

Avec Silverlight 2, qui est le sujet de ce livre, cette vision architecturale s'est encore affirmée et la richesse des interfaces utilisateurs RIA, que la technologie permet de réaliser, n'a pas grand chose à envier aux meilleures interfaces d'applications Windows natives réalisées en WPF sur le client.

Silverlight 2 se distingue particulièrement par la facilité avec laquelle il permet de mélanger et d'utiliser tous les formats et tous les médias existants : texte, données XML, son, vidéo, animations... Cette capacité, combinée à une gestion intelligente du streaming et à des fonctionnalités uniques telles que le Deep Zoom, permet de créer des expériences utilisateurs infiniment riches et agréables capitalisant sur des interfaces immersives et intuitives. Toutes sortes de scénarios applicatifs sont imaginables depuis l'amélioration de la navigation et des catalogues produits dans des sites de e-commerce jusqu'aux domaines du jeu et de l'*entertainment*, en passant par la dynamisation de la présentation des tableaux de bord en entreprises, etc.

De plus, si l'on se place du point de vue du développeur, Silverlight a été pensé pour une prise en main rapide (à travers les deux familles d'outils Suite Expression et Visual Studio) et pour une ubiquité de déploiement (les applications Silverlight peuvent s'exécuter sur Windows et sur Mac OS X, et demain également sur Linux via le projet Open Source nommé Moonlight, dans les navigateurs Web Internet Explorer, Safari et Firefox).

Silverlight 2 est accessible à tout développeur ayant une petite expérience en .NET. Cet ouvrage utilise d'ailleurs C# et VB.NET pour ses exemples. Si vous cherchez un livre pour apprendre C#, sachez qu'il en existe justement un, du même auteur et chez le même

éditeur. En effet, Gérard Leblanc n'en est pas vraiment à son coup d'essai avec cet ouvrage, il a déjà signé de nombreux titres sur les langages C, C++ et C#. C'est d'ailleurs avec deux de ses livres que j'ai appris le C, il y a maintenant quelques années de ça ;)

Merci Gérard pour ta passion toujours renouvelée au cours de ces années, merci pour avoir rédigé ce livre qui va devenir – j'en suis certain – le livre de chevet de bon nombre de développeurs curieux d'apprendre à maîtriser cette plate-forme si prometteuse que nous apprécions tout particulièrement !

Christophe LAUER

Responsable des Relations avec les Agences Interactives, Microsoft France

Table des matières

CHAPITRE 1

Introduction à Silverlight 2 et installation du logiciel	1
Introduction à Silverlight 2	1
Installation du logiciel	7

CHAPITRE 2

Création d'une application Silverlight	9
Description de l'application	9
Démarrage du projet avec Visual Studio	10
Les fichiers de l'application	12
Déploiements ASP.NET et HTML	16
Insertion de l'image de fond	18
Insertion de la vidéo	22
Le travail en couches transparentes	25
Première animation : le texte déroulant	26
Zone d'édition et bouton dans Silverlight	30
Image dans bouton	31
On tourne la page...	33
Accès au serveur d'images Flickr	38

CHAPITRE 3

Les conteneurs	47
Les conteneurs de base	47
Le canevas.	48
Le StackPanel	50
La grille.	53
Conteneurs spécifiques	61
Le GridSplitter	61
Le composant ScrollViewer	64
Le composant Border	65

CHAPITRE 4

Couleurs et pinceaux	67
Les couleurs	67
Un nom de couleur	68
Les représentations sRGB et scRGB	69
Les couleurs dans le code	70
Les pinceaux	71
Le pinceau SolidColorBrush	71
Le pinceau LinearGradientBrush	72
Le pinceau RadialGradientBrush	75
Le masque d'opacité	80
Les pinceaux dans Expression Blend	80
Utilisation d'Expression Blend	80
Créer un dégradé avec Expression Blend	84
Exemples d'utilisation des dégradés	87
Le bouton « gel »	87
L'effet « plastic »	88
L'effet « métal »	89
Le bouton « de verre »	90
L'effet d'ombre	90

CHAPITRE 5

Une première série de composants	93
Les rectangles et les ellipses	94
Les zones d’affichage (TextBlock)	98
Décomposer un texte en plusieurs parties	99
Les effets de relief	100
Les polices de caractères	102
Les polices fournies avec Silverlight	102
Afficher un texte dans une police non fournie avec Silverlight	103
Les zones d’édition (TextBox)	103
Les boutons	105
Créer un bouton dans Expression Blend	107
Les cases à cocher (CheckBox)	108
Les boutons radio (RadioButton)	108
Les boutons hyperliens	109
Le composant Slider	110
La barre de défilement (ScrollBar)	112
Le calendrier (Calendar)	113
Le composant DatePicker	114

CHAPITRE 6

Du code dans les applications Silverlight	117
Les événements	117
La notion d’événement	117
L’événement Loaded	118
Les événements liés à la souris	119
Comment signaler le traitement d’un événement ?	119
Comment détecter la position de la souris ?	122
Comment traiter le clic sur un bouton ?	123
Les événements liés au clavier	124

Le signal d'horloge (timer)	126
Exemple de traitement d'événement	128
La création dynamique d'objets	130
Comment modifier le contenu d'un conteneur ?	132
La création dynamique à partir du XAML	133
Programmes d'accompagnement	134

CHAPITRE 7

Les images, les curseurs et les vidéos	137
Les images	137
Les images en ressources	137
Les images qui ne sont pas en ressources	138
Taille des images et respect des proportions	139
Lire une image à partir du système de fichiers local	140
Le clipping d'image	142
Les images comme motifs de pinceau	143
Les curseurs	144
Les sons et les films	145
Deep Zoom	146
Tourner la page	155
Programmes d'accompagnement	158

CHAPITRE 8

Les figures géométriques	163
Line, Polyline et Polygon	163
Le Path	164
Les courbes de Bézier	167
Le Stroke	172
Dessiner avec Expression Blend	174
Programmes d'accompagnement	178

CHAPITRE 9

Les transformations et les animations	181
Les transformations	181
Forcer une transformation par programme.	188
Les animations	191
Les animations From-To	191
Les animations par key frames.	195
Les animations avec Expression Blend	197
Programmes d'accompagnement	205

CHAPITRE 10

Les liaisons de données	209
Les liaisons de données avec TextBlock et TextBox	209
Les boîtes de liste	213
Les propriétés des boîtes de liste	213
Remplir et modifier le contenu d'une boîte de liste	215
Retrouver l'article sélectionné	216
Cas des données provenant d'une liste d'objets.	217
La personnalisation des boîtes de liste	219
La grille de données	221
Présentation générale	221
Modifications simples de présentation de la grille	224
Assurer le suivi des données entre la source et la grille.	225
Définir ses propres colonnes.	225
Les templates de colonnes	226
Éditer le contenu d'une colonne.	227
Déterminer la ou les rangées sélectionnées	227
L'événement LoadingRow	227
L'événement CommittingEdit	228

CHAPITRE 11

L'accès aux fichiers	229
Le stockage isolé avec IsolatedStorage	229
La zone de stockage isolé	229
Stocker simplement des informations élémentaires	230
Le système de fichiers du stockage isolé	230
Localisation de la zone de stockage isolé	235
Lire des fichiers distants	236
Lire des fichiers locaux	237
Les fichiers en ressources	238

CHAPITRE 12

Accès XML avec Linq	239
Chargement du fichier XML	241
Les espaces de noms	242
Cas pratiques	242
Retrouver les noms des personnages	243
Retrouver les prénoms des personnages	244
Détection si une balise contient une ou plusieurs balises	244
Retrouver les attributs d'une balise	245
Amélioration du select	245
Convertir le résultat d'une recherche en un tableau ou une liste	245
Création d'objets d'une classe à partir de balises	246
Les contraintes et les tris	247

CHAPITRE 13

Accès à distance aux données	249
Les accès distants	249
L'objet WebClient	250
Application à la lecture d'une image	250
Application à la lecture d'un fichier XML	252

Application à un service Web météo	254
Application au service Web Flickr	260
CHAPITRE 14	
Les contrôles utilisateurs	263
Création d'un contrôle utilisateur	263
Traitement d'événements liés à un contrôle utilisateur	267
Utilisation d'un contrôle utilisateur	268
Contrôle utilisateur DLL	269
CHAPITRE 15	
Les styles et les templates	271
Les styles	271
Pinceaux et couleurs en ressources	273
Les templates	273
Donner un feedback visuel	279
Les styles des composants Silverlight de Microsoft	282
Modifier n'importe quel contrôle avec Expression Blend	282
Modification de l'apparence	282
Modification des transitions	288
CHAPITRE 16	
Interaction Silverlight/HTML	295
Blocs Silverlight dans une page Web	295
Accès aux éléments HTML depuis Silverlight	298
Modifier les attributs de la page	299
Accéder aux éléments HTML	299
Attacher des événements	302
Appeler une fonction JavaScript	303
Appeler une fonction C# à partir de JavaScript	303
Animation Flash dans une page Silverlight	304

ANNEXE

C#, VB et Visual Studio pour le développement Silverlight 2. 307

Index 327

Introduction à Silverlight 2 et installation du logiciel

Introduction à Silverlight 2

On peut faire remonter la naissance d'Internet en 1991, année où Timothy Berners-Lee et Robert Cailliau ont conçu et mis au point, à l'usage interne des scientifiques du CERN (Centre européen pour la recherche nucléaire), un système d'accès à des documents. Ce système, qui devait être indépendant des machines utilisées, était basé sur un protocole qu'ils ont appelé HTTP (règles de communication pour appeler et obtenir une page HTML), où HT signifie *Hyper Text*. Les documents devaient être formatés à l'aide de balises conformes à la norme qu'ils ont appelé HTML (règles et significations de ces balises), où HT signifie également *Hyper Text*. Tout tournait donc autour de ces deux lettres « HT » : du texte avec des liens, dits « hyperliens », pour passer d'un document à un autre.

Il a ensuite fallu peu de temps pour que cette innovation technologique sorte des laboratoires du CERN et fasse une entrée en force – avec le succès que l'on connaît – dans les milieux académiques et les grandes entreprises tout d'abord, puis parmi le grand public peu de temps après.

En 1994, Netscape version 1 est le premier navigateur à apparaître sur le marché. Il est basé sur le navigateur Mosaïc (développé dans un centre de recherche américain, le NCSA (*National Center for Supercomputing Applications*), le premier à intégrer les images dans le texte (elles étaient auparavant affichées dans une autre fenêtre).

À cette époque, Internet donne satisfaction pour la consultation de documents et même de catalogues commerciaux éparpillés dans le monde entier, tout cela sans que l'utilisateur

ait à se préoccuper de leur localisation. Cependant, il n'en va pas de même lorsqu'il s'agit de passer commande... En 1995, le concept de CGI (*Common Gateway Interface*) est alors introduit, technique qui permet de transmettre des données (nom, adresse de livraison et surtout numéro de carte bancaire) au serveur, chez l'hébergeur du site Web : l'utilisateur clique sur un bouton généralement libellé *Submit* (sur la machine de l'utilisateur donc) et un programme est exécuté sur le serveur. Même si cette programmation s'avère peu performante (un nouveau programme s'exécute à chaque requête d'un client), la technique a le mérite d'exister et offre de très encourageantes perspectives pour le commerce électronique. Pour que la réussite soit totale, il reste à améliorer le système et surtout à attirer le chaland !

En 1996, Netscape introduit un interpréteur dans son navigateur. Celui-ci, d'abord baptisé LiveScript, lit des instructions écrites en un langage dérivé du langage C mais moins puissant et nettement plus laxiste, sous couvert de simplicité. Le but est de permettre un traitement local, sur la machine de l'utilisateur. LiveScript est ensuite rebaptisé JavaScript, uniquement par opportunisme, pour profiter de la vague Java, le tout nouveau langage à la mode à cette époque. JavaScript représente donc une première tentative pour « déporter » des tâches dans le navigateur, sur la machine de l'utilisateur. Mais dans les faits, il faut bien reconnaître que JavaScript est surtout utilisé pour réaliser des effets de survol (par exemple, une image qui change lors du passage de la souris). Même normalisé en ECMAScript, JavaScript est certes adopté par tous les navigateurs dignes de ce nom mais cela se fait avec très peu de coordination, surtout dans la manière d'accéder et de manipuler par programme les différents éléments de la page HTML. Heureusement, le comité de normalisation W3C met un peu d'ordre dans tout cela en faisant adopter une norme pour régir ce que l'on appelle le DOM, c'est-à-dire le *Document Object Model* (méthodes d'accès et de manipulation des éléments HTML de la page). Cette norme nous influence encore, même dans Silverlight (il est en effet possible en Silverlight de manipuler des éléments traditionnels du HTML).

Les pages Web se répandent ensuite de manière phénoménale dans le grand public, qui fait preuve d'une belle constance dans sa demande incessante de nouveautés. À partir de 1996, on commence à voir apparaître des pages Web agrémentées d'animations grâce à Flash (d'abord Shockwave Flash, puis Macromedia Flash et aujourd'hui Adobe Flash). Si les graphistes sont alors séduits par Flash (et le sont souvent encore), son modèle de programmation suscite moins d'enthousiasme de la part des programmeurs. À quelques exceptions près (qui peuvent être remarquables d'ailleurs), Flash reste essentiellement un outil pour graphistes et ne bénéficie pas d'un réel support des équipes de développement. La brèche et l'angle d'attaque n'échappent manifestement pas à Microsoft...

En 2002, Microsoft introduit la technologie ASP.NET qui permet de préparer sur un serveur des pages Web (constituées exclusivement de HTML et de JavaScript) et de répondre (sur le serveur) à des événements déclenchés dans le navigateur, c'est-à-dire chez le client (par exemple, un clic sur un bouton mais pas seulement). ASP.NET, tout comme la technologie similaire et concurrente PHP (*Hypertext Preprocessor*), fait partie de ce que l'on appelle la « programmation serveur », généralement avec accès (à partir du serveur uniquement puisque presque toute l'intelligence est sur le serveur) à une base

de données se trouvant sur le même serveur ou sur un autre, dédié à la base de données. Les solutions basées sur ASP.NET et PHP sont bien adaptées au commerce électronique mais elles sont incapables de rivaliser avec les applications Windows en termes d'interactivité et de convivialité.

Pour pallier cette rigidité, Ajax apparaît alors. Il s'agit d'une technologie qui consiste à « déporter » sur le client, mais d'une manière qui reste quand même assez limitée et toujours basée sur JavaScript, des fonctions qui s'exécutaient auparavant sur le serveur (avec un impact sur les performances puisque celui-ci traite simultanément un grand nombre d'applications Web utilisées par un très grand nombre de clients).

L'histoire ne fait guère de mystère quant à la direction qu'il convient de prendre...

En 2007, Microsoft introduit Silverlight version 1, lequel est encore basé sur JavaScript (grosse déception...) mais permet d'écrire des pages Web intégrant essentiellement les animations (comme Flash) et la vidéo.

En 2008, avec Silverlight version 2, Microsoft met en marche une machine à gagner. Le modèle de programmation est cette fois basé sur .NET, l'environnement de développement de programmes bien connu des programmeurs Windows pour sa facilité d'utilisation et son incomparable puissance dans le développement d'applications. Ce que l'on appelle le framework .NET regroupe un ensemble de classes (autrement dit du code pré-écrit par Microsoft et distribué sous forme d'un *run-time* qui doit être en exécution sur la machine hôte). Ces classes facilitent considérablement la tâche des développeurs d'applications, qui n'ont plus qu'à assembler des briques logicielles. Le framework .NET constitue une couche logicielle autour de Windows, laquelle n'intéresse pas directement les utilisateurs (en pratique, sa présence ne change rien pour eux) mais facilite grandement le développement et l'exécution des programmes mis à leur disposition.

Le portage à un autre type de plate-forme de l'environnement d'exécution de programmes .NET n'a rien de nouveau. En 2003 déjà, Microsoft avait porté .NET (en réduisant drastiquement sa taille) sur les appareils mobiles que sont les Pocket PC et les smartphones, ces GSM tournant sous une version particulière de Windows. Les développeurs d'applications pour mobiles retrouvaient ainsi leur(s) langage(s) de prédilection, les mêmes outils et les mêmes classes (à quelques restrictions près) que pour les programmes Windows. Et par conséquent la même efficacité.

Avec Silverlight 2, Microsoft porte ce modèle de développement qui a fait ses preuves aux pages Web. Moyennant l'installation sur la machine des clients de ce que l'on appelle le run-time Silverlight (Flash ne procède pas autrement), on peut bénéficier de pages Web qui offrent les fonctionnalités (à quelques restrictions près, notamment pour des raisons de sécurité) et la convivialité tant appréciées dans les applications Windows.

Le run-time existe pour Windows (Vista et XP ainsi que 2000 prochainement), pour Mac ainsi que pour Linux, grâce au projet Moonlight de Novell, mené en partenariat avec Microsoft. Il est compatible Internet Explorer, Firefox et Safari. On peut donc affirmer que les applications Silverlight sont indépendantes des navigateurs et des systèmes d'exploitation. Un run-time Silverlight n'est cependant pas disponible pour les mobiles,

même si des accords (notamment avec le numéro 1 mondial) ont été conclus en vue de son portage à ce type d'appareils.

Une application Silverlight peut être déployée sur n'importe quel serveur (par exemple, Apache), sans qu'il soit nécessaire d'y installer des modules additionnels ou d'adopter une configuration spéciale (ce n'est pas aussi simple pour les applications ASP.NET, qui nécessitent IIS de Microsoft ou le module additionnel « mono » sur Apache).

Lors du développement d'applications Silverlight, le développeur retrouve les mêmes outils, les mêmes langages (C# et Visual Basic – VB.NET – mais aussi Python ou Ruby), les mêmes classes (avec quelques restrictions) et la même façon de développer qu'en programmation Windows.

Un programmeur .NET est donc très rapidement opérationnel pour créer des applications Web sous Silverlight version 2. Quand on connaît les coûts de développement d'un logiciel, le gain est appréciable.

Les applications Silverlight 2 sont donc écrites (essentiellement) en C# et VB.NET, les deux langages phares de l'environnement .NET (surtout C# qui supprime maintenant C++, en perdition). Par rapport à JavaScript, l'intérêt est considérable pour le développeur du fait de la puissance de ces deux langages (qui sont orientés objet) et surtout grâce à l'apport des classes .NET.

Lors de l'exécution d'une application Silverlight 2, les choses se passent de la manière suivante : les instructions C# ou VB ont été compilées (lors du développement de l'application) en un code binaire appelé CIL (*Common Intermediate Language*). Ce code binaire (indépendant de la plate-forme – Windows, Mac ou Linux) est envoyé au navigateur du client (c'est en fait le HTML qui réclame ce code binaire, voir chapitre 2), puis exécuté par le run-time Silverlight (ce qui, en termes de vitesse d'exécution, s'avère nettement plus performant que la technique d'interprétation en vigueur pour JavaScript).

Au chapitre 2, nous présenterons une application dans son intégralité et nous expliquerons par quel mécanisme le navigateur (inchangé pour Silverlight et qui pourtant n'interprète que les balises HTML et le JavaScript) en vient à faire exécuter ce code binaire.

Avec Silverlight, l'interface utilisateur est décrite en XAML (*eXtensible Application Markup Language*), un formalisme (basé sur XML) de description de page. XAML de Silverlight est un sous-ensemble du XAML déjà créé par Microsoft pour WPF (*Windows Presentation Foundation*), la nouvelle technologie de description des pages Windows. Avec XAML, le développeur (programmeur ou graphiste) décrit non seulement la page avec ses différents composants mais également les animations.

Silverlight est vraiment conçu pour que programmeurs et graphistes travaillent en parfaite collaboration : Visual Studio (l'outil des programmeurs) et Expression Blend (celui des graphistes) opèrent sur les mêmes fichiers et toute modification effectuée à l'aide de l'un d'eux est automatiquement reconnue et prise en compte par l'autre.

Que peut-on faire avec Silverlight ?

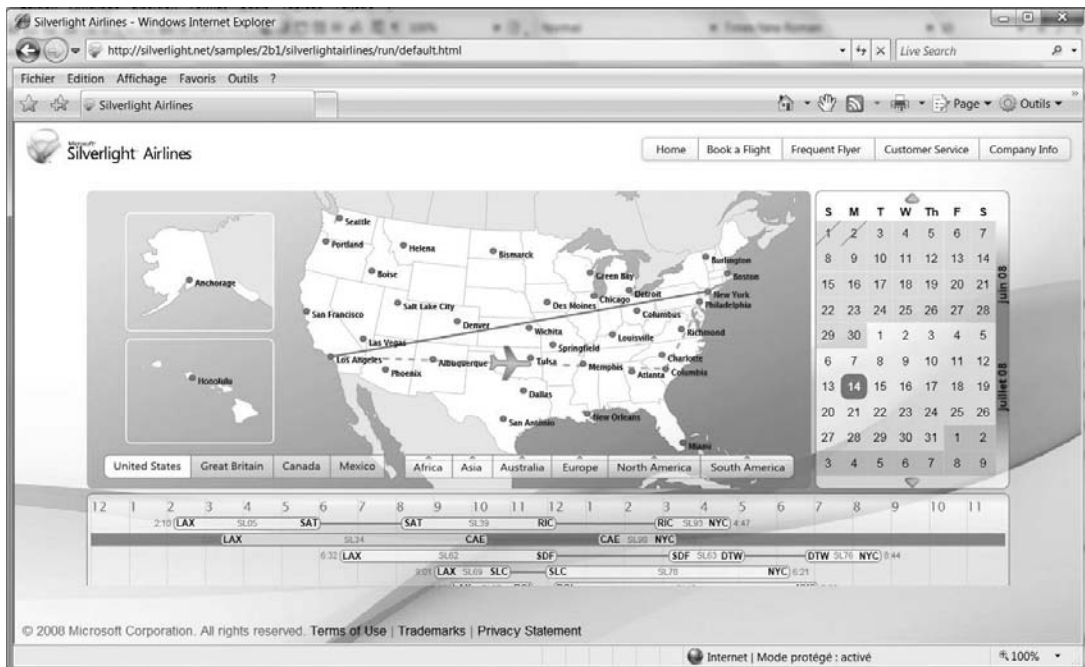
La réponse à cette question est simple : Silverlight rend possible le développement d'applications Web au moins aussi interactives et conviviales que les applications Windows. La palette des composants Silverlight 2 dans Visual Studio (boutons, zones d'édition, boîtes de listes, grilles de données, calendriers, etc.) est aussi impressionnante que celle en programmation Windows. Le modèle de programmation événementielle est aussi le même.

Mais l'avantage des applications Silverlight réside dans le déploiement : nul besoin de distribuer CD ou DVD, nul besoin d'installer (souvent avec des droits d'administration) des logiciels préalablement téléchargés et nul besoin d'avoir le framework .NET installé sur la machine (bien que le run-time Silverlight le soit). De plus, avec une application Web, l'utilisateur dispose toujours des dernières mises à jour : il suffit à l'administrateur du site Web de les installer sur le serveur pour que les utilisateurs puissent en bénéficier immédiatement.

Même si le framework .NET est installé (ce qui n'est possible que sur les machines Windows), le run-time Silverlight doit également l'être pour que les applications Silverlight puissent être exécutées. Les deux run-times n'interfèrent pas entre eux mais ne collaborent pas non plus. Comme nous l'avons déjà mentionné, le run-time Silverlight n'est plus limité aux ordinateurs sous Windows puisque des versions Mac et Linux existent.

De nombreuses applications Silverlight sont en démonstration sur le site <http://www.silverlight.net>, rubrique *Showcase*. La figure 1-1 illustre, par exemple, la nouvelle manière de réserver un vol en quelques clics :

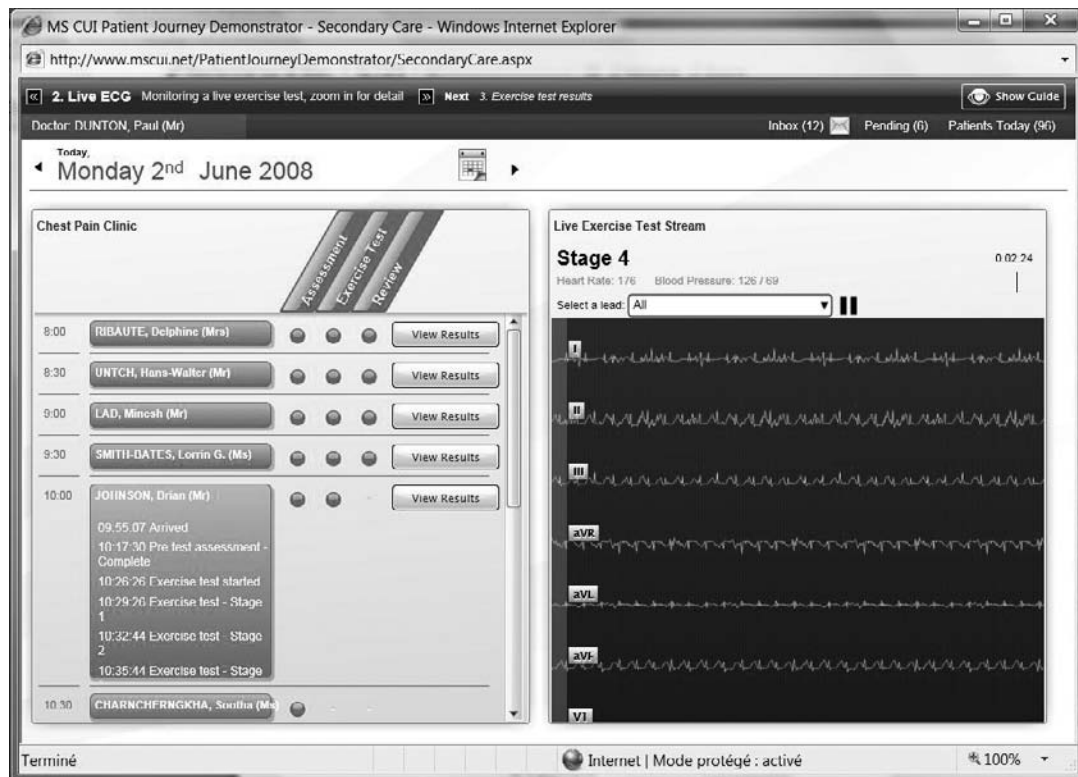
Figure 1-1



L'interface utilisateur est sans comparaison avec ce qui était disponible auparavant. Quant à l'accès aux données, on retrouve toutes les possibilités offertes par les programmes Windows, notamment grâce aux services Web.

Vous pourrez également visualiser une démonstration de gestion intégrée dans le domaine médical (figure 1-2).

Figure 1-2



Tout au long de cet ouvrage, nous aurons l'occasion de passer en revue les étonnantes possibilités de Silverlight version 2.

Microsoft autorise le déploiement d'applications Silverlight 2 depuis la sortie de Silverlight 2 bêta 2, sous licence « go-live », ce qui n'implique qu'une seule obligation : effectuer la mise à jour dès la sortie officielle de Silverlight 2 ou retirer les pages. Aucune royalties ne sont et ne seront réclamées par Microsoft pour l'utilisation ou le déploiement de sites basés sur Silverlight 2.

Les hébergeurs n'ont aucune raison de vous réclamer des coûts supplémentaires puisque le fait de déployer des applications Silverlight 2 ne change strictement rien pour eux. Si vos simples pages HTML sont acceptées par votre hébergeur, vos pages Silverlight 2 le seront également.

Installation du logiciel

Le run-time Silverlight 2 est disponible pour Windows Vista et XP (Windows 2000 est prévu), Mac OS X ainsi que les navigateurs IE7 (et suivants), Firefox (y compris la version 3) et Safari.

Une version (mise au point en partenariat avec Novell, sans être développée par Microsoft) devrait être disponible prochainement (elle est déjà en démonstration) pour les ordinateurs sous Linux grâce au projet Moonlight de l'équipe Mono (<http://www.mono-project.com/Moonlight/>). Pour information, Mono est l'implémentation de .NET sous Linux.

L'installation du run-time est automatique, après acceptation par l'utilisateur. Il suffit que celui-ci visite une page Web épicée de Silverlight pour qu'il soit invité à installer le run-time Silverlight. Celui-ci occupe 4,6 Mo et l'opération dure moins d'une minute, sans nécessiter de droits d'administration ou de redémarrage de la machine.

Pour développer des applications Silverlight, vous devez tout d'abord être sous Windows. Téléchargez et installez Visual Studio 2008 en français (une version d'évaluation de 90 jours est disponible à l'adresse suivante : <http://silverlight.net/GetStarted>). Dès la sortie de la version officielle, il sera possible d'utiliser Visual Web Developer, qui fait partie de la gamme de produits Microsoft Express Edition et correspond à la version gratuite de Visual Studio.

Par ailleurs, vous devez également installer Microsoft Silverlight Tools Bêta 2 pour Visual Studio 2008, disponible à l'adresse suivante : <http://www.microsoft.com/downloads> (effectuez une recherche sur Silverlight). Téléchargez et installez la version française (il s'agit d'un fichier nommé `silverlight_chainer.exe`).

L'outil graphique Expression Blend est le compagnon (pour les graphistes) de Visual Studio. Vous le trouverez sur le site <http://www.silverlight.net/GetStarted>. À la même adresse, vous trouverez Deep Zoom Composer, que nous utiliserons (voir la section « Deep Zoom » du chapitre 7).

Même si vous êtes programmeur, cela vaut la peine d'installer au moins les deux premiers outils suivants (même en version d'évaluation) de la gamme Microsoft Expression (<http://www.microsoft.com/Expression/>) :

- Expression Design pour préparer des dessins professionnels ;
- Expression Media Encoder pour la vidéo ;
- éventuellement, Expression Web qui remplace FrontPage.

Vous êtes maintenant prêt à écrire une première application Silverlight déjà très complète. Tournez la page, cela se passe au chapitre 2.

2

Création d'une application Silverlight

Description de l'application

Au cours de ce chapitre, nous allons créer une application Silverlight qui :

- réalise plusieurs animations, à savoir un texte déroulant et un carrousel ;
- permet de tourner les pages d'un catalogue touristique au moyen de la souris ;
- diffuse en permanence une vidéo au centre de la fenêtre ;
- recherche des images sur le site Flickr (avec critères de recherche spécifiés par l'utilisateur) et les affiche sur un carrousel tournant.

La figure 2-1 représente l'application terminée et visualisée dans Firefox version 3.

Figure 2-1



D'autres éléments auraient pu être ajoutés, tels que des animations Flash (voir la section « Animation Flash dans une page Silverlight » du chapitre 16), qui ont le mérite d'être largement répandues depuis des années, ou encore la technologie Deep Zoom (voir la section « Deep Zoom » du chapitre 7) qui permet de zoomer sur des sujets de plus en plus précis.

Démarrage du projet avec Visual Studio

Pour réaliser l'application souhaitée, nous utiliserons la version française de Visual Studio 2008. À noter qu'avec la version finale de Silverlight, il sera possible d'utiliser Visual Web Developer (y compris en version française) qui est la version gratuite de Visual Studio. En attendant, vous pouvez télécharger et utiliser la version d'évaluation française de Visual Studio 2008 Pro (voir la section « Installation du logiciel » du chapitre 1).

Pour commencer, vous allez devoir créer un nouveau projet dans Visual Studio en spécifiant que vous utiliserez le C#. À noter, vous pourriez tout aussi bien utiliser Visual Basic (VB.NET). En effet, les deux langages présentent les mêmes possibilités et le choix de l'un ou de l'autre est affaire purement personnelle. Ceux qui ne seraient pas familiers avec l'un de ces deux langages, mais sans être pour autant néophytes en programmation, trouveront dans l'annexe tout ce qu'il faut savoir pour débiter la programmation Silverlight 2 dans l'un de ces deux langages. Dans la suite de l'ouvrage, les versions C# et VB seront toujours présentées conjointement.

Pour créer le projet de l'application, sélectionnez le menu Fichier>Nouveau>Projet>Silverlight>Application Silverlight (onglet C# dans notre cas). Indiquez le répertoire dans lequel il sera enregistré dans le champ Emplacement et nommez le projet, ici, NotreApplication (figure 2-2).

Visual Studio vous demande maintenant (figure 2-3) si vous créez une véritable application qui devrait être déployée sur un serveur (proposition par défaut), ou un petit projet de test.

Le deuxième choix ne se justifie que s'il s'agit vraiment de créer une petite application de test. Vous gagnerez alors de la place sur le disque et un peu (si peu) de temps lors des compilations.

Validez la proposition par défaut (premier choix), qui correspond à un cas pratique avec développement d'une application et, éventuellement à la fin, déploiement de l'application sur un serveur Internet pour mise à disposition du public. De plus, cela vous en apprendra bien plus sur le fonctionnement d'une application Silverlight.

En procédant ainsi, Visual Studio va créer deux répertoires (NotreApplication et NotreApplicationWeb) et deux projets dans la solution (une solution regroupe un ou plusieurs projets). Dans le répertoire NotreApplicationWeb, seront enregistrés les fichiers qu'il faudra copier sur le serveur (chez votre hébergeur si vous ne disposez pas de votre propre serveur Internet).

Figure 2-2

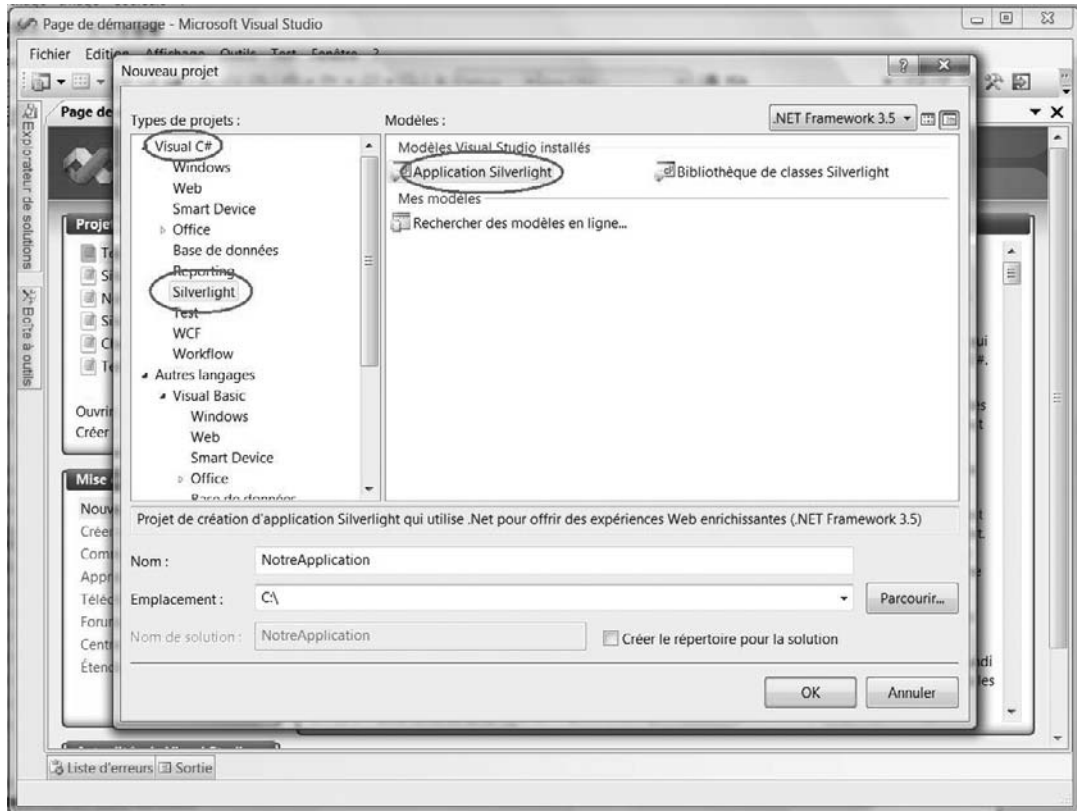
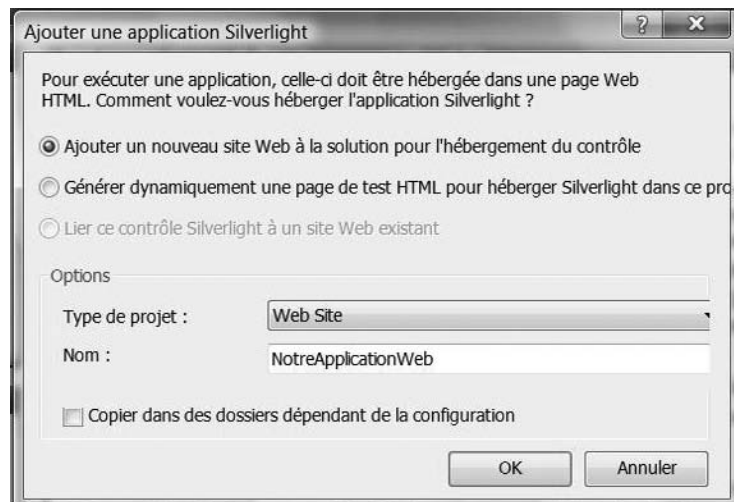


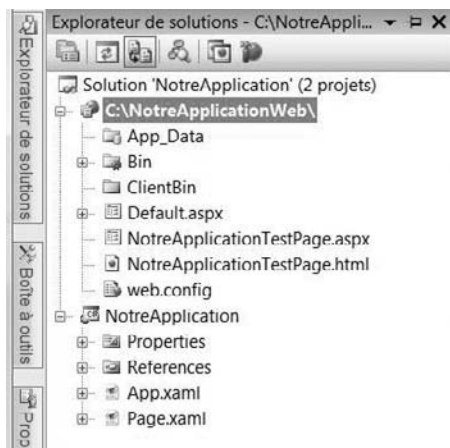
Figure 2-3



Côté hébergeur, aucun traitement spécial n'est à demander. Peu importe que le serveur d'accès à Internet tourne sous IIS (solution Microsoft) ou Apache (solution Linux). Si vos simples pages HTML sont acceptées, vos pages Silverlight le seront aussi.

La figure 2-4 représente l'Explorateur de solutions de Visual Studio tel qu'il apparaît après la génération des squelettes de fichiers de l'application (ces fichiers seront complétés durant la phase de développement de l'application Silverlight).

Figure 2-4



Les fichiers de l'application

Durant le développement de l'application, vous travaillerez essentiellement dans le projet `NotreApplication`, donc dans la partie Silverlight. Une fois l'application terminée, le déploiement (sur le serveur) se fera à partir des fichiers contenus dans `NotreApplicationWeb`, donc dans la partie Web du projet. Après compilation, des fichiers seront automatiquement copiés et mis à jour dans la partie Web du projet.

Les deux fichiers avec lesquels vous travaillerez le plus au cours du développement sont `Page.xaml` et `Page.xaml.cs`, mais examinons au préalable le contenu des différents fichiers générés par Visual Studio.

Visual Studio a créé dans le projet `NotreApplicationWeb` deux fichiers nommés par défaut `NotreApplicationTestPage.html` et `NotreApplicationTestPage.aspx`. Pour renommer ces fichiers à votre guise, cliquez droit sur l'un des noms et choisissez **Renommer**.

Les fichiers `NotreApplicationTestPage.html` et `NotreApplicationTestPage.aspx` correspondent à des déploiements différents. Pour un déploiement ASP.NET (avec fichier `.aspx`), le serveur doit être sous contrôle d'IIS (*Internet Information Server* de Microsoft). ASP.NET est très utilisé depuis quelques années pour des solutions dites de programmation serveur par les entreprises. Un déploiement HTML convenant parfaitement aux applications Silverlight, nous lui donnerons la préférence en raison de la plus grande facilité de déploiement qu'il

permet (un déploiement HTML est possible sur tous les types de serveurs, ce qui n'est pas le cas pour un déploiement ASP.NET).

Le fichier `Page.xaml` généré par Visual Studio contient la description de la page Web Silverlight. XAML (pour *eXtensible Application Markup Language*) est le « langage » de description de pages exécutées sous contrôle du run-time Silverlight. Il s'agit d'un formalisme XML avec des noms de balises et d'attributs bien particuliers. En général, les programmeurs préfèrent manipuler directement les balises XAML dans Visual Studio, alors que les graphistes préfèrent passer sous Expression Blend, l'outil graphique modifiant ou générant ces balises XAML. Les deux logiciels sont conçus pour un travail en collaboration : toute modification effectuée dans l'un est automatiquement détectée et prise en compte par l'autre. Dans ce chapitre, nous donnerons la préférence à la première possibilité (XAML sous Visual Studio) car vous en apprendrez ainsi bien plus. Tout programmeur Silverlight se doit de maîtriser le XAML (ce qui n'a vraiment rien de compliqué) même s'il lui arrive de passer à Expression Blend pour des opérations plus compliquées.

Au démarrage du développement d'une application, le fichier `Page.xaml` contient le code suivant :

```
<UserControl x:Class="NotreApplication.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">

    </Grid>
</UserControl>
```

Par défaut, Visual Studio, en générant ces balises, limite la page à un rectangle de 400 × 300 pixels. Supprimez les attributs `Width` et `Height` pour que la page Silverlight occupe toute la fenêtre du navigateur. Visual Studio a aussi créé une grille comme conteneur de la page Silverlight. Les composants (images, boutons, grilles de données, animations, etc.) seront insérés dans cette grille, ici avec un fond blanc (attribut `Background`). Les conteneurs et en particulier la grille, avec ses nombreuses possibilités, seront étudiés au chapitre 3.

Visual Studio génère également (dans les fichiers `Page.xaml.cs` ou `Page.xaml.vb` selon le langage retenu) un embryon de code avec les `using` (Imports en VB) les plus fréquemment utilisés. Il mentionne 11 espaces de noms (il s'agit de noms donnés à des regroupements de classes). C'est dans ce fichier (qui sera complété au fur et à mesure) que vous indiquerez le code à exécuter quand, par exemple, l'utilisateur clique sur un bouton. Pour le moment, il ne contient que le constructeur de la classe `Page`, qui est la classe de la page Silverlight.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace NotreApplication
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }
    }
}
```

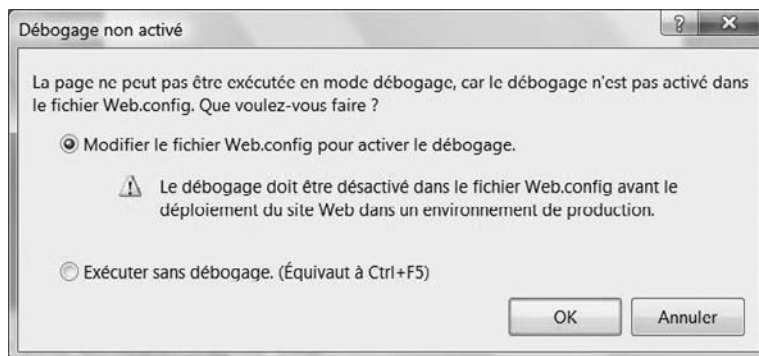
Ceux qui ont déjà pratiqué la programmation .NET pour le développement d'applications Windows se retrouvent en terrain connu avec les mêmes langages, les mêmes outils et quasiment les mêmes classes. Quant aux autres, nul doute qu'ils « entreront » avec une facilité déconcertante dans Visual Studio.

On peut déjà tester cette application, même si elle n'affiche encore qu'une page vierge. Pour lancer l'application, plusieurs techniques sont possibles :

- par le menu Débuguer>Démarrer le débogage, appuyer sur la touche F5 ou encore cliquer sur le petit triangle vert avec pointe vers la droite dans la barre des boutons ;
- par le menu Débuguer>Exécuter sans débogage pour une exécution sans contrôle du débogueur ;
- par l'Explorateur de solutions, cliquez droit sur le fichier .html ou .aspx puis sur Afficher dans le navigateur ou Naviguer avec... pour faire apparaître un autre navigateur que celui par défaut.

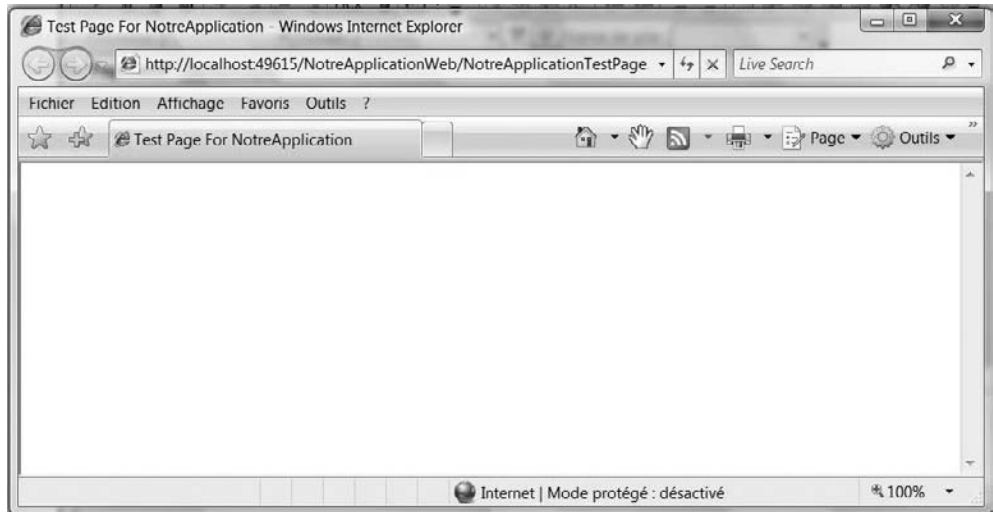
Lors de l'exécution par la touche F5 (en mode débogage, avec possibilité de placer des points d'arrêt), vous devez confirmer lors de la toute première exécution que le débogage de l'application est autorisé (ce qui nécessite une modification du fichier de configuration Web.config, figure 2-5).

Figure 2-5



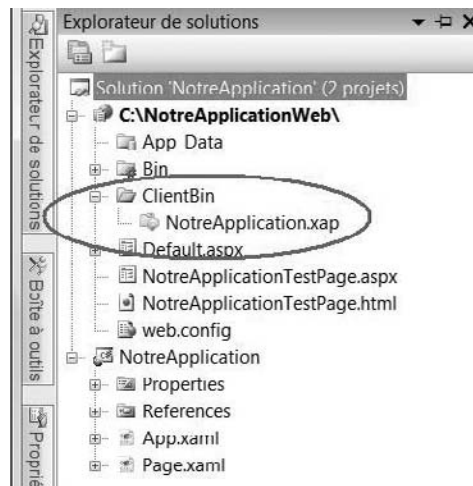
Vous obtenez alors une page Web vierge, ce qui est normal puisque aucun code n'a encore été saisi (figure 2-6).

Figure 2-6



Cette exécution a nécessité la compilation du programme. Un répertoire `ClientBin` a été créé dans la partie Web du projet, lequel contient un fichier d'extension `.xap` (figure 2-7). Ce fichier renferme le code binaire de l'application : il s'agit du résultat de la compilation de l'application (par le compilateur C# ou VB) en ce code binaire qu'on appelle CIL (*Common Intermediate Language*), code qui est indépendant de la plate-forme d'exécution de la page Web (Windows, Mac ou Linux).

Figure 2-7



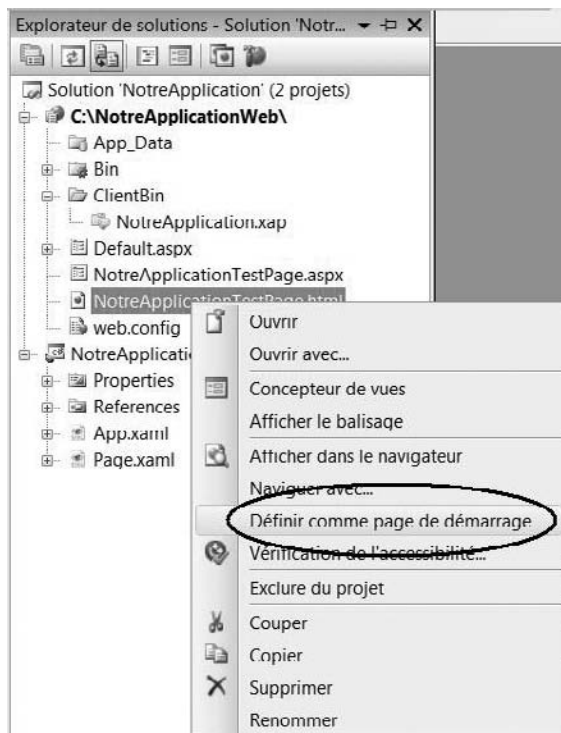
Une application Silverlight 2 est donc bien compilée et ce code est exécuté chez le client par le run-time Silverlight. Comme nous l'avons vu au chapitre 1, l'application détecte au démarrage si le run-time Silverlight est présent. Dans le cas contraire, elle propose à l'utilisateur de l'installer, ce qui prend moins d'une minute et ne réclame aucun droit d'administration de la machine.

Nous verrons bientôt par quel mécanisme le fichier XAP est appelé par le navigateur et exécuté sous contrôle du run-time Silverlight. À noter que le navigateur ne connaît pourtant que le HTML et le JavaScript et n'a pas été modifié pour Silverlight. Nous verrons également à cette occasion que le fichier XAP est en fait un fichier ZIP et qu'il contient, sous forme compressée, différents éléments.

Déploiements ASP.NET et HTML

Passons maintenant à l'examen des fichiers .aspx et .html. Par défaut, Visual Studio donne la priorité au déploiement ASP.NET, la solution Microsoft de programmation Web côté serveur. Pour modifier ce comportement, cliquez droit sur le nom du fichier HTML (dans la partie Web du projet) et sélectionnez Définir comme page de démarrage (figure 2-8).

Figure 2-8



Lors d'une exécution par la touche F5, c'est désormais le fichier HTML qui est pris en compte par le navigateur. En cours de développement, cette modification n'a d'importance que si vous modifiez un fichier (.html ou .aspx) sans effectuer la même modification dans l'autre. Pour le déploiement, c'est l'un ou l'autre fichier qu'il faudra copier sur le serveur Internet. Modifier les fichiers .html ou .aspx n'est vraiment nécessaire que dans des cas particuliers, qui seront présentés au chapitre 16.

Analysons maintenant le fichier .aspx généré par Visual Studio (fichier qu'il faudra copier sur le serveur, avec le contenu du répertoire ClientBin, en cas de déploiement ASP.NET) :

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Register Assembly="System.Web.Silverlight"
    Namespace="System.Web.UI.SilverlightControls"
    TagPrefix="asp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%;">
<head runat="server">
    <title>Test Page For NotreApplication</title>
</head>
<body style="height:100%;margin:0;">
    <form id="form1" runat="server" style="height:100%;">
        <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
        <div style="height:100%;">
            <asp:Silverlight ID="Xaml1" runat="server"
                Source="~/ClientBin/NotreApplication.xap"
                MinimumVersion="2.0.30523" Width="100%" Height="100%" />
        </div>
    </form>
</body>
</html>
```

Tout se passe dans la balise <asp:Silverlight> qui correspond à un composant écrit par Microsoft dans le cadre d'ASP.NET/Ajax/Silverlight. Facile à utiliser (une seule balise et quelques attributs seulement pour passer sous contrôle de Silverlight) mais malheureusement, cela nous cache le mécanisme de fonctionnement.

Analysons plutôt le fichier HTML qui nous en apprendra bien plus. Le HTML, tout à fait standard (rappelons que les navigateurs n'ont pas dû être modifiés pour Silverlight), comprend une balise object, largement utilisée depuis des années. Cette balise a été conçue à l'origine pour charger « quelque chose » dans le navigateur et demander à du code ainsi téléchargé d'assurer l'affichage de ce « quelque chose » (il s'agissait surtout de permettre la diffusion vidéo dans une page Web). Dans notre cas (voir la balise param avec source), « quelque chose » doit être téléchargé depuis ClientBin/NotreApplication.xap (cet emplacement étant relatif à celui du fichier HTML).

En toute logique, le fichier HTML est appelé en premier par le navigateur (c'est en effet ce fichier qui est mentionné dans l'URL saisie par l'utilisateur). Dès sa réception, ce fichier (de texte) est analysé ligne par ligne par le navigateur. Lorsque celui-ci arrive à la balise object, il appelle le fichier NotreApplication.xap du répertoire ClientBin :

```

<body>
.....
<div id="silverlightControlHost">
    <object data="data:application/x-silverlight,"
           type="application/x-silverlight-2-b2" width="100%" height="100%">
        <param name="source" value="ClientBin/NotreApplication.xap"/>
        <param name="onerror" value="onSilverlightError" />
        <param name="background" value="white" />
        <a href="http://go.microsoft.com/fwlink/?LinkId=115261"
           style="text-decoration: none;">
        
        </a>
    </object>
    <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
</div>
</body>

```

À ce stade du développement de l'application, la taille du fichier XAP est de 4 Ko (figure 2-9).

Figure 2-9



Nous verrons par la suite que la taille de ce fichier augmentera au fur et à mesure du développement de l'application tout en restant étonnamment faible. Ceci vient du fait que ce fichier contient le code compilé de l'application sous forme compressée. Celui-ci est exécuté directement chez le client, sans passer par une interprétation peu efficace, comme c'est le cas avec JavaScript.

Puisque nous en sommes à l'analyse des fichiers .html et .aspx, profitons-en pour modifier le titre de l'application (balise title dans la section head) :

```
<title>Une application Silverlight</title>
```

Insertion de l'image de fond

Après ces considérations générales mais indispensables, vous allez désormais pouvoir créer véritablement l'application.

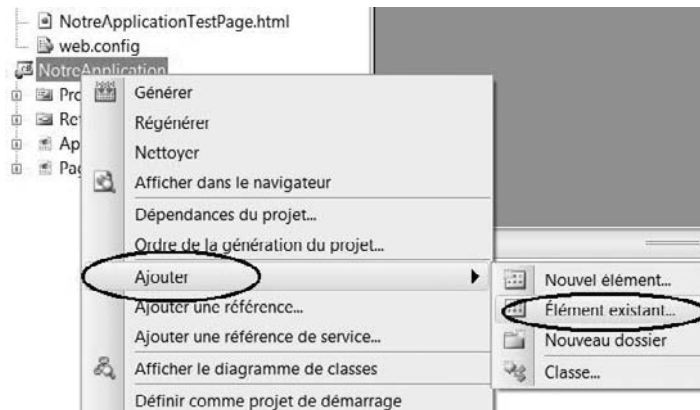
Pour cela, intéressons-nous au plus important des fichiers, à savoir Page.xaml. Ouvrez-le en double-cliquant dessus dans l'Explorateur de solutions. Du fait de la suppression précédente des balises Width et Height dans la balise UserControl1, le conteneur de l'application Silverlight est une grille qui s'adapte en permanence à la fenêtre du navigateur.

```
<UserControl x:Class="NotreApplication.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <Grid x:Name="LayoutRoot" Background="White">

    </Grid>
</UserControl>
```

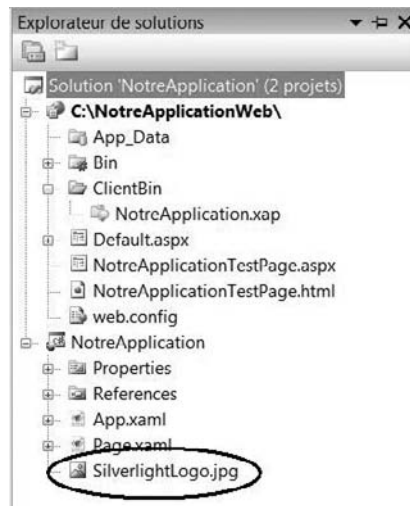
Ajoutons à présent une image (ici, le logo Silverlight) en fond de grille. Pour cela, vous devez charger l'image dans le projet (partie Silverlight) pour qu'elle soit intégrée en ressource de l'application (l'image de fond devra toujours être transmise au navigateur) : cliquez droit sur le nom du projet (partie Silverlight) et sélectionnez Ajouter>Élément existant (figure 2-10).

Figure 2-10



Recherchez ensuite l'image dans le système de fichiers. Elle est copiée dans le répertoire du projet (partie Silverlight) et apparaît dans le projet (figure 2-11) :

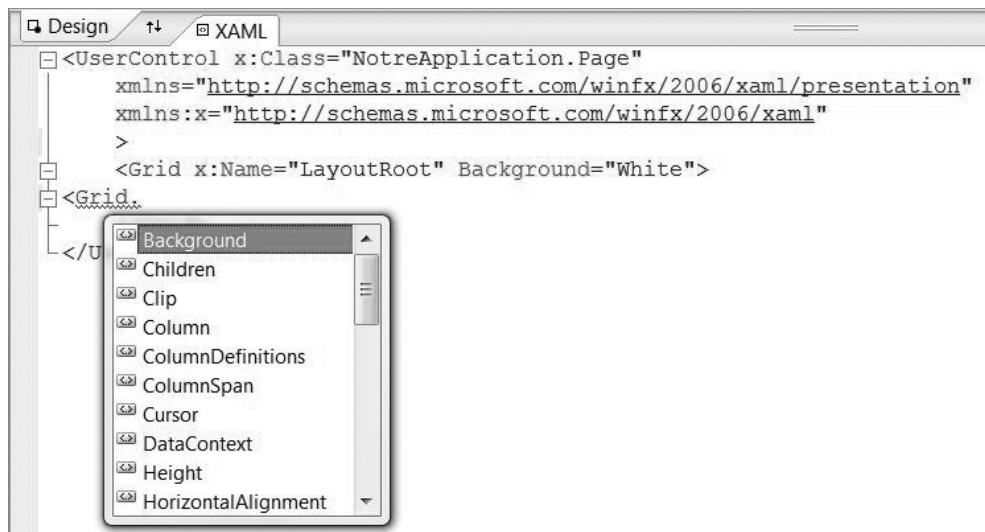
Figure 2-11



Pour le moment, le fond de la grille est blanc mais il va être remplacé par l'image. Pour cela, supprimez l'attribut `Background` et remplacez-le par une balise `Background` (de grille) plus complexe. Il s'agit d'une balise XAML.

Nul besoin de passer des heures à assimiler la syntaxe des balises XAML, l'aide contextuelle de Visual Studio rend les choses très intuitives (figure 2-12).

Figure 2-12



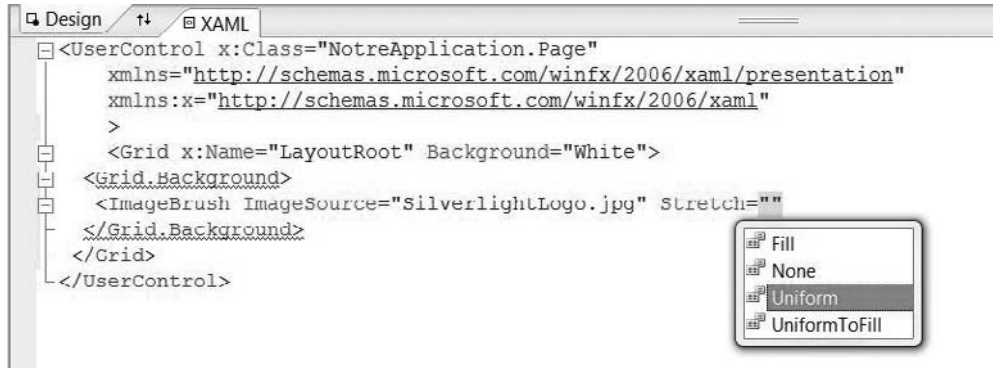
Dans la balise `Grid.Background`, spécifiez un pinceau basé sur une image (les pinceaux seront étudiés au chapitre 4) :

```
<Grid x:Name="LayoutRoot" >
  <Grid.Background>
    <ImageBrush ImageSource="SilverlightLogo.jpg" />
  </Grid.Background>
</Grid>
```

L'application n'a pas encore été compilée mais déjà Visual Studio nous signale une anomalie par un trait ondulé sous `Grid.Background` (figure 2-13). Cette erreur est due au fait que l'attribut `Background` n'a pas été supprimé alors qu'une sous-balise `Grid.Background` a été ajoutée. Supprimez l'attribut. Vous constatez alors que l'anomalie a disparu.

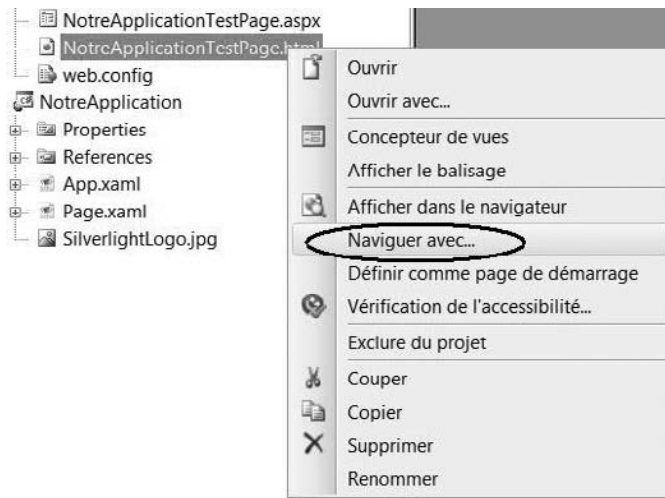
Une image est maintenant affichée en fond de grille mais elle est déformée (elle est, par exemple, aplatie) quand les dimensions de la grille (donc la fenêtre du navigateur) ne correspondent pas (en proportions) à celles de l'image. Pour résoudre ce problème, ajoutez l'attribut `Stretch` avec sa valeur `Uniform` (figure 2-13). Nous reviendrons sur cet attribut au chapitre 7).

Figure 2-13



Exécutez maintenant l'application (ce qui va impliquer une compilation) avec un autre navigateur que celui par défaut. Pour cela, cliquez droit sur le fichier .html dans l'Explorateur de solutions (partie Web), choisissez Naviguer avec... (figure 2-14) et sélectionnez Firefox dans la liste des navigateurs disponibles (c'est à vous de créer cette liste et de définir le navigateur par défaut).

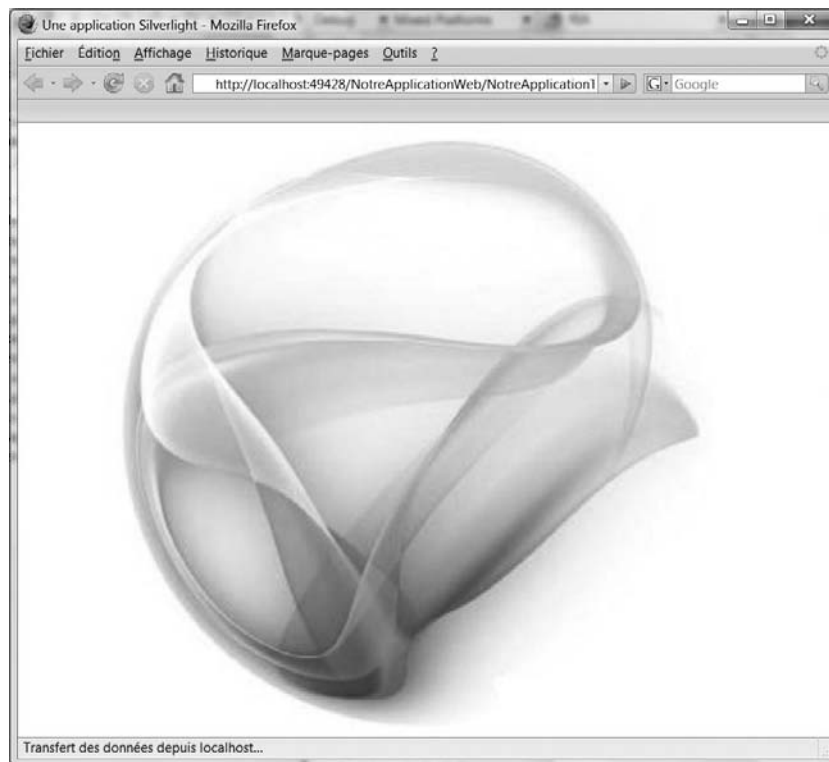
Figure 2-14



La figure 2-15 montre le résultat obtenu dans Firefox.

Quelle que soit la taille de la fenêtre du navigateur, l'image de fond est affichée à sa plus grande taille possible mais toujours en respectant ses proportions. Ainsi, elle n'est jamais déformée.

Figure 2-15



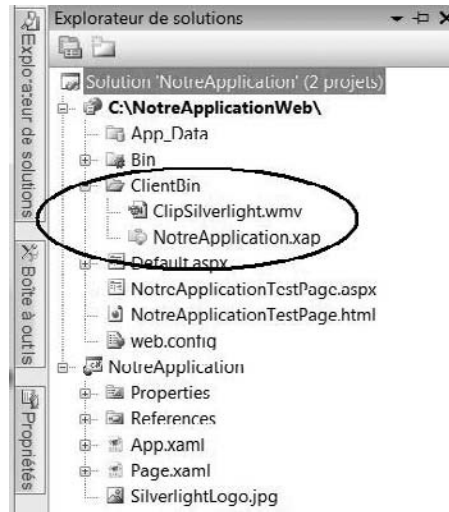
À ce stade, le fichier XAP (qui est un fichier compressé contenant le code binaire de l'application et désormais l'image incorporée en ressource) atteint 17 Ko.

Insertion de la vidéo

Nous allons à présent jouer un clip vidéo (le clip Silverlight de Microsoft) au milieu de la grille, qui se confond avec la fenêtre du navigateur. Pour cela, ajoutez la balise `MediaElement` et indiquez (attribut `AutoPlay`) qu'elle doit être jouée automatiquement et dès que possible (le temps qu'un premier *buffer* soit rempli afin d'assurer par la suite une diffusion fluide du film). Le fichier de la vidéo, `ClipSilverlight.wmv` doit être copié dans le répertoire `ClientBin` (celui du fichier XAP). Ceci s'effectue, par exemple, par un glisser-déposer depuis l'Explorateur de fichiers sur la ligne `ClientBin` de l'Explorateur de solutions, partie Web du projet (figure 2-16).

Insérez maintenant la balise `MediaElement` (voir chapitre 7) qui permet de lire de la musique ou des vidéos. Comme précédemment, vous pouvez procéder par glisser-déposer depuis la barre d'outils de Visual Studio vers un emplacement précis (le point de cliquage) dans le fichier `Page.xaml`. Spécifiez le nom du fichier vidéo (attribut `Source`) ainsi que la taille

Figure 2-16



du rectangle de diffusion (attributs `Width` et `Height`). Attribuez également un nom (attribut `x:Name`) au composant, ce qui n'est pas encore indispensable à ce stade mais le deviendra bientôt :

```
<Grid x:Name="LayoutRoot">
  <Grid.Background>
    <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
  </Grid.Background>
  <MediaElement x:Name="video" AutoPlay="True" Source="ClipSilverlight.wmv"
    Width="320" Height="340" />
</Grid>
```

Dans la mesure où les attributs `HorizontalAlignment` et `VerticalAlignment` n'ont pas été spécifiés (voir chapitre 3), `Center` est la valeur par défaut pour ces deux attributs. La vidéo sera donc toujours centrée dans la page du navigateur.

La figure 2-17 illustre le résultat obtenu dans Safari, avec l'image de fond et la vidéo.

La vidéo est maintenant jouée au démarrage (et à chaque rechargement de la page) mais elle ne l'est qu'une seule fois. Il faudra donc détecter la fin de la vidéo et, au moyen d'un programme, la relancer depuis le début. Pour cela, il convient de spécifier l'événement `MediaEnded` qui signale la fin de la vidéo. Il suffit de placer le curseur dans la balise `MediaElement` et d'appuyer sur la barre d'espace pour que Visual Studio affiche la liste des actions possibles (figure 2-18). Il s'agit de tous les attributs et événements applicables à un `MediaElement`, accompagnés d'une courte explication dans une info-bulle bien que les noms des attributs soient en général suffisamment explicites.

Sélectionnez l'attribut `MediaEnded` correspondant à un événement et demandez à Visual Studio de générer automatiquement la fonction de traitement (*event handler* en anglais). Pour cela, saisissez simplement le signe `=` à droite du nom de l'événement (figure 2-19).

Figure 2-17

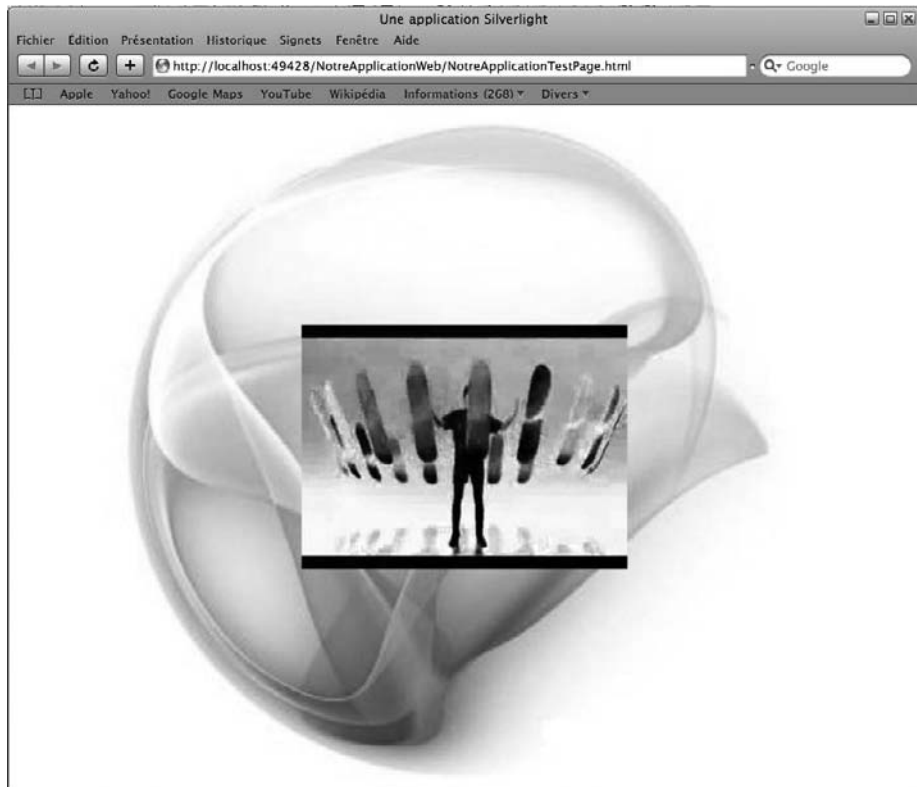


Figure 2-18

```
<MediaElement x:Name="video" | AutoPlay="True" Source="ClipSilverlight.wmv" />
</Grid>
</UserControl>
```

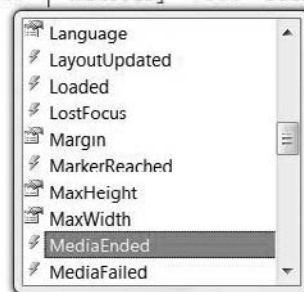
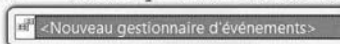


Figure 2-19

```
<Grid x:Name="LayoutRoot" >
  <Grid.Background>
    <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
  </Grid.Background>
  <MediaElement x:Name="video" MediaEnded="" AutoPlay="True" Source="ClipSilverlight.wmv" Width="320" />
</Grid>
</UserControl>
```



Liez les événements à une nouvelle méthode
gestionnaire d'événements pour naviguer ve

Confirmez ensuite la création de la fonction de traitement au moyen de la touche Tab (il s'agit ici d'une nouvelle fonction de traitement à créer mais on pourrait faire référence à une fonction existante qui traiterait un même événement pour plusieurs composants). La balise devient alors :

```
<MediaElement x:Name="video" MediaEnded="video_MediaEnded" AutoPlay="True"
              Source="ClipSilverlight.wmv" Width="320" Height="340" />
```

tandis que la fonction automatiquement générée pour traiter l'événement est (dans le fichier Page.xaml.cs) :

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

Le nom de la fonction de traitement est construit selon le schéma suivant : nom interne de l'élément (attribut `x:Name`), suivi du caractère de soulignement puis du nom de l'événement.

Dans cette fonction, repositionnez la vidéo à 0 seconde et jouez-la avec `Play`. Comme son nom l'indique, la classe `TimeSpan` (littéralement « étendue de temps ») permet de spécifier une durée. Il s'agit d'une classe que l'on retrouve telle quelle (comme la plupart des classes d'ailleurs) en programmation .NET pour Windows ou ASP.NET. Nous avons ici repris la forme la plus simple (ce qui est possible car il s'agit du début de la vidéo) mais on aurait pu construire une durée exprimée en jours, heures, minutes, secondes et millisecondes : `new TimeSpan(jj, hh, mm, ss, milli)`.

Avec ce petit ajout de deux lignes, la fonction de traitement devient :

```
private void video_MediaEnded(object sender, RoutedEventArgs e)
{
    video.Position = new TimeSpan(0);
    video.Play();
}
```

En VB, le XAML est identique et la fonction précédente s'écrit (elle est tout aussi automatiquement générée par Visual Studio) :

```
Private Sub video_MediaEnded(ByVal sender As Object, _
                             ByVal e As RoutedEventArgs)
    video.Position = New TimeSpan(0)
    video.Play()
End Sub
```

Passer d'un langage à l'autre ne présente vraiment aucune difficulté.

Le travail en couches transparentes

Nous allons maintenant nous occuper des animations. Bien que cela ne soit pas strictement nécessaire ici (mais cela nous simplifiera les choses tout en nous permettant de présenter cette intéressante fonctionnalité), nous allons ajouter deux couches transparentes à la grille et réaliser une animation sur chacune d'elles, sans toucher à la grille de fond (`LayoutRoot`) :

```

<Grid x:Name="LayoutRoot" >
  <Grid.Background>
    <ImageBrush ImageSource="SilverlightLogo.jpg" Stretch="Uniform" />
  </Grid.Background>
  <MediaElement x:Name="video" MediaEnded="video_MediaEnded" AutoPlay="True"
    Source="ClipSilverlight.wmv" Width="320" Height="340" />

  <Grid x:Name="GrImagesAnimées" Background="Transparent" >

  </Grid>

  <Grid x:Name="GrLogoAnimé" Background="Transparent" >

  </Grid>
</Grid>

```

Pour commencer, nommez (attribut `x:Name`) chacune des deux grilles transparentes. À ce stade, il s'agit avant tout de les nommer pour les identifier aisément mais les noms donnés aux composants sont aussi importants et jouent le même rôle que les noms donnés aux variables.

Comme aucune taille n'est spécifiée (ni aucun emplacement relatif), ces deux grilles transparentes se superposent à la grille de fond.

Première animation : le texte déroulant

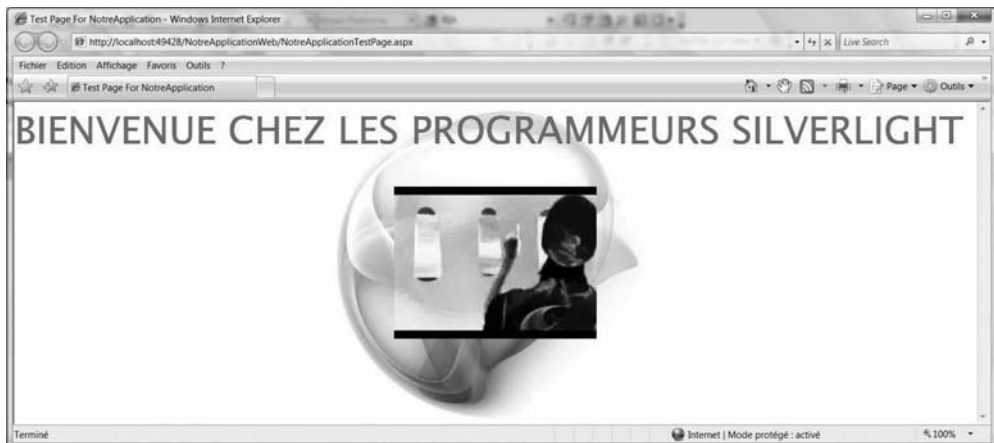
Dans la seconde grille transparente (`GrLogoAnimé`), ajoutez un texte en rouge et d'une taille de 60 pixels (figure 2-20) :

```

<Grid x:Name="GrLogoAnimé" Background="Transparent" >
  <TextBlock Text="BIENVENUE CHEZ LES PROGRAMMEURS SILVERLIGHT 2"
    FontSize="60" Foreground="Red" />
</Grid>

```

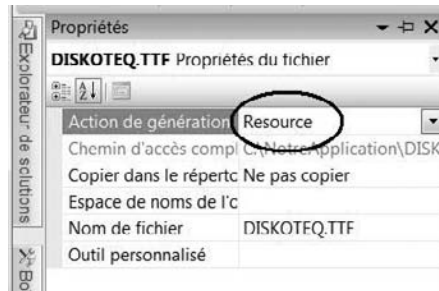
Figure 2-20



Le résultat est assez banal... Nous allons donc télécharger sur Internet une police de caractères (voir la section « Les polices de caractères » du chapitre 5) plus attrayante. Le site <http://www.coolgrafik.com> propose le fichier `Diskoteq.ttf` qui contient la police `Diskoteque`. Comment savoir que `Diskoteq.ttf` contient la police `Diskoteque` ? Dans l'Explorateur de fichiers, il suffit de double-cliquer sur le nom d'un fichier `.ttf` pour afficher les noms des polices de caractères qu'il contient.

Téléchargez ce fichier puis chargez-le dans le projet tout comme vous l'avez fait pour l'image de fond : cliquez droit sur le nom du projet (partie Silverlight), choisissez Ajouter>Élément existant et recherchez le fichier dans le système de fichiers. Vous devez ensuite indiquer que ce fichier doit être incorporé en ressource. Pour cela, cliquez droit sur le nom du fichier `.ttf` dans l'Explorateur de solutions, choisissez Propriétés et sélectionnez la valeur `Resource` dans le champ Action de génération (figure 2-21).

Figure 2-21



Procédez ensuite à un petit changement dans l'attribut `FontFamily` (nom de police) :

```
<TextBlock Text="Bienvenue chez les programmeurs Silverlight 2" FontSize="60"
            FontFamily="Diskoteq.ttf#Diskoteque" Foreground="Red" />
```

afin d'obtenir le résultat de la figure 2-22.

Figure 2-22



La taille du fichier XAP (qui incorpore maintenant la police) est désormais de 34,8 Ko. Comme nous l'avons déjà mentionné, un fichier XAP est en fait un fichier compressé au format ZIP. En l'analysant (il suffit de le renommer en .zip), on constate qu'il contient deux fichiers : AppManifest.xaml et NotreApplication.dll. Le premier donne des informations sur ce qui est transmis au navigateur du client (à l'exception de la page HTML). Le second contient quant à lui le code binaire (résultat d'une compilation) de l'application ainsi que les ressources (dans notre cas et jusqu'à présent, une image et une police de caractères).

Pour notre application, nous décidons de faire défiler le texte en permanence de droite à gauche, comme s'il entrerait par la droite et sortait par la gauche de la fenêtre du navigateur. Une animation sera nécessaire pour cela, ce que nous étudierons au chapitre 9.

Pour le moment, placez le texte (balise `TextBlock`) dans un canevas (qui est un type de conteneur, voir chapitre 3), lui-même inséré dans la grille. Spécifiez une marge de 10 pixels par rapport au bord supérieur afin d'aérer la présentation. Dans le canevas, animez la coordonnée `Left` (le long de l'axe horizontal) du texte que vous venez de créer. Cette coordonnée va passer d'un déplacement de 1 400 à -1 400 en 10 secondes et l'animation se répétera sans cesse (attribut `RepeatBehavior`). Il serait possible d'optimiser les valeurs 1 400 et -1 400 en tenant compte de la taille réelle du texte mais cela n'a pas d'importance à ce stade de l'étude.

Pour réaliser l'animation, il convient de placer ce que l'on appelle un *storyboard* dans le jargon Silverlight en ressource de son conteneur (ici, un canevas). Il faut également attribuer un nom (attribut `x:Name`) à l'animation (ici, `animLogo`) ainsi qu'au composant `TextBlock` (ici, `zaLogo`). Ce composant devra par ailleurs être déplacé dans la balise `Canvas`, juste avant la balise de fermeture. Pour plus de détails sur les animations et les transformations, reportez-vous au chapitre 9.

```
<Grid x:Name="GrLogoAnimé" Background="Transparent" >
  <Canvas Margin="0, 10, 0, 0" >
    <Canvas.Resources>
      <Storyboard x:Name="animLogo" RepeatBehavior="Forever" >
        <DoubleAnimation Storyboard.TargetName="zaLogo"
          Storyboard.TargetProperty="(Canvas.Left)"
          From="1400" To="-1400" Duration="0:0:10" />
      </Storyboard>
    </Canvas.Resources>
    <TextBlock x:Name="zaLogo"
      Text="Bienvenue chez les programmeurs Silverlight 2" FontSize="60"
      FontFamily="Diskoteq.ttf#Diskoteque" Foreground="Red" />
  </Canvas>
</Grid>
```

L'animation porte sur la propriété `Canvas.Left` de l'élément `zaLogo`, ce qui est spécifié dans les attributs `Storyboard.TargetProperty` et `Storyboard.TargetName`. Du fait que `Canvas.Left` désigne une propriété dite attachée (voir la section « Les rectangles et les ellipses » du chapitre 5), les parenthèses sont nécessaires.

Dans la mesure où il s'agit ici d'animer la propriété `Canvas.Left` qui contient une valeur numérique de type `double` (type privilégié par Silverlight pour les valeurs numériques), nous avons affaire à une animation de type `DoubleAnimation` (d'autres types d'animations seront étudiés à la section « Les animations » du chapitre 9). La durée d'un passage (de droite à gauche dans la fenêtre) est spécifiée dans l'attribut `Duration` et correspond ici à 0 heure, 0 minute et 10 secondes. La durée d'une animation peut aussi être exprimée en millisecondes mais cela n'aurait pas de sens ici.

L'animation sera lancée au démarrage de l'application. L'événement `Loaded` adressé à la grille (voir chapitre 6) signale ce moment. Cet événement est plus précisément signalé juste après le travail de préparation de la page en mémoire et juste avant le tout premier affichage dans la fenêtre du navigateur. Ici aussi, nous demandons à Visual Studio de générer la fonction de traitement. Comme nous l'avons vu précédemment, il suffit de taper `Loaded=` pour que Visual Studio enclenche le processus de création de la fonction de traitement (figure 2-23).

Figure 2-23



```
<Grid x:Name="LayoutRoot" Loaded="">
  <Grid.Background>
    <ImageBrush ImageSource="Silverlight.png" />
  </Grid.Background>
  <MediaElement x:Name="video" MediaEnded="video_MediaEnded" AutoPlay="true" />
</Grid>
```

La balise devient alors :

```
<Grid x:Name="LayoutRoot" Loaded="LayoutRoot_Loaded">
```

et une nouvelle fonction de traitement (de l'événement `Loaded` adressé à la grille `LayoutRoot`) est générée dans le fichier `Page.xaml.cs`. Il convient à présent de la compléter. Comme l'animation s'appelle `animLogo` (attribut `x:Name` du storyboard, figure 2-24), la fonction `Begin` appliquée à l'animation la fait démarrer :

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
    animLogo.Begin();
}
```

Figure 2-24



```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
    animLogo.Begin();
}
```

void Storyboard.Begin()
Initiates the set of animations associated with this System.Windows.Controls.AnimationClock.

Les figures 2-25 et 2-26 présentent le résultat de cette animation dans Safari à deux moments différents.

Figure 2-25



Figure 2-26



Zone d'édition et bouton dans Silverlight

Au bas de la fenêtre, il convient à présent de placer le libellé (Critère Flickr :), la zone d'édition (dans laquelle l'utilisateur saisira le critère de recherche) ainsi que le bouton permettant de lancer une recherche d'images sur le site Flickr. Ces trois éléments correspondent respectivement aux composants `TextBlock`, `TextBox` et `Button` qui font partie, avec beaucoup d'autres, de la panoplie des composants Silverlight. Ils devront être accolés au bord inférieur de la fenêtre au moyen de l'attribut `VerticalAlignment` pour lequel la valeur `Bottom` sera spécifiée, tout en prenant soin de définir une marge de manière à aérer la présentation. Ainsi, même si l'utilisateur redimensionne la fenêtre, ces trois éléments seront toujours affichés au bas de la fenêtre.

Pour cela, ajoutez ce qui suit à l'intérieur de la balise `Grid` pour `GrLogoAnimé` (juste avant la balise de fermeture `</Grid>`) :

```
<TextBlock Text="Critère Flickr :" HorizontalAlignment="Left"
           VerticalAlignment="Bottom" Margin="10, 20"
           FontSize="25" Foreground="Red" FontFamily="Verdana" />
<TextBox x:Name="zeCritFlickr" Text="Paris" HorizontalAlignment="Left"
          VerticalAlignment="Bottom" Margin="200, 25" Width="250" />
<Button x:Name="bFlickr" Content="Recherche Flickr" HorizontalAlignment="Left"
         VerticalAlignment="Bottom" Margin="500, 10" Width="100" Height="60"
         Click="bFlickr_Click" />
```

Profitez-en pour demander à Visual Studio de générer la fonction de traitement du clic sur le bouton (événement `Click`), fonction qui sera complétée par la suite :

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

La figure 2-27 illustre le résultat obtenu à ce stade.

Figure 2-27



Image dans bouton

Pour le moment, le libellé du bouton est un texte (`Recherche Flickr`), que nous allons remplacer par une image, à savoir le logo de Flickr. À l'aide du logiciel Microsoft Photo Editor, ou de tout autre logiciel proposant les mêmes fonctionnalités, rendez ce logo transparent en spécifiant une couleur de transparence et créez le fichier `Flickr.png` à partir du fichier `.jpg`. Incorporez ensuite cette image en ressource (Ajouter>Élément existant dans l'Explorateur de solutions (partie Silverlight)). Enfin, modifiez la balise `Button` (supprimez l'attribut `Content` et ajoutez une balise `Button.Content`), qui devient alors :

```
<Button x:Name="bFlickr" HorizontalAlignment="Left" VerticalAlignment="Bottom"
        Margin="500, 10" Width="100" Height="60" Click="bFlickr_Click">
    <Button.Content>
        <Image Source="Flickr.png" />
    </Button.Content>
</Button>
```

La figure 2-28 représente le résultat obtenu, le logo Flickr étant maintenant affiché en transparence dans le bouton.

Figure 2-28



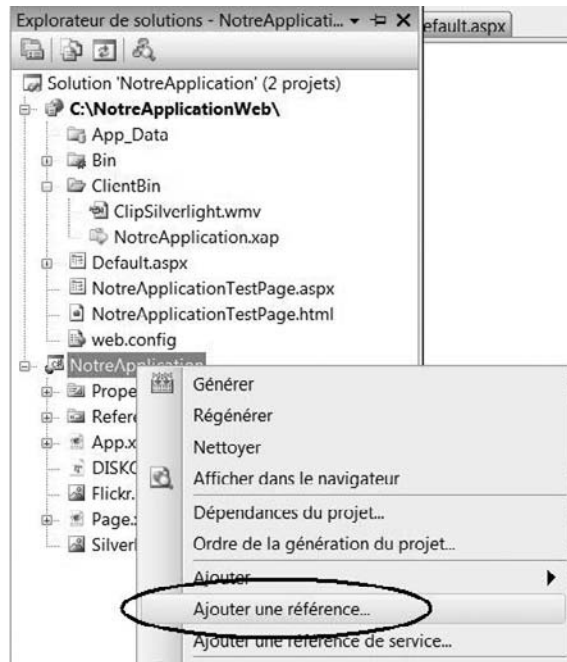
La modification du bouton a été ici très simple. Au chapitre 15, nous verrons comment un graphiste (de préférence...) peut, grâce au logiciel Expression Blend, modifier et personnaliser n'importe quel sous-élément de n'importe quel composant Silverlight (y compris les animations de transition d'état telles que le passage à l'état survol ou encore l'indication du clic) et cela, sans provoquer la moindre modification dans le travail du programmeur.

On tourne la page...

Nous allons maintenant insérer un composant *page turner* dans la partie inférieure droite de l'application, composant qui restera accolé au bord inférieur droit de la fenêtre du navigateur, quelle que soit sa taille. L'utilisateur pourra ainsi tourner les pages d'un catalogue touristique, tout comme il le ferait avec un catalogue papier, mais en se servant bien évidemment ici de la souris. Pour cela, vous allez utiliser le composant décrit à la section « Tourner la page » du chapitre 7.

Pour utiliser ce composant, l'application doit faire référence à (et intégrer) la DLL `SLMitsuControls.dll` (voir cette section « Tourner la page » au chapitre 7). Pour cela, sélectionnez Ajouter une référence... (figure 2-29) dans l'Explorateur de solutions (partie Silverlight).

Figure 2-29



Puis localisez (onglet Parcourir) la DLL `SLMitsuControls.dll` sur votre ordinateur (figure 2-30).

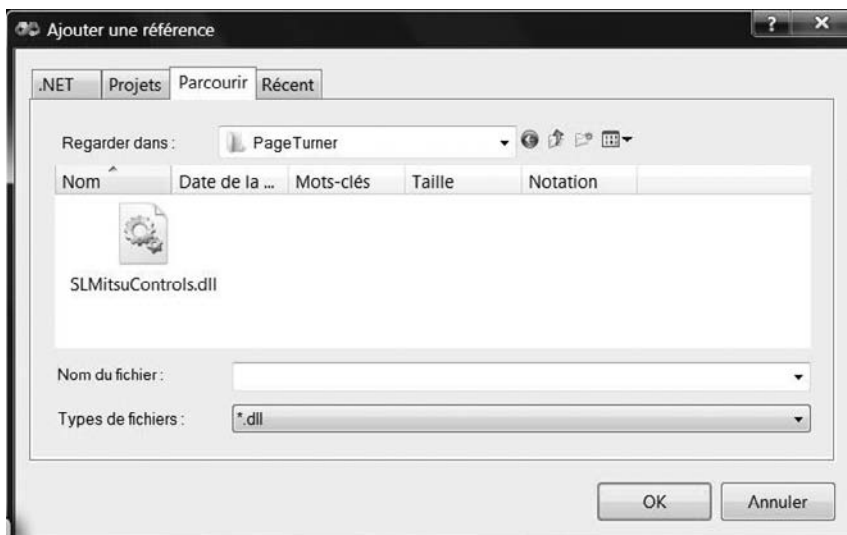
Après compilation, cette DLL sera greffée au fichier XAP, dont la taille augmentera alors de 34 Ko, ce qui est vraiment peu par rapport à ce qu'elle va nous permettre de faire.

Comme cela sera expliqué au chapitre 14, le contrôle utilisateur inclus dans cette DLL doit être spécifié avec un préfixe, de manière à le distinguer des composants intégrés au run-time Silverlight. Pour cela, il suffit d'ajouter la ligne de code suivante (en gras) dans

la balise UserControl du fichier Page.xaml (le préfixe est ici mf, d'après les initiales de l'auteur du composant, à qui nous rendons hommage) :

```
<UserControl x:Class="NotreApplication.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mf="clr-namespace:SLMitsuControls;assembly=SLMitsuControls"
>
```

Figure 2-30



Dans la grille GrLogoAnimé, ajoutez ce « tourneur de pages » et placez-le (avec une petite marge) dans le coin inférieur droit :

```
<mf:UCBook x:Name="géo" HorizontalAlignment="Right" VerticalAlignment="Bottom"
    Margin="20" Width="500" Height="220" />
```

Par ailleurs, les images du catalogue touristique doivent être copiées dans le répertoire ClientBin (partie Web du projet). Il suffit pour cela d'effectuer un copier-coller depuis l'Explorateur de fichiers jusqu'à l'entrée ClientBin dans l'Explorateur de solutions (figure 2-31).

Les fichiers de ces images apparaissent alors dans l'Explorateur de solutions, partie Web (figure 2-32).

En ressource dans la grille (celle du fond, LayoutRoot), indiquez les pages qui seront utilisées par le « tourneur de pages » :

```
<Grid x:Name="LayoutRoot" Loaded="LayoutRoot_Loaded">
    <Grid.Resources>
        <ItemsControl x:Name="pages" >
            <Image Source="Amsterdam.jpg" Stretch="Fill" />
```



```

<Image Source="Venise.jpg"    Stretch="Fill"  />
<Image Source="Bruxelles.jpg" Stretch="Fill" />
<Image Source="Moscou.jpg"   Stretch="Fill" />
<Image Source="Londres.jpg"  Stretch="Fill" />
</ItemsControl>
</Grid.Resources>
.....

```

Au chapitre 7, nous verrons qu'il pourrait s'agir de bien autre chose que de simples images. Des événements peuvent être traités, qui indiquent quand les pages sont tournées, quelles pages sont affichées (à gauche et à droite) et sur quelle page l'utilisateur a cliqué, vraisemblablement en vue d'obtenir de plus amples informations.

À ce stade, le fichier `Page.xaml.cs` doit subir quelques modifications. En effet, il faut indiquer que le compilateur doit tenir compte d'un nouvel espace de noms, que la classe `Page` doit implémenter l'interface `IDataProvider` et qu'il faut pour cela implémenter les fonctions `GetCount` et `GetItem` (c'est l'implémentation du composant « tourneur de pages » qui exige ces quelques lignes supplémentaires).

Figure 2-31

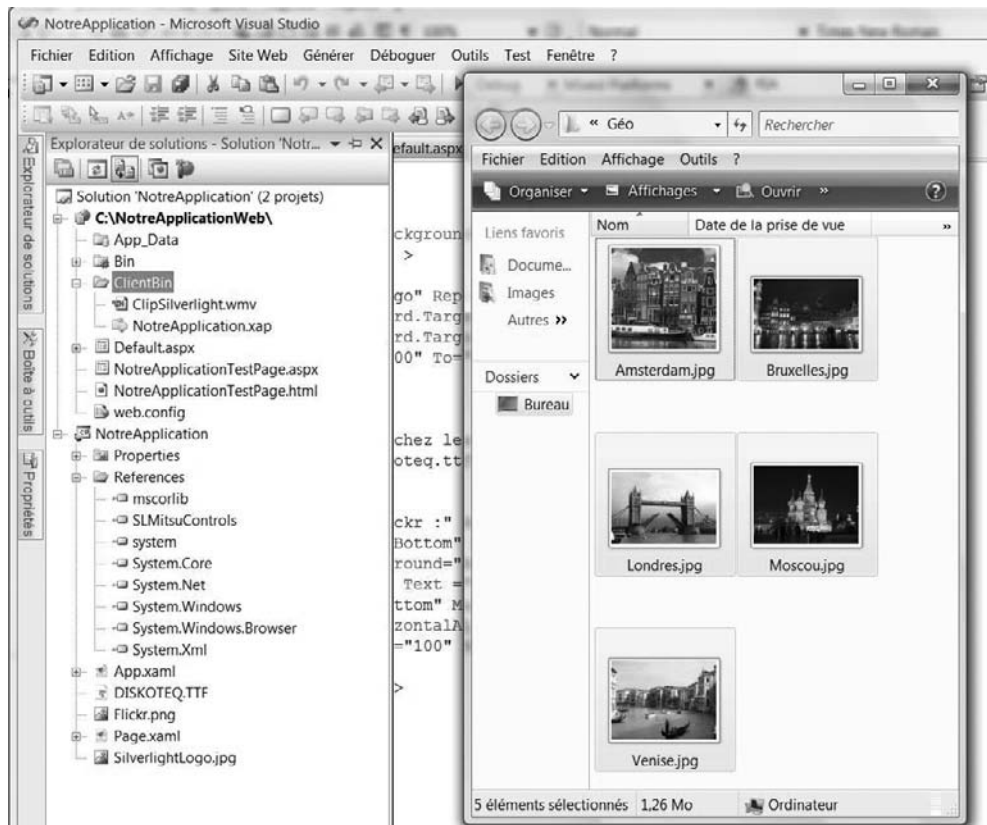
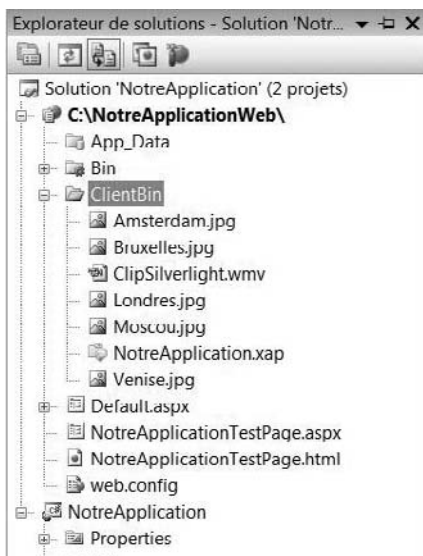


Figure 2-32



Dans la fonction qui traite l'événement `Loaded` adressé à la grille principale (`LayoutRoot`), associez le composant « tourneur de pages » aux données qu'il doit afficher dans ses pages (ici, une série d'images mentionnées dans la balise `ItemsControl`, avec pages comme nom interne). Les lignes de code à ajouter dans le fichier `Page.xaml.cs` sont indiquées ci-dessous :

```
using SLMitsuControls;
.....
public partial class Page : UserControl, IDataProvider
{
    .....
    private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
    {
        .....
        géo.SetData(this);
    }
    public object GetItem(int index)
    {
        return pages.Items[index];
    }
    public int GetCount()
    {
        return pages.Items.Count;
    }
}
```

Vous pouvez désormais lancer l'application et « tourner les pages » (figure 2-33).

Les figures 2-34 et 2-35 présentent le « tourneur de pages » en gros plan, lorsque l'utilisateur tourne la page.

Figure 2-33



Figure 2-34



Figure 2-35



Accès au serveur d'images Flickr

À la demande de l'utilisateur, c'est-à-dire lorsque celui-ci clique sur le bouton, l'application doit aller chercher des images sur le site Flickr et les afficher ensuite sur un carrousel tournant. Dans le cadre de cette application, nous nous limiterons à six photos par sujet (des milliers de photos sont parfois disponibles sur certains sujets).

Pour réaliser cette animation, il convient tout d'abord d'afficher l'anneau sur lequel tourneront les photos. Celui-ci est inséré dans la première grille transparente (GrImagesAnimées) :

```
<Grid x:Name="GrImagesAnimées" Background="Transparent" >
  <Ellipse x:Name="rail" StrokeThickness="10" Stroke="Gray"
    Width="500" Height="500" />
</Grid>
```

L'anneau est en fait une ellipse dont la largeur de trait est de 10 pixels. Comme les valeurs par défaut des attributs `HorizontalAlignment` et `VerticalAlignment` sont `Center`, l'anneau sera toujours centré dans la grille (figure 2-36).

Pour obtenir des photos correspondant au critère spécifié dans la zone d'édition, vous devez adresser une requête à Flickr (ceci sera expliqué au chapitre 13, ainsi que la procédure, très simple, d'inscription à Flickr). Après votre inscription sur le site Flickr, celui-ci vous communiquera un code utilisateur (à utiliser ici) ainsi qu'un mot de passe (non utilisé ici car il sert à envoyer des photos au serveur ou donne accès à des photos à usage privé). Cette requête sera exécutée dans la fonction qui traite l'événement `Click` du bouton.

Figure 2-36



La requête doit être adressée à une URL appartenant à Flickr, en mentionnant un code d'accès ainsi que la chaîne « critère de recherche ». Il n'y a pas de véritable problème à communiquer un code d'accès, sauf peut-être un usage immodéré et scabreux que certains pourraient en faire.

```
private void bFlickr_Click(object sender, RoutedEventArgs e)
{
    string clé = "1e1313a40fe8dbe????24141d90a1231"; // code d'accès
    string url = "http://api.flickr.com/services/rest/"
        + "?method=flickr.photos.search&api_key="
        + clé + "&text=" + zeCritFlickr.Text;
    WebClient client = new WebClient();
    ..... // ici, deux instructions, dont la requête
}
```

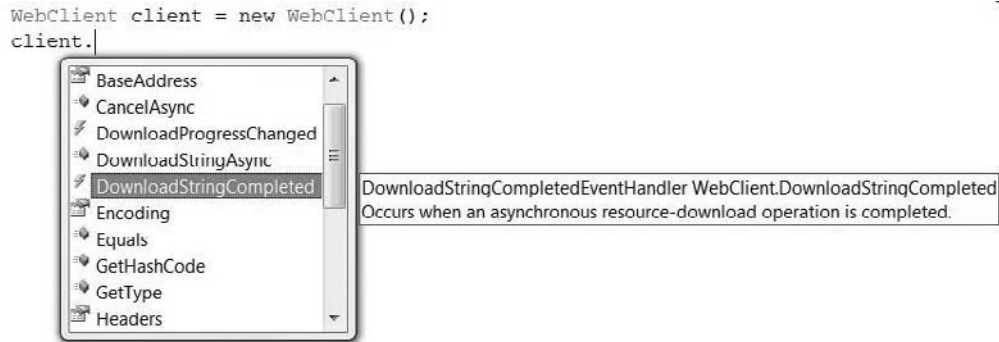
Flickr va lire la requête que vous lui avez envoyée et vous fournira en retour un fichier XML contenant des informations sur les photos répondant au critère (mais pas encore les photos elles-mêmes).

Pour effectuer une telle requête sur le Web, Silverlight met à votre disposition l'objet `WebClient`. Dans la mesure où cette requête peut durer un certain temps, l'opération est

effectuée de manière asynchrone afin de ne pas bloquer l'application Silverlight. L'objet WebClient signale l'événement DownloadStringCompleted quand le fichier XML a été reçu.

Cet événement est bien sûr repris dans l'aide contextuelle et il convient de signaler à Silverlight qu'il va être traité (figure 2-37).

Figure 2-37



Comme il s'agit d'un événement, saisissez += à droite du nom du nom de l'événement (figure 2-38).

Figure 2-38

```
WebClient client = new WebClient();
client.DownloadStringCompleted +=
```

new DownloadStringCompletedEventHandler(client.DownloadStringCompleted); (Appuyez sur TABULATION pour insérer)

Visual Studio vous propose alors de compléter l'instruction et de générer la fonction de traitement (appuyez sur la touche Tab pour valider, figure 2-39).

Figure 2-39

```
WebClient client = new WebClient();
client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(client.DownloadStringCompleted);
```

Appuyez sur TABULATION pour générer le gestionnaire

Visual Studio génère donc la ligne de code suivante (avant-dernière instruction de la fonction bFlickr_Click) :

```
client.DownloadStringCompleted
    += new DownloadStringCompletedEventHandler(client.DownloadStringCompleted);
```

ainsi que la fonction de traitement, qu'il convient de compléter :

```
void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
{
}
```

Dans cette fonction de traitement, `e.Result` contient (dans une chaîne de caractères) la réponse XML.

Ne reste ensuite qu'à lancer l'opération de requête auprès de Flickr (dernière instruction de la fonction `bFlickr_Click`) :

```
client.DownloadStringAsync(new Uri(url));
```

où l'argument est un objet basé sur l'URL où Flickr traite les requêtes (URL qui incorpore dans ses arguments votre code d'accès ainsi que le critère de recherche). Reportez-vous au chapitre 13 pour de plus de détails.

Nous allons maintenant réaliser le carrousel tournant dont la technique sera expliquée en détail au chapitre 9. Pour cela, il convient de définir une transformation de type `RotateTransform` et d'animer l'angle de rotation en le faisant passer de 0 à 360 degrés en 10 secondes et en répétant cette animation à l'infini (sauf pendant les temps de chargement d'une nouvelle série d'images). Le centre de rotation est défini en coordonnées relatives dans l'attribut `RenderTransformOrigin` :

```
<Grid x:Name="GrImagesAnimées"
      Background="Transparent" RenderTransformOrigin="0.5, 0.5" >
  <Grid.RenderTransform >
    <RotateTransform x:Name="rotCarrousel" Angle="0" />
  </Grid.RenderTransform>
  <Grid.Resources>
    <Storyboard x:Name="stbCarrousel" >
      <DoubleAnimation Storyboard.TargetName="rotCarrousel"
        Storyboard.TargetProperty="Angle" From="0" To="360" Duration="0:0:10"
        RepeatBehavior="Forever" />
    </Storyboard>
  </Grid.Resources>
  <Ellipse x:Name="rail" StrokeThickness="10" Stroke="Gray"
    Width="500" Height="500" />
</Grid>
```

Reste désormais à analyser la réponse XML de Flickr, ce que nous verrons en détail au chapitre 13.

Pour décortiquer ce XML de réponse, nous allons utiliser la technologie `Linq for Xml`, le langage d'interrogation et de manipulation de données maintenant intégré à C# et VB (reportez-vous au chapitre 12 pour plus d'informations à ce sujet).

Pour utiliser `Linq for Xml`, une bibliothèque de code doit être intégrée au programme. Pour cela, sélectionnez *Ajouter une référence...* dans l'Explorateur de solutions, partie Silverlight (figure 2-40). Choisissez ensuite `System.Xml.Linq` dans l'onglet .NET. La DLL de code pour `Linq for Xml` sera ainsi intégrée au projet et finalement greffée dans le fichier XAP envoyé au navigateur (figure 2-41).

Figure 2-40

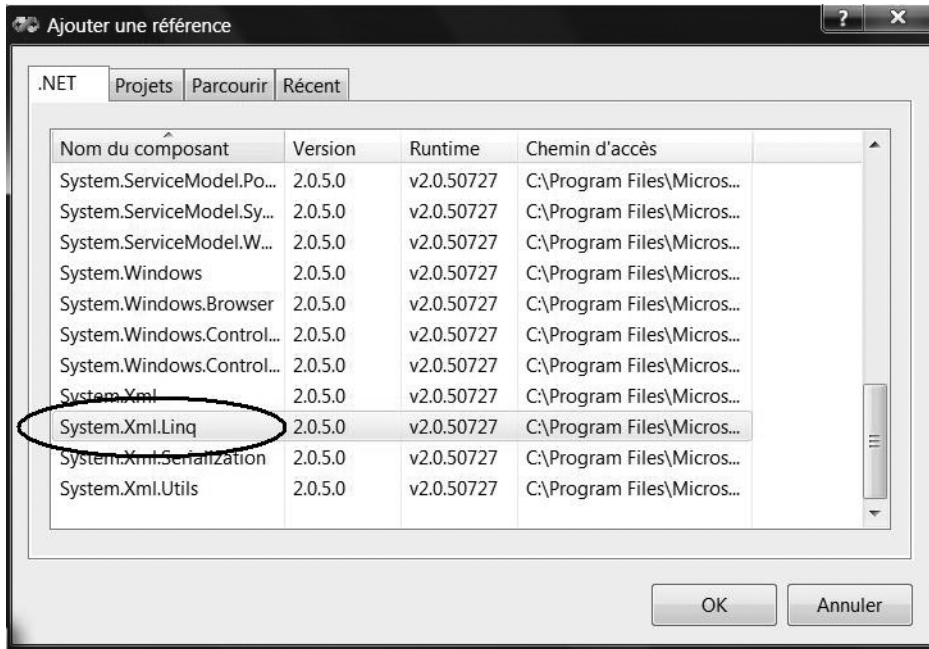
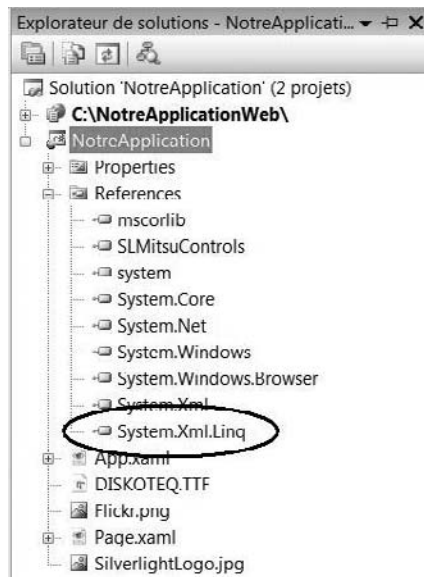


Figure 2-41



Nous allons maintenant décortiquer la réponse XML, créer dynamiquement les composants `Image` (dont on ignore exactement le nombre mais qui sera compris entre zéro et six), télécharger les images depuis Flickr et finalement les accrocher au carrousel tournant. Tout cela sera expliqué en détail aux chapitres 6 (création dynamique de composants), 12 (Linq for XML) et 13 (services Web).

Étant donné les objets utilisés pour réaliser cette animation, il convient d'ajouter les trois espaces de noms suivants en tête du fichier `Page.xaml.cs` :

```
using System.Xml.Linq;  
using System.Windows.Markup;  
using System.Windows.Media.Imaging;
```

Ce processus d'ajout d'espaces de noms peut être automatisé. Par ailleurs, si Visual Studio ne semble pas connaître une classe ou si une erreur est signalée sur un nom de classe, cliquez droit sur le nom de la classe et sélectionnez Résoudre. Visual Studio procédera alors automatiquement à l'ajout.

Dans la mesure où le nombre d'images qui seront affichées n'est pas connu (ce nombre varie en effet d'une requête à l'autre), il convient de placer les composants `Image` dans une liste dynamique d'images (il s'agit d'une liste qui s'agrandit automatiquement en fonction des insertions). La déclaration et l'instanciation dans la classe est effectuée en dehors des fonctions afin que `listeImages` soit accessible partout :

```
List<Image> listeImages = new List<Image>();
```

Dans la fonction qui traite l'événement `DownloadStringCompleted`, stoppez le carrousel et videz-le (ainsi que `listeImages`) des images qui s'y trouvaient :

```
void client_DownloadStringCompleted(object sender,  
                                   DownloadStringCompletedEventArgs e)  
{  
    stbCarrousel.Pause();  
  
    int N = listeImages.Count; // nombre actuel d'images  
    for (int n = 0; n < N; n++)  
    {  
        Image img = listeImages[0];  
        GrImagesAnimées.Children.Remove(img);  
        listeImages.RemoveAt(0);  
        img = null;  
    }  
    ....
```

Décortiquez à présent le XML de réponse en passant par les balises `rsp`, `photos` et `photo`. Cette dernière balise est répétée pour chaque image disponible et contient, dans ses attributs, le titre de la photo ainsi que des informations permettant de reconstituer l'URL de l'image sur le site de Flickr (toutes les opérations sont expliquées en détail dans la suite de l'ouvrage).


```
XDocument xml = XDocument.Parse(e.Result);
var liste = from p in xml.Element("rsp").Element("photos").Elements("photo")
            select p;
liste = liste.Take(6); // six photos au maximum
N = liste.Count();
```

Pour chaque photo recherchée sur le site de Flickr :

```
for (int n = 0; n < N; n++)
{
    .....
}
```

il faut construire dynamiquement un composant Image (ici, à partir d'une balise XAML, voir la section « Création dynamique à partir du XAML » du chapitre 6) :

```
string sXamlImage = "<Image xmlns='http://schemas.microsoft.com/client/2007' "
                  + " Width='200' Height='200' HorizontalAlignment='Left' "
                  + " VerticalAlignment='Top' />";
Image img = (Image)XamlReader.Load(sXamlImage);
```

Il convient ensuite de préparer l'URL de l'image (sur le site de Flickr) à partir d'attributs de la balise photo (voir la section « Application au service Web Flickr » du chapitre 13) :

```
string sFarm = liste.ElementAt(n).Attribute("farm").Value;
string sServer = liste.ElementAt(n).Attribute("server").Value;
string sId = liste.ElementAt(n).Attribute("id").Value;
string sSecret = liste.ElementAt(n).Attribute("secret").Value;
string sUrl = "http://farm" + sFarm + ".static.flickr.com/" + sServer + "/" + sId
              + "_" + sSecret + ".jpg";
```

La position de la photo sur le rail est ensuite calculée (s'il y a X photos, elles font entre elles un angle de 360/X degrés) :

```
double R = rail.Width / 2;
double angle = 2 * 3.14 * n / N;
double X = ActualWidth / 2 + R * Math.Cos(angle) - 75;
double Y = ActualHeight / 2 - R * Math.Sin(angle) - 75;
```

Celle-ci est ensuite positionnée sur le rail :

```
img.Margin = new Thickness(X, Y, 0, 0);
```

puis récupérée sur le site de Flickr :

```
ImageSource imgs = new BitmapImage(new Uri(sUrl));
img.SetValue(Image.SourceProperty, imgs);
```

L'image est alors « accrochée » sur la grille transparente et insérée dans la liste d'images :

```
GrImagesAnimées.Children.Add(img);
listeImages.Add(img);
```

Enfin, le carrousel tournant est lancé :

```
stbCarrousel.Begin();
```

La figure 2-42 montre le résultat final.

Figure 2-42



À noter que l'on pourrait stabiliser les images en leur donnant une rotation inverse à celle de la grille (ce qui sera le cas pour le programme d'accompagnement présenté au chapitre 7).

3

Les conteneurs

Les éléments visuels d'interface (*UI elements* en anglais, *UI* signifiant *User Interface*), tels que les boutons, les boîtes de listes, les images, les figures géométriques, etc., doivent être placés dans des conteneurs. Généralement (c'est le cas par défaut), on compte un conteneur Silverlight par page Web contenant du Silverlight. Au chapitre 16, nous verrons comment modifier les fichiers `.html` ou `.aspx` pour que la page Web ne soit que partiellement Silverlight, avec un ou plusieurs conteneurs Silverlight en superposition d'éléments HTML (ou ASP.NET ou PHP).

Dans ce chapitre, nous étudierons les trois conteneurs de base de Silverlight ainsi que `ScrollViewer` et `Border` (il s'agit de conteneurs car ces composants contiennent d'autres composants).

Les conteneurs de base

Silverlight 2 propose trois types de conteneurs (également appelés *layout panels*) :

- le canevas (objet `Canvas`), dans lequel les éléments sont déposés en spécifiant leurs coordonnées ;
- le `StackPanel`, dans lequel les objets sont automatiquement empilés les uns au-dessous des autres ou les uns à côté des autres, en fonction de l'orientation du panneau ;
- la grille (balise `Grid`), dans laquelle les objets sont placés dans des cellules (à l'intersection d'une rangée et d'une colonne), la grille pouvant s'adapter automatiquement à la taille de la fenêtre du navigateur.

Comme vous pouvez le constater en examinant le fichier XAML d'une page Silverlight, ces conteneurs sont eux-mêmes insérés dans une balise `UserControl` mais cela n'a pas d'importance à ce stade.

Disons déjà que les classes `Canvas`, `StackPanel` et `Grid` sont dérivées de la classe `Panel`.

Par défaut, Visual Studio propose la grille comme conteneur principal lors de la création d'un nouveau projet Silverlight (ici, le fichier nommé `Page.xaml` par défaut) :

```
<UserControl x:Class="SLProg.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">

        </Grid>
    </UserControl>
```

Par défaut, la grille présente un attribut `Background` correspondant à la couleur de fond (voir chapitre 4) ainsi que l'attribut `x:Name` correspondant au nom interne du conteneur. À l'intérieur de la balise `Grid`, d'autres balises permettront de décrire plus précisément la grille, notamment le nombre et la taille des rangées et des colonnes mais aussi le contenu des cellules.

Par ailleurs, le conteneur principal est également placé par défaut dans un *user control* qui limite la taille du conteneur à un rectangle de 400×300 pixels (attributs `Width` et `Height`). Ces deux attributs seront généralement modifiés, et même le plus souvent supprimés afin que la grille occupe toute la fenêtre du navigateur.

Dans ce chapitre, nous ne placerons dans les conteneurs qu'un seul type d'élément UI, à savoir le bouton (et encore, réduit à sa plus simple expression) afin de nous concentrer sur les conteneurs. À ce stade de l'étude de Silverlight, il n'est en effet pas question de réaliser une application spectaculaire mais bien d'en maîtriser les différents éléments de base.

Les sections suivantes détaillent les trois conteneurs de base de Silverlight.

Le canevas

Balise de canevas

On peut assimiler le canevas de Silverlight à celui utilisé pour créer une tapisserie ou un tableau : des éléments visuels (boutons, images, boîtes de listes, etc.) y sont déposés à des emplacements bien définis.

Pour imposer un canevas comme conteneur principal, il suffit de remplacer le mot-clé `Grid` par `Canvas` dans le fichier `Page.xaml` créé par défaut par Visual Studio. Profitez-en également pour modifier ses dimensions (portées ici à 800×600 pixels) ainsi que la couleur de fond (ici, la couleur `AliceBlue`) de manière à rendre les choses plus visibles lors de l'exécution de l'application Silverlight :

```
<UserControl x:Class="SLProg.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="800" Height="600">
    <Canvas Background="AliceBlue">

        </Canvas>
    </UserControl>
```

Le canevas a maintenant une largeur (attribut `Width`) de 800 pixels et une hauteur (attribut `Height`) de 600 pixels. Cette taille pourrait être spécifiée en attributs du `Canvas` plutôt qu'en attributs du `UserControl`. En supprimant les deux attributs `Width` et `Height` (dans les balises `UserControl` et `Canvas`), le canevas occupe toute la fenêtre du navigateur, mais sans adaptation automatique de son contenu (par *stretching*).

La couleur de fond est maintenant `AliceBlue` (voir la section « Les couleurs » du chapitre 4). Comme nous le verrons au chapitre 4, il pourrait s'agir d'un dégradé de couleurs mais aussi (voir la section « Les images comme motifs de pinceau » du chapitre 7) d'une image, voire d'une vidéo.

Le contenu du canevas

Nous allons à présent ajouter deux boutons au canevas (les boutons, tout comme la plupart des composants seront étudiés au chapitre 5). Le libellé d'un bouton est spécifié dans l'attribut `Content` de la balise `Button`. Au chapitre 15, nous verrons comment changer fondamentalement son apparence sans que cela ait d'influence sur son fonctionnement. Ici, nous nous contentons d'un bouton standard avec libellé.

```
<Canvas Background="AliceBlue">
  <Button Content="B1" Canvas.Left="50" Canvas.Top="50" Width="90" Height="70" />
  <Button Content="B2" Canvas.Left="250" Canvas.Top="80" Width="90" Height="70"/>
</Canvas>
```

Les coordonnées du coin supérieur gauche du bouton (mais aussi, de manière générale, de tout élément UI) sont spécifiées dans les attributs suivants :

- `Canvas.Left` pour l'axe des X ;
- `Canvas.Top` pour l'axe des Y.

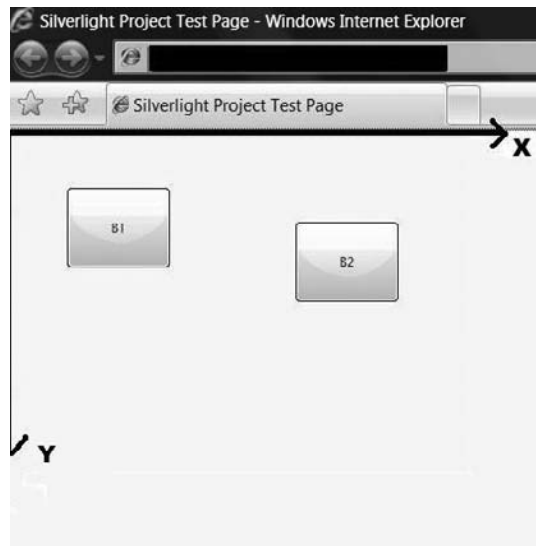
L'axe des X est disposé horizontalement, de gauche à droite, et se confond avec le bord supérieur du conteneur, donc ici (puisque'il s'agit du conteneur principal) avec celui de la fenêtre du navigateur (et plus précisément le bord supérieur situé à l'intérieur de cette fenêtre, ce que l'on appelle aussi la zone client ou *client area* en anglais). L'axe des Y est vertical, de haut en bas, et se confond avec le bord gauche du conteneur.

Ces coordonnées sont relatives au canevas dans lequel l'élément est directement inséré. Un canevas peut en effet être imbriqué dans un autre canevas.

L'unité est le pixel. Dans l'exemple présenté ici, les attributs `Canvas.Left` et `Canvas.Top`, mais aussi `Width` et `Height`, contiennent des valeurs entières, ce qui est normal puisqu'il s'agit de pixels. On pourrait cependant spécifier des valeurs décimales (avec le point comme séparateur, par exemple, 50.2). Les décimales prennent toute leur importance lors d'opérations dites de *stretching* qui permettent d'agrandir ou de rétrécir des éléments UI, y compris le canevas (voir la section « Les transformations » du chapitre 9).

Deux boutons apparaissent désormais dans la fenêtre du navigateur (figure 3-1) :

Figure 3-1



Attributs et propriétés

Dans la mesure où les balises du XAML correspondent à des objets (manipulables par programme) connus du run-time Silverlight (ici, un objet `Canvas` et deux objets `Button`), on peut confondre « balise » et « objet ». De même, les attributs de ces balises correspondent aux propriétés de ces objets. C'est le cas, par exemple, pour `Width` et `Height`. Les termes « attributs » et « propriétés » seront donc employés indifféremment pour désigner les mêmes choses.

Cependant, les attributs `Canvas.Left` et `Canvas.Top` ne correspondent pas à des propriétés intrinsèques d'un objet `Button`. On parle alors de propriétés attachées car il s'agit de propriétés d'un bouton relatives à un autre objet (ici, le canevas conteneur). La conséquence de cette distinction est que les propriétés attachées sont lues et modifiées différemment (avec `SetValue`, voir chapitre 6).

Nous aurons l'occasion de présenter des exemples où différents canevas (mais aussi d'autres conteneurs) se superposent avec des effets de transparence. Les graphistes sont en effet devenus experts dans ces combinaisons. Ces différentes couches (*layers* en anglais) superposées (qui apparaissent comme une seule à l'utilisateur) permettront de réaliser des effets visuels ou des animations spectaculaires.

Le `StackPanel`

Dans le conteneur qu'est le `StackPanel` (littéralement « panneau d'empilement »), les éléments UI sont disposés :

- les uns au-dessous des autres si l'attribut `Orientation` a pour valeur `Vertical`, ce qui est le cas par défaut ;

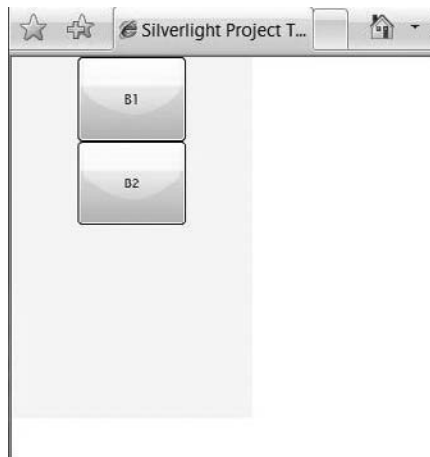
- les uns à côté des autres (de gauche à droite) si l'attribut `Orientation` a pour valeur `Horizontal`.

Par exemple (l'attribut `Orientation` étant ici superflu puisqu'il s'agit de la valeur par défaut) :

```
<StackPanel Background="Beige" Orientation="Vertical">
  <Button Content="B1" Width="90" Height="70" />
  <Button Content="B2" Width="90" Height="70" />
</StackPanel>
```

La figure 3-2 montre le résultat obtenu dans le navigateur :

Figure 3-2



Ici, il n'est plus nécessaire de spécifier la position exacte des composants car le `StackPanel` s'en charge automatiquement.

En général, un `StackPanel` est inséré dans un canevas ou, plus souvent, une grille (pour réaliser un menu, par exemple). Il est en effet rarement repris comme conteneur principal.

Si un panneau à orientation verticale a une largeur de `W` pixels et est inséré dans une grille ou une cellule de grille, il est par défaut centré horizontalement dans son conteneur et occupe toute la hauteur de celui-ci (la propriété `VerticalAlignment` du `StackPanel` prend alors automatiquement la valeur `Stretch`). Ce comportement par défaut peut être modifié en ajoutant l'attribut `HorizontalAlignment` et en lui attribuant la valeur `Left` ou `Right` (`Center` étant la valeur par défaut). `VerticalAlignment` peut aussi prendre les valeurs `Top`, `Center` et `Bottom` et dans ce cas, `Height` doit être spécifié.

Si vous spécifiez une hauteur pour ce panneau à orientation verticale, celui-ci sera accolé au bord supérieur ou au bord inférieur en fonction des valeurs `Top` ou `Bottom` de l'attribut `VerticalAlignment`. La valeur par défaut (en l'absence de `Height`) est `Stretch`, qui adapte automatiquement la hauteur du panneau à celle de son conteneur (et lui fait donc occuper

toute la hauteur du conteneur). En présence de Height, la valeur par défaut de VerticalAlignment est Center.

Un raisonnement analogue s'applique au panneau à orientation horizontale : le StackPanel occupe toute la surface de son conteneur (grille ou cellule de grille) si les attributs Width et Height sont absents (HorizontalAlignment et VerticalAlignment valent alors Stretch). Si le panneau horizontal a une hauteur de H pixels, il est centré verticalement et occupe toute la largeur de son conteneur (HorizontalAlignment vaut Stretch et VerticalAlignment vaut Center). Il faut faire passer VerticalAlignment à Top pour que le panneau soit accolé au bord supérieur de son conteneur.

Si le StackPanel est contenu dans un canevas, il faut spécifier Width et Height en attributs du StackPanel.

Dans l'extrait de code XAML précédent, les boutons sont accolés les uns aux autres, sans la moindre marge. Ce problème peut être aisément résolu en ajoutant un attribut Margin (ici dans la balise Button mais de manière plus générale dans n'importe quel élément UI). Cet attribut ajoute des marges à chaque bord de l'élément UI, celles-ci étant spécifiées comme indiqué dans le tableau 3-1.

Tableau 3-1 – Spécifications des marges via l'attribut Margin

Marges autour d'un élément UI	
Margin="10"	Marge de 10 pixels autour de chaque bord.
Margin="10, 20"	Marge de 10 pixels à gauche et à droite, et marge de 20 pixels au-dessus et au-dessous.
Margin="10, 20, 30, 40"	Marge de 10 pixels à gauche, de 20 pixels au-dessus, de 30 pixels à droite et de 40 pixels au-dessous.

Du point de vue de la programmation, Margin est de type « structure Thickness ». Ses champs Left, Top, Right et Bottom correspondent respectivement aux bords gauche, supérieur, droit et inférieur.

Nous venons de voir comment un StackPanel est inséré dans son conteneur. Voyons maintenant comment les éléments UI (boutons, etc.) sont accrochés dans un StackPanel.

Si un StackPanel à orientation verticale a une largeur de W pixels, les éléments UI insérés dans ce StackPanel :

- seront étendus à une largeur de W pixels s'ils ne contiennent pas d'attribut Width ;
- seront par défaut centrés dans le StackPanel s'ils ont un attribut Width inférieur à W pixels. En ajoutant l'attribut HorizontalAlignment dans la balise de l'élément UI, on obtient un alignement à gauche avec la valeur Left de cet attribut et à droite avec Right (la valeur par défaut est Center) ;
- ne seront que partiellement affichés si Width est supérieur à W pixels.

De même, dans le cas d'éléments UI insérés dans un StackPanel à orientation horizontale, vous pouvez spécifier VerticalAlignment avec ses valeurs Top, Bottom et Center (cette dernière étant la valeur par défaut).

Disons déjà qu'il en va de même pour les cellules de grille.

Si le texte contenu dans un `TextBlock` (zone d'affichage) ou un `TextBox` (zone d'édition) paraît aligné à gauche, cela est dû à la propriété `TextAlign` de ces deux composants (la valeur par défaut de `TextAlign` est `Left`) et cela n'a rien à voir avec `HorizontalAlignment` et `VerticalAlignment`.

La grille

Définition de la grille

La grille (balise `Grid` du XAML) constitue le conteneur le plus puissant. Sous réserve de supprimer les attributs `Width` et `Height` (dans la balise `Grid` mais aussi dans la balise `UserControl`), elle occupe toute la fenêtre du navigateur et s'adapte automatiquement aux redimensionnements effectués par l'utilisateur. La grille présente des similitudes avec la balise `table` du HTML, lorsque la largeur de celle-ci est de 100 %.

Pour définir une grille, il suffit de spécifier :

- le nombre de rangées : autant de rangées que de balises `RowDefinition` dans la balise `Grid.RowDefinitions` ;
- le nombre de colonnes : autant de colonnes que de balises `ColumnDefinition` dans la balise `Grid.ColumnDefinitions`.

En l'absence de balise `Grid.RowDefinitions`, la grille ne comporte qu'une seule rangée, et une seule colonne si la balise `Grid.ColumnDefinitions` est absente. En l'absence de ces deux balises, la grille se résume à une seule cellule.

Pour chaque élément UI, il convient également d'indiquer dans quelle cellule il se trouve : numéro de rangée (attribut `Grid.Row`) et numéro de colonne (attribut `Grid.Column`). La numérotation commence à zéro. La valeur par défaut de `Grid.Row` et de `Grid.Column` est zéro.

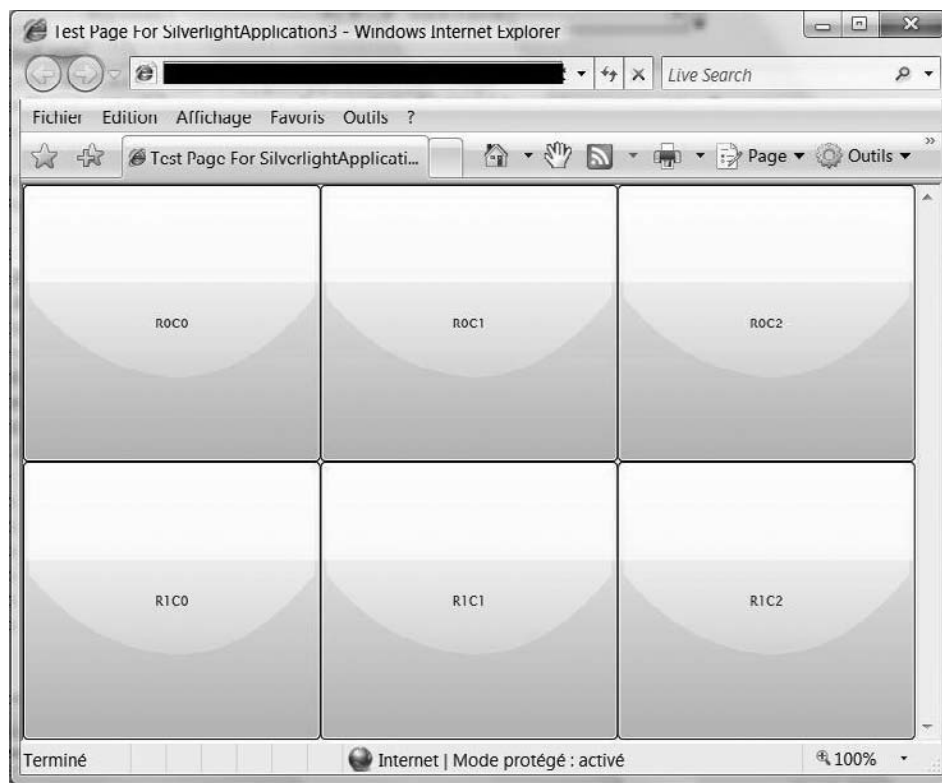
Par exemple (il s'agit ici de la forme la plus basique de balise `Grid`, avec six boutons répartis sur deux rangées et trois colonnes et sans rien préciser quant aux tailles relatives des cellules) :

```
<Grid Background="AliceBlue" >
  <Grid.RowDefinitions> <!-- deux rangées -->
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions> <!-- de trois colonnes -->
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <!-- contenu des six cellules -->
  <Button Content="ROC0" Grid.Row="0" Grid.Column="0" />
  <Button Content="ROC1" Grid.Row="0" Grid.Column="1" />
```

```
<Button Content="ROC2" Grid.Row="0" Grid.Column="2" />
<Button Content="R1C0" Grid.Row="1" Grid.Column="0" />
<Button Content="R1C1" Grid.Row="1" Grid.Column="1" />
<Button Content="R1C2" Grid.Row="1" Grid.Column="2" />
</Grid>
```

La figure 3-3 montre le résultat obtenu dans le navigateur (par défaut, un élément UI dont on ne spécifie pas la taille occupe toute la surface de sa cellule).

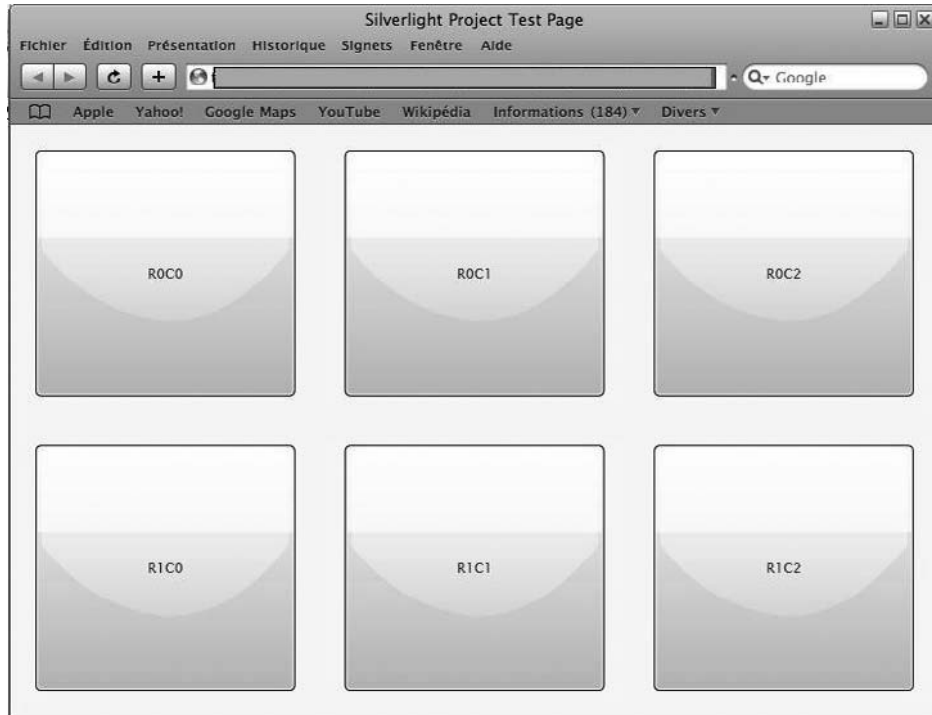
Figure 3-3



La grille occupe toute la fenêtre du navigateur et s'adapte automatiquement à celle-ci. De même, chaque bouton dans une cellule occupe toute la surface de sa cellule et s'adapte automatiquement à la taille de celle-ci. Rappelons que ceci n'est possible que si les attributs `Width` et `Height` ont été supprimés des balises `UserControl` et `Grid`, ainsi que dans celles des éléments UI.

La présentation obtenue serait plus satisfaisante en ajoutant un attribut `Margin` à chaque bouton (ces derniers ne seraient plus accolés les uns aux autres). La figure 3-4 montre le résultat obtenu dans Safari après ajout de `Margin="20"` à chaque balise `Button`.

Figure 3-4



L'utilisateur peut donc redimensionner la fenêtre du navigateur et voir la grille ainsi que son contenu s'adapter en conséquence (les boutons deviennent plus petits ou plus grands). Cependant, il n'en va pas de même pour la police de caractères des libellés, dont la taille ne s'adapte pas automatiquement.

Dans une grille, les attributs `Canvas.Left` et `Canvas.Top` ne sont pas pris en compte, même s'ils sont présents dans une balise d'élément UI. L'attribut `Canvas.ZIndex` (attribut de positionnement relatif des couches superposées, voir la section « Les rectangles et les ellipses » du chapitre 5) est cependant pris en compte.

Comme pour les autres conteneurs, une couleur de fond (attribut `Background`) peut être spécifiée. Des lignes en pointillés sont affichées et délimitent les rangées et colonnes si l'attribut `ShowGridLines` vaut `true` (`false` étant la valeur par défaut car cette possibilité n'est généralement retenue que durant le développement de l'application).

Si plusieurs composants occupent une même cellule (mêmes valeurs dans `Grid.Row` et `Grid.Column`), ils se superposent, ce qui crée autant de couches éventuellement transparentes. Ils peuvent être décalés les uns par rapport aux autres en appliquant une transformation de type `TranslateTransform` (voir la section « Les transformations » du chapitre 9). Pour que ces composants soient placés côte à côte, il faut les insérer dans un `StackPanel` (ou un `Canvas`), lui-même inséré dans la cellule de la grille.

Pour placer un élément UI (par exemple, un bouton) à un emplacement bien précis d’une cellule de grille, par exemple en (10, 20), on spécifie pour cet élément UI :

- `HorizontalAlignment` à `Left` ;
- `VerticalAlignment` à `Top` ;
- `Margin` à `"10, 20, 0, 0"`.

Pour qu’une cellule particulière de la grille ait une couleur de fond (ou un dégradé de couleurs ou encore une image), il suffit d’insérer au préalable un rectangle de la couleur désirée (attribut `Fill`) dans la cellule mais sans lui spécifier de largeur et de hauteur pour ce rectangle. D’autres éléments UI peuvent être insérés dans cette cellule et ils apparaîtront en superposition de ce rectangle.

Il n’est pas rare de rencontrer des grilles sans `Grid.RowDefinitions` ni `Grid.ColumnDefinitions`. Cette dernière est alors limitée à une seule cellule. Les composants « contrôles utilisateurs » (*user control*, voir chapitre 14) sont souvent réalisés de cette manière, avec la seule cellule d’une grille comme conteneur. Le composant s’adapte ainsi automatiquement à son conteneur si `Width` et `Height` sont absents ou prend une taille spécifique si `Width` et `Height` sont mentionnés dans la balise du contrôle utilisateur.

Taille minimale et maximale

Il est possible de spécifier une taille minimale et maximale pour un élément UI, y compris pour le conteneur principal. Dans ce dernier cas, l’utilisateur ne pourra pas réduire sa fenêtre de navigateur au-dessous de cette valeur (pour `MinWidth` et `MinHeight`) ou l’agrandir au-delà de cette valeur (pour `MaxWidth` et `MaxHeight`).

Tableau 3-2 – Les attributs de taille minimale et maximale

Attributs de taille minimale et maximale	
<code>MinWidth</code> <code>MinHeight</code>	Taille minimale. S’il s’agit d’attributs du conteneur principal, l’utilisateur ne pourra pas redimensionner sa fenêtre et la rétrécir au-delà de cette valeur.
<code>MaxWidth</code> <code>MaxHeight</code>	Taille maximale. S’il s’agit d’attributs du conteneur principal, l’utilisateur pourra redimensionner sa fenêtre et l’agrandir au-delà de cette taille mais le conteneur restera limité à (<code>MaxWidth</code> , <code>MaxHeight</code>). Par défaut, le conteneur est alors centré horizontalement dans la fenêtre du navigateur mais ce comportement dépend en fait de l’attribut <code>HorizontalAlignment</code> (qui peut prendre les valeurs <code>Center</code> , <code>Left</code> et <code>Right</code>).

`MinWidth` et `MinHeight` sont bien plus souvent spécifiés que `MaxWidth` et `MaxHeight`.

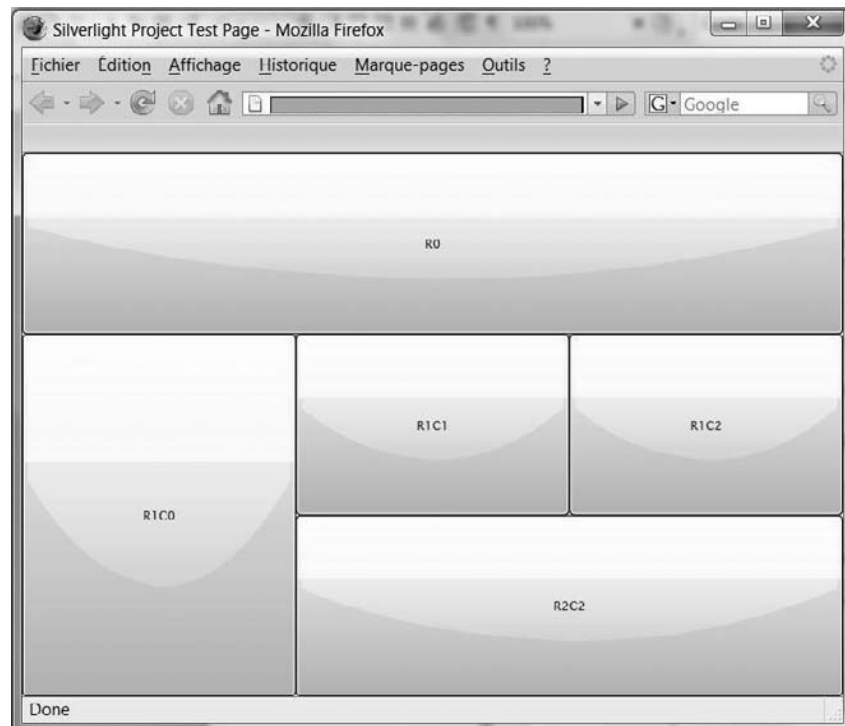
Les propriétés `ActualWidth` et `ActualHeight` indiquent la largeur et la hauteur lors de la lecture de la propriété. Elles ne peuvent donc être utilisées qu’en cours d’exécution de programme. Employez donc bien `ActualWidth` et `ActualHeight` et non `Width` et `Height`. En effet, si les attributs `Width` et `Height` ne sont pas spécifiés dans la balise `UserControl` ou dans la balise du conteneur, les propriétés `Width` et `Height` ne contiennent pas de valeur numérique (pour tester cette absence de valeur, il faut passer par `Width.Equals(Double.NaN)` qui renvoie `true` dans ce cas).

Les attributs RowSpan et ColumnSpan

Comme pour l'attribut `table` du HTML, il est possible d'étendre une cellule de grille sur plusieurs colonnes (attribut `Grid.ColumnSpan`) ou sur plusieurs rangées (attribut `Grid.RowSpan`). Ci-après, une grille de trois lignes et de trois colonnes mais avec fusion de certaines cellules :

```
<Grid Background="AliceBlue" >
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Button Content="R0" Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3" />
  <Button Content="R1C0" Grid.Row="1" Grid.Column="0" Grid.RowSpan="2" />
  <Button Content="R1C1" Grid.Row="1" Grid.Column="1" />
  <Button Content="R1C2" Grid.Row="1" Grid.Column="2" />
  <Button Content="R2C2" Grid.Row="2" Grid.Column="1" Grid.ColumnSpan="2" />
</Grid>
```

Figure 3-5



La figure 3-5 montre le résultat obtenu dans le navigateur :

- la première cellule de la première rangée est étendue à trois colonnes (attribut `Grid.ColumnSpan` à 3 pour cette cellule) ;
- la première cellule de la deuxième rangée est étendue sur deux rangées (attribut `Grid.RowSpan` à 2 pour cette cellule) ;
- la deuxième cellule de la troisième rangée est étendue à deux colonnes (attribut `Grid.RowSpan` à 2 pour cette cellule).

Les composants peuvent s'étendre sur plusieurs cellules, indépendamment de ce qui est spécifié pour les cellules de la grille. Il suffit de spécifier `Grid.RowSpan` et/ou `Grid.ColumnSpan` dans la balise du composant.

Largeur et hauteur des rangées et des colonnes

Il est possible de spécifier des hauteurs et/ou des largeurs de manière absolue ou relative en ajoutant un attribut `Height` dans une balise `RowDefinition` et/ou un attribut `Width` dans une balise `ColumnDefinition`.

Tableau 3-3 – Spécification des hauteurs et largeurs des rangées et colonnes

Hauteur de rangée (Height) / largeur de colonne (Width)	
Height="Auto"	La rangée s'adapte en hauteur à son contenu, autrement dit au plus grand (ici, en hauteur) des éléments UI de la rangée (on peut cependant encore ajouter un attribut <code>Margin</code> à ce contenu).
Height="100"	La hauteur de la rangée est de 100 pixels.
Height="3*"	Avec le signe *, on spécifie une hauteur (ou une largeur avec <code>Width</code>) en relation avec d'autres rangées (ou colonnes) dont la hauteur (ou la largeur) est spécifiée de la même manière. L'interpréteur XAML de Silverlight calcule la somme des * et, à partir de là, un pourcentage (voir exemple ci-dessous). La valeur préfixant * peut être décimale.

Le comportement est semblable pour l'attribut `Width`.

Considérons l'exemple suivant pour illustrer les hauteurs et largeurs relatives :

```
<Grid Background="AliceBlue" >
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="3*" />
    <RowDefinition Height="7*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="200" />
    <ColumnDefinition Width="2*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Button Content="R0C1" Grid.Row="0" Grid.Column="1" Width="100" Height="80" />
  <Button Content="R1C0" Grid.Row="1" Grid.Column="0" />
  <Button Content="R1C1" Grid.Row="1" Grid.Column="1" />
  <Button Content="R1C2" Grid.Row="1" Grid.Column="2" />
</Grid>
```

```
<Button Content="R2C0" Grid.Row="2" Grid.Column="0" />
<Button Content="R2C1" Grid.Row="2" Grid.Column="1" />
<Button Content="R2C2" Grid.Row="2" Grid.Column="2" />
</Grid>
```

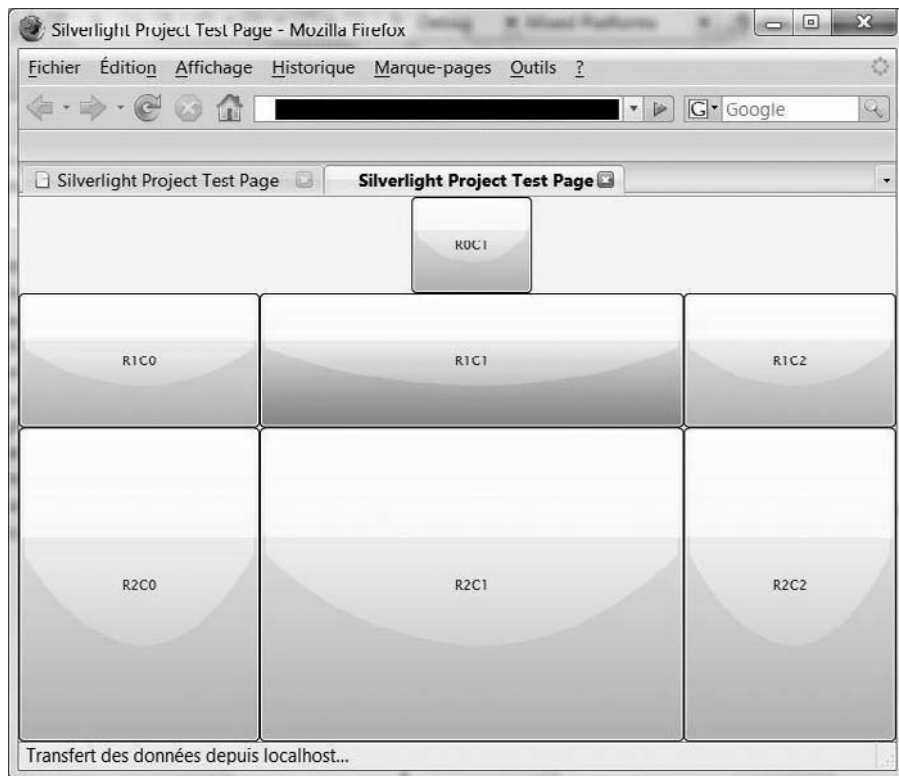
La hauteur de la première rangée dépend de son contenu, ici, un bouton d'une hauteur de 80 pixels. En l'absence de marges pour ce bouton (l'attribut `Margin` est absent), la première ligne a donc une hauteur de 80 pixels. La hauteur restante (dans la fenêtre du navigateur puisqu'il s'agit du conteneur principal) est partagée entre les deux autres lignes : 30 % pour la deuxième (3 divisé par 3 + 7) et 70 % pour la troisième (7 divisé par 3 + 7).

La largeur de la première colonne a été fixée à 200 pixels. La largeur restante (dans la fenêtre du navigateur ou dans le conteneur de la grille) est partagée entre les deux autres colonnes : 66 % pour la deuxième (2 divisé par 2 + 1) et 33 % pour la troisième (1 divisé par 2 + 1).

Rappelons que ces proportions seront respectées, même après un redimensionnement de la fenêtre du navigateur par l'utilisateur.

La figure 3-6 montre la page Web obtenue (bien sûr, dans la pratique, les cellules contiendront des éléments bien plus intéressants).

Figure 3-6



Rien ne s'affiche dans les première et troisième cellules de la première rangée puisque rien n'est spécifié pour celles-ci, mais on aurait pu préciser :

- `MinHeight` et `MaxHeight` dans une balise `RowDefinition`, ce qui donne respectivement la hauteur minimale et la hauteur maximale de cette rangée ;
- `MinWidth` et `MaxWidth` dans une balise `ColumnDefinition`, ce qui donne respectivement la largeur minimale et la largeur maximale de cette colonne.

Les grilles imbriquées

Les attributs `Grid.RowSpan` et `Grid.ColumnSpan` n'offrent pas toujours la souplesse nécessaire (on rencontre en fait le même problème avec `RowSpan` et `ColumnSpan` du HTML). En effet, il est parfois nécessaire de placer une grille à l'intérieur d'une cellule, notamment pour donner aux cellules des largeurs différentes d'une rangée à l'autre.

Dans l'exemple suivant, nous forçons :

- dans la grille extérieure (qui va occuper toute la fenêtre du navigateur) : trois rangées d'une seule colonne (aucune balise `Grid.ColumnDefinitions` n'est alors nécessaire) ;
- en première rangée (attribut `Grid.Row="0"`), une grille de deux colonnes est insérée sur une seule rangée (aucune balise `Grid.RowDefinitions` n'est alors nécessaire pour cette dernière grille) ;
- en deuxième rangée, une grille limitée à une seule cellule est insérée (aucun besoin dès lors de `Grid.RowDefinitions` et `Grid.ColumnDefinitions` pour cette dernière grille) ;
- en troisième rangée, une grille de trois colonnes est insérée sur une seule rangée.

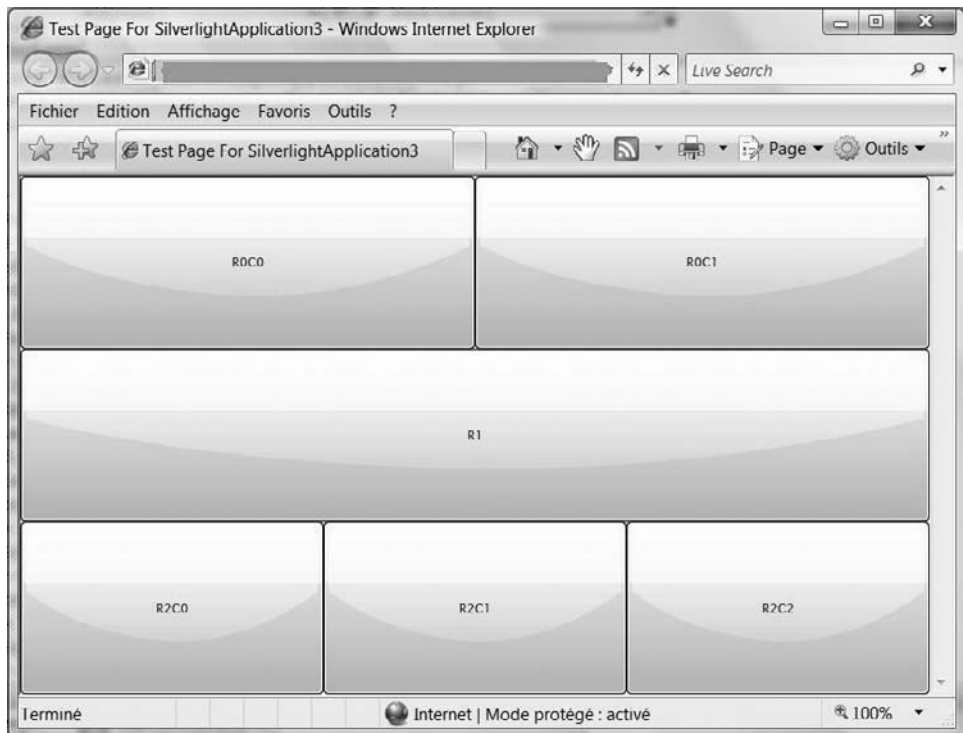
Le XAML correspondant à cette grille est le suivant :

```
<Grid> <!-- grille externe de trois rangées -->
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid Grid.Row="0"> <!-- grille en première rangée -->
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Content="ROC0" Grid.Row="0" Grid.Column="0" />
    <Button Content="ROC1" Grid.Row="0" Grid.Column="1" />
  </Grid>
  <Grid Grid.Row="1"> <!-- grille en deuxième rangée -->
    <Button Content="R1" Grid.Row="0" />
  </Grid>
  <Grid Grid.Row="2"> <!-- grille en troisième rangée -->
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
```

```
</Grid.ColumnDefinitions>  
<Button Content="R2C0" Grid.Row="2" Grid.Column="0" />  
<Button Content="R2C1" Grid.Row="2" Grid.Column="1" />  
<Button Content="R2C2" Grid.Row="2" Grid.Column="2" />  
</Grid>  
</Grid>
```

La figure 3-7 illustre le résultat obtenu dans le navigateur (à noter que pour cet exemple encore simple, on aurait pu s'en sortir avec des `RowSpan`).

Figure 3-7



L'étude des trois conteneurs de base étant terminée, passons maintenant en revue d'autres conteneurs aux utilisations plus spécifiques.

Conteneurs spécifiques

Le *GridSplitter*

Grâce au composant `GridSplitter`, l'utilisateur peut redimensionner les rangées et les colonnes d'une grille. Il lui suffit pour cela de cliquer sur une ligne de séparation et de la déplacer ensuite tout en maintenant le bouton de la souris enfoncé. Les utilisateurs de Windows et de nombreux logiciels sont habitués à ce type de manipulation.

Le `GridSplitter` ne fait pas partie du run-time Silverlight. Lorsque vous l'utilisez (par un glisser-déposer à partir de la boîte d'outils de Visual Studio), celui-ci ajoute le code suivant (en gras) :

```
<UserControl
  xmlns:my="clr-namespace:System.Windows.Controls;assembly=System.Windows.
    Controls.Extended"
  x:Class="SLProg.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
```

Ceci signifie que le code du `GridSplitter` ne fait pas partie du run-time Silverlight et doit de ce fait être inclus dans une DLL séparée, celle-ci étant incorporée dans le fichier XAP envoyé à l'application. De plus, la balise devra être préfixée (vous pouvez choisir un autre préfixe que `my` à condition de modifier la ligne `xmlns:my` dans le `UserControl`) :

```
<my:GridSplitter> ..... </my:GridSplitter>
```

Ainsi, on définit la grille (avec ses rangées et ses colonnes) et ensuite le `GridSplitter`.

Les attributs importants de `GridSplitter` sont `HorizontalAlignment` et `VerticalAlignment`. Si l'attribut `HorizontalAlignment` a pour valeur `Stretch`, l'utilisateur peut redimensionner une rangée. Si l'attribut `VerticalAlignment` a pour valeur `Stretch`, il peut redimensionner une colonne.

Analysons l'exemple suivant :

```
<Grid x:Name="LayoutRoot" Background="Beige" >
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>

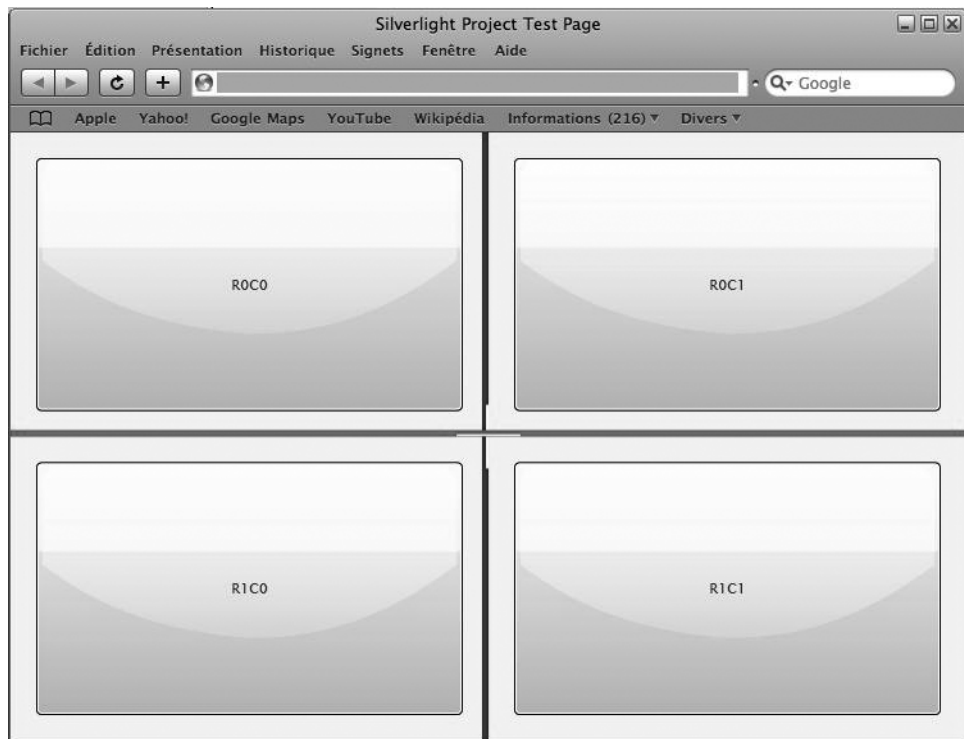
  <Button Content="R0C0" Grid.Row="0" Grid.Column="0" Margin="20" />
  <Button Content="R0C1" Grid.Row="0" Grid.Column="1" Margin="20" />
  <Button Content="R1C0" Grid.Row="1" Grid.Column="0" Margin="20" />
  <Button Content="R1C1" Grid.Row="1" Grid.Column="1" Margin="20" />
  <my:GridSplitter Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
    Width="5" Background="Blue"
    HorizontalAlignment="Right" VerticalAlignment="Stretch" />
  <my:GridSplitter Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2"
    Height="5" Background="Red"
    VerticalAlignment="Bottom" HorizontalAlignment="Stretch" />
</Grid>
```

Ce code permet de créer :

- une première ligne de séparation verticale (`VerticalAlignment` a pour valeur `Stretch`) de couleur bleue (`Background`) et épaisse de 5 pixels (`Width`). La ligne de séparation s'étend sur deux rangées (`Grid.RowSpan`) et se trouve à droite (`HorizontalAlignment` à `Right`) de la cellule en (0, 0) ;
- une ligne de séparation horizontale (`HorizontalAlignment` a pour valeur `Stretch`) de couleur rouge et haute (`Height`) de 5 pixels. La ligne de séparation s'étend sur deux colonnes (`Grid.ColumnSpan`) et se trouve sous (`VerticalAlignment` à `Bottom`) la cellule en (0, 0).

La figure 3-8 illustre le résultat obtenu.

Figure 3-8



Le curseur de la souris se transforme en double flèche lorsqu'il survole une ligne de séparation ; si l'utilisateur clique et maintient le bouton de la souris enfoncé, il peut redimensionner les cellules. La manipulation est donc bien intuitive et conforme à l'usage.

Même si `Grid.RowSpan`, ou `Grid.ColumnSpan`, vaut 1 (ou si cet attribut est absent, ce qui revient au même), la ligne de séparation est limitée à une seule cellule mais le redimensionnement implique toutes les cellules de la rangée ou de la colonne.

Si l'attribut `ShowPreview` du `GridSplitter` vaut `true`, la ligne de séparation suit la souris mais les cellules ne sont redimensionnées qu'après relâchement du bouton de la souris.

Intéressons-nous à présent à deux « petits » conteneurs.

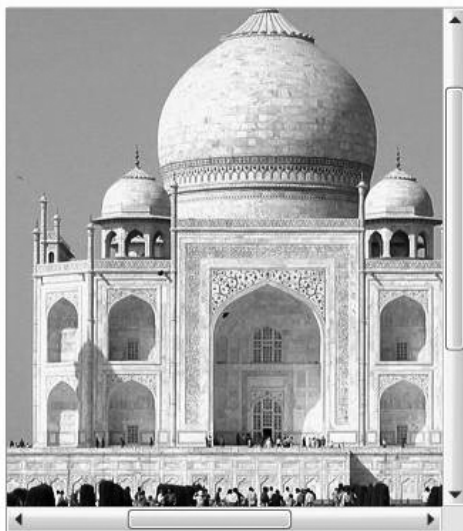
Le composant *ScrollView*

Le composant `ScrollView` peut contenir des composants (image, canevas, grille, etc.) qu'il peut faire défiler dans une zone de taille plus réduite. L'utilisateur peut visualiser tout le contenu à l'aide de barres de défilement.

Voici un exemple de code dans lequel le `ScrollView` est inséré dans la cellule en deuxième colonne de la deuxième rangée pour pouvoir visualiser une image de grande taille (figure 3-9) :

```
<ScrollView Grid.Row="1" Grid.Column="1"
    HorizontalScrollBarVisibility="Auto"
    VerticalScrollBarVisibility="Auto" >
    <ScrollView.Content>
        <Image Source="TajMahal.jpg" Stretch="Fill" />
    </ScrollView.Content>
</ScrollView>
```

Figure 3-9



Si les attributs `HorizontalScrollBarVisibility` et `VerticalScrollBarVisibility` ont la valeur `Auto` (il s'agit de l'une des valeurs de l'énumération `ScrollBarVisibility`), les barres de défilement ne seront affichées qu'en cas de nécessité (lorsque la taille du `ScrollView` est inférieure à celle du contenu). Avec la valeur `Visible`, la barre est systématiquement affichée. Les valeurs `Hidden` (pour caché) et `Disabled`, quant à elles, rendent les barres de défilement inopérantes.

Les propriétés `HorizontalOffset` et `VerticalOffset` spécifient les coordonnées du point qui est affiché dans le coin supérieur gauche du `ScrollView`. Ces coordonnées sont relatives à son contenu, ici l'image. Ces deux propriétés sont en lecture seule.

Le composant *Border*

Le composant `Border` crée une enveloppe pour des composants.

Analysons l'exemple suivant (sans oublier que le composant `Border` occupe toute la surface de sa cellule si `Width` et `Height` ne sont pas spécifiés) :

```
<Border .....  
    Background="LightGray" CornerRadius="3, 10, 60, 100" >  
    <Button Content="GO !" Width="60" Height="40" />  
</Border>
```

La figure 3-10 illustre le résultat obtenu.

Figure 3-10



En attributs de `Border`, il est possible de spécifier :

- une couleur de fond (`Background`) pour le conteneur qu'est `Border` ;
- une bordure, avec `BorderThickness` pour l'épaisseur (0 par défaut), et une couleur (`BorderBrush` car il s'agit plus précisément d'un pinceau) ;
- des coins arrondis : une (mêmes arrondis à chaque coin), deux (pour des coins opposés) ou quatre valeurs. Dans ce cas, on indique le rayon de l'arrondi pour, successivement, le coin supérieur gauche, le coin supérieur droit, le coin inférieur droit et le coin inférieur gauche.

4

Couleurs et pinceaux

Dans ce chapitre, nous allons voir comment spécifier les couleurs et les pinceaux. Nous étudierons également des techniques avancées telles que les dégradés et les masques d'opacité, permettant de personnaliser l'aspect des composants. À la fin de ce chapitre, vous découvrirez des exemples d'utilisation du logiciel Expression Blend, l'outil graphique générant du XAML et compagnon de Visual Studio et vous verrez comment les graphistes arrivent à obtenir certains effets.

Les couleurs

Une couleur peut être spécifiée comme valeur des attributs suivants :

- **Background** : couleur de fond d'un conteneur mais aussi d'un `TextBox` (zone d'édition) et d'un `Button` ;
- **Fill** : couleur de remplissage d'une figure géométrique comme `Rectangle` ou `Ellipse` ;
- **Foreground** : couleur de premier plan d'un `TextBlock` (zone d'affichage, encore appelée *label*), d'un `TextBox` ou d'un `Button` ;
- **Stroke** : couleur du contour des figures géométriques.

En XAML, la valeur de ces attributs peut être une chaîne de caractères (tout au moins pour les couleurs unies), bien qu'il s'agisse d'objets de la classe `Brush` ou d'une classe dérivée de celle-ci. L'interpréteur XAML est en effet capable de convertir une chaîne de caractères (représentant un nom de couleur) en un objet `Brush`. Nous verrons par la suite que ces attributs peuvent devenir des balises.

Un nom de couleur

Comme en HTML, les noms de couleurs suivants peuvent être utilisés en valeur des attributs mentionnés précédemment :

Tableau 4-1 – Les noms de couleurs en XAML

AliceBlue	AntiqueWhite	Aqua	AquaMarine	Azure
Beige	Bisque	Black	BlanchedAlmond	Blue
BlueViolet	Brown	BurlyWood		
CadetBlue	Chartreuse	Chocolate	Coral	Cornflower
Cornsilk	Crimson	Cyan		
DarkBlue	DarkCyan	DarkGoldenrod	Darkgray	DarkGreen
DarkKhaki	DarkMagenta	DarkOliveGreen	DarkOrange	DarkOrchid
DarkRed	DarkSalmon	DarkSeaGreen	DarkSlateBlue	DarkSlateGray
DarkTurquoise	DarkViolet	DeepPink	DeepSkyBlue	DimGray
DodgerBlue				
FireBrick	FloralWhite	ForestGreen	Fuchsia	Gainsboro
GhostWhite	Gold	Goldenrod	Gray	Green
GreenYellow				
Honeydew	HotPink			
IndianRed	Indigo	Ivory		
Khaki				
Lavender	LavenderBlush	LawnGreen	LemonChiffon	LightBlue
LightCoral	LightCyan	LightGoldenrodYellow	LightGray	LightGreen
LightPink	LightSalmon	LightSeaGreen	LightSlateGray	LightSteelBlue
LightYellow	Lime	LimeGreen	Linen	
Magenta	Maroon	MediumAquaMarine	MediumBlue	MediumOrchid
MediumPurple	MediumSeaGreen	MediumSlateBlue	MediumSpringGreen	MediumTurquoise
MediumVioletRed	MidnightBlue	MintCream	MistyRose	Moccasin
NavajoWhite	Navy			
OldLace	Olive	OliveDrab	Orange	OrangeRed
Orchid				
PaleGoldenrod	PaleGreen	PaleTurquoise	PaleVioletRed	PapayaWhip
PeachStiff	Peru	Pink	Plum	PowderBlue
Purple				
Red	RosyBrown	RoyalBlue		
SaddleBrown	Sienna	Silver	SkyBlue	SlateBlue
Tan	Teal	Thistle	Tomato	
Violet				
Wheat	White	WhiteSmoke		
Yellow	YellowGreen			

Les représentations sRGB et scRGB

Les 130 noms de couleurs du tableau 4-1 ne permettent pas de spécifier une couleur avec toutes les nuances souhaitées. Il existe dès lors d'autres moyens de spécifier une couleur : les représentations sRGB et scRGB. Dans les deux systèmes, une couleur est spécifiée par une combinaison de trois couleurs de base : rouge, vert et bleu (respectivement *red*, *green* et *blue*).

Un indice de transparence (aussi appelé *alpha channel*) peut également être spécifié. Si une couleur est totalement ou partiellement transparente, le fond reste totalement ou partiellement visible à l'emplacement des pixels de cette « couleur ».

Le système SRGB

Dans ce système, chaque quantité de rouge, de vert et de bleu est spécifiée par une valeur entière comprise entre 0 et 255. Une valeur de couleur, préfixée par le caractère #, est indiquée par trois ou quatre chiffres hexadécimaux (quatre si un *alpha channel* est spécifié pour la transparence) : #rrggbb ou #aarrggbb.

Il est à craindre que seuls ceux qui, en d'autres temps, ont pratiqué l'assembleur dans la douleur et l'abnégation qui seyaient à ce genre de programmation savent encore, pour en avoir vu l'intérêt, ce qu'est un nombre en représentation hexadécimale. L'hexadécimal permet de représenter les nombres de 0 à 15 à l'aide d'un seul signe : 0 à 9 (pour 0 à 9) puis A, B, C, D, E et F (pour les valeurs allant de 10 à 15). Ainsi :

- hexadécimal A (équivalent à 0A) correspond à la valeur (décimale) 10 ;
- hexadécimal 12 correspond à 18 ($1 \times (16 + 2)$) ;
- hexadécimal A4 correspond à 164 ($10 \times (16 + 4)$) ;
- hexadécimal FF correspond à 255 ($15 \times (16 + 15)$).

Le noir peut ainsi être représenté par #000000 (tous pixels éteints) et le blanc par #FFFFFF (tous pixels allumés) ; le rouge par #FF0000 (premier des trois octets à hexadécimal FF, équivalent à décimal 255 et deuxième octet ainsi que troisième à zéro) ; le jaune (obtenu par un mélange à parts égales de rouge et de vert) par #FFFF00. Puisque la valeur d'opacité (premier des quatre octets) est absente, ces couleurs sont opaques (autrement dit, le fond n'est pas du tout visible à cet emplacement).

La calculatrice scientifique de Windows peut s'avérer utile pour effectuer des conversions entre représentation décimale et hexadécimale d'un nombre. Mais souvent, les graphistes trouvent plus simple d'utiliser le logiciel Expression Blend, qui permet de sélectionner une couleur de manière bien plus visuelle (voir la section « Les pinceaux dans Expression Blend » de ce chapitre).

Pour ajouter de la transparence (une valeur comprise entre 0 et 255, avec 0 pour transparence totale et 255 pour opacité totale), spécifiez deux chiffres hexadécimaux en tête des trois octets de couleur (avec donc 00 pour fond entièrement visible et FF pour une opacité totale, les valeurs intermédiaires étant évidemment acceptées). Par exemple :

■ Fill = "#80FF0000"

pour du rouge transparent à 50 % (car hexadécimal 80 correspond à décimal 128, moitié de 256).

Seize millions de couleurs environ peuvent ainsi être représentées dans le système sRGB (plus précisément $256 \times 256 \times 256$ couleurs).

Le système scRGB

Dans ce système (avec `sc#` comme préfixe pour une valeur de couleur), chacune des trois couleurs de base est spécifiée par un nombre décimal compris entre 0 et 1. L'opacité est également représentée par un nombre décimal compris entre 0 et 1 (il s'agit alors de la première des quatre valeurs). Ces trois ou quatre valeurs décimales (quatre en cas de transparence) doivent être séparées par une espace ou une virgule (utilisez toujours le point comme séparateur des décimales).

Le nombre de couleurs susceptibles d'être ainsi représentées est substantiellement plus important que dans le système sRGB.

Voici deux exemples de représentations de couleurs en scRGB :

- `Background = "sc#1.0, 0, 0"`, pour du rouge, sans mention de transparence (donc, opacité totale) ;
- `Background = "sc#0.5, 1.0, 1.0, 0"`, pour du jaune semi-transparent.

Par ailleurs, la transparence peut souvent être également spécifiée en attribut `Opacity` de balise (valeur décimale comprise entre 0 et 1).

Les couleurs dans le code

Jusqu'à présent (sauf au chapitre 2), nous n'avons pas encore écrit la moindre ligne de code en C# ou VB. Voyons néanmoins comment spécifier une couleur dans le code.

Une manière simple – mais limitée – consiste à mentionner l'un des quatorze champs statiques de la classe `Colors`. Par exemple, `Colors.Black` (les autres valeurs sont `Blue`, `Brown`, `Cyan`, `DarkGray`, `Gray`, `Green`, `LightGray`, `Magenta`, `Orange`, `Purple`, `Red`, `White` et `Yellow`). Signalons également `Colors.Transparent`, qui correspond à la transparence totale : le fond reste parfaitement visible là où cette « couleur » est présente.

La classe `Color` contient la fonction `FromArgb` dont les quatre arguments (des valeurs entières comprises entre 0 et 255) correspondent respectivement aux composantes alpha (transparence, avec 0 pour transparence et 255 pour opacité), rouge, vert et bleu d'une couleur.

Par exemple, pour du jaune (mélange à parts égales de rouge et de vert) sans la moindre transparence, le code C# sera le suivant :

```
Color c;  
.....  
c = Color.FromArgb(255, 255, 255, 0);
```

et le code VB :

```
Dim c As Color
.....
c = Color.FromArgb(255, 255, 255, 0)
```

Les arguments peuvent être exprimés en décimal ou en hexadécimal. Par exemple, 0xFF en C# et &HFF en VB pour la valeur 255.

Un objet `Color` (même s'il s'agit plus précisément d'une structure) présente les quatre propriétés `A`, `R`, `G` et `B` qui correspondent respectivement au canal alpha (transparence), au rouge, au vert et au bleu. Ces quatre propriétés, dont les valeurs sont comprises entre 0 et 255, peuvent être modifiées. Ainsi, en exécutant :

```
c.A = 64;
```

on rend la couleur contenue dans la variable `c` opaque à 25 % (car 64 représente le quart de 255).

Les pinceaux

Les pinceaux (*brushes* en anglais) sont utilisés pour colorier le fond des éléments visuels (propriété `Fill` ou `Background` en fonction des objets), pour peindre le contour des formes (propriété `Stroke`) ou encore pour colorier les lettres (propriété `Foreground`).

Il est possible de créer des pinceaux de différentes sortes :

- `SolidColorBrush` : couleur unie et aucun effet ;
- `LinearGradientBrush` : dégradé de couleurs le long d'une ligne droite ;
- `RadialGradientBrush` : dégradé de couleurs à partir d'un point ;
- `ImageBrush` : une image est utilisée ;
- `VideoBrush` : une vidéo est jouée.

Nous allons voir comment représenter un pinceau en XAML. Bien qu'il soit plus simple (les graphistes semblent en tout cas préférer cette voie) d'utiliser Expression Blend, il est impératif pour un programmeur de connaître le XAML. Avec un peu d'habitude, vous trouverez peut-être que programmer directement dans ce langage est finalement plus simple...

Les pinceaux `ImageBrush` et `VideoBrush` seront traités au chapitre 7.

Le pinceau `SolidColorBrush`

Il s'agit du plus simple des pinceaux, celui qui permet de colorier de manière uniforme. Voici le code à saisir pour obtenir le pinceau `SolidColorBrush` (pour une couleur nommée) :

- `Background =`, pour un conteneur, un bouton ou une zone d'édition ;
- `Fill =`, pour une forme géométrique comme le rectangle ou l'ellipse ;
- `Foreground =`, pour le premier plan des `TextBox`, `TextBlock` et `Button`.

Pour que le fond du rectangle (voir la section « Les rectangles et les ellipses » du chapitre 5) suivant soit peint en rouge semi-transparent (l'image de fond dans le conteneur reste donc partiellement visible), saisissez le code suivant :

```
<Rectangle ..... Fill="#80FF0000" />
```

La balise précédente pourrait être écrite de manière tout à fait équivalente (puisqu'il s'agit du pinceau par défaut) en saisissant les lignes de code suivantes :

```
<Rectangle ..... >
  <Rectangle.Fill>
    <SolidColorBrush Color="#80FF0000" />
  </Rectangle.Fill>
</Rectangle>
```

La balise `SolidColorBrush` pourrait également s'écrire :

```
<SolidColorBrush Color="sc#0.5, 1, 0, 0" />
```

ou

```
<SolidColorBrush Color="Red" Opacity="0.5" />
```

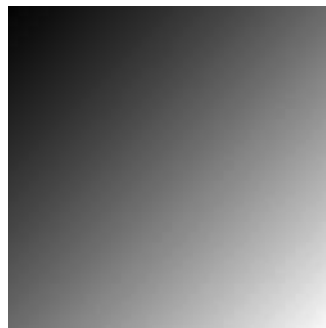
Le pinceau `LinearGradientBrush`

Ce pinceau permet d'obtenir un dégradé de couleurs le long d'une ligne. Dans le cas le plus simple, il suffit de spécifier la couleur au point de départ (*starting point* en anglais) et celle au point d'arrivée (*ending point*). À noter qu'il est également possible de spécifier des couleurs intermédiaires.

L'exemple de code suivant permet d'obtenir un dégradé du noir au blanc (par défaut, du coin supérieur gauche au coin inférieur droit, figure 4-1) :

```
<Rectangle ..... >
  <Rectangle.Fill>
    <LinearGradientBrush>
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Figure 4-1

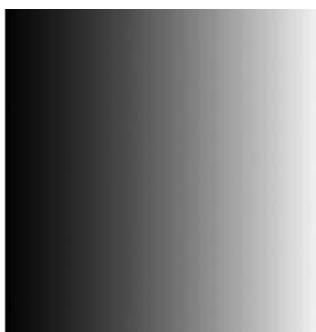


Par défaut, les déplacements (*offset* en anglais) le long de la ligne reliant le point de départ au point d'arrivée sont relatifs : 0 pour le point de départ et 1 pour le point d'arrivée. Par défaut aussi, cette ligne relie le coin supérieur gauche au coin inférieur droit. Les attributs `StartPoint` et `EndPoint` permettent de choisir d'autres points de début et de fin.

Dans l'exemple de code suivant, la ligne de dégradé est horizontale et située à mi-hauteur dans le rectangle (figure 4-2) :

```
<LinearGradientBrush StartPoint="0, 0.5" EndPoint="1, 0.5" >
  <GradientStop Offset="0" Color="Black" />
  <GradientStop Offset="1" Color="White" />
</LinearGradientBrush>
```

Figure 4-2



`StartPoint="0, 0.5"` signifie 0 sur l'axe des X et 0,5 sur l'axe des Y, autrement dit au milieu du bord gauche.

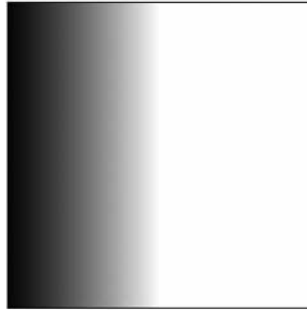
Les déplacements sont néanmoins absolus si l'attribut `MappingMode` de la balise `LinearGradientBrush` a pour valeur `Absolute`. Avec cet attribut ainsi initialisé et pour reprendre l'exemple précédent, `StartPoint` devrait être (0, 75) et `EndPoint` (200, 75) si le rectangle avait une largeur de 200 pixels et une hauteur de 150 pixels.

Les déplacements (attribut `Offset`), de même que `StartPoint` et `EndPoint`, ne sont pas limités à l'intervalle 0 à 1 (ce qui permet de réaliser des dégradés aux effets intéressants) mais rien n'est peint en dehors de l'intervalle relatif 0 à 1.

Dans l'exemple de code suivant, le dégradé suit une ligne horizontale à mi-hauteur (un rectangle au contour noir est affiché pour montrer la différence, figure 4-3) :

```
<Rectangle ..... Stroke="Black" >
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0, 0.5" EndPoint="1, 0.5" >
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="0.5" Color="White" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Figure 4-3



Si le dégradé commence ou finit au milieu d'une figure, quelle sera alors la couleur utilisée au début ou à la fin ? La couleur reprise jusqu'à ou à partir de ces emplacements dépend de l'attribut `SpreadMethod`, pour lequel trois valeurs sont possibles (si aucun attribut `SpreadMethod` n'est spécifié, le comportement est semblable à celui spécifié par la valeur par défaut `Pad`) :

- `Pad` : remplissage au début avec la première couleur spécifiée et à la fin avec la dernière couleur spécifiée (ce qui explique pourquoi la partie droite du rectangle de la figure 4-3 est restée blanche) ;
- `Repeat` : le motif (entre `StartPoint` et `EndPoint`) est répété au début et/ou à la fin ;
- `Reflect` : même comportement qu'avec la valeur `Repeat` mais avec un effet de miroir.

Les figures 4-4, 4-5 et 4-6 illustrent les dégradés obtenus en fonction des différentes valeurs de `SpreadMethod` pour le code suivant :

```
<Rectangle ..... >
<Rectangle.Fill>
  <LinearGradientBrush StartPoint="0.4, 0.5"
                        EndPoint="0.6, 0.5"
                        SpreadMethod="xyz" >
    <GradientStop Offset="0" Color="Black" />
    <GradientStop Offset="1" Color="White" />
  </LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
```

où `xyz` doit être remplacé par l'une des trois valeurs possibles, soit `Pad`, `Repeat` ou `Reflect` (n'oubliez pas qu'`Offset` est relatif à l'intervalle `StartPoint`, `EndPoint`).

Figure 4-4

SpreadMethod="Pad"

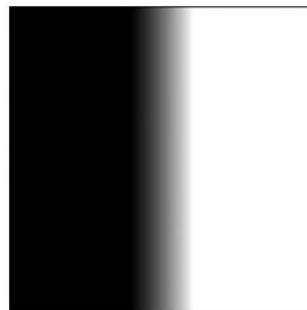
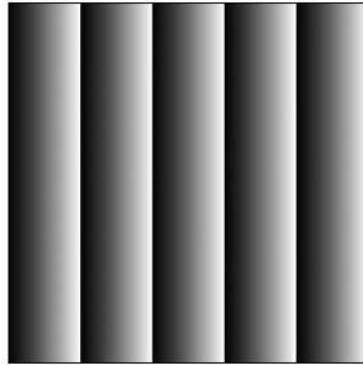
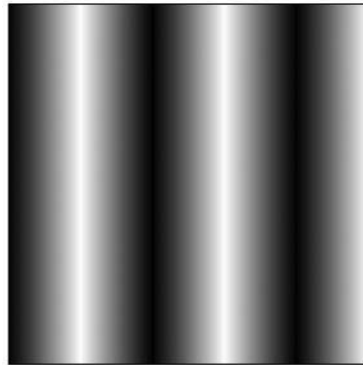


Figure 4-5

SpreadMethod="Repeat"

**Figure 4-6**

SpreadMethod="Reflect"



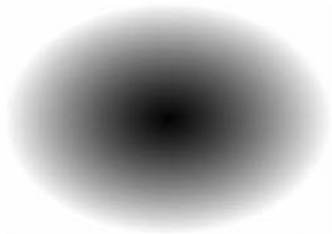
Le pinceau *RadialGradientBrush*

Le pinceau *RadialGradientBrush* diffère du pinceau *LinearGradientBrush* par le fait que le dégradé se fait le long de cercles concentriques. L'attribut *GradientOrigin* permet de spécifier les coordonnées de ce centre. Par défaut, il s'agit du milieu de la figure à laquelle on applique un tel pinceau.

L'exemple de code suivant correspond au cas le plus simple et permet d'obtenir le dégradé de la figure 4-7.

```
<Ellipse ..... >
  <Ellipse.Fill>
    <RadialGradientBrush>
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </RadialGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```


Figure 4-7



Le tableau 4-2 présente les propriétés du pinceau RadialGradientBrush pouvant être spécifiées :

Tableau 4-2 – Les propriétés du pinceau RadialGradientBrush

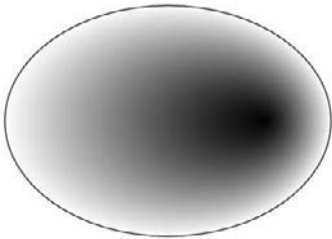
GradientOrigin	Coordonnées du centre des cercles concentriques de dégradé. Par défaut, il s'agit du centre de la figure (point 0.5, 0.5 en coordonnées relatives). Ces coordonnées peuvent néanmoins être exprimées en pixels (mais toujours relativement à la figure) à condition de faire passer l'attribut MappingMode à Absolute.
RadiusX RadiusY	Le dégradé peut être limité à une ellipse dont les rayons sont RadiusX et RadiusY. Par défaut, ces valeurs sont relatives à la figure, RadiusX est égal à la moitié de la largeur de la figure et RadiusY à la moitié de la hauteur.
Center	Coordonnées du centre de la plus grande ellipse de dégradé, cette ellipse ayant RadiusX et RadiusY comme rayons. Si GradientOrigin se trouve en dehors de cette ellipse, le dégradé est limité à cette ellipse plus un cône formé par deux tangentes à cette ellipse à partir de GradientOrigin.

Pour mieux comprendre l'influence de ces quatre propriétés, visualisons le résultat dans des cas pratiques de dégradés radiaux :

Exemple 1 :

```
<Ellipse Stroke="Black" >
  <Ellipse.Fill>
    <RadialGradientBrush
      GradientOrigin="0.8, 0.5" >
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </RadialGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```

Figure 4-8

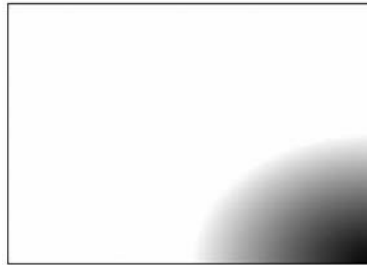


Le dégradé émane (propriété `GradientOrigin`) du point (0.8, 0.5) et s'étend à une ellipse ayant (0.5, 0.5) comme centre et de taille égale à l'ellipse (propriétés `Center`, `RadiusX` et `RadiusY`).

Exemple 2 :

```
<Rectangle Stroke="Black" >  
  <Rectangle.Fill>  
    <RadialGradientBrush GradientOrigin="1, 1"  
      Center="1, 1" >  
      <GradientStop Offset="0" Color="Black" />  
      <GradientStop Offset="1" Color="White" />  
    </RadialGradientBrush>  
  </Rectangle.Fill>  
</Rectangle>
```

Figure 4-9



Le dégradé émane du coin inférieur droit et est limité à une ellipse ayant également son centre au coin inférieur droit. Cette ellipse a pour rayons la moitié de la largeur et la moitié de la hauteur du rectangle (valeur par défaut en l'absence de `RadiusX` et `RadiusY`).

Exemple 3 :

```
<Rectangle Stroke="Black" >  
  <Rectangle.Fill>  
    <RadialGradientBrush GradientOrigin="1, 1"  
      Center="0.5, 1" >  
      <GradientStop Offset="0" Color="Black" />  
      <GradientStop Offset="1" Color="White" />  
    </RadialGradientBrush>  
  </Rectangle.Fill>  
</Rectangle>
```

Figure 4-10

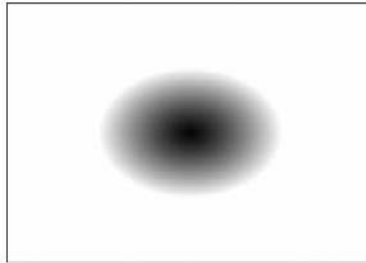


Le dégradé émane du coin inférieur droit et est limité à une ellipse ayant son centre en (0.5, 1).

Exemple 4 :

```
<Rectangle Stroke="Black" >
  <Rectangle.Fill>
    <RadialGradientBrush RadiusX="0.25"
                          RadiusY="0.25" >
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Figure 4-11

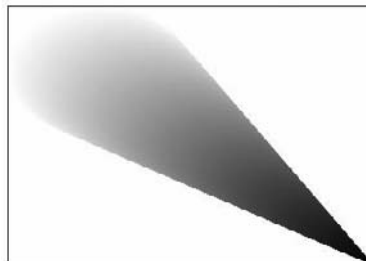


Le dégradé est limité à une ellipse ayant comme rayons le quart de la largeur et de la hauteur de la figure (ici, un rectangle). En l'absence de GradientOrigin, le dégradé radial émane du milieu de la figure.

Exemple 5 :

```
<Rectangle Stroke="Black" >
  <Rectangle.Fill>
    <RadialGradientBrush GradientOrigin="1,1"
                          Center="0.25, 0.25"
                          RadiusX="0.25" RadiusY="0.25" >
      <GradientStop Offset="0" Color="Black" />
      <GradientStop Offset="1" Color="White" />
    </RadialGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Figure 4-12



Le dégradé émane du coin inférieur droit et est limité à une ellipse ayant son centre en (0.25, 0.25) et dont les rayons sont égaux au quart de la figure. Comme `GradientOrigin` se trouve en dehors de cette ellipse, l'enveloppe de dégradé est complétée par un cône formé de deux tangentes à cette ellipse, à partir de `GradientOrigin`.

Exemple 6 :

```
<RadialGradientBrush RadiusX="0.1" RadiusY="0.1"
    SpreadMethod="Reflect" >
  <GradientStop Color="Transparent" Offset="0.1"/>
  <GradientStop Color="Black" Offset="0.9" />
</RadialGradientBrush>
```

Figure 4-13



Exemple 7 :

```
<Ellipse ..... >
<Ellipse.Fill>
  <RadialGradientBrush GradientOrigin="0.75,0.25" >
    <GradientStop Color="Silver" Offset="0" />
    <GradientStop Color="Brown" Offset="0.85" />
    <GradientStop Color="Brown" Offset="1.0" />
  </RadialGradientBrush>
</Ellipse.Fill>
</Ellipse>
```

Figure 4-14



Ce dégradé donne l'illusion d'une sphère éclairée en haut à droite (à cause de GradientOrigin et du choix des couleurs).

Le masque d'opacité

Le masque d'opacité ressemble en tout point à un dégradé (linéaire ou radial) à la différence que seule la composante de transparence (premier des quatre octets de la couleur) est prise en compte.

L'exemple de code suivant permet d'obtenir un dégradé radial de transparence appliqué à une image (figures 4-15 et 4-16) :

```
<Image Source="MonaLisa.jpg" ..... >
<Image.OpacityMask>
  <RadialGradientBrush>
    <GradientStop Offset="0" Color="#FF000000" />
    <GradientStop Offset="1" Color="#00000000" />
  </RadialGradientBrush>
</Image.OpacityMask>
</Image>
```

Figure 4-15

Image d'origine, sans dégradé de transparence



Figure 4-16

Résultat obtenu avec dégradé de transparence



Les pinceaux dans Expression Blend

Utilisation d'Expression Blend

À la base, le XAML n'a rien de bien compliqué (et doit donc être maîtrisé facilement) mais les choses deviennent encore plus intuitives avec Expression Blend, même si graphistes et programmeurs (qui doivent travailler main dans la main) n'ont pas toujours la même notion de la difficulté...

Pour illustrer l'utilisation de ce logiciel, nous allons tout d'abord commencer par créer un nouveau projet dans Visual Studio qui contiendra uniquement un rectangle dans un conteneur (il s'agit d'un canevas mais cela n'a aucune d'importance, l'objectif étant tout simplement ici de peindre le rectangle avec l'aide d'Expression Blend) :

```
<Rectangle x:Name="rc" Canvas.Left="10" Canvas.Top="10" Width="200" Height="150"
          Fill="Red"/>
```

Un nom est attribué au rectangle (ici, *rc*). Même si la raison essentielle de ce nommage, à savoir la manipulation des composants par programme, n'est pas encore présente, il sera ainsi plus aisé de retrouver un composant parmi tous les éléments UI déjà présents dans la fenêtre Expression Blend (avec un rectangle seulement, cela ne pose bien entendu pas de problème).

Pour passer à Expression Blend, il vous suffit d'effectuer les opérations suivantes dans Visual Studio :

- enregistrez le projet ;
- effectuez un clic droit sur le nom du fichier XAML (*Page.xaml*) dans l'Explorateur de solutions ;
- cliquez sur Ouvrir dans Expression Blend.

Expression Blend travaille sur les mêmes fichiers que Visual Studio, ce qui assure l'interopérabilité entre les deux logiciels, chacun permettant d'effectuer des tâches bien distinctes : le développement du programme pour Visual Studio et l'amélioration visuelle de l'interface utilisateur pour Expression Blend. Toute modification effectuée dans l'un des logiciels est automatiquement répercutée dans l'autre.

Expression Blend est maintenant prêt à modifier la page Silverlight (plus précisément le XAML).

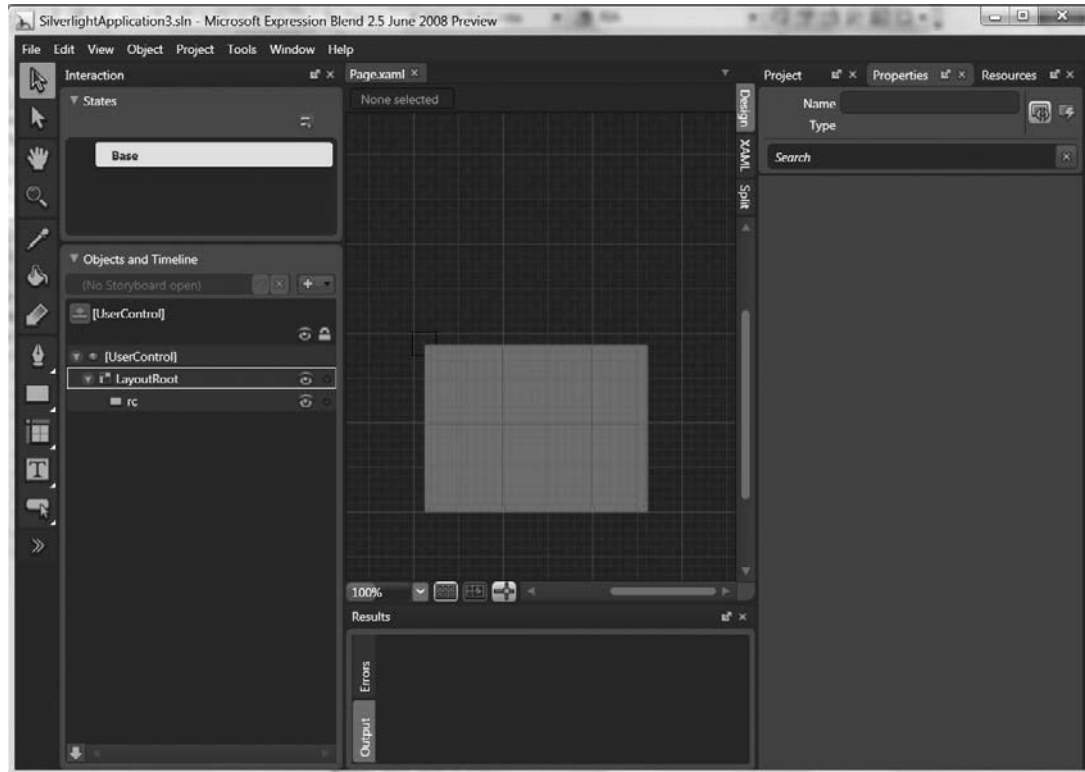
La fenêtre Expression Blend (figure 4-17) est constituée de quatre panneaux verticaux (évidemment coulissants) qui sont, de gauche à droite :

- la barre d'outils (l'outil de sélection étant le premier) ;
- le panneau des objets, parmi lesquels on retrouve le conteneur *LayoutRoot* et le rectangle *rc* ;
- le panneau d'affichage ou espace de travail (on y retrouve le rectangle rouge) ;
- le panneau des propriétés avec ses trois onglets (pour les propriétés du projet, celles de l'objet sélectionné et celles des ressources).

Les différentes icônes situées en bas de l'espace de travail permettent d'agrandir la vue, d'effectuer un zoom, d'afficher une grille (ce qui aide au positionnement des objets) ou encore de faire agir les fils de cette grille comme aimants.

Pour changer la couleur du rectangle, sélectionnez-le par un double clic sur le nom de l'élément UI nommé *rc* dans le panneau des objets ou par un clic sur l'élément lui-même dans l'espace de travail (au besoin, cliquez d'abord sur l'outil de sélection dans la barre d'outils).

Figure 4-17



Le rectangle apparaît alors sélectionné dans l'espace de travail et ses propriétés s'affichent dans le dernier panneau de droite (si ce n'est pas le cas, cliquez sur l'onglet Propriétés de ce panneau). Vous pouvez déplacer le rectangle, modifier sa taille, l'incliner, etc. Vous ne seriez pas en train de lire cet ouvrage si vous aviez besoin d'aide pour cela...

Les onglets Brushes et Appearance sont maintenant visibles pour l'élément UI sélectionné.

L'onglet Brushes représenté à la figure 4-18 permet de spécifier la couleur de remplissage (Fill ou Background en fonction des objets, ici Fill puisqu'il s'agit d'un rectangle) ou celle du contour (Stroke).

Ici, Fill est sélectionné ainsi que le pinceau SolidColorBrush de couleur unie (le deuxième rectangle au-dessus de l'éditeur de couleur est mis en évidence).

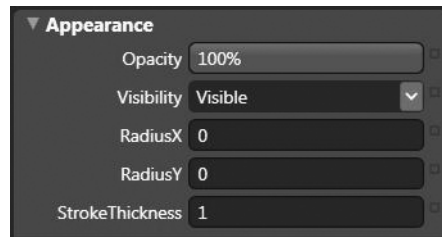
Une couleur peut être spécifiée par sa valeur hexadécimale (préfixée du caractère #, ici rouge opaque avec #FFFF0000), par ses composantes R, G et B (valeurs comprises entre 0 et 255) ainsi que par la valeur de A (transparence, soit un pourcentage compris entre 0 et 100) ou encore par sélection dans l'éditeur de couleurs (comme on en trouve dans la plupart des logiciels de dessin).

Figure 4-18



Sur la colonne de droite de l'éditeur de couleurs, vous trouverez deux petits triangles noirs se faisant face par la pointe (par défaut, ils sont situés en haut de la colonne). Déplacez-les afin d'afficher d'autres nuances de couleur. Pour sélectionner une couleur particulière, cliquez simplement dans le rectangle des nuances. Sa représentation hexadécimale apparaît alors immédiatement dans la partie de droite.

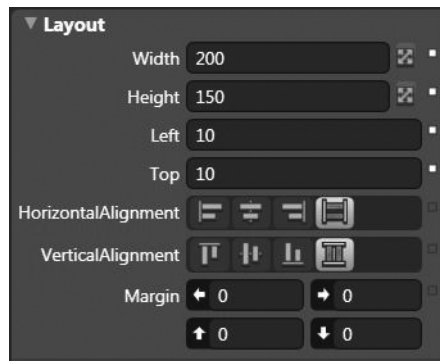
Figure 4-19



L'onglet Appearance représenté à la figure 4-19 permet de modifier les attributs du rectangle : opacité, rayons des coins arrondis et épaisseur du contour (StrokeThickness).

Pour changer une valeur (par exemple, `StrokeThickness`), vous pouvez la modifier manuellement mais aussi cliquer, en maintenant le clic, sur la zone d'édition et déplacer la souris vers la droite (pour augmenter la valeur) ou vers la gauche (pour la diminuer).

Figure 4-20



L'onglet Layout représenté à la figure 4-20 permet de spécifier la taille du rectangle, les coordonnées de son coin supérieur gauche, les marges ainsi que les alignements (ce qui présente surtout de l'intérêt si le conteneur est une cellule de grille ou un `StackPanel`).

Pour modifier ces valeurs, vous pouvez également cliquer sur le rectangle dans l'espace de travail pour le sélectionner et le déplacer tout en maintenant le bouton de la souris enfoncé. Pour modifier sa taille, il suffit de cliquer sur l'un de ses bords.

Toutes les modifications ainsi effectuées dans Expression Blend sont répercutées dans le XAML. Pour vous en assurer, cliquez sur l'onglet vertical XAML de l'espace de travail. Avant de repasser à Visual Studio, enregistrez votre travail dans Expression Blend via le menu `File>Save all`. Visual Studio détecte que le fichier XAML a été modifié par un programme externe et vous demande votre accord avant de prendre en compte ces changements. Confirmez en répondant Oui pour tout.

Créer un dégradé avec Expression Blend

Les dégradés linéaires

Pour créer un pinceau avec dégradé de couleurs dans Expression Blend, sélectionnez le troisième rectangle situé au-dessus de l'éditeur de couleurs. Un autre éditeur de couleurs apparaît alors, légèrement différent du précédent et à couleur unie (figure 4-21).

Sélectionnez tout d'abord le type de dégradé souhaité (linéaire ou radial) en cliquant sur l'un des deux boutons situés en bas à gauche ainsi que sur le bouton Options pour spécifier les valeurs de l'attribut `SpreadMethod` (`Pad`, `Repeat` et `Reflect`). Par défaut, il s'agit du dégradé linéaire, avec `Pad` comme option.

Vous pouvez maintenant sélectionner les couleurs aux points intermédiaires (les `GradientStop` du XAML). Pour cela, cliquez l'un des curseurs situés juste au-dessous de

l'échelle du dégradé, déplacez-le jusqu'à l'emplacement souhaité et sélectionnez la couleur correspondante en cliquant dans le rectangle des couleurs. La marque reflète aussitôt la couleur sélectionnée.

Figure 4-21



La marque de couleur intermédiaire peut être déplacée le long d'une ligne horizontale par une opération bien connue de glisser-déposer. Pour ajouter des marques intermédiaires, cliquez sur l'échelle du dégradé à l'endroit où vous souhaitez ajouter une couleur. Pour supprimer une marque, cliquez dessus et faites-la glisser en dehors de l'échelle du dégradé.

Pour changer la ligne de dégradé (par défaut, elle va du coin supérieur gauche au coin inférieur droit), cliquez sur le bouton en forme d'épaisse flèche vers le bas et vers la gauche dans la barre d'outils (figure 4-22). Une flèche s'affiche alors en superposition de l'élément UI (ici, un rectangle).

Vous pouvez agrandir, rétrécir et faire tourner la flèche de dégradé en cliquant à ses alentours. Ces manipulations ont pour effet de modifier les attributs `StartPoint` et `EndPoint` du XAML. Le nouveau dégradé est immédiatement répercuté dans le rectangle.

Les dégradés radiaux

La figure 4-23 illustre l'exemple 6 de la section « Le pinceau `RadialGradientBrush` » précédente (ici, dans une ellipse plutôt qu'un rectangle). C'est tellement intuitif avec Expression Blend que cela en devient plus simple à réaliser qu'à expliquer ! En effet, il suffit de manipuler avec la souris la ligne et les enveloppes de dégradés.

Figure 4-22

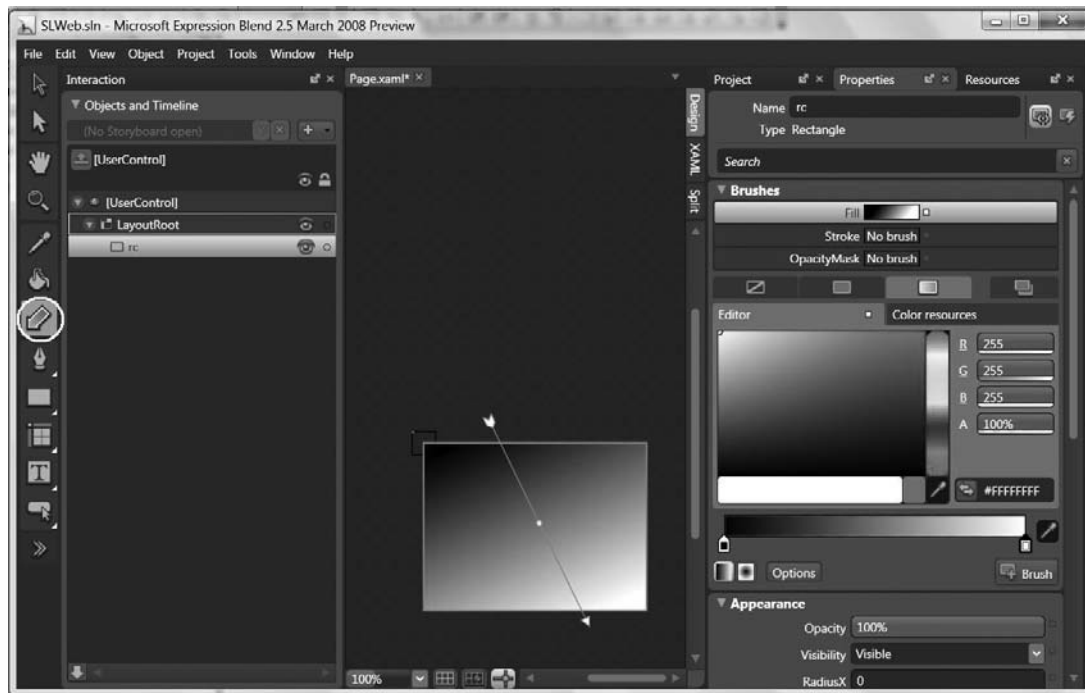
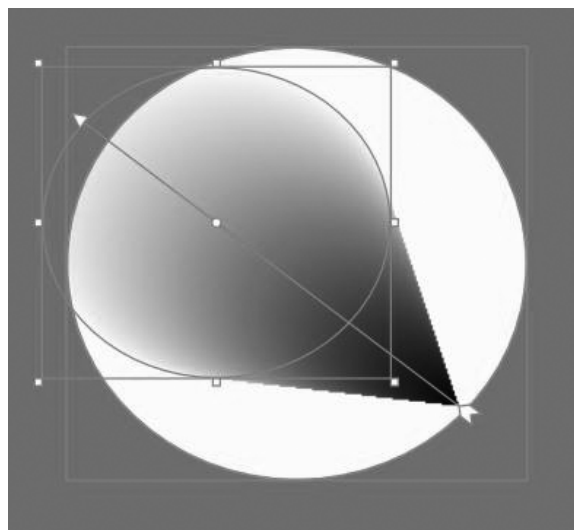


Figure 4-23



Exemples d'utilisation des dégradés

Dans cette section, vous allez découvrir certains effets obtenus par les graphistes et les techniques employées à cet effet.

Le bouton « gel »

Ce bouton est ainsi appelé en raison de son aspect « gélatineux ». Pour créer un tel bouton (figure 4-24), l'astuce consiste à superposer deux rectangles aux bords arrondis, le second étant très légèrement plus petit et centré dans le premier. Dans le premier bouton, on réalise un dégradé linéaire du haut vers le bas entre deux couleurs qui sont ici, le vert foncé (green) et le vert clair (lime), tandis que dans le second rectangle, on réalise un dégradé (avec la même ligne de dégradé) allant du blanc opaque au noir transparent.

Au chapitre 15, nous verrons comment réaliser les effets visuels qui reflètent le clic ou le survol de la souris.

Figure 4-24



```
<Canvas ..... >
<!-- premier rectangle -->
<Rectangle Width="200" Height="50" RadiusX="15" RadiusY="15"
    Stroke="Black" StrokeThickness="2" >
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1" >
            <GradientStop Offset="0" Color="Green" />
            <GradientStop Offset="1" Color="Lime" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
<!-- second rectangle, en superposition du premier -->
<Rectangle Width="198" Height="48" RadiusX="14" RadiusY="14"
    Canvas.Left="1" Canvas.Top="1" >
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1" >
            <GradientStop Offset="0" Color="White" />
            <GradientStop Offset="1" Color="#00000000" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
<!-- libellé du bouton -->
<TextBlock Text="Effet Gel" Foreground="White"
    FontFamily="Verdana" FontSize="17" FontWeight="Bold"
    Canvas.Left="60" Canvas.Top="12" />
</Canvas>
```

Le bouton gel correspondant à ce code (figure 4-24) est basé sur un canevas et les propriétés attachées que sont `Canvas.Left` et `Canvas.Top`, ce qui est plus simple à ce stade de l'étude. Le code suivant correspond au même bouton mais cette fois basé sur une grille, ce qui présente l'avantage que la taille du bouton s'adapte automatiquement à la cellule de la grille ou prend la taille imposée par l'utilisateur.

```
<Grid Grid.Row="1" Grid.Column="1" >
<!-- premier rectangle -->
<Rectangle RadiusX="15" RadiusY="15"
    Stroke="Black" StrokeThickness="2" >
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1" >
            <GradientStop Offset="0" Color="Green" />
            <GradientStop Offset="1" Color="Lime" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
<!-- second rectangle, en superposition du premier -->
<Rectangle RadiusX="14" RadiusY="14"
    RenderTransformOrigin="0.5, 0.5" >
    <Rectangle.RenderTransform>
        <ScaleTransform ScaleX="0.98" ScaleY="0.98" />
    </Rectangle.RenderTransform>
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1" >
            <GradientStop Offset="0" Color="White" />
            <GradientStop Offset="1" Color="#00000000" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
<!-- libellé du bouton -->
<TextBlock Text="Effet Gel" Foreground="White"
    FontFamily="Verdana" FontSize="17" FontWeight="Bold"
    HorizontalAlignment="Center" VerticalAlignment="Center"
/>
</Grid>
```

L'effet « plastic »

Cet effet (figure 4-25), aussi qualifié de tube éclairé, est obtenu à l'aide d'un seul rectangle, en spécifiant plusieurs couleurs intermédiaires (ici, dans les tons orange) le long de la ligne de dégradé.

Figure 4-25



```

<Grid ..... >
  <Rectangle Width="200" Height="40" >
    <Rectangle.Fill >
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1" >
        <GradientStop Color="Red" Offset="0" />
        <GradientStop Color="OrangeRed" Offset="0.07" />
        <GradientStop Color="Orange" Offset="0.15" />
        <GradientStop Color="OrangeRed" Offset="0.30" />
        <GradientStop Color="Red" Offset="0.35" />
        <GradientStop Color="FireBrick" Offset="0.45" />
        <GradientStop Color="Sienna" Offset="0.9" />
        <GradientStop Color="Sienna" Offset="1" />
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
  <TextBlock Text="Effet plastic" Foreground="White"
    FontFamily="Verdana" FontSize="25" FontWeight="Bold"
    HorizontalAlignment="Center" VerticalAlignment="Center" />
</Grid>

```

L'effet « métal »

Cet effet est obtenu en superposant deux rectangles et constitue donc une variante du bouton gel. La couleur du premier rectangle est unie (ici, le gris) tandis que le second (en superposition du premier) offre un dégradé de blanc de plus en plus transparent le long d'une verticale allant du bord supérieur au milieu du bouton.

Figure 4-26



```

<Grid ..... >
  <!-- premier rectangle -->
  <Rectangle Width="120" Height="80" RadiusX="7" RadiusY="7" Fill="DimGray" />
  <!-- second rectangle, avec dégradé -->
  <Rectangle Width="110" Height="70" RadiusX="5" RadiusY="5"
    Canvas.Left="5" Canvas.Top="5" >
    <Rectangle.Fill>
      <LinearGradientBrush StartPoint="0, 0" EndPoint="0, 1" >
        <GradientStop Color="#D0FFFFFF" Offset="0" />
        <GradientStop Color="#00FFFFFF" Offset="0.5" />
      </LinearGradientBrush>
    </Rectangle.Fill>
  </Rectangle>
  <TextBlock Text="Effet métal"
    HorizontalAlignment="Center" VerticalAlignment="Center"
    FontSize="18" Foreground="White" FontWeight="Bold" />
</Grid>

```

Le bouton « de verre »

Il s'agit d'un bouton rond dont l'aspect convexe est obtenu à l'aide d'une petite ellipse avec dégradé dans la partie supérieure (figure 4-27). L'effet est réalisé à l'aide de trois ellipses :

- une ellipse extérieure pour le collier ;
- une ellipse pour le bouton lui-même (uniformément vert) ;
- une ellipse pour l'effet de lumière rendant la forme convexe du bouton.

Figure 4-27



```
<Canvas ..... >
  <!-- collier autour du bouton de verre -->
  <Ellipse Canvas.Left="13" Canvas.Top="13" Width="114" Height="114"
    Stroke="Black" Fill="White" />
  <!-- bouton de verre -->
  <Ellipse Canvas.Left="20" Canvas.Top="20" Width="100" Height="100"
    Stroke="Gray" Fill="Green" />
  <!-- effet de lumière dans la partie supérieure -->
  <Ellipse Canvas.Left="31" Canvas.Top="21" Height="78" Width="78" >
    <Ellipse.Fill>
      <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1" >
        <GradientStop Color="#C0FFFFFF" Offset="0" />
        <GradientStop Color="#70FFFFFF" Offset="0.3" />
        <GradientStop Color="#30FFFFFF" Offset="0.55" />
        <GradientStop Color="Transparent" Offset="0.8" />
      </LinearGradientBrush>
    </Ellipse.Fill>
  </Ellipse>
</Canvas>
```

L'effet d'ombre

L'exemple de code suivant montre comment utiliser les rectangles, les pinceaux et les transparences pour réaliser un effet d'ombre donnant du relief aux pages (figure 4-29). L'astuce consiste à dessiner en fond d'image des rectangles légèrement décalés et de plus en plus opaques.

```
<Canvas ..... >
  <Rectangle Canvas.Left="2" Canvas.Top="2" Width="300" Height="380"
    Fill="Black" Opacity="0.50" />
  <Rectangle Canvas.Left="4" Canvas.Top="4" Width="300" Height="380"
    Fill="Black" Opacity="0.40" />
```

```
<Rectangle Canvas.Left="6" Canvas.Top="6" Width="300" Height="380"  
    Fill="Black" Opacity="0.20" />  
<Rectangle Canvas.Left="8" Canvas.Top="8" Width="300" Height="380"  
    Fill="Black" Opacity="0.10" />  
<Image Source="VG.jpg" Width="300" Height="380" Stretch="Fill" />  
</Canvas>
```

Figure 4-28*Sans effet d'ombre***Figure 4-29***Avec un léger effet d'ombre*

5

Une première série de composants

Dans ce chapitre, nous allons présenter une première – mais déjà large – série de composants Silverlight : les figures géométriques telles que les rectangles ou les ellipses, les zones d’affichage (`TextBlock`), les zones d’édition (`TextBox`), les boutons, les barres de défilement, les calendriers, etc. Ceci nous permettra d’introduire le plus rapidement possible (mais dans des chapitres ultérieurs) les transformations, les animations ainsi que la manipulation d’éléments UI par du code.

Nous compléterons l’étude de ces composants au cours des chapitres suivants, nous reviendrons ensuite sur les composants liés aux données (chapitre 10) et nous apprendrons à personnaliser l’apparence de ces composants (chapitre 15) en permettant à un graphiste d’en modifier chaque sous-élément, sans que cela implique un changement dans le code du programme.

Dans la foulée des zones d’affichage étudiées dans ce chapitre, nous verrons également comment spécifier des polices de caractères.

Les composants liés aux images et à la vidéo seront traités au chapitre 7. Les boîtes de listes ainsi que les grilles de données seront traitées au chapitre 10.

Les rectangles et les ellipses

Les rectangles et les ellipses sont des composants très utilisés, notamment pour en créer et en personnaliser d'autres.

Pour illustrer leur fonctionnement, nous allons créer une nouvelle application Silverlight (appelons-la *SLProg*) et nous éditerons le fichier nommé par défaut *Page.xaml*, qui contient la description XAML de la page Web. Pour cela, double-cliquez sur le nom du fichier dans l'Explorateur de solutions de l'application. Éditer du XAML est très simple grâce à l'aide contextuelle fournie par Visual Studio en cours de frappe. En effet, il suffit de taper les premières lettres et Visual Studio vous indique ce qui peut suivre selon le contexte. Utilisez la combinaison de touches *Ctrl* + *espace* pour visualiser ce qu'il est possible de faire et la touche *Entrée* pour valider une proposition de Visual Studio.

Au lieu de taper le texte de la balise (par exemple, *<Rectangle>*), il est possible de procéder par un glisser-déposer (*drag & drop* en anglais) entre la boîte d'outils et la fenêtre d'édition du code XAML. Il suffit alors de compléter les attributs de l'embryon de balise ainsi générée.

Placez à présent un rectangle rouge et un cercle vert dans un canevas. Les éléments UI pourraient être placés dans une cellule de grille (cas le plus fréquent), un *StackPanel* ou encore dans un conteneur inséré dans une cellule de grille ou un *StackPanel*. L'option retenue pour cette application est le canevas mais la plupart des exemples mentionnés dans la suite de l'ouvrage utilisent la grille comme conteneur. Ajoutez ensuite deux balises (l'une pour le rectangle et l'autre pour l'ellipse) dans le fichier *Page.xaml* :

```
<UserControl x:Class="SLProg.Page"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
  <Canvas x:Name="LayoutRoot" Background="Beige">
    <Rectangle Canvas.Left="50" Canvas.Top="100" Width="120"
      Height="80" Fill="Red" />
    <Ellipse Canvas.Left="200" Canvas.Top="200" Width="150"
      Height="150" Fill="Green" />
  </Canvas>
</UserControl>
```

Qu'en serait-il de ces deux balises si le conteneur était d'un autre type que le canevas ?

Dans le cas d'une ellipse ou d'un rectangle inséré dans un *StackPanel* :

- *Canvas.Left* et *Canvas.Top* ne doivent pas être spécifiés et s'ils le sont, ces attributs sont ignorés ;
- il faut au moins spécifier *Width* si le panneau est à orientation verticale et *Height* si l'orientation est horizontale.

Dans le cas d'une ellipse ou d'un rectangle inséré dans une cellule de grille :

- *Canvas.Left* et *Canvas.Top* ne doivent pas être spécifiés et s'ils le sont, ces attributs sont ignorés (*Canvas.ZIndex*, qui indique le positionnement relatif quand il y a une superposition, est cependant pris en compte) ;

- Width et Height sont également optionnels : en leur absence, l'élément UI occupe la plus grande surface possible dans la cellule.

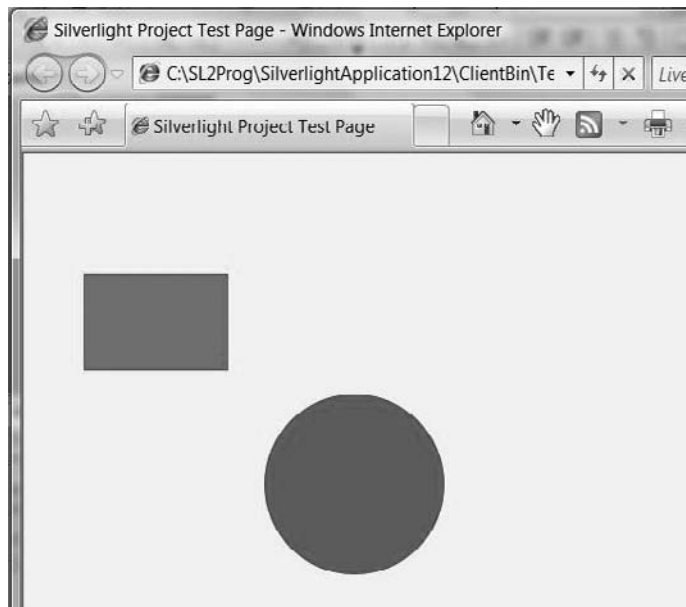
Dans le cas d'un rectangle inséré dans une cellule de grille, les attributs Canvas.Left et Canvas.Top sont simulés au moyen du code suivant :

```
<Rectangle HorizontalAlignment="Left" VerticalAlignment="Top"
            Margin="50, 100, 0, 0" Width="120" Height="80" Fill="Red" />
```

Le conteneur, c'est-à-dire le canevas pour cet exemple, contient désormais deux objets. On dit aussi qu'il a deux « enfants ». À la section « La création dynamique d'objets » du chapitre 6, nous verrons comment construire, ajouter et supprimer de tels éléments UI par programme (autrement dit, comment construire dynamiquement l'interface Web).

La figure 5-1 représente le résultat de la page Web obtenue. Il s'agit d'une page statique et peu attrayante (et somme toute impossible à réaliser en HTML pur) mais cela n'a que peu d'importance car nous n'en sommes qu'à la mise en place des objets.

Figure 5-1



Les éléments visuels (ici, un rectangle et une ellipse) sont affichés à l'intérieur du conteneur (ici, un canevas).

Le run-time Silverlight comprend que la balise `Rectangle` correspond à un objet `Rectangle`, évidemment affiché comme un rectangle. Les classes `Rectangle` et `Ellipse` sont dérivées de la classe `Shape`, elle-même dérivée de `FrameworkElement`. Comme n'importe quel objet,

un rectangle peut être modifié et animé par programme, donc en cours d'exécution de programme. Nous aurons l'occasion de rencontrer plusieurs animations de ce genre.

Un rectangle peut avoir les coins arrondis. Dans ce cas, les attributs `RadiusX` et `RadiusY` correspondent respectivement au rayon selon l'axe des X et au rayon selon l'axe des Y de l'ellipse (partiellement visible) affichée à chacun des quatre coins du rectangle.

Les éléments affichés dans la fenêtre peuvent se superposer (voir l'attribut `Canvas.ZIndex` présenté dans le tableau 5-1) ou être partiellement transparents (attribut `Opacity`).

Le tableau 5-1 présente les attributs utilisés dans les extraits de code précédents et qui sont communs à tous les éléments d'interface (mais parfois sous un autre nom pour `Fill` ou avec des restrictions, comme pour `Canvas.Left` et `Canvas.Top`).

Tableau 5-1 – Les attributs des éléments UI

Nom de l'attribut	Description
Fill	<p>La propriété <code>Fill</code> indique comment peindre l'intérieur de la figure. Son équivalent pour un conteneur mais aussi les boutons, les cases à cocher et les zones d'édition est la propriété <code>Background</code>.</p> <p>Des pinceaux de coloriage plus complexes, avec dégradés de couleurs, peuvent être spécifiés, comme nous l'avons vu à la section « Les pinceaux » du chapitre 4. Aucun contour n'est affiché en l'absence de la propriété <code>Stroke</code>.</p> <p>La propriété <code>Fill</code> est de type <code>Brush</code>. Les types spécialisés de <code>Brush</code> (voir également la section « Les pinceaux » du chapitre 4) sont <code>SolidColorBrush</code> (pinceau de couleur unie), <code>LinearColorBrush</code> et <code>RadialColorBrush</code> (pinceau avec dégradé linéaire ou radial).</p> <p>Un nom de couleur peut être spécifié en valeur de l'attribut <code>Fill</code> car l'interpréteur XAML est capable de transformer un nom de couleur en objet <code>SolidColorBrush</code> de cette couleur. Pour un pinceau plus complexe, il faut transformer <code>Fill</code> en une sous-balise de <code>Rectangle</code> ou <code>Ellipse</code> :</p> <pre><Rectangle > <Rectangle.Fill> <LinearGradientBrush /> </Rectangle.Fill> </Rectangle></pre>
Opacity	<p>La propriété <code>Opacity</code> (valeur décimale comprise entre 0 et 1) indique la transparence de l'élément. Par défaut, les éléments sont affichés de manière opaque (<code>Opacity</code> vaut alors 1 et rien n'est visible sous l'élément). Si <code>Opacity</code> vaut 0, l'élément est entièrement transparent et n'est donc pas du tout visible. Comparons des opacités de 0.75 et de 0.25 : avec la valeur 0.75, le fond reste partiellement visible mais moins visible qu'avec une <code>Opacity</code> de 0.25.</p>
Visibility	<p>Cette propriété indique si un élément est visible ou non (par défaut, il l'est évidemment). Les valeurs possibles sont <code>Visible</code> et <code>Collapsed</code> (correspondant à « non visible »).</p>
x:Name	<p>Bien que cela ne soit pas encore nécessaire dans une page Silverlight aussi élémentaire, il est possible d'attribuer un nom interne à n'importe quel élément (attribut <code>x:Name</code>). Ceci permet de manipuler l'élément à partir du code C# ou VB, le nom de l'élément devenant alors une variable dans la partie « programme ». Très accessoirement, cela permet aussi de repérer plus aisément un élément quand on passe à Expression Blend.</p>

Tableau 5-1 – Les attributs des éléments UI (*suite*)

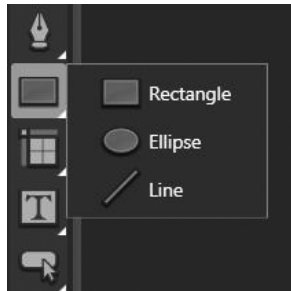
Nom de l'attribut	Description
Canvas.Left Canvas.Top	Canvas.Left et Canvas.Top correspondent aux coordonnées de l'élément (plus précisément, celles de son coin supérieur gauche), par rapport au coin supérieur gauche du canevas parent. Dans la mesure où un canevas peut en contenir d'autres, il convient de préciser que ces coordonnées sont relatives au canevas parent le plus proche. Dans le cas d'une ellipse, il s'agit de la coordonnée du coin supérieur gauche du rectangle (non affiché) entourant l'ellipse. Nous avons vu, avec HorizontalAlignment et VerticalAlignment, comment simuler Canvas.Left et Canvas.Top quand le conteneur est une cellule de grille.
Canvas.ZIndex	Dans la mesure où des éléments affichés dans la fenêtre du navigateur peuvent se superposer, ceux qui sont déclarés (dans le XAML) avant les autres peuvent être cachés (partiellement ou totalement) par ces derniers. Ce comportement par défaut (l'ordre d'affichage suit l'ordre des déclarations) peut être modifié par la propriété Canvas.ZIndex (n'importe quelle valeur entière, avec 0 comme valeur par défaut). L'élément ayant le ZIndex le plus faible est affiché en premier (et peut donc être partiellement ou totalement caché) tandis que celui qui bénéficie du ZIndex le plus élevé est affiché en dernier et est donc toujours visible. En cas d'égalité des ZIndex, les éléments sont affichés dans l'ordre de leur déclaration.

À noter qu'il existe d'autres propriétés dont :

- la famille *Stroke* (voir la section « Le Stroke » du chapitre 8), pour spécifier le contour de la figure (et notamment *Stroke* pour la couleur et *StrokeThickness* pour l'épaisseur de ce contour) ;
- Cursor*, pour indiquer la forme que prend le curseur lorsque la souris survole l'élément (voir la section « Les curseurs » du chapitre 7).

Voyons maintenant comment créer un rectangle ou une ellipse dans Expression Blend. Si ces éléments n'apparaissent pas dans la boîte d'outils d'Expression Blend, cliquez sur le composant affiché (l'ellipse ou le rectangle) et maintenez enfoncé le bouton de la souris pendant une seconde. Sélectionnez ensuite le composant souhaité qui s'affiche alors dans la boîte d'outils (figure 5-2).

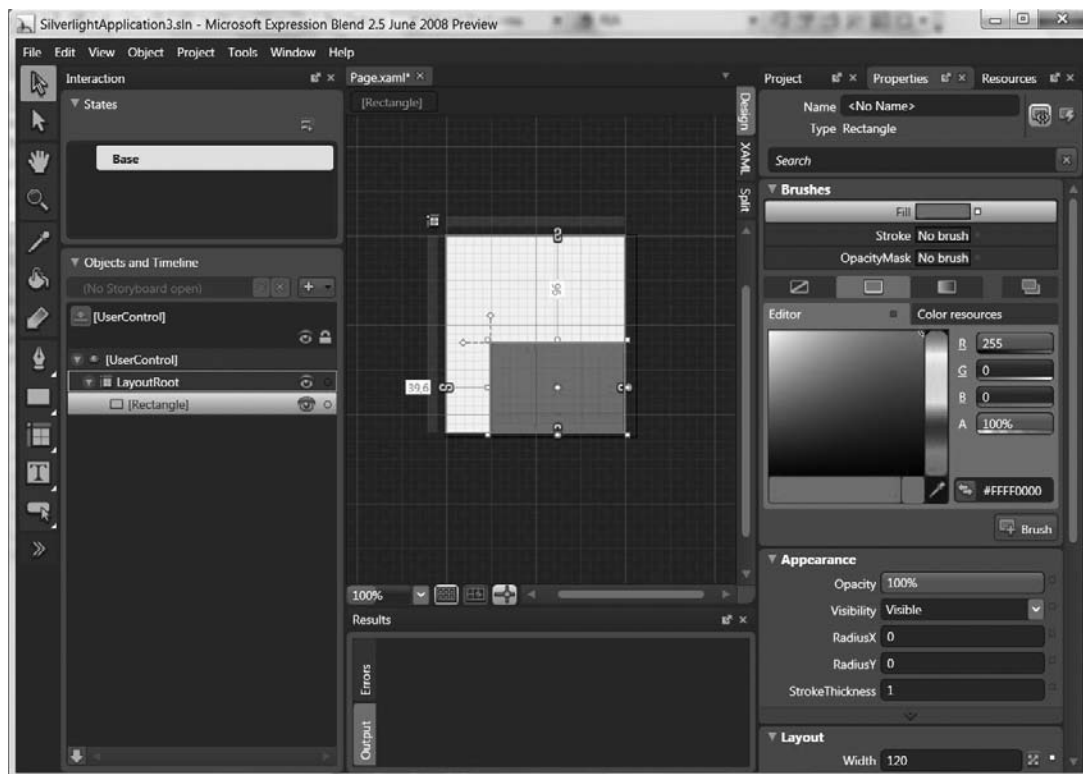
Figure 5-2



Déplacer le rectangle et le redimensionner est un jeu d'enfant dans la surface de travail d'Expression Blend (figure 5-3). En cliquant près de l'un des coins du rectangle, une flèche en forme d'arc de cercle s'affiche, permettant d'incliner, voire de faire tourner le

rectangle. Une transformation de type `RotateTransform` (voir la section « Les transformations » du chapitre 9) est alors appliquée au rectangle. Vous pouvez aussi utiliser le panneau de droite pour modifier les caractéristiques du rectangle ou de l'ellipse.

Figure 5-3



Pour tracer un carré ou un cercle, maintenez simplement la touche Maj enfoncée lors des redimensionnements à l'aide de la souris afin de conserver les proportions.

Les zones d'affichage (TextBlock)

La balise `TextBlock` permet d'afficher du texte, celui-ci étant spécifié dans l'attribut `Text`. Il n'est pas nécessaire de spécifier `Width` et `Height` (la zone d'affichage s'étend automatiquement) bien que cela soit possible : le texte est alors limité au rectangle (`Width`, `Height`) et ne s'affiche pas en dehors de celui-ci. Il est néanmoins possible d'afficher le texte sur plusieurs lignes à l'intérieur du rectangle. Pour cela, il suffit d'initialiser la propriété `TextWrapping` à `Wrap`, le rectangle servant alors d'enveloppe au texte.

Par ailleurs, les lettres du texte sont peintes avec la couleur (plus précisément le pinceau) spécifiée dans la propriété `Foreground`. Il peut s'agir d'une couleur avec dégradé (linéaire ou radial), d'une image ou encore d'une vidéo (celle-ci étant alors jouée à l'intérieur des lettres).

Les caractéristiques de la police sont spécifiées dans les propriétés `FontFamily`, `FontSize`, `FontStretch`, `FontStyle`, `FontHeight` (voir la section suivante) mais aussi `TextDecorations` (qui peut contenir la chaîne `Underline` pour forcer un soulignement) et `TextWrapping`.

Par exemple :

```
<TextBlock Text="Hello Silverlight" ..... />
```

Décomposer un texte en plusieurs parties

Un texte peut être divisé en plusieurs parties, chacune d'elles comportant des attributs de présentation différents. Pour cela, il faut utiliser :

- la balise `Run` pour forcer une division de texte (avec attributs de présentation propres à cette section) ;
- la balise `LineBreak` pour provoquer un saut de ligne.

L'exemple de code suivant affiche un texte sur deux lignes, en bleu, en blanc et en rouge, avec un espace entre `Allez` et `la` :

```
<TextBlock ..... FontSize="12">
  <Run Foreground="Blue">Allez</Run>
  <Run Foreground="White" FontSize="15">la</Run>
  <LineBreak />
  <Run Foreground="Red" FontSize="12">France</Run>
</TextBlock>
```

Au lieu d'utiliser les balises précédentes, ce code pourrait également s'écrire :

```
<TextBlock ..... FontSize="12">
  <Run Foreground="Red" Text="Allez" />
  <Run Foreground="Black" FontSize="15" Text="  la  " />
  <LineBreak/>
  <Run Foreground="Green" FontSize="12" Text="      France" />
</TextBlock>
```

Cette syntaxe rend plus aisé l'ajout d'espaces, non pris en compte dans l'exemple précédent au-delà de l'unique espace de séparation.

Si la balise `TextBlock` contient un attribut `Text`, son contenu est affiché avant le texte spécifié dans les balises `Run`.

Lors de l'initialisation par programme de la propriété `Text` d'une zone d'affichage, ajoutez l'attribut `Environment.NewLine` pour provoquer un saut de ligne, ce qui présente l'avantage d'être indépendant de la plate-forme d'exécution du programme Silverlight.

Par exemple (za désignant le nom interne d'une zone d'affichage) :

```
za.Text = "Hello" + Environment.NewLine + "Silverlight"; // syntaxe C#
```

ou

```
za.Text = "Hello" & Environment.NewLine & "Silverlight" ' syntaxe VB
```

Les effets de relief

Il est possible de créer des effets de relief (de préférence sur un fond gris comme LightGray mais d'autres combinaisons seraient possibles) en combinant astucieusement position du texte et couleurs.

Pour « graver » du texte, affichez tout d'abord le texte en blanc à partir du point (x + 1, y + 1) puis en noir à partir de (x, y) :

```
<TextBlock Canvas.Left="101" Canvas.Top="101" Foreground="White"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !"/>
<TextBlock Canvas.Left="100" Canvas.Top="100" Foreground="Black"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !"/>
```

Pour le faire ressortir, affichez tout d'abord le texte en blanc en (x - 1, y - 1), puis en gris en (x + 1, y + 1) et finalement en blanc en (x, y) :

```
<TextBlock Canvas.Left="199" Canvas.Top="199" Foreground="White"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !"/>
<TextBlock Canvas.Left="201" Canvas.Top="201" Foreground="Gray"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !"/>
<TextBlock Canvas.Left="200" Canvas.Top="200" Foreground="Black"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !"/>
```

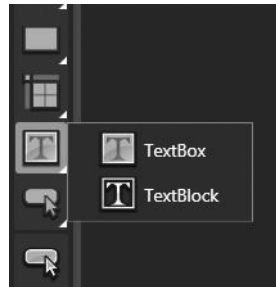
Dans une cellule de grille, cet effet est obtenu en passant par HorizontalAlignment, VerticalAlignment et Margin comme indiqué précédemment ou en modifiant comme suit les balises :

```
<TextBlock Foreground="White" Grid.Row="1" Grid.Column="1"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !">
    <TextBlock.RenderTransform>
        <TranslateTransform X="-1" Y="-1" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock Foreground="Gray" Grid.Row="1" Grid.Column="1"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !">
    <TextBlock.RenderTransform>
        <TranslateTransform X="1" Y="1" />
    </TextBlock.RenderTransform>
</TextBlock>
<TextBlock Foreground="Black" Grid.Row="1" Grid.Column="1"
    FontFamily="Verdana" FontSize="40" Text ="Hello Silverlight !" />
```

Nous pourrions réaliser des effets plus saisissants encore lorsque nous aborderons les transformations et animations au chapitre 9.

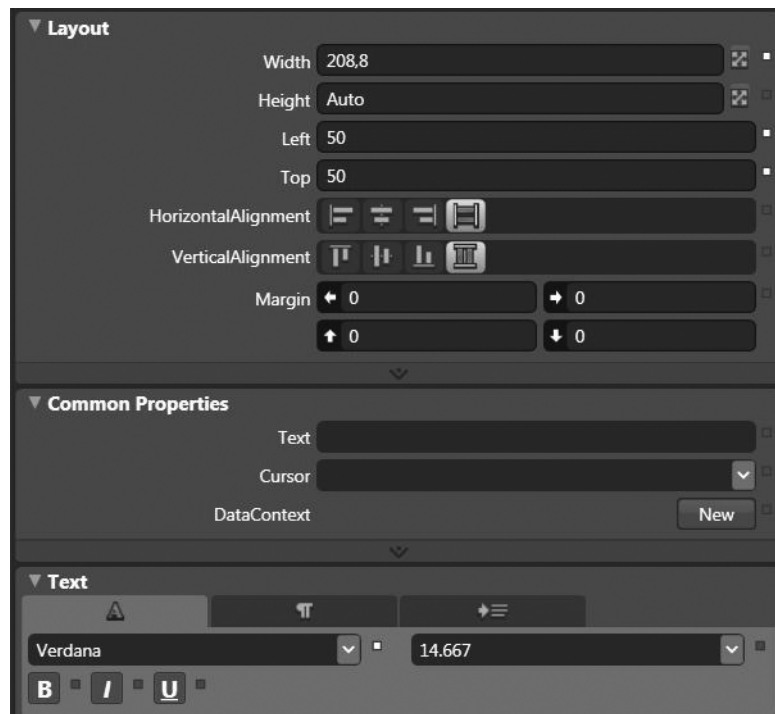
Dans Expression Blend, `TextBlock` (zone d’affichage) et `TextBox` (zone d’édition) font partie de la même famille. Cliquez donc sur le bouton correspondant dans la boîte d’outils (figure 5-4) pour sélectionner l’outil `TextBox` ou `TextBlock` (maintenez le bouton de la souris enfoncé pendant une seconde pour sélectionner l’un ou l’autre).

Figure 5-4



Dans le panneau de droite (figure 5-5), vous pouvez spécifier les caractéristiques d’affichage (libellé, alignements, marges, etc.). Les attributs XAML que nous venons d’étudier sont ainsi générés par Expression Blend.

Figure 5-5



Les polices de caractères

Même si les polices de caractères ne font pas partie des composants d'interface visuels, nous les présentons ici en raison de leur rôle dans les zones d'affichage mais aussi dans la plupart des éléments UI (libellé des boutons mais aussi de bien d'autres composants).

Les polices fournies avec Silverlight

Pour afficher du texte autrement qu'avec ses attributs de police (*font* en anglais) par défaut, il suffit de spécifier des valeurs pour les attributs de la famille `Font`. Par défaut, la police (attribut `FontFamily`) est `Lucida Sans Unicode`, `Lucida Grande` (aussi appelée *Portable User Interface*), avec la valeur `FontSize` (taille) égale à 11 points, soit 14,66 pixels.

Le tableau 5-2 présente les différentes propriétés des polices de caractères.

Tableau 5-2 – Les propriétés des polices de caractères

Nom de l'attribut	Description
<code>FontFamily</code>	Nom de la police. Plusieurs noms de polices (séparés par des virgules) peuvent être spécifiés. Les polices nativement supportées sont <code>Arial</code> , <code>Arial Black</code> , <code>Comic Sans MS</code> , <code>Courier New</code> , <code>Georgia</code> , <code>Lucida Grande/Lucida Sans Unicode</code> , <code>Times New Roman</code> , <code>Trebuchet MS</code> et <code>Verdana</code> (voir figure 5-6).
<code>FontSize</code>	Hauteur des caractères en pixels.
<code>FontStretch</code>	Une des valeurs de l'énumération <code>FontStretches</code> indiquant la compression ou l'expansion des caractères : <code>Condensed</code> , <code>Expanded</code> , <code>ExtraCondensed</code> , <code>ExtraExpanded</code> , <code>Medium</code> , <code>Normal</code> , <code>SemiCondensed</code> , <code>SemiExpanded</code> , <code>UltraCondensed</code> et <code>UltraExpanded</code> .
<code>FontStyle</code>	Une des valeurs de l'énumération <code>FontStyles</code> spécifiant l'inclinaison des caractères : <code>Italic</code> , <code>Normal</code> et <code>Oblique</code> .
<code>FontWeight</code>	Une des valeurs de l'énumération <code>FontWeights</code> indiquant la graisse (ici, du très foncé au très clair) : <code>Black</code> , <code>Bold</code> , <code>DemiBold</code> , <code>ExtraBlack</code> , <code>ExtraLight</code> , <code>Heavy</code> , <code>Light</code> , <code>Medium</code> , <code>Normal</code> , <code>Regular</code> , <code>SemiBold</code> , <code>Thin</code> , <code>UltraBlack</code> , <code>UltraBold</code> et <code>UltraLight</code> .

`Normal` est la valeur par défaut pour les trois derniers attributs.

La figure 5-6 présente le nom des polices fournies avec Silverlight, affichés dans les polices correspondantes.

Figure 5-6

Arial	Arial Black	Comic Sans MS	Courier New	
Georgia	Par défaut	Times New Roman	Trebuchet	Verdana

Afficher un texte dans une police non fournie avec Silverlight

Pour afficher du texte dans une police non fournie avec Silverlight, il faut disposer du fichier .ttf de cette police. D'innombrables polices (fichiers .ttf) peuvent être téléchargées sur Internet, souvent gratuitement. Pour connaître le nom de la police contenue dans un fichier .ttf, il suffit de double-cliquer sur le nom du fichier à partir de l'Explorateur de fichiers. Dans le cas de notre seconde application Web, nous allons incorporer la police Scriptina contenue dans le fichier ScriptIn.ttf, librement téléchargeable à partir du site de Smashing Magazine, <http://www.smashingmagazine.com/>

Ajoutez tout d'abord le fichier ScriptIn.ttf au projet de l'application. Pour cela, dans l'Explorateur de solutions, effectuez un clic droit sur le nom du projet Silverlight puis sélectionnez Ajouter>Élément existant. Localisez ensuite sur votre ordinateur le fichier d'extension .ttf souhaité. Celui-ci sera ensuite copié dans le répertoire du projet et figurera parmi les éléments faisant partie du projet.

Cet élément de projet doit être inclus en ressource (il sera ainsi ajouté au fichier d'extension .xap envoyé au client avec la page Web). Pour cela, effectuez un clic droit sur le fichier .ttf puis sélectionnez Propriétés>Action de génération>Resource.

Ça n'est pas plus compliqué ! Il suffit à présent de modifier l'attribut Font dans la balise de l'élément visuel de la manière suivante :

```
<TextBlock Text="Police Scriptina" FontSize="50" .....  
FontFamily="ScriptIn.ttf#Scriptina" />
```

Figure 5-7



La figure 5-7 présente le résultat obtenu. Cette opération augmente certes la taille du fichier d'extension .xap transmis au client mais de quelques dizaines de Ko seulement, soit guère plus qu'une image, même en basse définition. Cela est dû notamment au fait que le fichier XAP est un fichier compressé (avec compression ZIP).

Les zones d'édition (TextBox)

Une zone d'édition (objet TextBox) permet de saisir du texte sur une ou plusieurs lignes (voir l'attribut AcceptsReturn pour cette dernière possibilité). L'application Silverlight peut lire son contenu (propriété Text), initialiser celui-ci ou encore sélectionner du texte

dans ce contenu ou lire la partie de texte sélectionnée par l'utilisateur. Les opérations de couper-coller sont donc possibles.

Nous ne reviendrons pas sur les attributs `Background` (couleur de fond), `Foreground` (couleur d'affichage), ceux de la famille `Font` et les attributs `HorizontalAlignment` et `VerticalAlignment` qui indiquent l'alignement de la zone d'édition dans un conteneur comme une cellule de grille ou un `StackPanel`. Les attributs `Width` et `Height` seront également ignorés ici.

Le tableau 5-3 présente un certain nombre de propriétés (notamment celles liées à la sélection de texte) présentant un intérêt en cours d'exécution de programme seulement.

Tableau 5-3 – Les propriétés des zones d'édition

Nom de l'attribut	Description
<code>AcceptsReturn</code>	Si cette propriété vaut <code>true</code> (ce qui est le cas par défaut), la zone d'édition peut s'étendre sur plusieurs lignes (la touche <code>Entrée</code> permet de forcer un saut de ligne).
<code>BorderBrush</code>	Pinceau (donc couleur) utilisé pour dessiner la ligne de contour. Par exemple : <code>BorderBrush = "Red"</code>
<code>BorderThickness</code>	Épaisseur du contour autour de la zone d'édition. Aucun contour n'est affiché si <code>BorderThickness</code> vaut 0 (la valeur par défaut est 1).
<code>IsReadOnly</code>	L'utilisateur ne peut pas modifier le contenu de la zone d'édition si <code>IsReadOnly</code> vaut <code>true</code> (<code>false</code> par défaut).
<code>IsTabStop</code>	Si cet attribut a <code>false</code> pour valeur (<code>true</code> est la valeur par défaut), il n'y a pas d'arrêt sur cette zone d'édition lors d'une navigation de composant à composant par la touche <code>Tab</code> .
<code>Padding</code>	Par défaut, le contour borde au plus près le contenu de la zone d'édition ou s'ajuste sur <code>Width</code> et <code>Height</code> . <code>Padding</code> permet d'ajouter une marge entre la zone d'édition et la bordure. Une, deux ou quatre valeurs peuvent être spécifiées, comme pour <code>Margin</code> (voir la section « La grille » du chapitre 3). <code>Padding</code> correspond à un espacement à l'intérieur de la bordure tandis que <code>Margin</code> correspond à un espacement à l'extérieur du composant. Par exemple :
	<code>Padding="10"</code> 10 pixels sont ajoutés à chaque bord.
	<code>Padding="10, 5"</code> 10 pixels sont ajoutés à gauche et à droite et 5 pixels sont ajoutés au-dessus et au-dessous.
<code>SelectedText</code>	Texte sélectionné soit par l'utilisateur (par l'opération usuelle de sélection de texte à l'aide de la souris ou du clavier), soit par du code (à l'aide de la fonction <code>Select</code> appliquée à la zone d'édition).
<code>SelectionBackground</code>	Couleur de fond du texte sélectionné.
<code>SelectionLength</code>	Nombre de caractères du texte sélectionné.
<code>SelectionStart</code>	Déplacement (0 pour le premier) du premier caractère de la zone de sélection.
<code>TabIndex</code>	Ordre de passage sur la zone d'édition lors d'une navigation (passage d'un élément UI à l'autre) par la touche <code>Tab</code> .
<code>Text</code>	Contenu de la zone d'édition.
<code>TextAlignment</code>	Alignement du texte dans la zone d'édition. Les valeurs possibles sont <code>Left</code> (valeur par défaut), <code>Center</code> et <code>Right</code> .

Tableau 5-4 – Les événements relatifs aux zones d'édition

Nom de l'événement	Description
SelectionChanged	Événement signalé aussitôt qu'une sélection de texte démarre ou est modifiée.
TextChanged	Événement signalé aussitôt que le contenu de la zone d'édition est modifié (voir la section « Les événements » du chapitre 6 pour le traitement des événements).

Une zone d'édition destinée à recevoir un mot de passe est obtenue en attribuant la même couleur à Background et Foreground (ce qui présente l'avantage d'empêcher tout curieux d'estimer le nombre de caractères du mot de passe).

Les boutons

Nul besoin de présenter le bouton de commande puisqu'il s'agit du plus connu des composants et cela depuis les premiers programmes Windows il y a un quart de siècle. Nous verrons au chapitre 6 comment traiter le clic sur le bouton (événement Click) et au chapitre 15, comment personnaliser un bouton de commande. À ce stade de l'étude, nous nous contenterons d'un simple libellé.

On retrouve dans une balise Button des attributs désormais familiers : Background, Foreground, ceux de la famille Font, Width, Height, Margin, Padding, etc.

Le tableau 5-5 présente les propriétés propres aux boutons.

Tableau 5-5 – Les propriétés des boutons

Nom de l'attribut	Description
ClickMode	Indique à quel moment l'événement Click est signalé (et peut dès lors être traité par programme) :
Hover	Quand la souris survole le bouton (plus précisément lorsqu'elle entre dans la surface du bouton).
Press	Au moment où le bouton est enfoncé.
Release	Au moment où le bouton est relâché. Il s'agit bien sûr de la valeur par défaut. Les deux autres valeurs (et surtout la première) ne peuvent mener qu'à une utilisation atypique susceptible de désorienter les utilisateurs.
Content	Libellé du bouton. Par défaut, il s'agit d'un texte mais le libellé peut prendre une forme bien plus complexe (voir la personnalisation des boutons au chapitre 15).

À noter que l'événement Click signale que l'utilisateur a cliqué sur le bouton.

Pour un bouton placé dans une cellule d'une grille, la balise peut être aussi simple que :

```
<Button Content="GO" />
```

Voici à quoi ressemble le code suite à l'ajout de couleurs et à la spécification de la taille :

```
<Button Content="GO" Foreground="Red" Background="Silver"
        FontSize="20" Width="100" Height="80" />
```

Il est également possible d'ajouter une info-bulle (affichée quand la souris survole le bouton) :

```
<Button ..... ToolTipService.ToolTip="Ceci est une aide sur GO" />
```

L'attribut `Content` peut être redéfini pour modifier considérablement l'apparence du bouton. Celui-ci (cas simple) pourrait contenir une image ou une vidéo. Nous étudierons la personnalisation des boutons au chapitre 15, où nous aborderons les *templates* qui permettent une modification complète de l'apparence (et surtout, à destination des graphistes, Expression Blend avec sa technologie Visual State Manager).

Grâce aux connaissances acquises jusqu'à présent, vous pouvez déjà personnaliser quelque peu (si peu au regard de ce qui est possible) le bouton. Dans l'exemple de code suivant, le contenu d'affichage du bouton est inséré dans une grille limitée à une seule cellule :

```
<Button FontSize="20" Width="100" Height="80"
        Grid.Row="1" Grid.Column="1">
    <Button.Content>
        <Grid>
            <Ellipse Fill="IndianRed" Width="60" Height="40"/>
            <TextBlock Text="GO" VerticalAlignment="Center"
                HorizontalAlignment="Center" />
        </Grid>
    </Button.Content>
</Button>
```

La figure 5-8 illustre le résultat obtenu.

Figure 5-8



La boîte à outils de Silverlight propose deux variantes du bouton : le `RepeatButton` et le `ToggleButton`.

Voyons en quoi elles diffèrent du bouton « normal ». Tout d'abord, celui-ci génère l'événement `Click` lors du relâchement du bouton (événement `MouseLeftButtonUp`). Le bouton `RepeatButton`, quant à lui, génère l'événement `Click` à intervalles très rapprochés tant que l'utilisateur garde le bouton de la souris enfoncé. Le tableau 5-6 présente les deux attributs propres aux boutons `RepeatButton`.

Tableau 5-6 – Les propriétés propres aux boutons RepeatButton

Nom de l'attribut	Description
Delay	Délai (en millisecondes) entre le clic (événement MouseButtonDown) et la génération du premier événement Click. Par défaut, cette valeur est de 250 millisecondes.
Interval	Durée (en millisecondes) entre deux événements Click (tant que le bouton de la souris reste enfoncé). Par défaut, cette valeur est de 250 millisecondes.

Par ailleurs, le bouton `ToggleButton` garde son état visuel après relâchement. La propriété booléenne `IsChecked` donne l'état du bouton à un moment donné. Sans personnalisation du bouton (voir chapitre 15), la différence est malheureusement peu perceptible au niveau visuel.

Créer un bouton dans Expression Blend

Créer et modifier un bouton à l'aide d'Expression Blend est un jeu d'enfant. Si l'icône du bouton n'est pas visible dans la boîte d'outils, cliquez sur l'icône symbolisée par des chevrons fermants (ici, dans la partie inférieure de la boîte d'outils) et maintenez enfoncé le bouton de la souris pendant une seconde.

Le tableau des composants susceptibles d'être utilisés apparaît alors (figure 5-9). Sélectionnez le bouton, insérez-le dans la surface de travail (par un clic dans la boîte d'outils suivi d'un autre dans la surface de travail) et modifiez ses propriétés dans le panneau des propriétés d'Expression Blend (panneau de droite).

Figure 5-9



Les cases à cocher (CheckBox)

Les cases à cocher sont aussi trop connues pour être présentées longuement et dans le détail. Disons simplement qu'elles peuvent être à deux ou trois états :

- deux états : état `Checked` (case cochée) et `Unchecked` (case non cochée) si la propriété `IsThreeState` vaut `false` (valeur par défaut),
- trois états : les deux états précédents ainsi qu'un état dit indéterminé si la propriété `IsThreeState` vaut `true`.

Le tableau 5-7 présente les propriétés propres aux cases à cocher.

Tableau 5-7 – Les propriétés des cases à cocher

Nom de l'attribut	Description
<code>ClickMode</code>	Voir l'attribut <code>ClickMode</code> (avec ses valeurs <code>Hover</code> , <code>Press</code> et <code>Release</code>) des boutons présentés au tableau 5-5, avec la réserve qui a été émise concernant une utilisation non standard.
<code>IsChecked</code>	Indique si la case est cochée ou non (<code>true</code> ou <code>false</code>).
<code>IsEnabled</code>	Indique si la case est activable ou non (l'utilisateur ne peut pas modifier l'état de la case si <code>IsEnabled</code> a <code>false</code> pour valeur).
<code>Content</code>	Libellé de la case.
<code>IsThreeState</code>	Indique s'il s'agit d'une case à trois états (<code>true</code> ou <code>false</code>).

Tableau 5-8 – Les événements liés aux cases à cocher

Nom de l'événement	Description
<code>Checked</code>	Événement signalé quand la case passe à l'état « case cochée ».
<code>Indeterminate</code>	Événement signalé quand la case passe à l'état « indéterminé ».
<code>Unchecked</code>	Événement signalé quand la case passe à l'état « case non cochée ».

Le traitement des événements est présenté au chapitre suivant.

Les boutons radio (RadioButton)

Un bouton radio fait généralement partie d'un groupe (attribut `GroupName`, de type chaîne de caractères). Un seul bouton radio du groupe peut être coché à un moment donné (celui pour lequel `IsChecked` vaut `true`). Tout clic sur un bouton a pour effet de le cocher et de décocher le bouton qui était coché auparavant.

Par exemple :

```
<RadioButton x:Name="rbCélibataire" Content="Célibataire"
    GroupName="EtatCivil" IsChecked="true" />
<RadioButton x:Name="rbMarié" Content="Marié"
    GroupName="EtatCivil"/>
<RadioButton x:Name="rbDivorcé" Content="Divorcé"
    GroupName="EtatCivil"/>
<RadioButton x:Name="rbVeuf" Content="Veuf"
    GroupName="EtatCivil"/>
```

La figure 5-10 illustre le résultat obtenu.

Figure 5-10



Les boutons hyperliens

Ce composant, qui permet de naviguer d'une page Web à une autre, est lui aussi d'une simplicité enfantine et a le même effet que la balise `<a>` du HTML (l'apparence du composant Silverlight peut néanmoins être complètement redéfinie).

Le tableau 5-9 présente les propriétés propres aux boutons `HyperlinkButton`.

Tableau 5-9 – Les propriétés des boutons `HyperlinkButton`

Nom de l'attribut	Description
Content	Libellé du lien.
NavigateUri	URL du site de redirection.
TargetName	Si cet attribut vaut <code>_blank</code> , la nouvelle page Web sera affichée dans une autre page (par défaut, la nouvelle page remplace la page existante).
Tooltip	Contenu de l'info-bulle affichée quand la souris marque l'arrêt au-dessus du composant.

Par exemple :

```
<HyperlinkButton Content="Silverlight"
    ToolTipService.ToolTip
        ="Vers le site Microsoft de Silverlight"
    NavigateUri="http://silverlight.net"
    TargetName="_blank" />
```

La figure 5-11 illustre le résultat obtenu.

Figure 5-11



Le composant Slider

Le composant `Slider` permet de sélectionner une valeur (comprise entre `Minimum` et `Maximum`) en faisant glisser un curseur à l'aide de la souris.

Le tableau 5-10 présente les propriétés propres au composant `Slider`.

Tableau 5-10 – Les propriétés du composant `Slider`

Nom de l'attribut	Description
<code>IsDirectionReversed</code>	Si cette propriété vaut <code>true</code> , les plus petites valeurs sont situées à droite (quand <code>Orientation</code> vaut <code>Horizontal</code>) ou en bas (quand <code>Orientation</code> vaut <code>Vertical</code>). La valeur par défaut est <code>false</code> .
<code>LargeChange</code>	Valeur d'incrément ou de décrémentation du curseur, lorsque l'utilisateur clique sur la barre.
<code>Orientation</code>	Orientation : <code>Horizontal</code> (valeur par défaut) ou <code>Vertical</code> .
<code>Maximum</code>	Valeur maximale, de type <code>double</code> (avec 0 comme valeur par défaut).
<code>Minimum</code>	Valeur minimale, de type <code>double</code> (avec 10 comme valeur par défaut).
<code>Value</code>	Valeur correspondant à la position du curseur, entre <code>Minimum</code> et <code>Maximum</code> .

À noter que l'événement `ValueChanged` est signalé quand le curseur est déplacé.

Par exemple :

```
<Slider x:Name="slider" Minimum="1" Maximum="100"
        LargeChange="10" Width="200" Height="20"
        ValueChanged="slider_ValueChanged" />
```

La figure 5-12 illustre le résultat obtenu.

Figure 5-12



Les attributs `Width` et `Height` doivent être spécifiés car, sinon, le `Slider` se comporte comme un panneau et occupe toute la largeur et toute la hauteur de la fenêtre du navigateur.

La fonction `slider_ValueChanged` est automatiquement exécutée lorsque l'utilisateur déplace le curseur. Dans cette fonction de traitement, `slider.Value` donne la valeur correspondant au curseur (ici, entre 1 et 100).

Ces propriétés peuvent être initialisées dans le panneau de droite d'Expression Blend (figure 5-13).

Figure 5-13



La barre de défilement (ScrollBar)

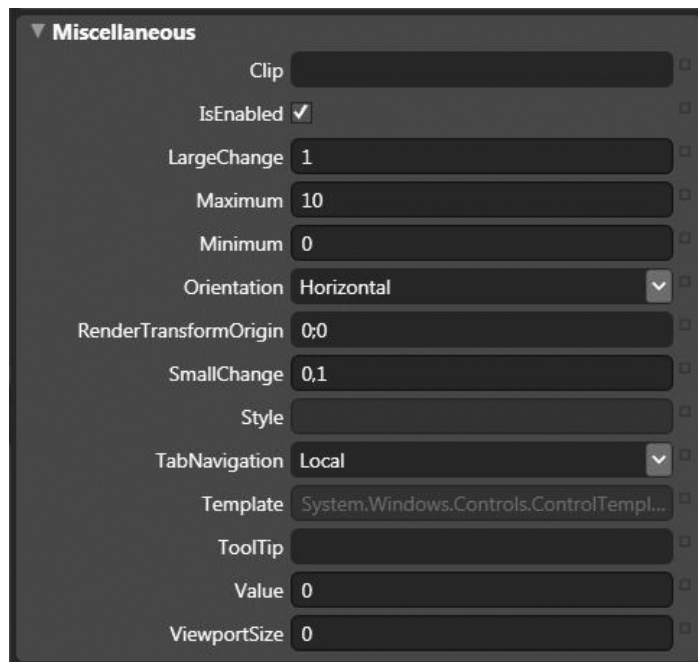
La barre de défilement `ScrollBar` (figure 5-14) permet de sélectionner une valeur, tout comme on le fait avec un `Slider`.

Figure 5-14



Ses propriétés peuvent également être initialisées dans le panneau de droite d'Expression Blend (figure 5-15).

Figure 5-15



À la section « Modifier n'importe quel contrôle avec Expression Blend » du chapitre 15, nous verrons comment modifier avec Expression Blend n'importe quel sous-élément de la barre de défilement, par exemple le curseur.

Le calendrier (Calendar)

Le composant `Calendar` affiche un calendrier et permet à l'utilisateur de sélectionner une date ou une période. Par défaut, la date du jour est présélectionnée.

Comme c'est le cas pour le `GridSplitter` (voir la section « Le `GridSplitter` » du chapitre 3), le code du calendrier ne fait pas partie du run-time Silverlight. À la suite d'un glisser-déposer de la boîte d'outils vers la fenêtre du code XAML, Visual Studio ajoute le code suivant :

```
xmlns:my="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Extended"
```

qui signifie :

- le composant `Calendar` fait partie de l'espace de noms `System.Windows.Controls` ;
- son code se trouve dans la DLL `System.Windows.Controls.Extended.dll` qui sera greffée au fichier XAP et ainsi transmise avec le code de l'application au navigateur ;
- il faut préfixer `Calendar` de `my:` (vous pouvez choisir un autre préfixe, il suffit de modifier l'attribut `xmlns:my`) pour faire référence au composant `Calendar` dans les balises XAML.

La figure 5-16 représente un exemple de calendrier.

Figure 5-16



L'utilisation du calendrier est intuitive et trop connue pour s'y attarder. Passons donc directement à ses propriétés (tableau 5-11).

À noter que l'événement `DateSelected` est signalé quand l'utilisateur sélectionne une date (par un clic).

Toutes ces propriétés peuvent être initialisées dans le panneau de droite d'Expression Blend.

Tableau 5-11 – Les propriétés du composant Calendar

Nom de l'attribut	Description
SelectedDate.HasValue	Booléen qui indique si l'utilisateur a sélectionné une date.
SelectedDate	Propriété de type DateTime qui indique la date sélectionnée.
DisplayDate	Par défaut, Silverlight affiche initialement le mois de la date du jour. En initialisant DisplayDate (de type DateTime), un autre mois peut être affiché.
DisplayDateStart DisplayDateEnd	Propriétés de type DateTime qu'il est toujours plus prudent d'initialiser par programme pour des raisons de formats de dates (fort différents d'une région à l'autre du monde). Par défaut, toutes les dates peuvent être sélectionnées. DisplayDateStart et DisplayDateEnd permettent de spécifier un intervalle de dates en dehors duquel les dates ne sont pas affichées, ce qui limite automatiquement le choix de la date : cal.DisplayDateStart = new DateTime(2008, 12, 25);
SelectableDateStart SelectableDateEnd	Si ces dates sont spécifiées, l'utilisateur ne peut pas sélectionner une date en dehors de cette période.

Le composant DatePicker

Ce composant permet de saisir une date, soit dans une zone d'édition, soit en la sélectionnant dans un calendrier. Il peut donc apparaître de deux manières à l'utilisateur (figures 5-17 et 5-18). Il suffit de cliquer sur l'icône en forme de calendrier pour passer d'une représentation à l'autre.

Figure 5-17



Figure 5-18



L’affichage, ainsi que le format de la date, s’adapte automatiquement aux caractéristiques régionales de l’utilisateur.

Son code fait partie de la même DLL que `Calendar`. Dans une balise, il faut donc préfixer `DatePicker` de `my` (préfixe par défaut).

On retrouve les mêmes propriétés que pour `Calendar`.

Le composant `DatePickerTextBox` (de la même famille) correspond à une zone d’édition de saisie de date.

6

Du code dans les applications Silverlight

Les événements

La notion d'événement

Lors d'un clic sur un bouton, d'une frappe sur une touche, du chargement de la page dans le navigateur, d'un clic de la souris, lorsque celle-ci se déplace ou survole un composant (plus précisément lorsqu'elle entre ou sort de la surface occupée par le composant), etc., donc « lorsque quelque chose se passe », le run-time Silverlight vous en informe en appelant une fonction (dite de traitement) de votre programme.

Il n'y a là rien de nouveau pour ceux qui ont déjà pratiqué la programmation Windows. La technique devient maintenant applicable aux pages Web, ce qui rend possible la création d'applications Web aussi conviviales et interactives que les applications Windows !

La fonction de traitement (écrite en C# ou VB) est exécutée sur la machine client, dans le navigateur, sous contrôle du run-time Silverlight (autrement dit, sans pouvoir interférer avec les autres programmes et sans pouvoir accéder au système de fichiers, sauf la zone de stockage isolé, comme nous l'expliquerons à la section « Le stockage isolé avec Isolated-Storage » du chapitre 11).

Dans cette fonction de traitement, vous avez accès à toutes les possibilités du C# ou de VB et pouvez utiliser les classes de l'environnement .NET, dont on sait qu'elles facilitent grandement la réalisation des programmes. Ces classes, à quelques restrictions près, sont celles qui sont utilisées en programmation Windows ou pour la programmation des mobiles.

Le fonctionnement sous Silverlight est très différent du fonctionnement des programmes Web sous ASP.NET ou PHP, type de programmation également appelé programmation serveur. Ces programmes Web s'exécutent sur le serveur (par exemple, chez votre hébergeur) et génèrent du code HTML (contenant souvent du JavaScript) à partir des balises ASP.NET ou PHP incluses dans la page. Ces lignes de code HTML sont alors envoyées au navigateur du client, qui les interprète et en affiche le rendu dans la fenêtre du navigateur.

En programmation côté serveur, le navigateur (qui s'exécute sur la machine client) informe le serveur (qui peut être situé à des milliers de kilomètres) qu'un bouton a été cliqué. Une fonction (qui traite le clic sur le bouton) est alors exécutée sur le serveur. Comme celui-ci traite un grand nombre de requêtes en provenance d'une multitude de clients, il n'est pas difficile d'imaginer l'impact sur ses performances. Impossible donc de rivaliser avec les applications Windows.

Généralement, cette fonction de traitement exécutée sur le serveur génère une nouvelle page (contenant du HTML, éventuellement du JavaScript, mais rien d'autre) qui est envoyée au navigateur du client. Avec des techniques comme Ajax, il est cependant possible de faire en sorte que cette fonction n'effectue que la mise à jour d'une partie de la page, ce qui améliore déjà les performances et l'interactivité. Mais rien encore de comparable avec les programmes Windows et maintenant les applications Silverlight. C'est pour cette raison (due au modèle de la programmation serveur) qu'un programme ASP.NET ou PHP ne peut pas traiter des événements comme `MouseMove` correspondant au déplacement de la souris. Cela serait possible en théorie mais en pratique, le délai de traitement ne serait pas raisonnable, sans compter de la charge additionnelle sur le trafic.

Avec Silverlight, le traitement (à l'exception des services Web qui sont déportés sur un serveur, voir chapitre 13) reste local, sur la machine du client. Ceci explique pourquoi une application Silverlight répond bien plus promptement aux sollicitations des utilisateurs, ce qui offre une interactivité inimaginable avec les programmes Web s'exécutant sur le serveur.

Nous allons maintenant nous intéresser aux événements liés aux applications Silverlight.

L'événement Loaded

L'un des premiers événements signalés à l'application Silverlight est `Loaded`. Il est signalé quand la page Silverlight est chargée par le navigateur, avant même qu'elle ne s'affiche à l'écran. Traiter cet événement donne l'occasion de peaufiner la page (avant son premier affichage) et de l'adapter au client, à l'environnement, au contexte, à l'heure, etc.

Pour traiter un événement, il faut lui associer une fonction de traitement. Il existe plusieurs techniques pour cela, ce que nous verrons avec les événements liés à la souris, exemples les plus parlants. Une fois la fonction (automatiquement générée par Visual Studio) associée à l'événement, vous la retrouverez dans le fichier `Page.xaml.cs` ou `Page.xaml.vb`, selon le langage choisi pour le développement de l'application. Dans l'Explorateur de solutions, ces fichiers se trouvent sous l'article `Page.xaml`. Cliquez sur le signe + en regard

de `Page.xaml` pour faire apparaître le nom du fichier de code et double-cliquez sur celui-ci pour l'éditer.

Les événements liés à la souris

Quand le run-time Silverlight détecte un événement lié à la souris, une fonction de l'application Silverlight (celle de traitement de l'événement) est automatiquement exécutée, avec de l'information additionnelle (propre à l'événement en question) passée en arguments. Encore faut-il avoir signalé que l'on désire traiter l'événement en question.

Le tableau 6-1 présentent les événements liés à la souris et susceptibles d'être traités pour chaque élément UI (rectangle, zone d'affichage, bouton, etc.).

Tableau 6-1 – Les événements liés à la souris

Nom de l'événement	Description
MouseMove	La souris se déplace au-dessus de la surface de l'élément UI. Le second argument de la fonction de traitement indique la position de la souris. Cet événement est signalé de manière très rapprochée tant que la souris est en mouvement.
MouseLeftButtonDown	Le bouton de la souris vient d'être enfoncé (première phase du clic).
MouseLeftButtonUp	Le bouton est relâché (dernière phase du clic).
MouseEnter	La souris entre dans la surface de l'élément UI.
MouseLeave	La souris sort de la surface de l'élément UI.

Comment signaler le traitement d'un événement ?

Pour cela, il convient tout d'abord de nommer (attribut `x:Name`) l'élément UI concerné par l'événement. Il est préférable de lui attribuer un nom qui désigne clairement le composant, ce qui facilitera le travail du programmeur mais aussi celui de la personne qui devra en assurer la maintenance (laisser des noms par défaut comme `button1`, `button2`, etc., devrait être proscrit).

Par exemple, pour ce rectangle rouge :

```
<Rectangle x:Name="rcRouge" Fill="Red" ..... />
```

La première technique pour traiter un événement (ici, `MouseMove`) consiste à ajouter un attribut dans la balise de l'élément UI :

```
MouseMove =
```

Dès la saisie du signe `=` après un nom d'événement, Visual Studio propose de compléter la balise et de construire automatiquement la fonction de traitement. Il vous suffit d'appuyer sur la touche `Tab` une première fois (pour confirmer la génération automatique de la valeur de l'attribut) et puis une seconde fois (pour la génération automatique de la fonction de traitement). Il est également possible de retenir une fonction existante comme fonction de traitement (celle-ci traite alors un même événement pour plusieurs éléments UI).

Le tableau 6-2 présente la balise automatiquement complétée par Visual Studio ainsi que la fonction de traitement pour C# et VB automatiquement générée.

Tableau 6-2 – Génération de la fonction de traitement par ajout d'un attribut

XAML
<Rectangle x:Name="rcRouge" MouseMove="rcRouge_MouseMove" />
Fonction de traitement C# automatiquement générée
<pre>void rcRouge_MouseMove(object sender, MouseEventArgs e) { // saisissez votre code de traitement ici }</pre>
Fonction de traitement VB automatiquement générée
<pre>Private Sub rcRouge_MouseMove(ByVal sender As System.Object, _ ByVal e As System.Windows.Input.MouseEventArgs) ' saisissez votre code de traitement ici End Sub</pre>

En C#, Visual Studio génère une instruction `throw` dans le corps de la fonction de traitement mais vous pouvez supprimer cette ligne et la remplacer par votre propre code (en l'occurrence, ce que vous souhaitez exécuter comme instructions quand la souris se déplace au-dessus du rectangle rouge).

En VB, vous pouvez réduire `System.Windows.Input.MouseEventArgs` à `MouseEventArgs` mais assurez-vous tout de même que `System.Windows.Input` fait bien partie des Imports.

Pour l'événement `MouseEnter`, la fonction de traitement serait alors ici `rcRouge_MouseEnter`, et ainsi de suite pour les autres événements.

L'argument `sender` de la fonction de traitement indique l'objet qui a généré l'événement. En raison de la propagation d'événements (*bubbling*, voir plus loin), il ne s'agit pas nécessairement de l'objet qui est à l'origine de l'événement (celui-ci est donné par `e.Source`). Le second argument donne des informations sur l'événement et est de type `MouseEventArgs`, dérivé de `EventArgs`, dans le cas de la souris.

La seconde technique permettant de traiter un événement consiste à l'associer par programme (en cours d'exécution donc) à une fonction de traitement du programme. Généralement, cette association est effectuée dans le constructeur de la classe `Page` (après l'appel à `InitializeComponent`) ou dans la fonction traitant l'événement `Loaded` (cette dernière solution étant préférable car on est alors sûr que toutes les initialisations ont été effectuées).

En C#, il suffit de saisir `rc.MouseMove +=` puis d'appuyer (à l'invitation de Visual Studio) deux fois sur touche `Tab` pour que Visual Studio complète la phrase et génère la fonction de traitement. En VB, c'est moins automatique mais deux solutions sont possibles (tableau 6-3).

Tableau 6-3 – Ajout dynamique d'une fonction de traitement

C#
<code>rcRouge.MouseMove += new MouseEventHandler(rcRouge_MouseMove);</code>
VB
<p>Première solution : aucune instruction à exécuter mais une clause <code>Handles</code> doit être ajoutée à la fonction de traitement (la fonction <code>rcRouge_MouseMove</code> devient ici la fonction de traitement de l'événement <code>MouseMove</code> adressé à l'élément <code>rcRouge</code>).</p> <pre>Public Sub rcRouge_MouseMove(ByVal o As Object, ByVal e As MouseEventArgs) _ Handles rcRouge.MouseMove</pre> <p>End Sub</p> <p>Deuxième solution : exécuter l'instruction suivante (dans ce cas, pas de clause <code>Handles</code> et on se rapproche de ce qu'il faut faire en C#) :</p> <pre>AddHandler rcRouge.MouseMove, AddressOf rcRouge_MouseMove</pre>

La fonction de traitement `rcRouge_MouseMove` sera automatiquement exécutée (elle est en fait appelée par le run-time Silverlight) quand la souris se déplacera dans les limites du rectangle `rcRouge`, exception faite des parties de ce rectangle qui seraient recouvertes par d'autres éléments UI. L'événement `MouseMove` est signalé, à intervalles très rapprochés, pour chaque déplacement élémentaire de la souris.

L'instruction suivante :

```
rcRouge.MouseMove += new MouseEventHandler(rcRouge_MouseMove);
```

automatiquement générée par Visual Studio, pourrait être simplifiée et s'écrire :

```
rcRouge.MouseMove += rcRouge_MouseMove;
```

mais autant s'en tenir au code automatiquement généré.

Une fonction de traitement peut être annulée à tout moment en exécutant :

```
rcRouge.MouseMove -= rcRouge_MouseMove;
```

en C#, et en VB :

```
RemoveHandler rcRouge.MouseMove, AddressOf rcRouge_MouseMove
```

Pour information, l'instruction et la fonction de traitement précédentes pourraient également s'écrire (ici, dans le constructeur de la classe `Page`, après l'appel de la fonction `InitializeComponent`) :

```
MouseMove += delegate (object sndr, MouseEventArgs evt)
{
    .....
}
```

Les événements liés à la souris (sauf `MouseEnter` et `MouseLeave`) sont reportés (*bubbled* en anglais) le long de la hiérarchie enfant-parents donc, ici, au conteneur du rectangle ainsi qu'à l'éventuel conteneur de ce conteneur (et ainsi de suite jusqu'au conteneur principal).

Autrement dit, vous pouvez traiter l'événement `MouseMove` pour le rectangle (dans la fonction `rcRouge_MouseMove`) mais aussi pour le conteneur (par exemple, dans `LayoutRoot_MouseMove`). La fonction `rcRouge_MouseMove` est d'abord exécutée, puis c'est au tour de la fonction `LayoutRoot_MouseMove`, dans laquelle l'argument `e.Source` indique quel élément UI est concerné et `((FrameworkElement)e.Source).Name` donne le nom de l'élément (ici, "rcRouge" ou "LayoutRoot"). On pourrait écrire :

```
FrameworkElement fe = e.Source as FrameworkElement;
```

en C#, et en VB :

```
Dim fe as FrameworkElement = e.Source
```

puis accéder à `fe.Name`, qui indique, sous la forme d'une chaîne de caractères, le nom (attribut `x:Name`) de l'élément concerné.

Pour mettre fin à cette propagation, il suffit de passer le champ `e.Handled` du second argument à `true`.

Comment détecter la position de la souris ?

Le premier argument (appelé `sender`) de la fonction de traitement indique quel élément a généré l'événement (sans oublier qu'à cause de la propagation, il ne s'agit pas nécessairement de l'élément qui en est à l'origine). Généralement, vous savez quel élément est concerné puisque vous avez écrit une fonction de traitement pour celui-ci (dans les exemples précédents, il s'agissait du rectangle `rcRouge`). Mais quand une même fonction traite un même événement pour plusieurs éléments ou quand un même événement peut être traité à différents niveaux (dans la hiérarchie enfant-parents, en tirant parti de la propagation d'événements), il est nécessaire de déterminer l'élément qui signale l'événement.

Le second argument de la fonction de traitement donne accès à diverses informations, notamment `e.Source`, qui désigne l'élément à l'origine de l'événement mais aussi la position de la souris.

Le tableau 6-4 présente les différents attributs possibles pour le second argument (de type `MouseEventArgs`) de la fonction de traitement.

Tableau 6-4 – Les attributs du second argument de la fonction de traitement

Nom de l'attribut	Description
<code>GetPosition(UI Element)</code>	La fonction <code>GetPosition</code> renvoie un objet de type <code>Point</code> qui donne les coordonnées de la souris, relativement à l'élément UI passé en argument.
<code>Handled</code>	Valeur booléenne par laquelle on indique qu'un traitement a été effectué. Si <code>e.Handled</code> passe à <code>true</code> , l'événement n'est plus signalé aux niveaux supérieurs dans la hiérarchie enfants-parents.
<code>Source</code>	Référence à l'élément qui est à l'origine de l'événement.

Prenons un exemple. Voici le code permettant d'afficher (dans un `TextBlock` nommé `zaInfo`) la position de la souris dans le rectangle `rc` :

```
<TextBlock x:Name="zaInfo" ..... />
<Rectangle x:Name="rc" MouseMove="rc_MouseMove" ..... />
```

En C#, cela donne :

```
void rc_MouseMove(object sender, MouseEventArgs e)
{
    Point p = e.GetPosition(rc);
    zaInfo.Text = p.X + " - " + p.Y;
}
```

Et en VB :

```
Public Sub rc_MouseMove(ByVal sender As Object, _
                        ByVal e As MouseEventArgs)
    Dim p As Point
    p = e.GetPosition(rc)
    zaInfo.Text = p.X & " - " & p.Y
End Sub
```

On aurait pu se passer de la déclaration de la variable de type `Point` et écrire directement :

```
e.GetPosition(rc).X
```

Pour déterminer la position de la souris dans la page, l'argument de `GetPosition` aurait dû être `this` en C# et `Me` en VB. Une meilleure solution consiste à passer en argument le nom interne du conteneur (`LayoutRoot` pour le conteneur principal).

Comment traiter le clic sur un bouton ?

Considérons le code suivant (remplacez par d'autres attributs comme la position, la largeur, la hauteur, etc.) :

```
<Button x:Name="bG0" Content="GO !" ..... />
```

et voyons comment traiter le clic sur le bouton (événement `Click`).

Pour cela, en attribut de la balise `Button`, saisissez `Click=`. Comme pour n'importe quel événement, Visual Studio propose de compléter la balise et de générer automatiquement la fonction de traitement. Le nom de cette fonction sera ici `bG0_Click` et l'argument de type `RoutedEventArgs`. L'événement `Click` n'est pas reporté au conteneur, bien que les événements `MouseDown` et `MouseUp` sur le bouton le soient.

Le seul champ de la classe `RoutedEventArgs` qui présente de l'intérêt (quand une même fonction traite le clic pour plusieurs boutons) est `Source`, de type `object`. Celui-ci fait référence au bouton sur lequel l'utilisateur a cliqué. Il est généralement converti en un `FrameworkElement`, qui est la classe de base des éléments UI.

En C#, cela donne :

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

Et en VB :

```
Private Sub bGO_Click(ByVal sender As System.Object, _
    ByVal e As System.Windows.RoutedEventArgs)

End Sub
```

Plusieurs boutons peuvent partager la même fonction de traitement :

```
<Button x:Name="bAction1" Content="Action 1" Click="bAction_Click" ..... />
<Button x:Name="bAction2" Content="Action 2" Click="bAction_Click" ..... />
```

En C#, cela donne :

```
private void bAction_Click(object sender, RoutedEventArgs e)
{
    FrameworkElement fe = e.Source as FrameworkElement;
    // fe.Name contient le nom interne du bouton (ici, bAction1 ou bAction2)
    if (e.Name == "bAction1") .....
}
```

Et en VB :

```
Private Sub bAction_Click(ByVal sender As System.Object, _
    ByVal e As System.Windows.RoutedEventArgs)

    Dim fe As FrameworkElement = e.Source
    ' fe.Name contient le nom interne du bouton (ici, bAction1 ou bAction2)

End Sub
```

Ici, on aurait pu utiliser sender plutôt que e.Source car l'événement est traité au niveau même de l'objet qui en est responsable et non à un niveau supérieur dans la hiérarchie enfant-parents.

Les événements liés au clavier

En ce qui concerne le clavier, il est possible de traiter les événements présentés au tableau 6-5.

Tableau 6-5 – Les événements liés au clavier

Nom de l'événement	Description
KeyDown	Enfoncement d'une touche.
KeyUp	Relâchement de la touche.

Ces événements ne sont généralement traités que pour détecter la touche Entrée et faire jouer à la frappe de cette touche le rôle du clic sur un bouton.

Le second argument de la fonction de traitement est de type `EventArgs`. Les champs `Handled` et `Source` ont déjà été abordés et ont la même signification que pour les événements liés à la souris.

Le tableau 6-6 présente les champs de l'argument (de type `EventArgs`) de la fonction de traitement.

**Tableau 6-6 – Les différents champs de l'argument
(de type `EventArgs`) de la fonction de traitement**

Nom du champ	Description
Key	Code de la touche. La valeur de <code>Key</code> est l'une des valeurs de l'énumération <code>Key</code> : <code>Key.A</code> à <code>Key.Z</code> (on retrouve uniquement les majuscules car aucune distinction n'est effectuée lors du traitement de l'événement entre majuscule et minuscule), <code>Key.Back</code> , <code>Key.CapsLock</code> (enclenchement de majuscule), <code>Key.Alt</code> (Alt Gr), <code>Key.Ctrl</code> , <code>Key.Escape</code> , <code>Key.Shift</code> , <code>Key.Home</code> , <code>Key.End</code> , <code>Key.Enter</code> , <code>Key.Space</code> , <code>Key.Insert</code> , <code>Key.Left</code> , <code>Key.Top</code> , <code>Key.Right</code> , <code>Key.Down</code> (touches de direction), <code>Key.Add</code> , <code>Key.Multiply</code> , <code>Key.Divide</code> , <code>Key.NumPad0</code> à <code>Key.NumPad9</code> (pavé numérique). Les touches de fonction (F0 à F12) sont traitées par le navigateur avant que l'application Silverlight n'ait l'occasion de le faire.
PlatformKeyCode	Si <code>Key</code> a pris la valeur <code>Key.Unknown</code> , cela signifie que la touche est propre à un système d'exploitation particulier. <code>PlatformKeyCode</code> contient alors un code propre à ce système.
Source	Élément qui est à la base de l'événement. Il s'agit ici du composant qui a le focus au moment de la frappe.

Souvent, il faut déterminer si une action (à la souris, par exemple) a été entreprise avec la touche `Ctrl`, ou `Maj` (*shift* en anglais), enfoncée. Pour cela, il suffit d'écrire le code suivant (ici, pour la touche `Ctrl`) :

```
if ((Keyboard.Modifiers & ModifierKeys.Control) != 0) // touche Ctrl enfoncée
```

où `&` désigne l'opérateur « ET au niveau binaire ». Chacun des bits dans `Keyboard.Modifiers` va faire l'objet d'un ET binaire avec le bit correspondant dans `ModifierKeys.Control`. Si l'un au moins des bits du résultat vaut 1, cela signifie que la touche `Ctrl` est enfoncée.

En VB, cette instruction s'écrit :

```
If (Keyboard.Modifiers And ModifierKeys.Control) <> 0 Then
```

Les autres valeurs de l'énumération `ModifierKeys` sont `Alt`, `Apple`, `Control`, `None`, `Shift` et `Windows`.

Pour qu'une frappe sur la touche Entrée ait le même effet qu'un clic sur un bouton, l'événement `KeyUp` est traité et le code suivant est ajouté dans la fonction qui traite l'événement `KeyUp` :

```
if (e.Key==Key.Enter) Trt();
```

La fonction de traitement du clic sur le bouton n'est pas directement appelée. Son code est déporté dans une nouvelle fonction `Trt`. Ainsi, la fonction de traitement du clic mais aussi celle du `KeyUp` appellent maintenant cette fonction `Trt`.

Le signal d'horloge (timer)

Même si la technique des animations (voir chapitre 9), introduite dans WPF et reprise avec quelques restrictions dans Silverlight (par exemple, pas de 3D), constitue une meilleure solution, le signal d'horloge (*timer* en anglais) est encore très utilisé, et ce depuis des lustres, pour réaliser des effets d'animation.

Pour créer une horloge (ce qui n'a ici aucune représentation visuelle) signalant l'événement `Tick` à intervalles réguliers, il faut :

- déclarer une variable (appelons-la `timer`) de type `DispatcherTimer` (cette classe fait partie de l'espace de noms `System.Windows.Threading`) dans la classe de la page Silverlight, en dehors des fonctions (pour que cette variable `timer` soit accessible à partir des différentes fonctions) ;
- instancier cet objet, par exemple dans la fonction traitant l'événement `Loaded` adressé au conteneur ;
- spécifier un intervalle de temps (par exemple, un dixième de seconde) dans la propriété `Interval`, qui est de type `TimeSpan` ;
- associer à ce timer une fonction de traitement de l'événement `Tick`, laquelle étant automatiquement générée par Visual Studio lorsque vous saisissez `timer.Tick +=` (il vous suffit alors d'appuyer sur la touche `Tab` pour confirmer la génération de la fonction) ;
- démarrer le timer avec `Start` (et `Stop` pour l'arrêter).

Voyons comment spécifier l'intervalle de temps. Le constructeur de la classe `TimeSpan` peut accepter trois, quatre ou cinq arguments (soyez attentif au fait que le constructeur à quatre arguments fait intervenir les jours et non les millisecondes) :

- `TimeSpan(heures, minutes, secondes) ;`
- `TimeSpan(jours, heures, minutes, secondes) ;`
- `TimeSpan(jours, heures, minutes, secondes, millisecondes).`

L'exemple de code XAML suivant permet d'afficher l'heure (celle de la machine de l'utilisateur) dans la zone d'affichage dont le nom interne (attribut `x:Name`) est `za` :

```
<UserControl x:Class="SLProg.Page"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="800" Height="600">
    <Grid x:Name="LayoutRoot" Background="Beige" Loaded="LayoutRoot_Loaded">
        <TextBlock x:Name="za" />
    </Grid>
</UserControl>
```

Ce qui donne en C# :

```
namespace SLProg
{
    public partial class Page : UserControl
    {
        public Page()
        {
            InitializeComponent();
        }
        System.Windows.Threading.DispatcherTimer timer;
        private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
        {
            timer = new System.Windows.Threading.DispatcherTimer();
            timer.Interval = new TimeSpan(0, 0, 1); // toutes les secondes
            timer.Tick += new EventHandler(timer_Tick);
            timer.Start(); // démarre le timer
        }
        void timer_Tick(object sender, EventArgs e)
        {
            za.Text = DateTime.Now.ToLongTimeString();
        }
    }
}
```

Et en VB :

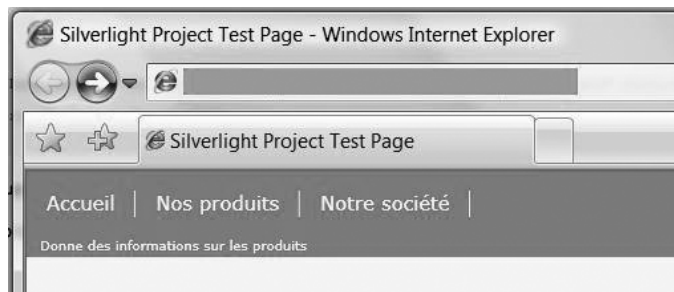
```
Partial Public Class Page
    Inherits UserControl
    Public Sub New()
        InitializeComponent()
    End Sub
    Private timer As System.Windows.Threading.DispatcherTimer
    Private Sub LayoutRoot_Loaded(ByVal sender As System.Object, _
        ByVal e As System.Windows.RoutedEventArgs)
        timer = New System.Windows.Threading.DispatcherTimer()
        timer.Interval = New TimeSpan(0, 0, 1)
        AddHandler timer.Tick, AddressOf timer_Tick
        timer.Start()
    End Sub
    Private Sub timer_Tick(ByVal sender As Object, _
        ByVal e As EventArgs)
        za.Text = DateTime.Now.ToLongTimeString()
    End Sub
End Class
```

Au chapitre 9, consacré aux transformations et aux animations, nous améliorerons ce programme en créant une véritable horloge, avec aiguilles en rotation.

Exemple de traitement d'événement

Pour illustrer le traitement d'un événement, nous allons réaliser un menu simple mais utile (figure 6-1). Une information complémentaire sur l'article de menu (une sorte de *tooltip*) sera affichée dès que la souris survolera l'article et le curseur de la souris prendra alors la forme d'une main. Un article sélectionné sera affiché en rouge (le reste de la page Silverlight, non représenté ici, sera alors en rapport avec cet article).

Figure 6-1



Le menu est réalisé à l'aide d'un panneau vertical occupant toute la largeur de la page et accolé au bord supérieur de la fenêtre. Ce panneau vertical contient deux articles : la ligne de menu (en fait un panneau horizontal) et la ligne d'aide. Les articles du menu, ainsi que les barres de séparation verticales, sont inclus dans un panneau à orientation horizontal. Chaque article du menu est un `TextBlock`. Pour chaque article, les événements `MouseEnter` (pour afficher la ligne d'aide), `MouseLeave` (pour l'effacer) et `MouseLeftButtonUp` (pour la sélection d'articles) sont traités. Les fonctions de traitement sont communes à tous les articles.

Dans le code XAML suivant, les articles sont codés « en dur ». Dans la pratique, il faudrait en faire un « contrôle utilisateur » (voir chapitre 14) avec les données qui proviennent d'un fichier XML (voir chapitre 12).

Le `StackPanel` repris ci-dessous doit être inséré dans le conteneur de la page Silverlight.

```
<StackPanel HorizontalAlignment="Stretch" VerticalAlignment="Top"
    Background="DodgerBlue" >
    <StackPanel Orientation="Horizontal" Height="50" Background="DodgerBlue"
        HorizontalAlignment="Stretch" VerticalAlignment="Top" >
        <TextBlock x:Name="mnuAccueil" Text="Accueil"
            FontFamily="Verdana" Foreground="White" Margin="15"
            MouseEnter="mnu_MouseEnter" MouseLeave="mnu_MouseLeave"
            MouseLeftButtonUp="mnu_MouseLeftButtonUp" />
        <Line X1="0" Y1="15" X2="0" Y2="35" Stroke="White" />
        <TextBlock x:Name="mnuProduits" Text="Nos produits" FontFamily="Verdana"
            Foreground="White" Margin="15"
            MouseEnter="mnu_MouseEnter" MouseLeave="mnu_MouseLeave"
            MouseLeftButtonUp="mnu_MouseLeftButtonUp" />
    </StackPanel>
</StackPanel>
```

```

<Line X1="0" Y1="15" X2="0" Y2="35" Stroke="White" />
<TextBlock x:Name="mnuSociété" Text="Notre société" FontFamily="Verdana"
    Foreground="White" Margin="15"
    MouseEnter="mnu_MouseEnter" MouseLeave="mnu_MouseLeave"
    MouseLeftButtonUp="mnu_MouseLeftButtonUp"/>
<Line X1="0" Y1="15" X2="0" Y2="35" Stroke="White" />
</StackPanel>
<TextBlock x:Name="mnuExplication" Text="Explication" FontFamily="Verdana"
    FontSize="10" Foreground="White" Margin="10,0,0,3"/>
</StackPanel>

```

En C#, cela donne :

```

private void mnu_MouseEnter(object sender, MouseEventArgs e)
{
    TextBlock x = e.Source as TextBlock;
    switch (x.Name)
    {
        case "mnuAccueil":
            mnuExplication.Text = "Donne des informations générales";
            break;
        case "mnuProduits":
            mnuExplication.Text = "Donne des informations sur les produits";
            break;
        case "mnuSociété":
            mnuExplication.Text = "Donne des informations sur la société";
            break;
    }
    Cursor = Cursors.Hand;
}
private void mnu_MouseLeave(object sender, MouseEventArgs e)
{
    Cursor = null;
    mnuExplication.Text = "";
}
TextBlock mnuSelected; // article actuellement sélectionné
private void mnu_MouseLeftButtonUp(object sender,
                                    MouseButtonEventArgs e)
{
    if (mnuSelected!=null)
        mnuSelected.Foreground = new SolidColorBrush(Colors.White);
    mnuSelected = e.Source as TextBlock;
    mnuSelected.Foreground = new SolidColorBrush(Colors.Red);
    // traiter ici la sélection d'article
}

```

Et en VB :

```

Private Sub mnu_MouseEnter(ByVal sender As Object, _
                           ByVal e As MouseEventArgs)
    Dim x As TextBlock = TryCast(e.Source, TextBlock)
    Select Case x.Name
        Case "mnuAccueil"
            mnuExplication.Text = "Donne des informations générales"

```

```

    Case "mnuProduits"
        mnuExplication.Text = "Donne des informations sur les produits"
    Case "mnuSociété"
        mnuExplication.Text = "Donne des informations sur la société"
    End Select
    Cursor = Cursors.Hand
End Sub
Private Sub mnu_MouseLeave(ByVal sender As Object, _
                        ByVal e As MouseEventArgs)

    Cursor = Nothing
    mnuExplication.Text = ""
End Sub
Private mnuSelected As TextBlock ' article actuellement sélectionné
Private Sub mnu_MouseLeftButtonUp(ByVal sender As Object, _
                        ByVal e As MouseButtonEventArgs)

    If mnuSelected IsNot Nothing Then
        mnuSelected.Foreground = New SolidColorBrush(Colors.White)
    End If
    mnuSelected = TryCast(e.Source, TextBlock)
    mnuSelected.Foreground = New SolidColorBrush(Colors.Red)
    ' traiter ici la sélection d'article
End Sub

```

La création dynamique d'objets

Puisque les balises XAML correspondent à des objets connus du run-time Silverlight, il est possible de créer des éléments UI (mais aussi des transformations et des animations) par programme, à n'importe quel moment en cours d'exécution de l'application Silverlight. Il pourrait s'agir de la phase de chargement (événement `Loaded`), avant le tout premier affichage de la page, mais aussi de tout autre moment.

Pour illustrer la création dynamique d'objets, nous allons créer un rectangle de toutes pièces en cours d'exécution de programme. Pour cela, il convient de déclarer une variable dans la classe de la page Silverlight (classe appelée `Page` par défaut), en dehors des fonctions (la variable sera ainsi accessible à partir des différentes fonctions).

En C#, cette déclaration s'écrit :

```
Rectangle rc;
```

Et en VB :

```
Dim rc as Rectangle
```

Ensuite, le rectangle est créé en mémoire, ce qui serait également le cas pour tout autre objet.

En C#, cette création en mémoire du rectangle s'écrit :

```
rc = new Rectangle();
```

Et en VB :

```
rc = New Rectangle
```

À ce stade, rien ne s’affiche encore, même si l’objet a été créé en mémoire.

Pour initialiser les propriétés du rectangle `rc` (bien que, jusqu’à présent, rien ne soit encore affiché), il convient d’écrire le code suivant (ici, en C#) :

```
rc.Width = 200; rc.Fill = new SolidColorBrush(Colors.Red);
```

La syntaxe est identique en VB mais soyez attentif aux trois petites différences suivantes :

- pas de point-virgule (;) en VB pour terminer une instruction ;
- deux ou plusieurs instructions peuvent être placées en VB sur une même ligne à condition d’être séparées par le deux-points (:)
- En VB, on termine une ligne avec le caractère de soulignement (*underscore*) pour signaler que l’instruction se poursuit à la ligne suivante.

Pour positionner le rectangle dans un canevas, deux techniques peuvent être utilisées.

La première consiste à appeler les fonctions statiques `SetLeft`, `SetTop` et `SetZIndex` de la classe `Canvas` :

```
Canvas.SetLeft(rc, 200);  
Canvas.SetTop(rc, 100);
```

La seconde consiste à appeler la fonction `SetValue` appliquée à l’objet (ici, quand le conteneur est un canevas).

En C#, cela donne :

```
rc.SetValue(Canvas.LeftProperty, 200.0);
```

et en VB :

```
rc.SetValue(Canvas.TopProperty, 100.0)
```

Le second argument de `SetValue` doit être de type `double`, d’où les valeurs `200.0` et `100.0` dans l’exemple ci-dessus. Aucune conversion ne peut être implicitement réalisée car la signature de la fonction indique que cet argument est de type `object`. Si ce second argument est le nom d’une variable, un transtypage (*casting* en anglais) est nécessaire :

```
rc.SetValue(Canvas.LeftProperty, (double)n);
```

Pour placer le rectangle dans une cellule de grille (ici, la cellule en quatrième rangée et cinquième colonne), il convient d’écrire le code suivant :

```
rc.SetValue(Grid.RowProperty, 3);  
rc.SetValue(Grid.ColumnProperty, 4);
```

Pour positionner le rectangle dans une cellule de grille ou un `StackPanel`, il faut imposer un alignement en haut à gauche et spécifier les marges. Par exemple :

```
rc.HorizontalAlignment = HorizontalAlignment.Right;  
rc.VerticalAlignment = VerticalAlignment.Top;  
rc.Margin = new Thickness(10, 20, 0, 0);
```


L'objet `rc` a été créé en mémoire et initialisé, mais il ne fait pas encore partie des objets « enfants » de son conteneur (et de ce fait, il ne peut pas encore être affiché). Il faut donc l'ajouter à la collection des nœuds enfants de son conteneur (ici, `LayoutRoot`, qui est par défaut le nom interne du conteneur le plus externe) :

```
LayoutRoot.Children.Add(rc);
```

Au début de ce chapitre, nous avons vu comment associer dynamiquement une fonction de traitement (existante) à un événement relatif à un élément UI. La technique est la même pour les éléments UI créés dynamiquement.

Comment modifier le contenu d'un conteneur ?

La liste des nœuds enfants d'un conteneur peut être manipulée comme n'importe quelle collection sous le framework .NET (la propriété `Children` d'un conteneur désigne une collection). Le tableau 6-7 présente les méthodes et les propriétés applicables à une collection.

Tableau 6-7 – Les méthodes et les propriétés applicables à une collection

Méthode	Description
<code>Clear()</code>	Supprime tous les nœuds enfants. Par exemple : <code>xyz.Children.Clear()</code> où <code>xyz</code> désigne le nom interne – propriété <code>x.Name</code> – d'un conteneur, par exemple <code>LayoutRoot</code> . Les objets restent néanmoins en mémoire (mais ils sont désormais invisibles) et peuvent encore être utilisés pour être accrochés à un conteneur.
<code>Count</code>	Propriété indiquant le nombre de nœuds enfants (de premier niveau) : <code>n = xyz.Children.Count</code>
<code>Add(obj)</code>	Ajoute un nœud enfant à la fin de la liste.
<code>Insert(n, obj)</code>	Insère un nœud enfant en <i>n</i> ^{ième} position (0 pour la première) : <code>xyz.Children.Insert(1, rc)</code>
<code>Remove(obj)</code>	Supprime un objet déterminé : <code>xyz.Children.Remove(rc)</code>
<code>RemoveAt(n)</code>	Supprime un objet en un emplacement déterminé (0 pour le premier emplacement).

Dans le code C# suivant, le rectangle est inséré dans une cellule de grille (en deuxième rangée et troisième colonne), avec placement fixe dans cette cellule :

```
Rectangle rc; // déclaration dans la classe Page, en dehors des fonctions
.....
rc = new Rectangle();
rc.SetValue(Grid.RowProperty, 1);
rc.SetValue(Grid.ColumnProperty, 2);
rc.HorizontalAlignment = HorizontalAlignment.Left;
rc.VerticalAlignment = VerticalAlignment.Top;
rc.Margin = new Thickness(10, 20, 0, 0);
rc.Width = 200; rc.Height = 150;
```

```
rc.Stroke = new SolidColorBrush(Colors.Red);
rc.StrokeThickness = 3;
LayoutRoot.Children.Add(rc);
```

Dans le code VB suivant, le rectangle est cette fois inséré dans un canevas :

```
Dim rc As Rectangle ' déclaration dans la classe Page, en dehors des fonctions
.....
rc = New Rectangle()
rc.Width = 200 : rc.Height = 150
rc.Stroke=New SolidColorBrush(Colors.Red) : rc.StrokeThickness = 3
rc.SetValue(Canvas.LeftProperty, 10)
rc.SetValue(Canvas.TopProperty, 20)
LayoutRoot.Children.Add(rc)
```

Dans ce cas, le rectangle est transparent (et donc limité à un cadre) car la propriété `Fill` n'a pas été initialisée.

La création dynamique à partir du XAML

La fonction `Load` de la classe `XamlReader` permet de créer dynamiquement des éléments UI à partir d'une chaîne de caractères contenant le XAML de cet élément UI.

Le XAML proviendra ici d'une chaîne de caractères enregistrée en mémoire. Il pourrait également provenir d'une ressource ou être téléchargé depuis un serveur.

La classe `XamlReader` fait partie de l'espace de noms `System.Windows.Markup`. Il ne faut donc pas oublier le `using` (en C#) ou l'`Imports` (en VB) correspondant en tête du fichier `Page.xaml.cs` ou `Page.xaml.vb`.

Ensuite, il suffit :

- de spécifier le XAML dans une chaîne de caractères, sans oublier d'ajouter une clause `xmlns='http://schemas.microsoft.com/client/2007'` ;
- de créer l'élément UI en mémoire avec `XamlReader.Load`, qui renvoie une référence à l'élément UI mais sous forme d'un objet (car le véritable type dépend du contenu de la chaîne XAML, qui n'est pas nécessairement connu au moment de la compilation) ou `null` (`Nothing` en VB) en cas d'erreur ;
- d'ajouter l'objet ainsi créé à la liste des nœuds enfants d'un conteneur.

Par exemple, pour créer un rectangle rouge dans un conteneur (ici, un canevas), à partir du point (X, Y) et avec les dimensions (W, H), le code C# sera :

```
using System.Windows.Markup;
.....
int X=50, Y=150, W=100, H=75;
string s = "<Rectangle xmlns='http://schemas.microsoft.com/client/2007' "
          + "Canvas.Left='" + X + "' Canvas.Top='" + Y + "' "
          + "Width='" + W + "' Height='" + H + "' Fill='Red' />";
Rectangle rc = (Rectangle)XamlReader.Load(s);
LayoutRoot.Children.Add(rc);
```

et le code VB :

```
Imports System.Windows.Markup ' avant la déclaration de la classe
.....
Dim s As String
Dim X = 20, Y = 200, H = 200, W = 150
s = "<Rectangle xmlns='http://schemas.microsoft.com/client/2007' " _
    & "Canvas.Left=' " & X & "' Canvas.Top=' " & Y & "' Width=' " _
    & W & "' Height=' " & H & "' Fill='Red' />"
Dim rc As Rectangle
rc = XamlReader.Load(s)
LayoutRoot.Children.Add(rc)
```

Les apostrophes doubles (") ont été remplacées par des apostrophes simples (') dans le XAML afin de ne pas entrer en conflit avec la syntaxe du C# et du VB qui considèrent les " comme des caractères de début et de fin de chaîne. Si le XAML provient d'une source extérieure comme un service Web (voir chapitre 13), il n'y a cependant pas de problème.

Il est impossible de spécifier un attribut `x:Name` dans la balise XAML contenue dans `s`. Pour donner un nom interne (ce que nous avons fait jusqu'à présent avec l'attribut `x:Name`) à l'objet ainsi créé et pouvoir le manipuler en cours d'exécution de programme, il faut spécifier un attribut `Name` et récupérer une référence à l'élément UI créé dynamiquement en utilisant la fonction `FindName`. Une autre solution consiste à retenir la référence qui avait été renvoyée par `XamlReader.Load`. Par exemple :

```
<Rectangle Name="rc" ..... />
```

On retrouve en C# une référence au rectangle mentionné dans `s` (chaîne dans laquelle on a ajouté `Name="rc"`) en écrivant :

```
Rectangle r = (Rectangle)FindName("rc");
r.Width *= 2; // on double sa largeur
```

Ce qui s'écrit en VB :

```
Dim r As Rectangle
r = FindName("rc")
r.Width *= 2
```

Programmes d'accompagnement

Vous trouverez les sources des programmes Silverlight qui suivent sur le site d'accompagnement de l'ouvrage.

Exemple 1 :

Pour créer un rectangle semi-transparent, cliquez sur un coin du rectangle et déplacez la souris à l'endroit souhaité pour le coin opposé, tout en maintenant le bouton de la souris enfoncé. La construction du rectangle suit donc le déplacement de la souris (figure 6-2).

Figure 6-2

**Exemple 2 :**

Tracé d'une courbe à l'aide de la souris. Lors du relâchement du bouton, une balle suit la courbe qui vient d'être dessinée (figure 6-3).

Figure 6-3

**Exemple 3 :**

Tracé d'une courbe à l'aide de la souris, courbe suivie ensuite par un Pacman qui mange la ligne en suivant sa courbure (figure 6-4).

Figure 6-4



Les images, les curseurs et les vidéos

Les images


Silverlight peut évidemment afficher des images, à condition qu'elles soient au format .jpg ou .png (format qui permet l'affichage d'images présentant des zones transparentes).

Intéressons-nous d'abord à la source de l'image, c'est-à-dire à son emplacement.

Les images en ressources

Plusieurs techniques sont possibles pour insérer une image dans une page Silverlight. La première consiste à insérer l'image en ressource de l'application. Pour cela, ouvrez l'Explorateur de solutions, effectuez un clic droit sur le nom du projet, sélectionnez Ajouter>Élément existant... et localisez l'image souhaitée sur votre ordinateur. Visual Studio copie alors l'image dans le répertoire du projet. L'image sera ainsi greffée en ressource de l'application (effectuez un clic droit sur le nom de l'image dans le projet puis sélectionnez Propriétés, vous constatez alors que l'attribut Action de génération a pour valeur Resource).

La forme la plus élémentaire de la balise Image s'écrit :

```
 <Image Source="MonaLisa.jpg" ..... />
```

où doit être remplacé par des attributs tels que Grid.Row ou Canvas.Left en fonction du conteneur choisi.

Pour mieux organiser le projet de l'application Silverlight, les images sont généralement regroupées dans des sous-répertoires du projet. Pour créer un sous-répertoire contenant des données d'un type particulier (par exemple, les images de l'application), effectuez un clic droit sur le projet (partie Silverlight) et sélectionnez Ajouter>Nouveau dossier. Insérez alors l'image en ressource en partant de ce sous-répertoire, comme expliqué précédemment.

Après compilation, l'image (ici, `MonaLisa.jpg`) est greffée dans la DLL (qu'on appelle *assembly* dans le jargon .NET) contenant le code C# ou VB compilé, DLL qui est elle-même greffée dans le fichier XAP (dans le sous-répertoire `ClientBin` de la partie Web) qui est envoyé au navigateur avec le fichier HTML.

Notez aussi cette subtile différence car cela aura bientôt une répercussion : avec la valeur `Resource` dans l'attribut `Action` de génération des propriétés de l'image (propriété dans l'Explorateur de solutions), cette dernière n'est pas directement greffée au fichier XAP mais bien à la DLL, elle-même greffée au fichier XAP.

L'image est néanmoins directement greffée au fichier XAP, mais en dehors de la DLL contenant le code compilé de l'application Silverlight si l'attribut `Action` de génération de l'image est `Content` (pour visualiser les propriétés de l'image, effectuez un clic droit sur le nom de celle-ci et sélectionnez `Propriétés`). Cela ne change rien pour le déploiement de notre application Web ou son utilisateur, mais nous verrons bientôt que cela peut avoir une influence pour le programmeur.

Les images qui ne sont pas en ressources

Une image ne doit pas nécessairement être insérée en ressource de l'application Silverlight mais dans ce cas, elle doit être copiée dans le répertoire `ClientBin` qui doit également être créé sur le serveur. L'image n'est pas greffée au fichier XAP mais elle est chargée par le navigateur (indépendamment du fichier XAP) lors de la phase de préparation de la page.

Sur le serveur (chez votre hébergeur si vous ne disposez pas de l'infrastructure nécessaire), les applications Web sont généralement sous contrôle d'IIS de Microsoft ou d'Apache si le serveur est sous Linux. S'il s'agit d'Apache, soyez attentif à respecter la casse : les lettres minuscules et majuscules sont en effet considérées comme différentes.

Que se passe-t-il si un même nom d'image (par exemple, `MonaLisa.jpg`) se trouve à la fois en ressource et dans le répertoire `ClientBin` ? Si Silverlight trouve l'image en ressource (directement dans le fichier XAP ou dans la DLL incorporée à celui-ci), il prend cette image. Sinon, il poursuit sa recherche dans le répertoire `ClientBin`.

Néanmoins, si la balise est (notez le / en tête du nom) :

```
<Image Source="/MonaLisa.jpg" ..... />
```

le préfixe `/` signifie : « le fichier est d'abord recherché dans le fichier XAP mais sans le rechercher dans les DLL incluses dans ce fichier XAP et la recherche se poursuit, si nécessaire, dans le répertoire `ClientBin` ». La valeur de la propriété `Action` de génération a donc bien une influence.

L'association entre le composant Image et le nom de l'image peut être effectuée en cours d'exécution de programme en écrivant :

```
<Image x:Name="img" ..... />
.....
using System.Windows.Media.Imaging;
.....
img.Source = new BitmapImage(new Uri("MonaLisa.jpg", UriKind.Relative));
```

Pour utiliser la classe `BitmapImage`, il faut mentionner l'espace de noms `System.Windows.Media.Imaging` dans le programme (directive `using` en C# et `Imports` en VB, en tête du fichier `.cs` ou `.vb` du programme). Ici aussi, l'image mentionnée en premier argument du constructeur d'`Uri` doit se trouver dans le répertoire du fichier XAP (ou un sous-répertoire de celui-ci, à condition évidemment de le mentionner dans le nom du fichier).

Au chapitre 13, nous verrons comment accéder, via l'objet `WebClient`, à une image stockée sur un serveur, y compris un serveur d'images comme Flickr.

Taille des images et respect des proportions

Si les propriétés `Width` et/ou `Height` ne sont pas spécifiées pour une image, celle-ci est affichée dans sa taille d'origine. Si `Width` et/ou `Height` sont spécifiées, la propriété `Stretch` permet d'indiquer comment l'image doit être affichée dans sa surface d'affichage. En l'absence de clause `Stretch`, l'image est affichée sans déformation mais sans respecter `Width` et `Height` (par exemple, si `Width` vaut 300 et `Height` 10, une image carrée sera affichée dans un rectangle de 10 × 10 pixels).

Pour illustrer l'utilisation de la propriété `Stretch`, nous allons afficher l'image de la figure 7-1 dont la taille est de 91 × 142 pixels. Pour une bonne compréhension des différentes valeurs de l'attribut `Stretch`, nous avons modifié les attributs `Width` et `Height` (tableau 7-1) de la balise `Image` pour les images des figures 7-2 à 7-5, ceci afin de mettre en évidence les problèmes rencontrés.

Figure 7-1



Tableau 7-1 – Les différentes valeurs de l’attribut Stretch de la balise Image

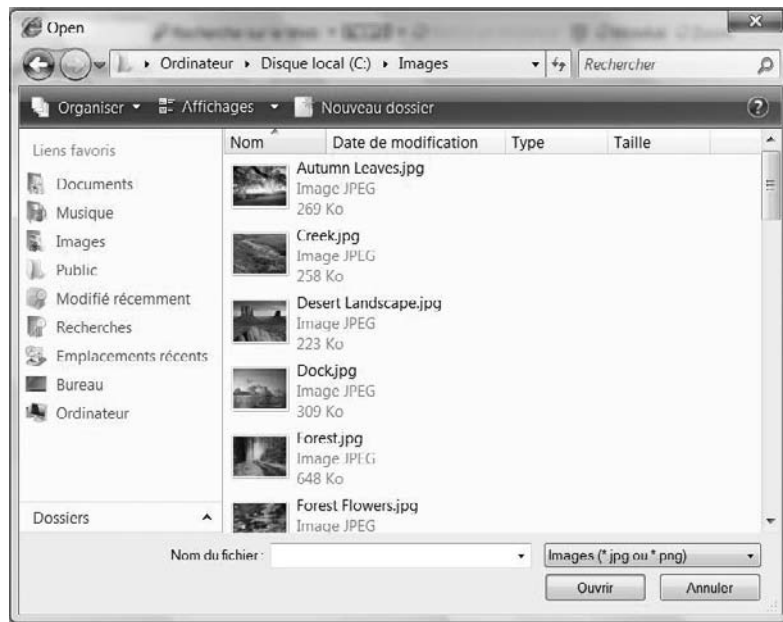
Nom de l'attribut	Résultat visuel	Description
None	Figure 7-2 	L'image est affichée à sa taille réelle et n'est donc pas déformée. Il se peut qu'une partie seulement de l'image (coin supérieur gauche) soit affichée, ou qu'une partie seulement de la zone dévolue à l'image soit occupée (quand la surface d'affichage est plus grande que l'image).
Fill	Figure 7-3 	L'image est étendue (<i>stretched</i> en anglais) pour être entièrement affichée dans le rectangle (Width, Height). De ce fait, elle est généralement déformée.
Uniform	Figure 7-4 	L'image n'est pas déformée et est affichée dans son intégralité (ici, dans un rectangle de 100 x 200 pixels). La plus grande image possible est affichée (ici, le rectangle est affiché dans le seul but de montrer l'effet). Des bords peuvent donc apparaître à gauche, à droite, au-dessus ou au-dessous (ici, au-dessus et au-dessous).
UniformToFill	Figure 7-5 	L'image (la plus grande possible) est affichée sans déformation mais pas nécessairement dans son intégralité. L'image est donc généralement coupée au-dessus et au-dessous (ce qui est le cas ici) ou à gauche et à droite.

Les événements liés à la souris (MouseMove, etc.) peuvent être traités pour une image, comme pour n'importe quel composant. L'événement ImageFailed est signalé si l'image ne peut être affichée (image non trouvée ou problème de format).

Lire une image à partir du système de fichiers local

Une application Silverlight peut lire des images qui se trouvent sur la machine de l'utilisateur mais sous contrôle de celui-ci (c'est lui, impérativement, qui mène à ce fichier et l'application Silverlight ignore tout du chemin qui mène à celui-ci). Elle peut en effet ouvrir une boîte de dialogue et permettre à l'utilisateur de naviguer dans son système de fichiers local pour sélectionner une ou plusieurs de ses images (figure 7-6). Cette boîte de dialogue est un objet OpenFileDialog.

Figure 7-6



Dans le filtre (propriété `Filter` de l'objet `OpenFileDialog`), deux informations séparées par une barre verticale doivent être spécifiées :

- le libellé qui sera affiché tel quel, ici `Images (*.jpg ou *.png)` ;
- les filtres eux-mêmes (ici, ne seront retenus que les fichiers ayant pour extensions `.jpg` et `.png`, quel que soit le nom du fichier image).

La boîte de dialogue est alors affichée par `ofd.ShowDialog()` :

Si l'utilisateur sélectionne une image, la fonction `ofd.ShowDialog()` se termine et renvoie la valeur `DialogResult.OK`. Le programme ouvre alors le fichier de l'image afin de lire son contenu et crée un objet `BitmapImage` à partir de celui-ci (qui est lu dans le flux `stream`).

Voici le code correspondant en C# :

```
using System.IO;
using System.Windows.Media.Imaging;
.....
OpenFileDialog ofd = new OpenFileDialog();
ofd.Filter = "Images (*.jpg ou *.png)|*.jpg;*.png";
if (ofd.ShowDialog() == DialogResult.OK)
{
    Stream stream = ofd.SelectedFile.OpenRead();
    BitmapImage bi = new BitmapImage();
    bi.SetSource(stream);
    img.Source = bi;
    stream.Close();
}
```

et le même code en VB :

```
Imports System.Windows.Media.Imaging
Imports System.Windows.Resources
Imports System.IO
.....
Dim ofd As OpenFileDialog
ofd = New OpenFileDialog()
ofd.Filter = "Images (*.jpg ou *.png)|*.jpg;*.png"
If ofd.ShowDialog() = DialogResult.OK Then
    Dim stream As Stream = ofd.SelectedFile.OpenRead()
    Dim bi As BitmapImage = New BitmapImage()
    bi.SetSource(stream)
    img.Source = bi
    stream.Close()
End If
```

Le clipping d'image

Une image (mais aussi une vidéo) peut être découpée selon les contours de n'importe quelle forme (par exemple, une ellipse mais il pourrait s'agir d'une forme bien plus complexe, voir la section « Le Path » du chapitre 8). La zone de découpe est définie par une « géométrie » telle que `EllipseGeometry`.

Expliquons tout d'abord ce qu'est une géométrie. Alors qu'un objet `Ellipse` a une représentation visuelle (une ellipse dessinée dans son conteneur), `EllipseGeometry` consiste en la définition, en mémoire, d'une ellipse. Si un objet `Ellipse` peut être enfant d'un conteneur, un objet `EllipseGeometry` ne peut pas l'être : il doit être enfant (autrement dit, défini sous une balise) d'une propriété telle que `Clip` (ce qui nous intéresse ici) ou `Data` d'un objet `Path` (voir la section « Le Path » du chapitre 8).

Silverlight définit quatre géométries différentes (tableau 7-2) ainsi que `GeometryGroup` qui permet de créer des géométries complexes par combinaison de géométries de base.

Tableau 7-2 – Les différentes géométries définies dans Silverlight

Nom de la géométrie	Description
<code>EllipseGeometry</code>	Une ellipse définit la géométrie, avec <code>Center</code> , <code>RadiusX</code> et <code>RadiusY</code> comme propriétés. Les coordonnées sont relatives au coin supérieur gauche de l'image (voir exemple ci-dessous).
<code>LineGeometry</code>	Une ligne définit la géométrie par un point de départ (propriété <code>StartPoint</code>) et un point d'arrivée (<code>EndPoint</code>) : <code><LineGeometry StartPoint="10,10" EndPoint="100,80" /></code>
<code>PathGeometry</code>	Un <code>Path</code> définit la géométrie, ce qui permet de créer des régions de coupe de n'importe quelle forme, souvent même très complexe.
<code>RectangleGeometry</code>	Un rectangle définit la géométrie.

Pour illustrer le fonctionnement des géométries, nous allons créer une zone de *clipping* en forme d'ellipse pour l'image `MonaLisa.jpg` (figure 7-7). Voici le code correspondant :

```
<Image Width="182" Height="284" Source="MonaLisa.jpg" ..... >  
  <Image.Clip>  
    <EllipseGeometry Center="91,142"  
      RadiusX="91" RadiusY="142" />  
  </Image.Clip>  
</Image>
```

Figure 7-7



Pour réaliser une découpe avec Expression Blend, créez tout d'abord une ellipse et superposez-la à l'image (au besoin, modifiez temporairement la transparence de l'ellipse pour que l'image de fond soit encore partiellement visible, ce qui vous aidera dans le positionnement et la taille de l'ellipse de découpe). Sélectionnez ensuite l'image et l'ellipse au moyen de l'outil de sélection et cliquez sur chacun de ces éléments tout en maintenant enfoncée la touche Ctrl. Enfin, procédez à la découpe de l'image via le menu `Object>Path>Make Clipping Path`.

Les images comme motifs de pinceau

Une image peut être utilisée comme pinceau (ici, pour le fond d'un rectangle) :

```
<Rectangle Width="300" Height="200" Stroke="Black" >  
  <Rectangle.Fill>  
    <ImageBrush ImageSource="SilverlightLogo.jpg" />  
  </Rectangle.Fill>  
</Rectangle>
```

Dans la mesure où une balise `ImageBrush` n'a pas (contrairement à `LinearGradientBrush` et `RadialGradientBrush`) d'attribut `SpreadMethod` (avec sa valeur `Repeat`), l'image n'est jamais répétée mais est étendue (par *stretching*) à toute la surface du rectangle. Cela permet aussi de charger une image en fond de conteneur (utilisez dans ce cas la propriété `Background`). L'attribut `Stretch` peut être spécifié dans la balise `ImageBrush`.

Les curseurs

Il est très simple de modifier le curseur de la souris pour tout le conteneur ou pour un élément UI particulier (quand la souris le survole). En effet, il suffit d'ajouter ou de changer la valeur de l'attribut `Cursor` dans la balise de cet élément. Les figures 7-8 à 7-13 présentent le rendu visuel pour les différentes valeurs (qui proviennent de l'énumération `Cursors`) de l'attribut `Cursor`.

Figure 7-8

Arrow



Figure 7-9

Eraser

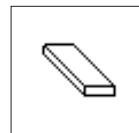


Figure 7-10

Hand



Figure 7-11

IBeam

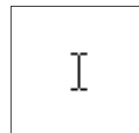


Figure 7-12

Stylus

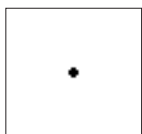
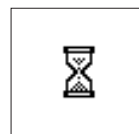


Figure 7-13

Wait



À ces six valeurs, il convient d'ajouter les valeurs `Default` (curseur par défaut) et `None` (aucun curseur, ce qui s'avère utile lorsque l'on crée son propre curseur à partir d'une forme géométrique ou d'une image avec transparence).

Pour changer de curseur par programme, il suffit d'initialiser la propriété `Cursor` de l'élément UI avec la valeur `Cursors.xyz`, où `xyz` doit être remplacé par l'une des huit valeurs mentionnées précédemment. Par exemple, si `rc` est le nom interne (propriété `x:Name`) d'un élément UI, on aura :

```
rc.Cursor = Cursors.Eraser;
```

Le curseur prendra alors la forme d'une gomme quand la souris survolera le rectangle `rc`.

Les sons et les films

Silverlight fournit les codecs nécessaires pour lire de la vidéo haute performance et haute qualité en mode streaming ou en mode de chargement progressif (*progressive download* en anglais), ce qui implique le remplissage d'une mémoire tampon avant de commencer à jouer la vidéo. On s'accommode ainsi des variations de débit dans le transfert des données, sans perturbation pour l'utilisateur (sauf lorsque le tampon devient vide par manque d'approvisionnement en données).

Silverlight n'accepte que le format .wmv pour la vidéo et les formats .wma et .mp3 pour l'audio. Si vos vidéos sont aux formats .avi, .mpg ou .flv, vous devrez au préalable les convertir. Microsoft Expression Encoder accepte les formats .avi, .mpg, .asf, .vob, .avs, .mov, .m4v, .mp4, .3gp, .3g2 et .dv pour la vidéo ainsi que les formats .wav, .aiff, .bwf, .m4a et .m4b pour l'audio, et peut les convertir en .wmv pour la vidéo et .mp3 ou .wma pour l'audio. On trouve aussi sur Internet de nombreux convertisseurs en *shareware*.

La balise XAML pour la diffusion de musique et de vidéo est `MediaElement`. La plupart de ses propriétés sont maintenant déjà bien connues (voir la section « Taille des images et respect des proportions » de ce chapitre pour la propriété `Stretch`). Si vous ne spécifiez pas d'attribut `Width` et `Height`, la vidéo est jouée à sa taille d'origine. Le tableau 7-3 présente les différentes propriétés de cette balise.

Tableau 7-3 – Les propriétés de la balise `MediaElement`

Nom de la propriété	Description
<code>AutoPlay</code>	Si <code>AutoPlay</code> vaut <code>true</code> , le média est joué dès le chargement. Avec <code>false</code> , vous devez exécuter <code>Play</code> pour démarrer la lecture. Un média n'est jamais joué qu'une seule fois ! Il faut donc détecter la fin (événement <code>MediaEnded</code>) et relancer la vidéo par programme.
<code>BufferingTime</code>	Temps de mise en cache (<i>buffering</i>). Par défaut, ce temps est de 5 secondes. <code>BufferingTime</code> , qui est de type <code>TimeSpan</code> , est spécifié dans le format "hh:mm:ss.mmm".
<code>IsMuted</code>	Si <code>IsMuted</code> vaut <code>true</code> , le média est joué en silence.
<code>Position</code>	Position courante dans le média (il s'agit d'un objet <code>TimeSpan</code>).
<code>Source</code>	Nom du fichier.
<code>Volume</code>	Une valeur entre 0 et 1 indiquant le volume sonore.

Par programme, vous pouvez exécuter les fonctions suivantes, dont les noms parlent d'eux-mêmes : `Play`, `Pause` et `Stop`.

Quand la fin d'un média est atteinte, l'événement `MediaEnded` est signalé. Pour rejouer le média, vous devez traiter cet événement, remettre `Position` à zéro et jouer à nouveau le média. `Stop` a le même effet :

```
<MediaElement x:Name="video" MediaEnded="video_MediaEnded" ..... />
.....
void video_MediaEnded(object sender, RoutedEventArgs e)
{
    video.Position = new TimeSpan(0);
    video.Play();
}
```

La propriété `CurrentState` indique l'état de la vidéo, soit l'une des valeurs de l'énumération `MediaElementState` dont les différentes valeurs sont `AcquiringLicence`, `Buffering`, `Closed`, `Opening`, `Paused`, `Playing` et `Stopped`.

Comme pour les images, copiez les fichiers `.wmv` et `.mp3` dans le répertoire `ClientBin` de la partie `Web`.

Une vidéo peut être utilisée comme pinceau `VideoBrush`. Par exemple :

```
<MediaElement x:Name="vid" Source="iINTERACT.wmv" Opacity="0" />
.....
<TextBlock Text="iINTERACT" FontFamily="Verdana" FontSize="50" >
  <TextBlock.Foreground>
    <VideoBrush SourceName="vid" />
  </TextBlock.Foreground>
</TextBlock>
```

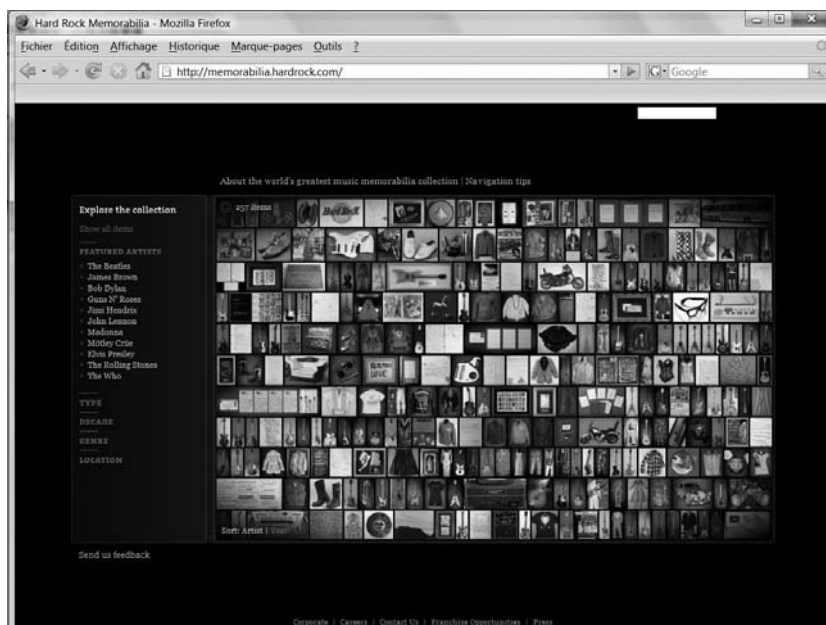
Lorsque la fonction `vid.Play()` est exécutée, la vidéo est jouée dans les lettres.

Deep Zoom

La technologie `Deep Zoom` permet de zoomer de plus en plus profondément dans une image et cela sans devoir télécharger une image de taille déraisonnable pour le Web et sans provoquer de délai d'attente chez l'utilisateur.

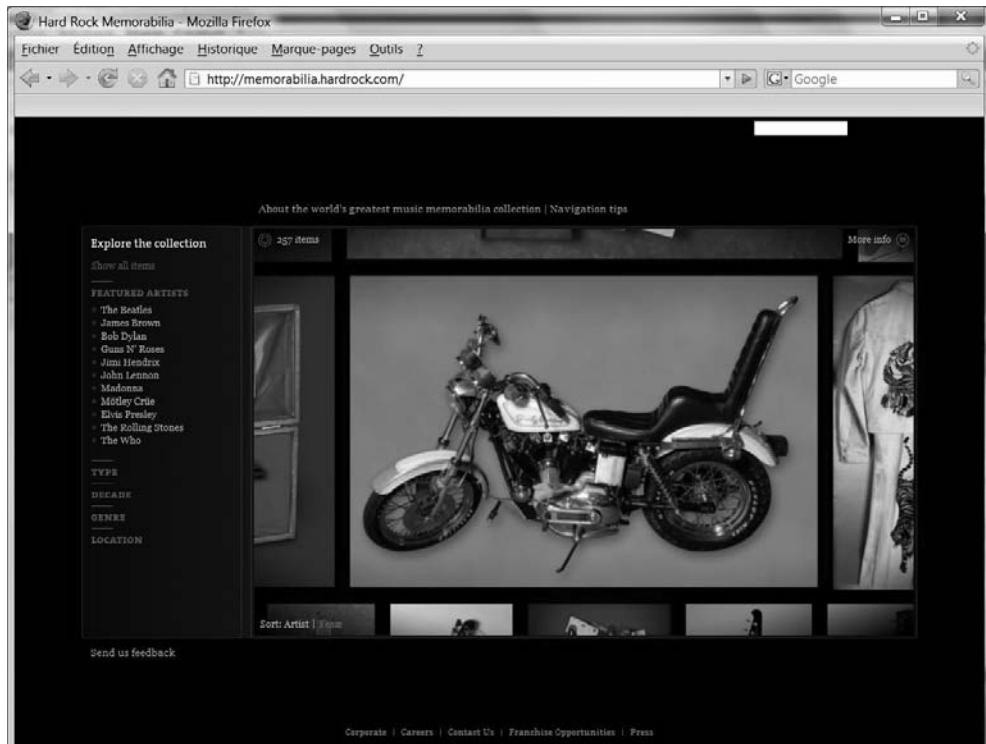
Pour expérimenter `Deep Zoom` et vous faire une idée de ses prodigieuses possibilités (par exemple, une visite virtuelle ou les caractéristiques détaillées d'un produit), rendez-vous par exemple sur le site <http://memorabilia.hardrock.com>, consacré aux légendes du rock (figure 7-14).

Figure 7-14



Amusez-vous ensuite (par exemple) à déplacer le curseur de la souris au-dessus de la miniature représentant la Harley-Davidson d'Elvis Presley et zoomez progressivement (grâce à la molette de la souris ou en cliquant sur l'image), sur des détails de plus en plus précis (figures 7-15 et 7-16).

Figure 7-15



S'il s'agissait d'une seule image (de taille gigantesque), il faudrait des heures, même avec une connexion haut débit, avant qu'elle ne s'affiche dans votre navigateur.

Pour développer une application Silverlight faisant appel à la technologie Deep Zoom, vous devez installer Deep Zoom Composer, téléchargeable depuis le site <http://www.microsoft.com/download> (effectuez une recherche sur « Deep Zoom »). Deep Zoom Composer est également proposé sur la page d'installation des outils de développement Silverlight. Téléchargez le fichier et installez Deep Zoom Composer sur votre ordinateur. Ce logiciel ne concerne que les développeurs, les utilisateurs n'ayant, eux, rien à installer.

Lancez Deep Zoom Composer (figure 7-17).

Figure 7-16

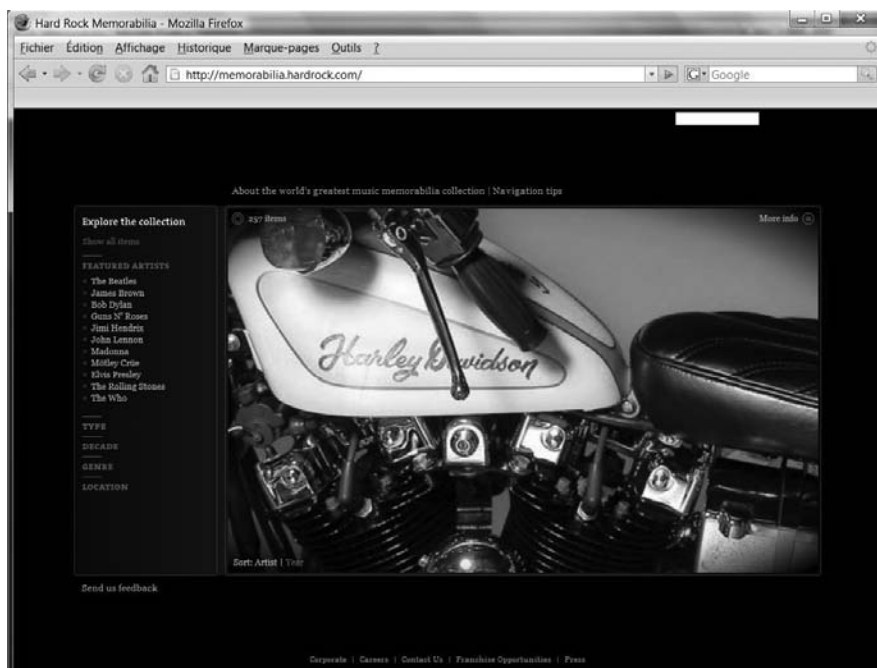
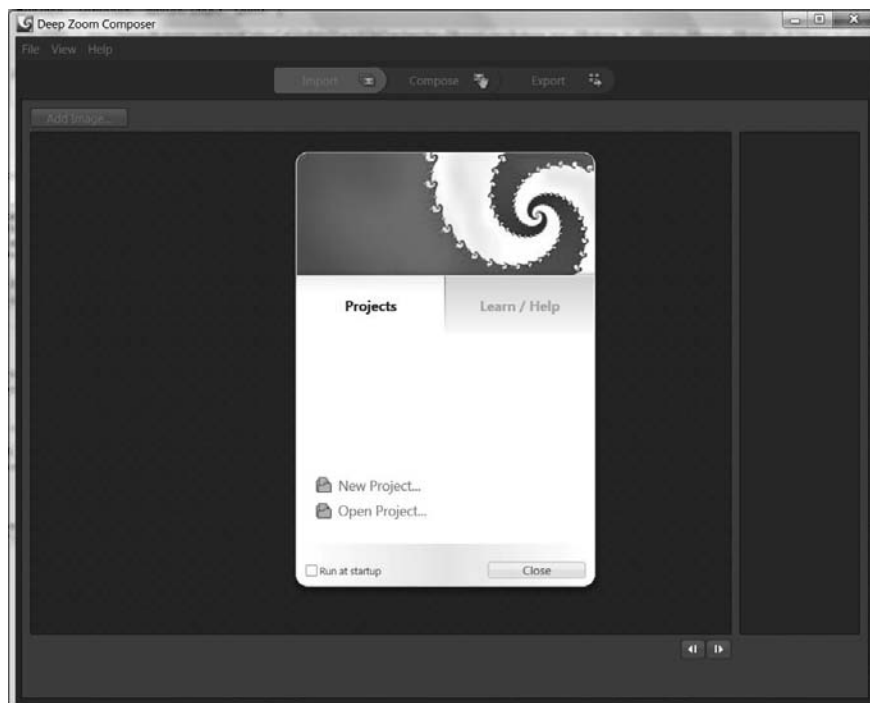


Figure 7-17



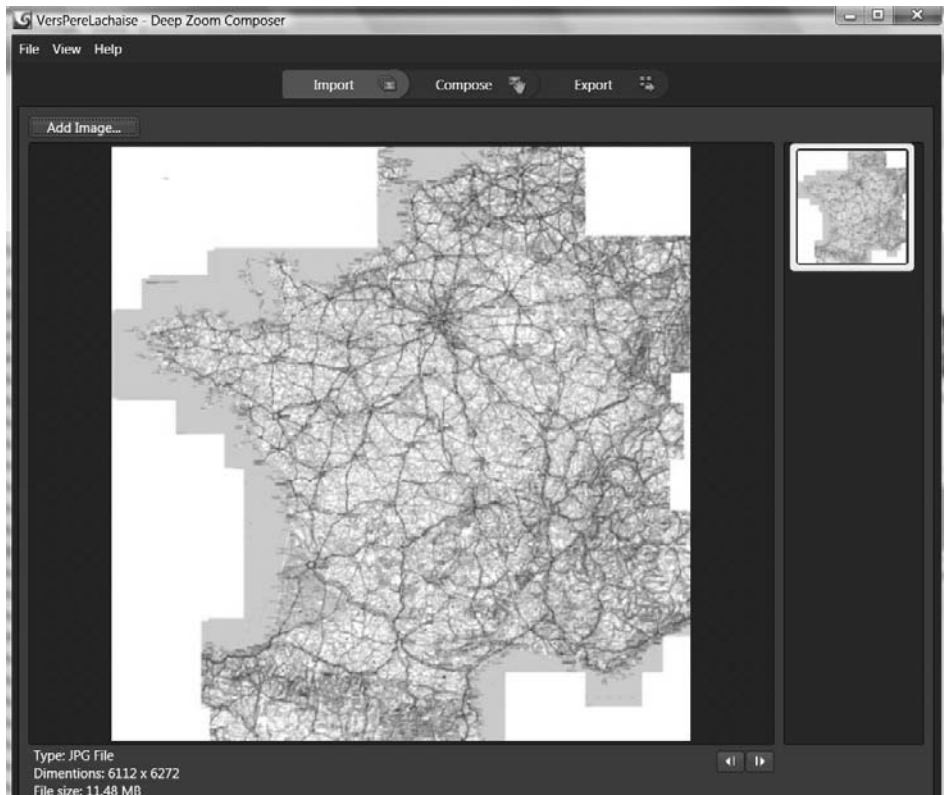
Nous allons créer un nouveau projet et, pour cela, Deep Zoom Composer va passer par trois étapes :

- Import : importation de vos différentes images (dont certaines peuvent être de très grande taille) ;
- Compose : phase de composition dans laquelle vous devez indiquer les chevauchements d'images afin de rentrer dans des zones de plus en plus précises ;
- Export : phase de création des images intermédiaires (Deep Zoom Composer va ainsi créer plusieurs dizaines d'images de taille optimale pour des transferts de qualité sur le Web).

Pour illustrer l'utilisation de la technologie Deep Zoom, nous allons créer une application Silverlight qui permettra de zoomer sur une carte de France pour en arriver à une visite du cimetière du Père-Lachaise.

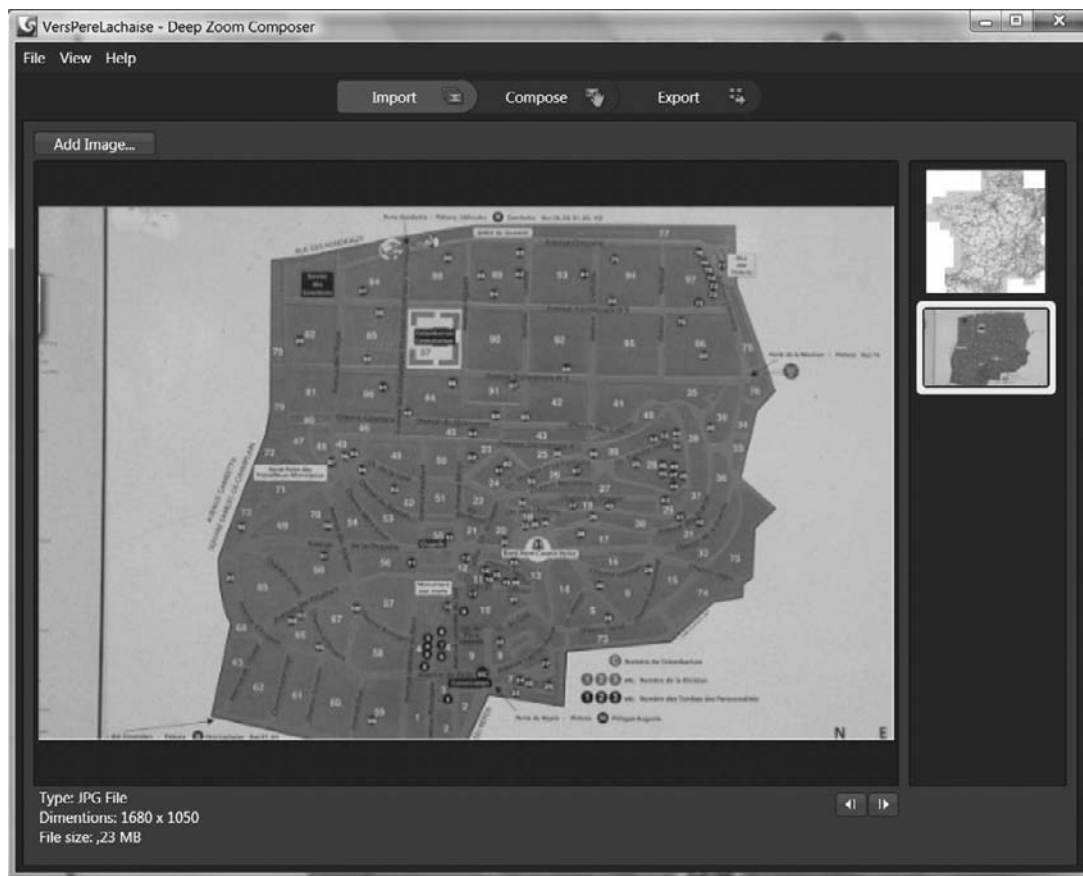
Chargez tout d'abord une carte de France, qui provient d'une image de très grande taille (plus de 10 Mo). Pour cela, cliquez sur le bouton Add Image... et sélectionnez l'image sur votre ordinateur (figure 7-18).

Figure 7-18



Chargez ensuite le plan du Père-Lachaise selon la même procédure (figure 7-19). Idéalement, il aurait fallu charger également une carte de la région parisienne, une autre de Paris et éventuellement une dernière du 20^e arrondissement, de manière à zoomer avec une précision de plus en plus grande tout en offrant un luxe de détails à chaque stade de notre visite virtuelle.

Figure 7-19



Passons maintenant à la phase de composition. Pour cela, cliquez sur l'onglet Compose et faites glisser la carte de France dans le cadre central. Sélectionnez ensuite l'outil Zoom situé en bas de ce cadre central et zoomez sur la région parisienne (figure 7-20).

Puis, faites glisser la vignette de la seconde image (plan du Père-Lachaise) dans le cadre central et placez-la à l'endroit où l'utilisateur entrera en zoomant au Père-Lachaise (heureusement pour lui, confortablement assis devant son ordinateur).

Figure 7-20



Passons maintenant à la dernière phase, au cours de laquelle Deep Zoom Composer calcule et crée toutes les images intermédiaires (plusieurs dizaines). Cliquez pour cela sur l'onglet Export.

À ce stade, il est possible de réclamer la création d'un projet Visual Studio (cochez pour cela l'option Export Images and Silverlight Project, figure 7-21). Dans ce cas, les images seront automatiquement copiées dans un sous-répertoire `GeneratedImages` du répertoire `ClientBin` de la partie Web du projet (celui du fichier XAP). Il faut un peu chercher pour le trouver et passer par plusieurs niveaux de sous-répertoires, dont `OutputSdi` et `source Images`, mais le projet est bien là. Vous pouvez le récupérer et l'utiliser comme base d'une application bien plus étoffée.

Un fichier `MouseWheelHelper` est également créé, qui contient le code de détection des mouvements de la molette de la souris. L'utilisateur de l'application Silverlight pourra ainsi, de manière très naturelle, zoomer de plus en plus ou de moins en moins profondément lorsque la souris survolera l'image dans une page Silverlight.

Sans quitter Deep Zoom Composer, on peut déjà tester le résultat de cette application. Pour cela, cliquez sur le bouton Export et cliquez sur Preview in Browser (figure 7-22).

Figure 7-21

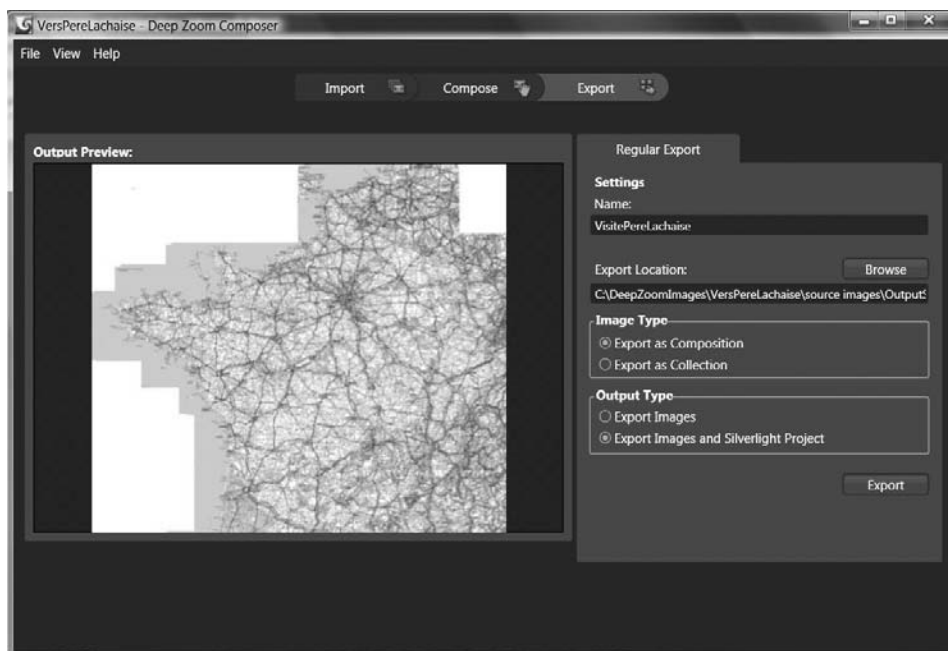
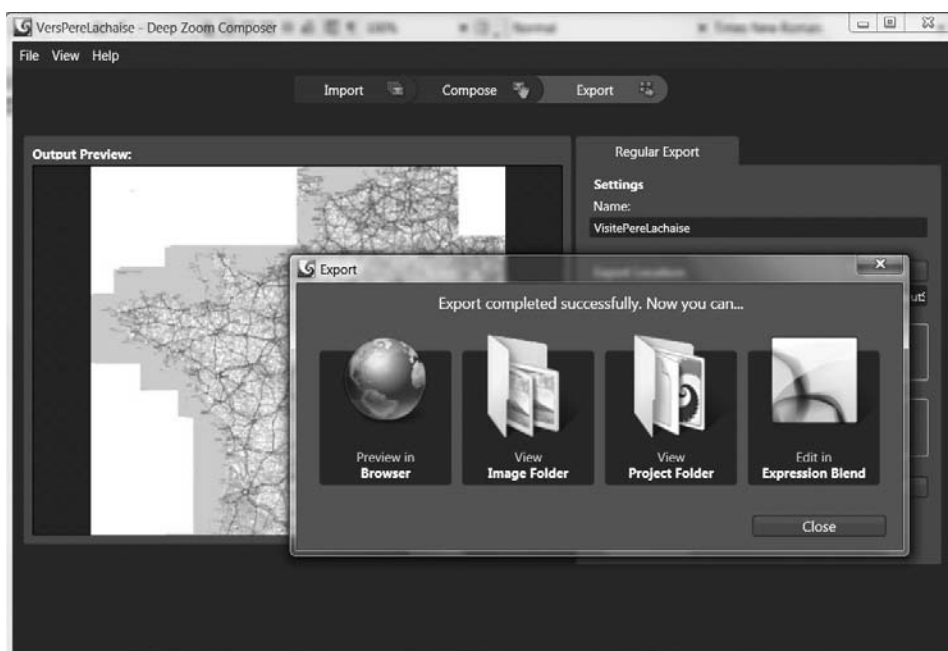


Figure 7-22



Dans le projet de l'application Silverlight, il suffit ensuite d'insérer un composant `MultiScaleImage` (en fait, c'est déjà fait par Deep Zoom Composer) qui s'utilise comme le composant `Image`, sauf que l'attribut `Source` doit faire référence à un fichier d'extension `.bin` créé par Deep Zoom Composer :

```
<MultiScaleImage Source="GeneratedImages/dzc_output/info.bin"
  Height="440" x:Name="msi" Width="600" Canvas.Left="20" Canvas.Top="20"/>
```

Il reste à tester l'application, ici avec Safari (figures 7-23 et 7-24) et Firefox (figure 7-25) à différents stades du zoom. Impressionnant et prometteur d'applications Web d'un nouveau type !

Figure 7-23

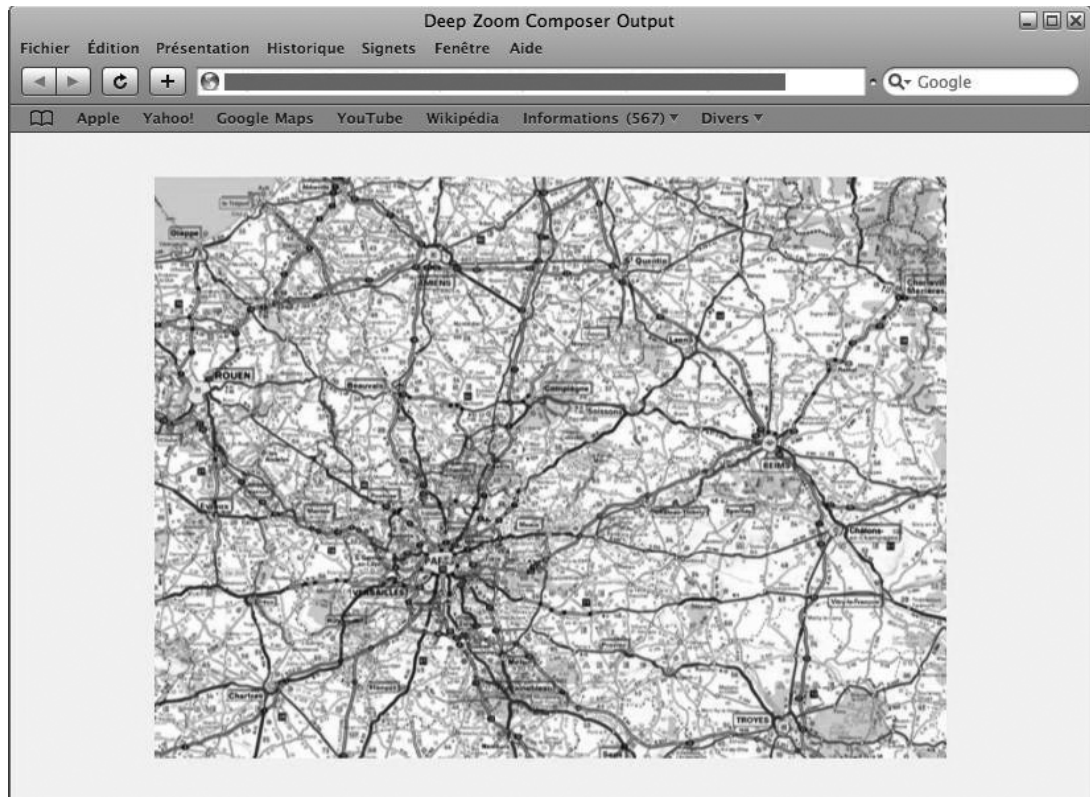


Figure 7-24

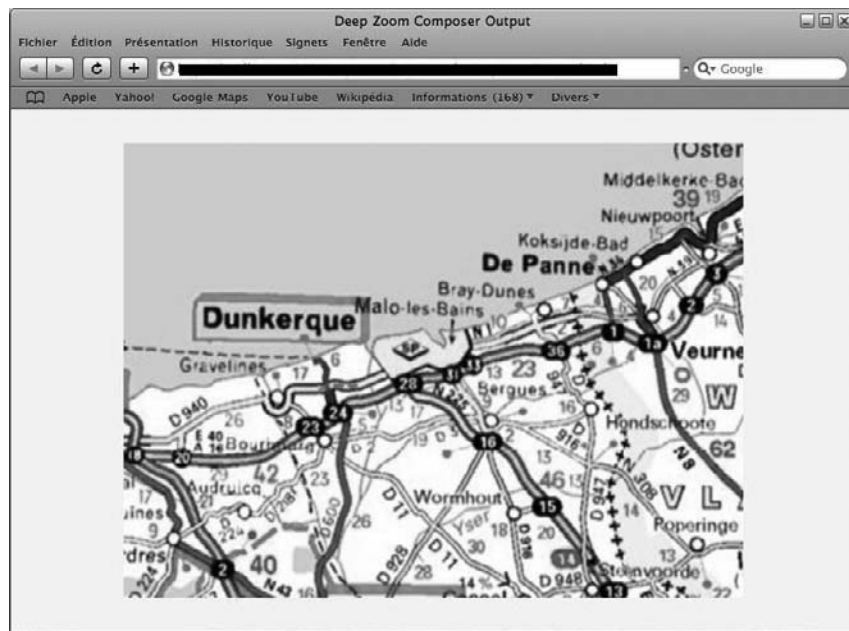
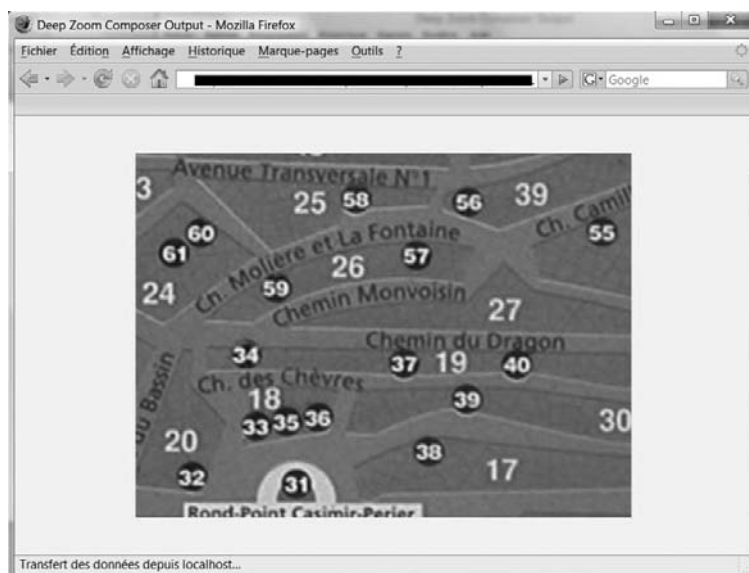


Figure 7-25



Tourner la page

Nous allons maintenant apprendre à tourner la page... Pour cela, prenons l'exemple d'un album de peintures de Breughel dont on peut tourner les pages comme on le ferait avec n'importe quel album « papier » à la différence ici que les pages sont tournées au moyen de la souris (en cliquant sur un coin de la page et en déplaçant la souris, bouton enfoncé, vers la gauche ou vers la droite (figures 7-26 à 7-28).

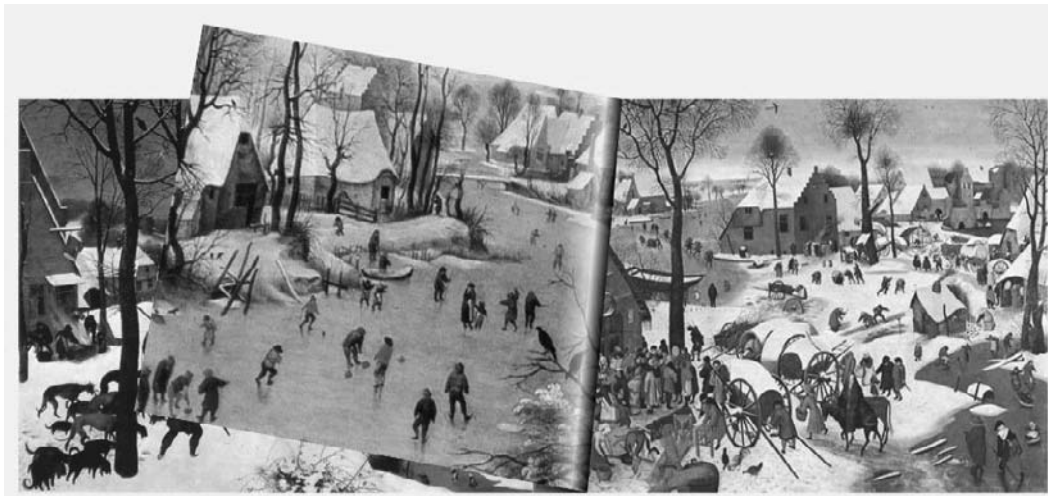
Figure 7-26



Figure 7-27



Figure 7-28



Pour réaliser cet album « virtuel », il faut utiliser un composant développé par Mitsu Furuta de Microsoft France et téléchargeable sur le site <http://www.codeplex.com/wpfbookcontrol>. Vous pouvez utiliser ce composant sans être redevable de royalties. La seule obligation est de respecter les règles Codeplex, qui sont vraiment de bon sens et légitimes envers un contributeur qui met gratuitement un composant d'un tel intérêt à la disposition de tous.

Téléchargez le fichier ZIP mis à disposition et décompressez-le. Vous constatez alors qu'il contient un fichier nommé `SLMitsuControls.dll`, dont la taille est de 34 Ko (cette DLL sera greffée au fichier XAP envoyé au client).

Voyons maintenant comment insérer ce composant dans une page Silverlight. Pour cela, vous devez tout d'abord faire référence à la DLL `SLMitsuControls.dll` dans le projet de l'application. Effectuez un clic droit sur le projet Silverlight dans l'Explorateur de solutions, sélectionnez le menu *Ajouter une référence...* > *Parcourir...* et localisez le fichier `SLMitsuControls.dll` sur votre ordinateur.

Dans la balise `UserControl` du fichier `Page.xaml`, ajoutez la ligne de code suivante, qui indique que les composants préfixés de `local` sont acceptés et que le code correspondant se trouve dans `SLMitsuControls` (vous pourriez remplacer `local` par tout autre préfixe de votre choix) :

```
xmlns:local="clr-namespace:SLMitsuControls;assembly=SLMitsuControls"
```

Ajoutez ce composant dans la page Silverlight, dans `Page.xaml` (ici, dans une cellule de grille, sans spécifier de largeur ni de hauteur pour que le composant s'adapte automatiquement à la taille de la cellule) :

```
<local:UCBook x:Name="book" Grid.Row="1" Grid.Column="1" />
```

Il convient ensuite de traiter l'événement `Loaded` pour le conteneur qu'est la grille afin de lier les données du composant à des éléments de la page. Complétez également le fichier

Page.xaml.cs comme suit, avec les fonctions `GetItem` et `GetCount`, et en indiquant que la classe `Page` implémente l'interface `IDataProvider` :

```
using SLMitsuControls;
.....
public partial class Page : UserControl, IDataProvider
{
    .....
    private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
    {
        book.SetData(this);
    }
    public object GetItem(int index)
    {
        return pages.Items[index];
    }
    public int GetCount()
    {
        return pages.Items.Count;
    }
}
```

En ressource du conteneur qu'est la grille, indiquez les images à reprendre dans l'album :

```
<Grid.Resources>
<ItemsControl x:Name="pages" >
    <Image Source="Breughel1.jpg" Stretch="Fill" />
    <Image Source="Breughel2.jpg" Stretch="Fill" />
    <Image Source="Breughel3.jpg" Stretch="Fill" />
    <Image Source="Breughel4.jpg" Stretch="Fill" />
    <Image Source="Breughel5.jpg" Stretch="Fill" />
</ItemsControl>
</Grid.Resources>
```

Ces images doivent ensuite être copiées dans le répertoire `ClientBin` de la partie Web de la solution. Pour cela, effectuez un glisser-déposer depuis l'Explorateur de fichiers jusqu'au nom du répertoire `ClientBin` dans l'Explorateur de solutions de Visual Studio, partie Web.

L'utilisateur peut désormais tourner les pages !

Pour savoir quelles sont les pages affichées à l'écran, il suffit de traiter l'événement :

```
<local:UCBook x:Name="book" ..... OnPageTurned="book_OnPageTurned" />
```

Cette fonction de traitement est :

```
private void book_OnPageTurned(int leftPageIndex, int rightPageIndex)
{
}
}
```

Elle est automatiquement exécutée chaque fois que l'utilisateur tourne une page. `leftPageIndex` et `rightPageIndex` contiennent respectivement le numéro de la page de gauche et celui de la page de droite, avec `-1` en cas d'absence de page.

Avec un attribut d'image `Stretch` tel que `Uniform` (qui garde les proportions de l'image mais laisse des bords vides), il est préférable d'insérer l'image dans une cellule de grille (sinon, tout se passe comme si les pages de l'album étaient en plastique transparent). Par exemple :

```
<Grid Background="AliceBlue" >
  <Image Source="Breughel1.jpg" Stretch="Uniform" />
</Grid>
```

Des événements comme `MouseLeftButtonDown` peuvent être traités pour l'image, ce qui peut indiquer que l'utilisateur souhaite des informations complémentaires sur l'article de la page.

Le contenu d'une page n'est pas limité à une image. Il peut être une image avec du texte d'accompagnement :

```
<Grid Background="AliceBlue" >
  <StackPanel>
    <Image Source="Breughel-VolIcare.jpg" Stretch="Fill" />
    <TextBlock Text="Vol d'Icare.jpg" Foreground="Red" TextAlignment="Center" />
  </StackPanel>
</Grid>
```

une boîte de listes ou encore n'importe quel composant ou combinaison de composants Silverlight :

```
<ListBox >
  <ListBoxItem Content="Allemagne" FontFamily="Verdana" FontSize="25" />
  <ListBoxItem Content="Belgique" FontFamily="Verdana" FontSize="25" />
  <ListBoxItem Content="France" FontFamily="Verdana" FontSize="25" />
  <ListBoxItem Content="Italie" FontFamily="Verdana" FontSize="25" />
  <ListBoxItem Content="Luxembourg" FontFamily="Verdana" FontSize="25" />
  <ListBoxItem Content="Pays-Bas" FontFamily="Verdana" FontSize="25" />
</ListBox>
```

Programmes d'accompagnement

Exemple 1 :

Scrolling d'une image de grande taille à l'aide de la souris et sans utiliser le composant `ScrollViewer` (figure 7-29).

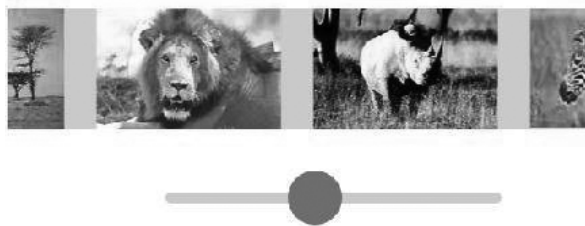
Figure 7-29



Exemple 2 :

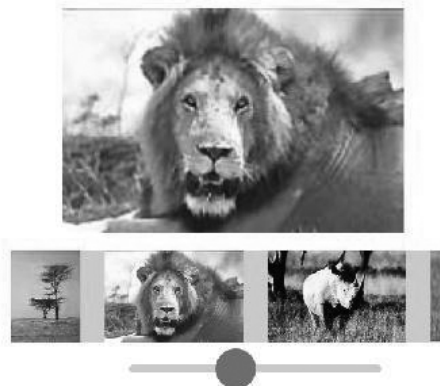
Défilement d'images (figure 7-30).

Figure 7-30

Défilement d'images**Exemple 3 :**

Défilement d'images avec animation lors de l'affichage (en grande taille) de l'image sélectionnée (figure 7-31).

Figure 7-31

Défilement + animation sur sélection**Exemple 4 :**

Carrousel d'images avec animation de l'image survolée par la souris (figure 7-32).

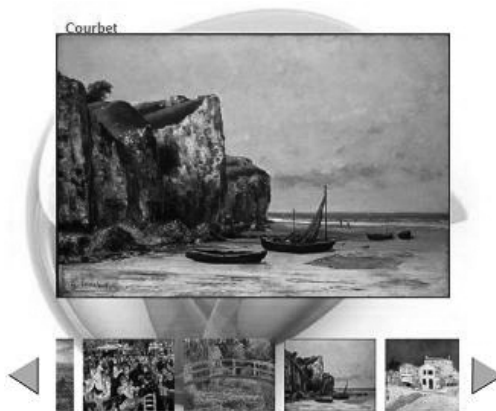
Figure 7-32



Exemple 5 :

Autre version du carrousel (figure 7-33).

Figure 7-33

**Exemple 6 :**

Version tournante d'un carrousel. Les images tournent en permanence tout en restant verticales. Un extrait de film est projeté suite à un clic sur une affiche (les fichiers .wmv ne sont pas proposés en téléchargement, figure 7-34).

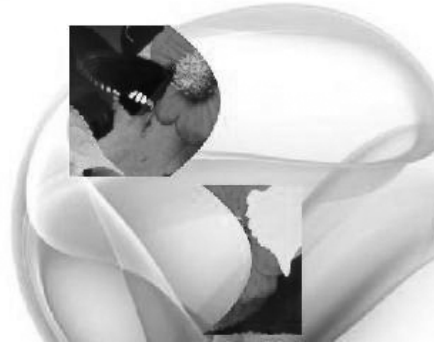
Figure 7-34



Exemple 7 :

Puzzle (ramené à deux pièces pour simplifier la démonstration) avec projection d'une partie de film dans chaque pièce. Le film est reconstitué en plaçant les différentes pièces du puzzle à leur emplacement correct (figure 7-35).

Figure 7-35

Puzzle pour film

Les figures géométriques

Line, Polyline et Polygon

Vous allez à présent apprendre à tracer des lignes, des courbes (notamment de Bézier) ainsi que des formes plus complexes que les simples rectangles et ellipses, en XAML bien sûr, mais aussi avec l’outil graphique Expression Blend.

Pour commencer, tracez une ligne ou une série de lignes. Les propriétés de la famille Stroke (Stroke, StrokeThickness, etc., voir la section « Le Stroke » de ce chapitre) qui s’appliquent au contour des figures jouent un rôle important avec les éléments UI dont il est question ici, mais aussi avec les rectangles et les ellipses. Les différentes balises permettant de tracer des lignes sont présentées dans le tableau 8-1.

Tableau 8-1 – Les balises/objets permettant de tracer des lignes

Balise / objet	Description
Line	Trace une ligne entre un premier point en (X1, Y1) et un second en (X2, Y2). Par exemple, pour tracer une ligne en bleu, épaisse de 3 pixels entre le point (10, 10) et le point (100, 100) : <code><Line X1="10" Y1="10" X2="100" Y2="100" Stroke="Blue" StrokeThickness="3" /></code>
Polyline	Trace une série de lignes. Les différents points (attribut Points) reliant les segments de droite sont séparés par au moins un espace tandis que les coordonnées X et Y d'un point sont séparées par un espace ou par une virgule (celle-ci étant même préférable car elle assure une meilleure lisibilité). La propriété Fill peut être spécifiée pour peindre l'intérieur, même si aucune ligne ne relie le dernier point au premier. Ici une ligne noire épaisse d'un pixel démarrant en (10, 10) et tracée jusqu'en (10, 100), une verticale donc, suivie d'une seconde (oblique) reliant le point précédent au point (50, 50) : <code><Polyline Points="10,10 10,100 50,50 Stroke="Black" /></code>
Polygon	Mêmes propriétés que Polyline à la différence qu'une ligne est tracée pour relier le dernier point au premier. <code><Polygon Points="10,10 10,100 50,50 Stroke="Black" /></code>

À noter que la propriété `StrokeDashArray` (voir la section « Le Stroke » de ce chapitre) permet de créer des lignes en pointillés.

Le Path

Passons maintenant aux figures de n'importe quelle forme. Avec la balise `Path`, via son attribut `Data`, il est possible de tracer une série de lignes et de courbes, autrement dit des figures qui peuvent devenir aussi complexes que réalistes. Un `Path` constituant un élément UI (comme `Rectangle` ou `Ellipse`), on retrouve les propriétés `Fill` avec ses différents pinceaux pour le coloriage intérieur, `Stroke` pour le contour, `Opacity`, etc.

Un `Path` peut être défini de différentes manières :

- à l'aide d'une chaîne de caractères, en attribut `Data` de la balise `Path` ;
- dans une balise `Path.Data`, elle-même insérée dans une balise `Path` ;
- en passant à `Expression Blend` (outil graphique générant des balises XAML qui peuvent devenir très complexes) mais sans oublier que ce logiciel ne fait que générer la chaîne de caractères, parfois extrêmement complexe, mentionnée au premier point.

La première manière consiste à spécifier les différentes opérations de tracé dans une chaîne de caractères. Chaque commande dans la chaîne `Data` commence par une lettre, suivie de coordonnées. Dans cette chaîne `Data`, une distinction doit être faite entre majuscules et minuscules :

- les coordonnées sont absolues si la lettre de la commande est une majuscule ;
- les coordonnées sont relatives au dernier point si la lettre de la commande est une minuscule.

`Canvas.Left` et `Canvas.Top` peuvent être spécifiés en attributs de `Path`, ce qui donne les coordonnées du point de départ (encore faut-il que la balise ait un canevas comme conteneur).

Le tableau 8-2 présente les différentes commandes de génération de la balise `Path`.

Tableau 8-2 – Les différentes commandes de génération de la balise `Path`

Nom de la commande	Description
<code>Mx,y</code>	Déplacement sans tracé jusqu'au point (x, y). Avec <code>M</code> (majuscule), la coordonnée (x, y) est relative au coin supérieur gauche du conteneur. Avec <code>m</code> (minuscule), les coordonnées sont relatives au point précédemment atteint. Il est possible de spécifier <code>Canvas.Left</code> et <code>Canvas.Top</code> en attributs de la balise <code>Path</code> . Si le conteneur est un canevas, les coordonnées sont alors relatives à ce point.
<code>Lx,y</code>	Trace une ligne reliant le point courant au point (x, y).
<code>Hx</code>	Trace une horizontale depuis le point courant jusqu'au point x.
<code>Vy</code>	Trace une verticale depuis le point courant jusqu'au point y.
<code>Arx,ry d f1 f2 x,y</code>	Trace un arc jusqu'au point (x, y). L'arc est une partie de l'ellipse ayant <code>rx</code> et <code>ry</code> comme rayons et un angle de rotation de <code>d</code> degrés (dans le sens des aiguilles d'une montre si <code>f1</code> vaut 0). <code>f2</code> doit avoir la valeur 0 si l'arc est inférieur à 180 degrés et la valeur 1 s'il est supérieur.

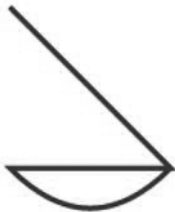
Tableau 8-2 – Les différentes commandes de génération de la balise Path (suite)

Nom de la commande	Description
Cx1,y1 x2,y2 x,y	Trace une courbe de Bézier cubique depuis le point courant jusqu'au point (x, y). Les deux points (x1, y1) et (x2, y2) agissent en tant que points de contrôle (autrement dit, comme des aimants). Le point courant et le point (x, y) sont donc situés sur la courbe, contrairement aux points de contrôle. Ceux-ci exercent un effet d'attraction sur la courbe. Voir la section « Les courbes de Bézier » de ce chapitre pour une étude approfondie des courbes de Bézier.
Qx1,y1 x,y	Trace une courbe de Bézier quadratique (un seul point d'attraction) entre le point courant et le point (x, y). Le point (x1, y1) agit comme un aimant pour le tracé de la courbe.
Z	Termine une série de commandes, traçant une ligne jusqu'au point de départ.

Le code suivant correspond au tracé représenté à la figure 8-1 :

```
<Path Stroke="Blue" StrokeThickness="3"
      Data="M100,100 L200,200 h-100 Q150,250 200,200" />
```

Figure 8-1



Ce code produit :

- un positionnement, sans rien tracer, au point (100, 100) ;
- le tracé d'une ligne droite (ici, en oblique) jusqu'au point (200, 200) ;
- le tracé d'une ligne horizontale de 100 pixels à gauche, autrement dit jusqu'au point (100, 200) ;
- le tracé d'une courbe (dite de Bézier) jusqu'au point (200, 200), avec le point (150, 250) qui agit comme point d'attraction.

La deuxième manière permettant de définir un Path (peut-être la moins pratique des trois) consiste à insérer des balises dans la balise Path.Data, elle-même contenue dans la balise Path :

```
<Path .....>
  <Path.Data>
    .....
  </Path.Data>
</Path>
```

Dans la balise Path.Data, vous pouvez spécifier des balises dites de géométrie. Dans le jargon Silverlight, une géométrie consiste en la description 2D d'une ligne, d'une courbe

ou d’une surface. Il ne s’agit donc pas d’un élément UI, comme Rectangle, Ellipse ou Path. Une géométrie ne présente donc pas de propriétés Fill, Stroke ou Opacity.

Le tableau 8-3 décrit les différentes balises de géométrie. Les trois premières sont évidentes et n’ont d’intérêt que dans un groupe (balise GeometryGroup).

Tableau 8-3 – Les différentes balises de géométrie

Nom de la balise	Description
EllipseGeometry	Permet d’insérer une ellipse dans un path. Spécifiez Center, RadiusX et RadiusY. Par exemple : <code><EllipseGeometry Center="100,100" RadiusX="50" RadiusY="50" /></code>
RectangleGeometry	Permet d’insérer un rectangle dans un path. Spécifiez Rect (une chaîne de caractères avec X, Y, W et H) et, éventuellement, RadiusX et RadiusY pour les coins arrondis. Par exemple : <code><RectangleGeometry Rect="100,100, 80, 60" /></code>
LineGeometry	Permet d’insérer une ligne dans un path. Spécifiez StartPoint et EndPoint (chacun de la forme X, Y).
PathGeometry	Permet d’insérer n’importe quelle figure. Une balise PathGeometry peut contenir une balise PathGeometry.Figures, laquelle peut contenir une ou plusieurs balises PathFigure contenant une balise PathFigure.Segments. La balise PathFigure.Segments peut, quant à elle, contenir des balises ArcSegment, BezierSegment, LineSegment, PolyBezierSegment, PolyLineSegment, PolyQuadraticBezierSegment et QuadraticBezierSegment.

Dans une balise GeometryGroup, il est possible de combiner deux géométries de base ou davantage (y compris des PathGeometry).

L’exemple de code suivant correspond au tracé de la figure 8-2 :

```
<Path Stroke="Blue" StrokeThickness="5" >
  <Path.Data>
    <GeometryGroup >
      <LineGeometry StartPoint="100,100" EndPoint="200,200" />
      <RectangleGeometry Rect="100,200,100,100" />
    </GeometryGroup>
  </Path.Data>
</Path>
```

Figure 8-2



À partir d'un certain niveau de complexité de la figure, il devient plus simple de passer à Expression Blend qui génère automatiquement les commandes de l'attribut Data (voir la section « Dessiner avec Expression Blend » de ce chapitre).

Les courbes de Bézier

Les courbes et surfaces de Bézier jouent un rôle considérable dans les représentations graphiques de courbes et de surfaces lisses. Elles ont été imaginées pour répondre aux problèmes de la conception et de la fabrication assistées par ordinateur, en particulier dans le domaine de l'automobile. Silverlight étant limité à la 2D, le run-time Silverlight offre des fonctions pour dessiner ces courbes lisses, connues sous le nom de courbes de Bézier.

Les courbes dont nous parlons ici tirent leur nom de Pierre Bézier, ingénieur français né à Paris en 1910 et diplômé en tant qu'ingénieur mécanicien et ingénieur électricien. Il a passé toute sa vie professionnelle chez Renault, débutant comme ouvrier manuel, puis comme modelleur et ajusteur. À la fin des années 1950, il était responsable de la production mais une nouvelle tâche lui fut assignée (Pierre Bézier lui-même donna une interprétation toute personnelle à cette « promotion ») : il s'agissait de préparer l'introduction des ordinateurs comme aide à la fabrication, ce qui était généralement considéré comme utopique, voire farfelu, à l'époque et tout particulièrement à la Régie. Si l'histoire vous intéresse, rendez-vous sur le site de Régis Le Boité (<http://www.le-boite.com/idee.htm>) et lisez le papier désopilant rédigé par Pierre Bézier lui-même dans lequel il explique, avec autant d'humour que de férocité, comment ses idées étaient reçues par ses employeurs.

Pierre Bézier devait faire face au problème suivant : trouver des méthodes pour alimenter les machines numériques à partir des dessins à main levée des designers de voitures. Il mit alors au point des algorithmes, qu'il présenta d'abord comme issus des travaux de recherche du fameux Professeur Onésime Durant, nés de l'esprit potache de l'ingénieur. Ces algorithmes sont aujourd'hui communément utilisés à travers le monde et dans de très nombreux domaines. Quand il partit à la retraite à l'âge de 65 ans, Pierre Bézier prépara une thèse de doctorat en mathématiques et fut honoré du titre de Docteur deux ans plus tard pour sa contribution au domaine, à savoir la mise en équation des courbes et des surfaces lisses. Pierre Bézier mourut à l'âge de 89 ans, encore très actif dans le domaine de la conception et de la fabrication assistées par ordinateur. Il n'est jamais arrivé à la conférence technique où il était attendu et devait faire part de ses dernières idées...

Après ce rapide rappel historique, intéressons-nous maintenant à la troisième et dernière manière de définir un Path, soit au moyen des courbes de Bézier. Ces courbes partent d'un point S, arrivent en un point E et sont attirées par un ou plusieurs points de contrôle agissant comme des aimants. En XAML :

- une courbe `BezierSegment` a deux points de contrôle (autrement dit, d'attraction) ;
- une courbe `QuadraticBezierSegment` a un seul point de contrôle.

Commençons avec la courbe `QuadraticBezierSegment` qui, malgré son nom impressionnant, est la plus simple puisqu'elle n'a qu'un seul point d'attraction. Une courbe de Bézier est

un Path (un path pouvant représenter n'importe quelle figure) et est définie comme suit en XAML (on aurait pu utiliser la commande Q dans l'attribut Data du Path) :

```
<Path Stroke="Red" ..... >
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="50,150" >
        <QuadraticBezierSegment
          Point1="100,300" Point2="500,300" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

La courbe de Bézier ainsi définie (figure 8-3) :

- démarre au point courant ou au point spécifié dans l'attribut StartPoint de la balise PathFigure ;
- se termine au point Point2 ;
- est attirée par le point Point1.

Figure 8-3



Dans la balise Path, vous pouvez ajouter d'autres attributs (Fill, par exemple) ainsi qu'un point de départ (propriétés attachées Canvas.Left et Canvas.Top). Pour dessiner plusieurs figures (pas nécessairement plusieurs QuadraticBezierSegment), il suffit de placer une balise PathFigures dans la balise PathGeometry.

Pour mieux comprendre, voyons comment sont créées les courbes de Bézier et, pour cela, il convient d'expliquer comment est créé chaque point de la courbe.

Pour une courbe de Bézier avec un seul point d'attraction, S constitue le point de départ, E le point d'arrivée et C le point de contrôle (autrement dit, le point d'attraction, voir figure 8-4). Pour simplifier, nous supposons que la courbe de Bézier est construite en 10 étapes (ce qui est évidemment insuffisant dans la pratique). Imaginons ensuite que nous traçons deux lignes droites : entre S et C pour la première ligne, entre E et C pour la seconde. Lors d'une étape (par exemple, 3 de 10), nous prenons un point au 3/10 le long de la ligne SC (le point A) et un autre au 3/10 le long de la ligne EC (le point B, voir figure 8-5). Nous joignons ensuite les points A et B et prenons le point qui se trouve au 3/10 le long de la ligne AB, soit le point X, qui est un point (pour l'étape 3 de 10) de la courbe

de Bézier (figure 8-6). Il suffit à présent de joindre les points ainsi obtenus à chaque étape pour obtenir la courbe de Bézier.

Figure 8-4

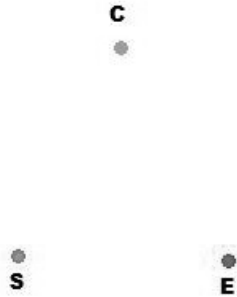


Figure 8-5

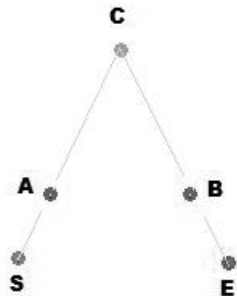
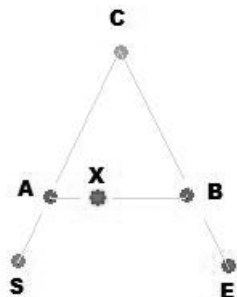


Figure 8-6



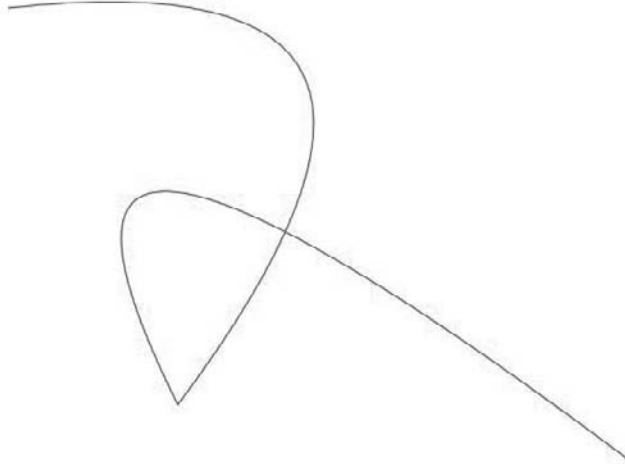
Plutôt simple, n'est-ce pas ? Bien sûr, les mathématiciens ont une manière bien plus impressionnante d'arriver au point X...

Passons maintenant au `PolyQuadraticBezierSegment` qui consiste en une succession de `QuadraticBezierSegment`.

L'exemple de code suivant correspond au tracé représenté à la figure 8-7 :

```
<Path Stroke="Red" ..... >
  <Path.Data>
    <PathGeometry>
      <PathGeometry.Figures>
        <PathFigure StartPoint="50,50" >
          <PolyQuadraticBezierSegment
            Points="500,0 200,400 0,0 600,450" />
        </PathFigure>
      </PathGeometry.Figures>
    </PathGeometry>
  </Path.Data>
</Path>
```

Figure 8-7



Deux courbes `QuadraticBezierSegment` ont été définies. La première présente les caractéristiques suivantes :

- elle démarre en (50, 50) ;
- se termine en (200, 400) ;
- est attirée par (500, 0).

La seconde courbe présente quant à elle les caractéristiques suivantes :

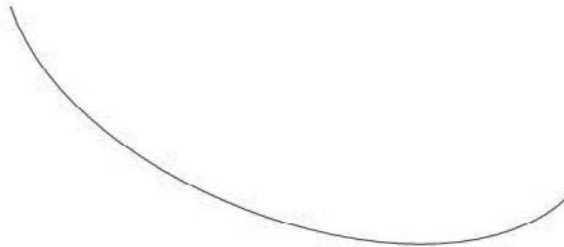
- elle démarre en (200, 400) qui est le point courant après avoir dessiné la courbe précédente,
- se termine en (600, 450) ;
- est attirée par le point (0, 0).

Dans un programme, l'attribut `Points` de `PolyQuadraticBezierSegment` consiste en un tableau de `Points`, autrement dit un `Point[]`. Chaque courbe du `PolyQuadraticBezierSegment` utilise deux entrées dans les valeurs de l'attribut `Points` : la première pour le point d'attraction et la seconde pour le point terminal, qui devient le point de départ de la prochaine courbe.

Présentons maintenant le `BezierSegment`, qui est une courbe de Bézier avec deux points d'attraction. L'exemple de code suivant correspond au tracé représenté à la figure 8-8 :

```
<Path Stroke="Red" ..... >
  <Path.Data>
    <PathGeometry>
      <PathFigure StartPoint="50,150" >
        <BezierSegment
          Point1="100,300" Point2="400,400"
          Point3="500,300" />
      </PathFigure>
    </PathGeometry>
  </Path.Data>
</Path>
```

Figure 8-8



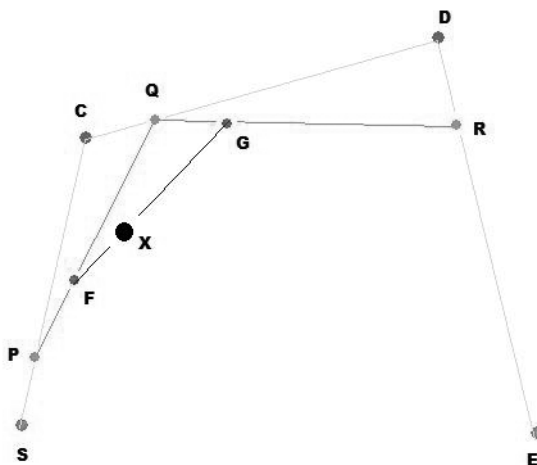
La courbe de Bézier ainsi définie :

- démarre au point courant ou au point spécifié dans `StartPoint` de la balise `PathFigure` ;
- se termine au point `Point3` ;
- est attirée par les points de contrôle `Point1` et `Point2`.

Là aussi, pour mieux comprendre, voyons comment est créée une courbe de Bézier avec deux points d'attraction. Considérons que *S* est le point de départ, *E* le point d'arrivée et que *C* et *D* sont des points de contrôle. Pour simplifier, prenons le cas d'une courbe de Bézier construite en 10 étapes. Imaginons ensuite que nous traçons trois lignes droites : une entre *S* et *C* (la ligne *SC*), une autre entre *C* et *D* (la ligne *CD*) et une troisième entre *D* et *E* (la ligne *CE*). Lors de l'étape 3 (3 de 10), nous prenons un point au 3/10 le long de la ligne *SC* (le point *P*), un autre au 3/10 le long de la ligne *CD* (le point *Q*) et un dernier au 3/10 le long de la ligne *DE* (le point *R*). Nous traçons ensuite des lignes entre les points *P* et *Q* (ligne *PQ*) et *Q* et *R* (ligne *QR*). Le long de ces lignes, nous prenons des points au 3/10, soit les points *F* et *G*. Nous traçons ensuite la ligne *FG* et prenons un

point (X) au 3/10 le long de cette ligne. Le point X appartient à la courbe BezierSegment pour l'étape 3 de 10 (figure 8-9).

Figure 8-9



Le PolyBezierSegment est semblable dans son principe au PolyQuadraticBezierSegment : il contient une série de BezierSegment. Dans l'attribut Points, il faut spécifier une série de trois points (deux points de contrôle et un point terminal) pour chaque courbe. Le point terminal d'une des courbes devient le point de départ de la courbe suivante.

Le Stroke

Pour tracer (peindre serait sans doute plus approprié) le contour de figures (rectangles, ellipses, etc.), il faut initialiser les attributs de la famille Stroke. Il ne s'agit pas d'un attribut général et de sous-attributs particuliers, mais bien d'une série d'attributs, de Stroke à StrokeThickness.

Le contour des figures est réellement peint à l'aide d'un pinceau (SolidColorBrush, LinearGradientBrush, RadialGradientBrush, ImageBrush ou encore VideoBrush) dont l'effet n'est généralement perceptible que sur des lignes épaisses. Par défaut, aucun contour n'est dessiné. Si l'attribut Stroke est présent, l'épaisseur du contour est de 1 pixel (valeur par défaut). Le tableau 8-4 présente les cinq principaux attributs de la famille Stroke.

L'exemple de code suivant correspond au tracé de la figure 8-10 :

```
<Line X1="20" Y1="20" X2="150" Y2="150"
      Stroke="Red" StrokeThickness="20"
      StrokeStartLineCap="Triangle"
      StrokeEndLineCap="Triangle" />
```

Figure 8-10

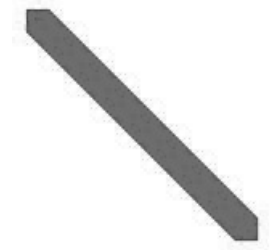


Tableau 8-4 – Les cinq principaux attributs de la famille Stroke

Nom de l'attribut	Description
Stroke	Pinceau utilisé pour peindre le contour d'une figure, tracer une ligne ou une courbe. Comme attribut XAML, il peut s'agir d'un nom de couleur ou d'une valeur exprimée dans les systèmes sRGB ou scRGB. Stroke est un objet d'une classe dérivée de Brush (voir chapitre 4).
StrokeStartLineCap	Indique comment commence une ligne (effet uniquement perceptible sur des lignes épaisses). Une des valeurs de l'énumération LineCap : Flat (valeur par défaut, rien de spécial), Round (rond ajouté au début de la ligne, de manière à la lisser), Triangle (triangle ajouté au début de la ligne) et Square (carré ajouté).
StrokeEndLineCap	Comme StrokeStartLineCap mais pour la fin de la ligne.
StrokeLineJoin	Jointure de lignes. Une des valeurs de l'énumération : Bevel (angles biseautés), Miter (angles en forme de mitre d'évêque) et Round (angles arrondis).
StrokeThickness	Épaisseur, en pixels. Peut être une valeur décimale (l'œil perçoit une différence entre 0.5 et 1).

Et le code suivant correspond à la figure 8-11 :

```
<Line X1="20" Y1="150" X2="150" Y2="20"
      StrokeThickness="60" >
  <Line.Stroke>
    <ImageBrush ImageSource="Fleur.jpg" />
  </Line.Stroke>
</Line>
```

Figure 8-11



Dans le premier exemple (figure 8-10), la ligne est peinte à l'aide d'un pinceau rouge. Dans le second exemple (figure 8-11), le pinceau est plus complexe car il est basé sur une image incorporée en ressource. Il pourrait s'agir d'un pinceau à dégradé linéaire ou radial, ou encore d'un pinceau basé sur une image ou même une vidéo (ce qui est quand même rare pour les contours).

Pour créer des lignes en pointillés, utilisez la propriété `StrokeDashArray`, qui peut prendre une ou deux valeurs : une longueur de segment tracé et une longueur d'espacement (par défaut, égale à la première valeur).

L'exemple de code suivant correspond au tracé représenté à la figure 8-12 :

```
<Line Stroke="Black"
      StrokeThickness="5" StrokeDashArray="2"
      X1="10" Y1="10" X2="200" Y2="10" />
```

Figure 8-12



Et le code suivant, au tracé de la figure 8-13 :

```
<Line Stroke="Black"
      StrokeThickness="5" StrokeDashArray="6 2"
      X1="10" Y1="10" X2="200" Y2="10" />
```

Figure 8-13



Le `StrokeDashArray` est un tableau d'une ou deux valeurs de type `double`. Par programme, le code à écrire en C# est :

```
li.StrokeDashArray = new double[] { 6, 2 };
```

et en VB :

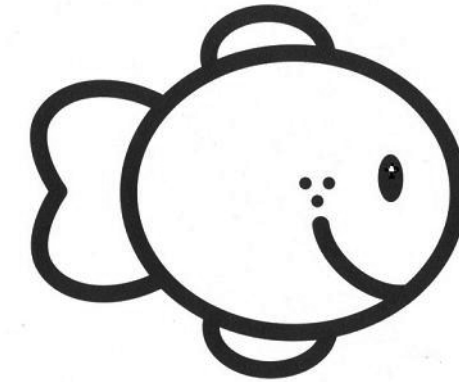
```
Dim tabDash() As Double = {6, 2}
li.StrokeDashArray = tabDash
```

Dessiner avec Expression Blend

Supposons que l'on désire représenter en XAML le poisson de la figure 8-14, en vue de le déplacer ou de l'animer. Celui-ci est formé d'une succession de lignes courbes, autrement dit de paths. Nous ne montrerons ici que les premiers pas, la suite n'étant qu'une répétition de ceux-ci.

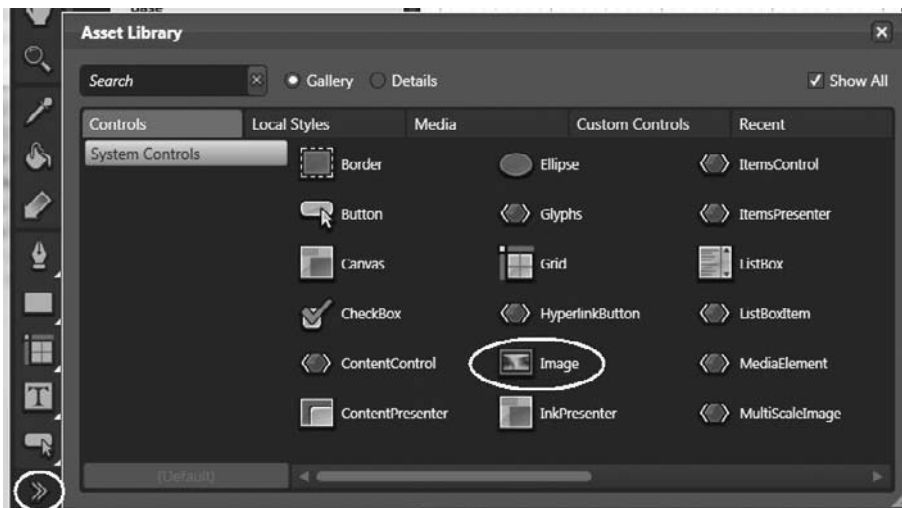
Les virtuoses de la tablette graphique n'auront aucun problème à reproduire ce dessin à main levée. Pour les autres, la solution consiste à scanner ce dessin et à le reproduire par une succession de courbes XAML (des ellipses pour la partie centrale et l'œil, et des courbes de Bézier pour le reste).

Figure 8-14



Pour réaliser le poisson de la figure 8-14, démarrez un projet sous Visual Studio et passez à Expression Blend. Pour rappel, il suffit pour cela d'effectuer un clic droit sur le nom du fichier XAML dans l'Explorateur de solutions et de sélectionner le menu Open in Expression Blend. À noter qu'il est possible de démarrer en créant directement un projet avec Expression Blend (créer alors un projet Silverlight 2). Au cas où l'outil Image n'est pas déjà présent dans la boîte d'outils, cliquez sur l'icône symbolisant des chevrons fermants et double-cliquez sur l'icône Image (figure 8-15).

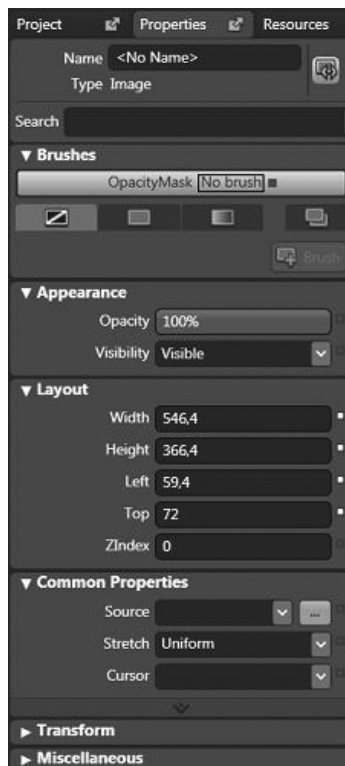
Figure 8-15



Placez ensuite l'image du poisson dans la surface de travail d'Expression Blend (cliquez sur l'outil Image dans la boîte d'outils, puis sur la surface de travail) et redimensionnez-la de manière à pouvoir travailler confortablement. Dans le panneau des propriétés, indiquez le nom du fichier contenant l'image dans le champ Source de l'onglet Common Properties.

Pour rappel, l'image doit être incorporée en ressource ou être copiée dans le répertoire ClientBin de la partie Web de l'application.

Figure 8-16



L'image sera supprimée par la suite ; pour le moment, elle vous aidera pour le tracé. La figure 8-17 présente la surface de travail d'Expression Blend à ce stade de la création du dessin.

Créez maintenant un rectangle sur la surface de travail (prenez l'habitude de sélectionner au préalable le conteneur parent). Pour cela, cliquez sur l'outil Rectangle (à noter qu'en maintenant le bouton de la souris enfoncé pendant une seconde, d'autres outils apparaissent tels que Ellipse ou Ligne). Placez ce rectangle au même emplacement que l'image et attribuez-lui les mêmes dimensions que cette dernière. Réglez ensuite la propriété Opacity à 0. L'image est maintenant visible mais vous allez dessiner sur le rectangle. Pour tracer des courbes, sélectionnez l'outil Pen (figure 8-18) : le curseur de la souris affiche désormais un signe +.

Cliquez ensuite sur deux points d'inflexion (le signe + du curseur se transforme en signe – si vous approchez d'un point déjà sélectionné, ce qui permet en cliquant dessus de le supprimer). Une ligne est alors tracée entre ces deux points (figure 8-19). Augmentez l'épaisseur (propriété StrokeThickness) de la ligne.

Figure 8-17

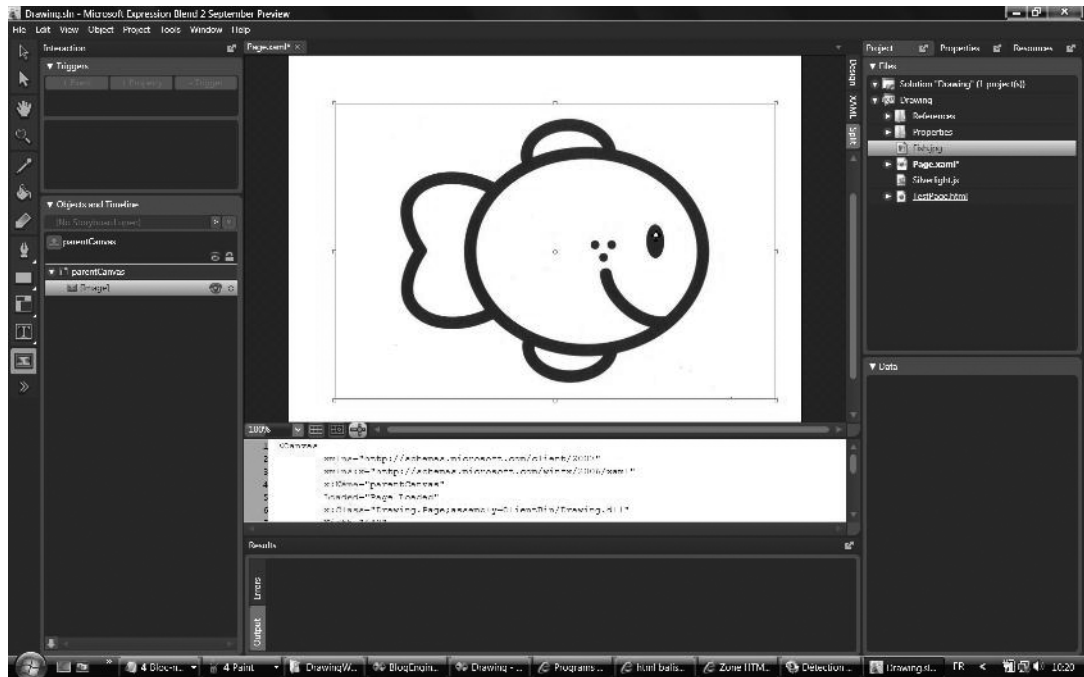
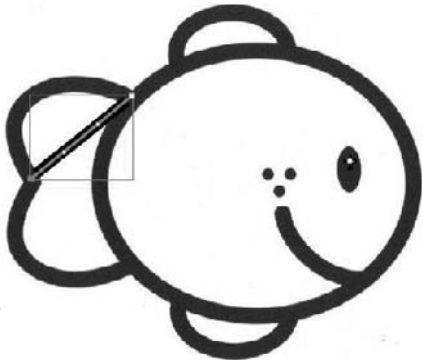


Figure 8-18

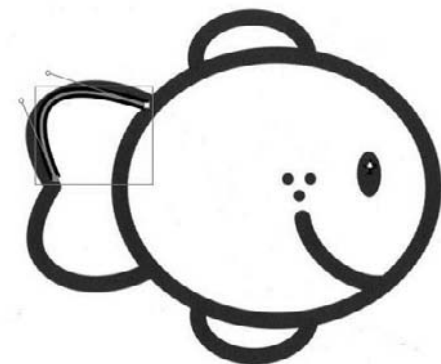


Figure 8-19



Il s'agit maintenant d'incurver la ligne. Pour cela, appuyez sur la touche Alt et déplacez le curseur de la souris près de la ligne (il se transforme alors en ^). Faites glisser la souris pour donner à la ligne la courbure désirée : la courbe épouse désormais celle de la figure d'origine (figure 8-20).

Figure 8-20



Procédez de la même manière pour les autres courbes. À la fin, supprimez l'image d'origine ainsi que le rectangle (invisible). Pour cela, effectuez un clic droit sur la fenêtre des objets (Objects window) et supprimez-les avec Delete.

Une série de paths vient d'être créée. Pour les manipuler par programme, donnez-leur un nom (propriété Name dans le panneau des propriétés).

Il est possible de créer un Path simple à partir de plusieurs Path. Pour cela, sélectionnez-les en cliquant dessus avec la touche Ctrl enfoncée et choisissez le menu Objects>Make Compound Path.

Programmes d'accompagnement

Exemple 1 :

Création et coloriage d'un clown (clic sur une couleur suivi d'un clic sur le personnage pour en colorier une partie).

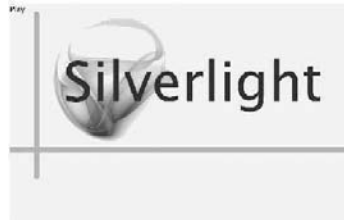
Figure 8-21



Exemple 2 :

Logo animé. Des lignes se déplacent, s'agrandissent et se redressent pendant que le texte apparaît progressivement en s'agrandissant.

Figure 8-22

**Exemple 3 :**

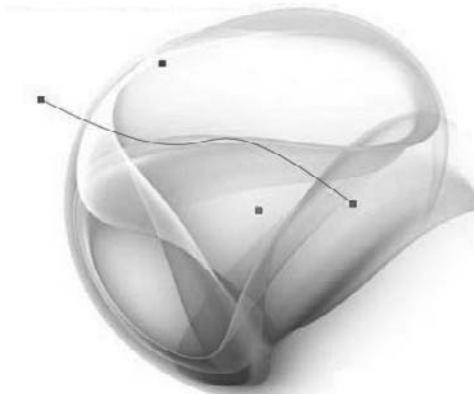
Tracé d'une courbe de Bézier avec un seul point de contrôle. On peut déplacer les trois points à l'aide de la souris : la courbe de Bézier se redessine alors automatiquement.

Figure 8-23

**Exemple 4 :**

Même chose pour une courbe de Bézier à deux points de contrôle.

Figure 8-24



9

Les transformations et les animations

Les transformations

Au cours des chapitres précédents, nous avons vu comment placer des éléments visuels dans une page Web Silverlight. Mais ceci est encore banalement traditionnel...

Nous allons à présent étudier les transformations qui vous permettront, par exemple, d'appliquer une rotation à un élément UI. Pour cela, il suffit d'attribuer un nom à la transformation pour pouvoir agir sur celle-ci par programme, donc en cours d'exécution.

L'étude des transformations nous mènera tout naturellement aux animations.

Le tableau 9-1 présente les différents types de transformations susceptibles d'être appliquées à un élément visuel.

Au lieu de spécifier `CenterX` et `CenterY` (pour rappel, il s'agit de coordonnées relatives au composant mais exprimées en pixels), il est possible de notifier `RenderTransformOrigin` (en coordonnées relatives) dans la balise d'élément.

Ainsi, au lieu d'écrire :

```
<TextBlock Text="HELLO" FontSize="50" Width="120" Height="80" >
  <TextBlock.RenderTransform>
    <RotateTransform Angle="45" CenterX="60" CenterY="40" />
  </TextBlock.RenderTransform>
</TextBlock>
```

on peut écrire :

```
<TextBlock Text="HELLO" FontSize="50" Width="120" Height="80"
    RenderTransformOrigine="0.5, 0.5" >
    <TextBlock.RenderTransform>
        <RotateTransform Angle="45" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Tableau 9-1 – Les différents types de transformations

Nom de la transformation	Description
TranslateTransform	<p>L'élément UI est déplacé de X pixels le long de l'axe des X et de Y pixels le long de l'axe des Y. X et Y sont des attributs de TranslateTransform. Par exemple (ici, un rectangle mais il pourrait s'agir de n'importe quel élément UI) :</p> <pre><Rectangle > <Rectangle.RenderTransform> <TranslateTransform X="100" Y="50" /> </Rectangle.RenderTransform> </Rectangle></pre> <p>Cette solution est générale et s'applique quel que soit le conteneur, ce qui n'est pas le cas de la modification directe par Canvas.Left et Canvas.Top (solution uniquement possible si le conteneur est un canevas).</p>
ScaleTransform	<p>L'élément UI est agrandi ou rétréci, comme spécifié dans les attributs ScaleX et ScaleY : Width est multiplié par ScaleX et Height par ScaleY. La mise à l'échelle est réalisée (dans toutes les directions) à partir du point (CenterX, CenterY). Par défaut, il s'agit du point (0, 0), qui correspond au coin supérieur gauche. ScaleX et/ou ScaleY peuvent être négatifs, ce qui crée un effet de miroir (voir exemples ci-dessous).</p>
RotateTransform	<p>L'élément UI subit une rotation, dont l'angle (attribut Angle) est mesuré en degrés, dans le sens des aiguilles d'une montre. La rotation est opérée autour d'un point spécifié par CenterX et CenterY (voir les exemples ci-dessous où l'ellipse noire du second exemple est présente uniquement pour montrer le point de rotation, qui est par défaut le coin supérieur gauche). Les coordonnées CenterX et CenterY sont relatives au composant mais exprimées en pixels.</p>
SkewTransform	<p>Transformation qui provoque un effet oblique le long de l'axe des X et/ou l'axe des Y. Un carré peut ainsi devenir un parallélogramme.</p>
TransformGroup	<p>Permet de combiner plusieurs transformations. Par exemple :</p> <pre><TextBlock > <TextBlock.RenderTransform> <TransformGroup> <RotateTransform Angle="45" /> <TranslateTransform X="100" /> </TransformGroup> </TextBlock.RenderTransform> </TextBlock></pre>
MatrixTransform	<p>La transformation est définie par une matrice (voir plus loin).</p>

Les extraits de code suivants correspondent à des exemples de transformations sur le texte « Hello ».

Effet miroir sur du texte (figure 9-1) :

```
<TextBlock Text="HELLO" FontSize="20" >
  <TextBlock.RenderTransform>
    <ScaleTransform ScaleX="-1"
      CenterX="31" CenterY="15" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Figure 9-1



Rotation (figure 9-2) :

```
<Ellipse Canvas.Left="195" Canvas.Top="195"
  Width="10" Height="10" Fill="Black" />
<TextBlock Canvas.Left="200" Canvas.Top="200"
  Text="HELLO" FontSize="20" >
  <TextBlock.RenderTransform>
    <RotateTransform Angle="45" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Figure 9-2



Effet oblique (figure 9-3) :

```
<TextBlock Text="HELLO" FontSize="20" >
  <TextBlock.RenderTransform>
    <SkewTransform AngleX="25" />
  </TextBlock.RenderTransform>
</TextBlock>
```

Figure 9-3



Les exemples suivants correspondent à des transformations appliquées à une image (figure 9-4) dont le code XAML est :

```
<StackPanel >
  <Image Source="TM.jpg" Width="360" Height="300" />
</StackPanel>
```

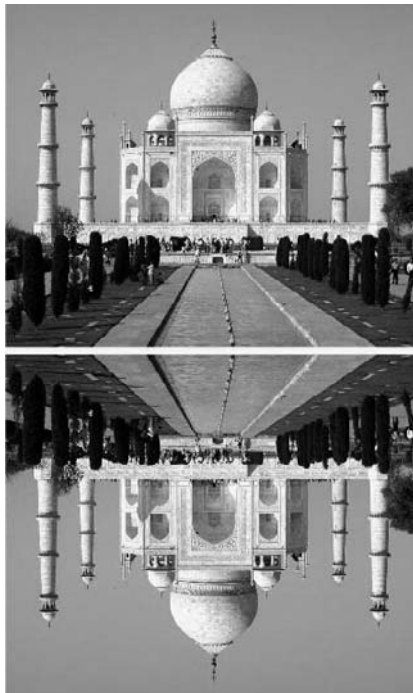
Figure 9-4



Effet miroir (figure 9-5) :

```
<StackPanel>
  <Image Source="TM.jpg" Width="360" Height="300" />
  <Image Source="TM.jpg" Width="360" Height="300" >
    <Image.RenderTransform>
      <ScaleTransform ScaleY="-1" CenterY="150" />
    </Image.RenderTransform>
  </Image>
</StackPanel>
```

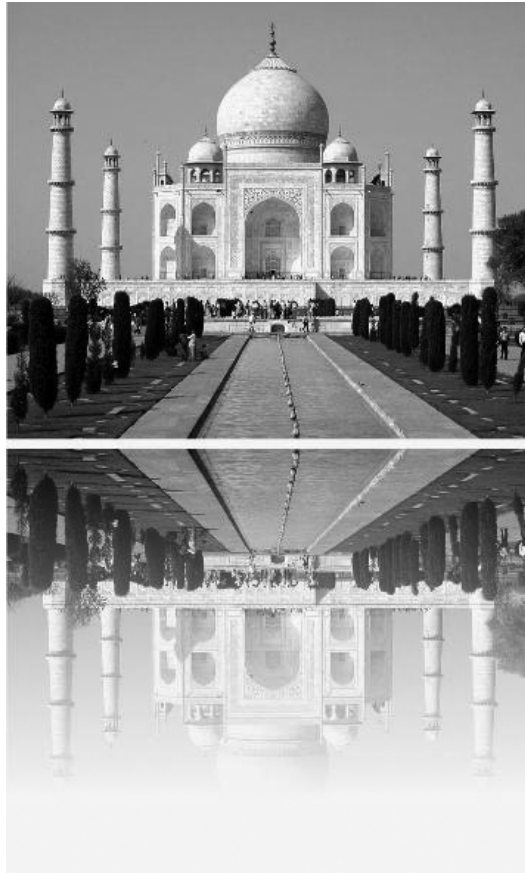
Figure 9-5



Effet miroir et dégradé de transparence (figure 9-6) :

```
<StackPanel>
  <Image Source="TM.jpg" Width="360" Height="300"/>
  <Image Source="TM.jpg" Width="360" Height="300">
    <Image.RenderTransform>
      <ScaleTransform ScaleY="-1" CenterY="150" />
    </Image.RenderTransform>
    <Image.OpacityMask>
      <LinearGradientBrush StartPoint="0,1"
                          EndPoint="0,0" >
        <GradientStop Offset="0" Color="Black" />
        <GradientStop Offset="0.8" Color="Transparent" />
      </LinearGradientBrush>
    </Image.OpacityMask>
  </Image>
</StackPanel>
```

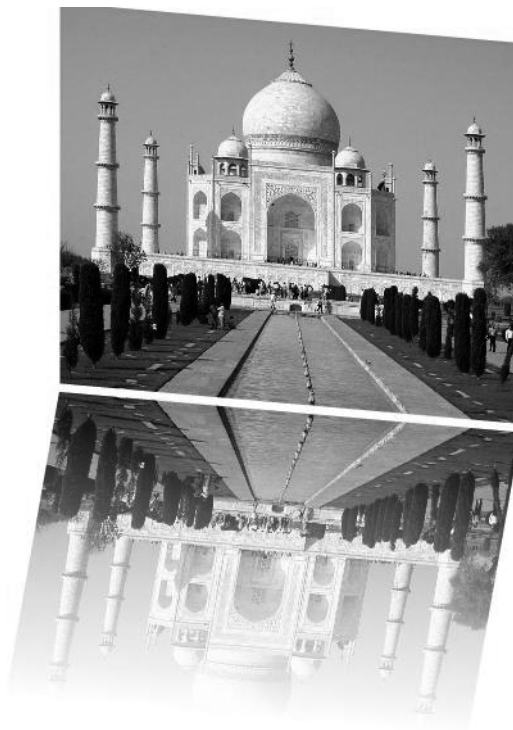
Figure 9-6



Effets précédents auxquels un effet oblique a été ajouté (figure 9-7) :

```
<StackPanel" >
  <Image Source="TM.jpg" Width="360" Height="300" >
    <Image.RenderTransform>
      <SkewTransform AngleY="5" />
    </Image.RenderTransform>
  </Image>
  <Image Source="TM.jpg" Width="360" Height="300" >
    <Image.RenderTransform>
      <TransformGroup>
        <ScaleTransform ScaleY="-1" CenterY="150" />
        <SkewTransform AngleX="-10" AngleY="5" />
      </TransformGroup>
    </Image.RenderTransform>
    <Image.OpacityMask>
      <LinearGradientBrush StartPoint="0,1"
        EndPoint="0,0" >
        <GradientStop Offset="0" Color="Black" />
        <GradientStop Offset="0.8" Color="Transparent" />
      </LinearGradientBrush>
    </Image.OpacityMask>
  </Image>
</StackPanel>
```

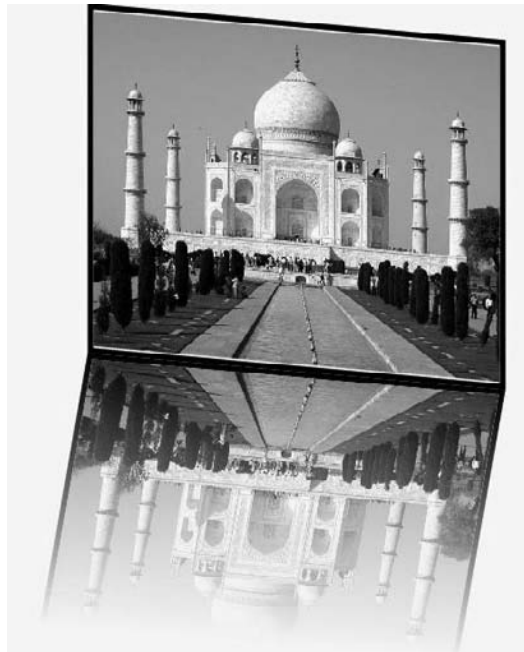
Figure 9-7



Effets précédents auxquels un cadre a été ajouté (figure 9-8) :

```
<StackPanel >
  <Border BorderBrush="Black" BorderThickness="5"
    Width="370">
    <Border.RenderTransform>
      <SkewTransform AngleY="5" />
    </Border.RenderTransform>
    <Image Source="TM.jpg" Width="360" Height="300" />
  </Border>
  <Border BorderBrush="Black" BorderThickness="5"
    Width="370" Height="300" >
    <Border.RenderTransform>
      <TransformGroup>
        <ScaleTransform ScaleY="-1" CenterY="150" />
        <SkewTransform AngleX="-10" AngleY="5" />
      </TransformGroup>
    </Border.RenderTransform>
    <Border.OpacityMask>
      <LinearGradientBrush StartPoint="0,1"
        EndPoint="0,0">
        <GradientStop Offset="0" Color="Black" />
        <GradientStop Offset="0.8" Color="Transparent" />
      </LinearGradientBrush>
    </Border.OpacityMask>
    <Image Source="TM.jpg" Width="360" Height="300" />
  </Border>
</StackPanel>
```

Figure 9-8



Une transformation peut être définie par une transformation particulière (comme `RotateTransform`) ou un groupe de transformations mais aussi, en termes mathématiques, par une matrice.

Le calcul matriciel nous apprend qu'un point (X, Y) peut être transformé en un point (A, B) par l'opération matricielle suivante :

```
A = X*M11 + Y*M21 + deplX
B = X*M12 + Y*M22 + deplY
```

`MatrixTransform` est définie par :

```
<MatrixTransform Matrix="M11, M12, M21, M22, deplX, deplY" />
```

où M11, M12, M21, M22, deplX et deplY doivent être remplacés par des valeurs.

Pour déplacer un rectangle de 100 pixels à gauche, il faut appliquer la transformation suivante :

```
<Rectangle .....>
<Rectangle.RenderTransform>
  <MatrixTransform Matrix="1, 0, 0, 1, -100, 0 " />
</Rectangle.RenderTransform>
</Rectangle>
```

Et pour réaliser un effet de miroir autour du bord supérieur :

```
<MatrixTransform Matrix="1, 0, 0, -1, 0, 0 " />
```

Forcer une transformation par programme

Les transformations peuvent être initiées par programme. Si vous attribuez un nom interne (ici, `rotText`) à la balise `RotateTransform` :

```
<TextBlock x:Name="info" Text="HELLO" ..... >
<TextBlock.RenderTransform>
  <RotateTransform x:Name="rotText" Angle="45" />
</TextBlock.RenderTransform>
</TextBlock>
```

le texte subit une rotation supplémentaire (ici, de 10 degrés) chaque fois que l'on exécute :

```
rotText.Angle += 10;
```

Le texte subit une rotation autour du point (`CenterX`, `CenterY`), qui est par défaut le coin supérieur gauche. Pour le faire tourner autour du centre du texte, exécutez au préalable :

```
rotText.CenterX = info.ActualWidth / 2;
rotText.CenterY = info.ActualHeight / 2;
```

Une autre solution consiste à spécifier `RenderTransformOrigin="0.5, 0.5"` dans la balise `TextBlock` (jusqu'ici, cela paraît simple) mais il faut alors aussi spécifier `Width` et `Height` dans cette balise (ce qui dépend du texte et de la police).

Même si une balise d'élément UI ne contient aucune balise `RenderTransform`, il est possible de forcer une transformation par programme.

Pour illustrer notre propos, partons de la balise suivante :

```
<TextBlock x:Name="za" Text="HELLO" ..... />
```

et forçons une transformation par programme. En C#, cela donne :

```
RotateTransform rt = new RotateTransform();
rt.Angle = 90;
za.RenderTransform = rt;
```

Et en VB :

```
Dim rt As RotateTransform = New RotateTransform()
rt.Angle = 90
za.RenderTransform = rt
```

À la section « Le signal d'horloge (timer) » du chapitre 6, nous avons vu comment afficher l'heure en permanence, mais sous forme de texte. Nous allons à présent afficher l'heure au moyen d'une horloge dont les aiguilles tournent (l'image de l'horloge a été insérée en ressource de l'application Silverlight, voir figure 9-9).

Figure 9-9



Le code XAML pour réaliser cela est le suivant :

```
<Canvas x:Name="hor1" Width="200" Height="200" Loaded="hor1_Loaded" >
  <Image Source="Horloge.jpg" Width="200" Height="200" />
  <!-- aiguille des heures -->
  <Path Data="M100,96 165,4 1-65,4 z" Fill="Black" >
    <Path.RenderTransform>
      <RotateTransform x:Name="rotHeure" CenterX="100" CenterY="100" />
    </Path.RenderTransform>
  </Path>
  <!-- aiguille des minutes -->
  <Path Fill="Black" Data="M100,98 180,2 1-80,2 z" >
    <Path.RenderTransform>
      <RotateTransform x:Name="rotMin" CenterX="100" CenterY="100" />
    </Path.RenderTransform>
  </Path>
```

```

<!-- aiguille des secondes -->
<Rectangle Fill="Black" Width="85" Height="2"
           Canvas.Left="99" Canvas.Top="99" >
  <Rectangle.RenderTransform >
    <RotateTransform x:Name="rotSec" CenterX="1" CenterY="1" />
  </Rectangle.RenderTransform>
</Rectangle>
</Canvas>

```

ce qui donne en C# :

```

using System.Windows.Threading;
.....
private void horl_Loaded(object sender, RoutedEventArgs e)
{
    DispatcherTimer timer = new DispatcherTimer();
    timer.Interval = new TimeSpan(0, 0, 1);
    timer.Tick += new EventHandler(timer_Tick);
    MettreAJourHorloge();
    timer.Start();
}
void timer_Tick(object sender, EventArgs e)
{
    MettreAJourHorloge();
}
void MettreAJourHorloge()
{
    int sec = DateTime.Now.Second;
    int min = DateTime.Now.Minute;
    int heure = DateTime.Now.Hour % 12;
    rotSec.Angle = sec * 6 - 90; // 6 degrés par seconde
    rotMin.Angle = min * 6 - 90; // 6 degrés par minute
    rotHeure.Angle = heure * 30 - 90; // 30 degrés par heure
}

```

et en VB :

```

Imports System.Windows.Threading
.....
Private Sub horl_Loaded(ByVal sender As System.Object,
                       ByVal e As System.Windows.RoutedEventArgs)
    Dim timer As New DispatcherTimer()
    timer.Interval = New TimeSpan(0, 0, 1)
    AddHandler timer.Tick, AddressOf timer_Tick
    MettreAJourHorloge()
    timer.Start()
End Sub
Private Sub timer_Tick(ByVal sender As Object, ByVal e As EventArgs)
    MettreAJourHorloge()
End Sub

```

```
Private Sub MettreAJourHorloge()  
    Dim sec As Integer = DateTime.Now.Second  
    Dim min As Integer = DateTime.Now.Minute  
    Dim heure As Integer = DateTime.Now.Hour Mod 12  
    rotSec.Angle = sec * 6 - 90 ' 6 degrés par seconde  
    rotMin.Angle = min * 6 - 90 ' 6 degrés par minute  
    rotHeure.Angle = heure * 30 - 90 ' 30 degrés par heure  
End Sub
```

Les animations

Les animations From-To

Les animations figurent parmi les fonctionnalités les plus puissantes offertes par Silverlight. Elles sont définies en XAML mais peuvent aussi être créées et/ou manipulées par programme (pour démarrer une animation, l'arrêter ou en modifier les paramètres). Avec cette technique, il suffit de demander à Silverlight de faire passer un élément UI d'un état à un autre en un nombre défini de secondes (y compris les fractions de seconde). Nul besoin de fournir des informations intermédiaires souvent compliquées à calculer car Silverlight effectue tous les calculs intermédiaires à votre place.

Pour commencer, nous allons réaliser une animation très simple : agrandir une image en une fraction de seconde (mais avec une phase d'agrandissement bien perceptible) quand la souris entre dans la surface de l'image (événement `MouseEnter`).

Pour cela, il convient tout d'abord de spécifier une transformation (ici, une mise à l'échelle par `ScaleTransform`) pour l'image et de traiter l'événement `MouseEnter` :

```
<Image x:Name="imgElle" Source="Elle.jpg" Width="264" Height="140"  
    MouseEnter="imgElle_MouseEnter" >  
    <Image.RenderTransform>  
        <ScaleTransform x:Name="scaleElle" CenterX="132" CenterY="70" />  
    </Image.RenderTransform>  
</Image>
```

On pourrait déjà écrire la fonction `imgElle_MouseEnter` qui contiendrait :

```
scaleElle.ScaleX *= 1.5; scaleElle.ScaleY *= 1.5;
```

mais l'agrandissement serait instantané. Nous allons donc le rendre progressif à l'aide d'une animation et, pour cela, nous allons devoir écrire un story-board (constitué d'une ou de plusieurs animations). Ici, les propriétés (de type décimal et plus précisément `Double`) `ScaleX` et `ScaleY` de la transformation `scaleElle` vont donc être animées. Le story-board doit être compris dans une balise `Resources` de son conteneur.

Dans la mesure où les valeurs animées sont de type `Double`, il s'agit ici d'une animation de type `DoubleAnimation` (les autres types étant `PointAnimation` et `ColorAnimation`). En attribut de la balise `DoubleAnimation`, il convient donc de spécifier :

- l'élément concerné (attribut `Storyboard.TargetName`), ici `scaleElle` ;

- la propriété concernée (propriété `Storyboard.TargetProperty`), ici `ScaleX` pour la première balise, et `DoubleAnimation` et `ScaleY` pour la seconde (il y a en effet deux propriétés à animer) ;
- la valeur de `ScaleX` (`ScaleY` pour l'autre animation) qui doit passer de 1 à 1.5 : propriétés `From` et `To`, bien que `From` soit optionnelle (on part alors de la taille de l'image au moment d'exécuter `Begin` qui démarre l'animation) ;
- que l'animation doit être réalisée en une demi-seconde : propriété `Duration`, dont la valeur est une chaîne de caractères au format `hh:mm:ss.mmm` (avec `hh` pour les heures, `mm` pour les minutes, `ss` pour les secondes et `mmm` pour les millisecondes).

Le code suivant permet d'agrandir une image sur `MouseEnter` :

```
<Grid x:Name="LayoutRoot" ..... >
  <Grid.Resources>
    <Storyboard x:Name="stbAgrandirImage">
      <DoubleAnimation From="1" To="1.5" Duration="0:0:0.500"
        Storyboard.TargetName="scaleElle"
        Storyboard.TargetProperty="ScaleX" />
      <DoubleAnimation From="1" To="1.5" Duration="0:0:0.500"
        Storyboard.TargetName="scaleElle"
        Storyboard.TargetProperty="ScaleY" />
    </Storyboard>
  </Grid.Resources>
  .....
  <Image x:Name="imgElle" Source="Elle.jpg" Width="264" Height="140"
    MouseEnter="imgElle_MouseEnter" RenderTransformOrigin="0.5, 0.5" >
    <Image.RenderTransform>
      <ScaleTransform x:Name="scaleElle" />
    </Image.RenderTransform>
  </Image>
</Grid>
```

L'animation aurait également pu être définie en ressource de l'image :

```
<Image x:Name="imgElle" Source="Elle.jpg" Width="264" Height="140"
  MouseEnter="imgElle_MouseEnter" RenderTransformOrigin="0.5, 0.5" >
  <Image.Resources>
    <Storyboard x:Name="stbAgrandirImage">
      <DoubleAnimation From="1" To="1.5" Duration="0:0:0.500"
        Storyboard.TargetName="scaleElle"
        Storyboard.TargetProperty="ScaleX" />
      <DoubleAnimation From="1" To="1.5" Duration="0:0:0.500"
        Storyboard.TargetName="scaleElle"
        Storyboard.TargetProperty="ScaleY" />
    </Storyboard>
  </Image.Resources>
  <Image.RenderTransform>
    <ScaleTransform x:Name="scaleElle" CenterX="132" CenterY="70" />
  </Image.RenderTransform>
</Image>
```

À ce stade, il ne reste plus qu'à lancer l'animation dans la fonction `imgElle_MouseEnter` :

```
stbAgrandirImage.Begin();
```

Sur un story-board, on peut exécuter `Begin` (pour démarrer l'animation) et `Stop` (pour l'arrêter) mais aussi `Pause` (pour la suspendre) et `Resume` (pour la reprendre).

À la fin de l'animation, l'événement `Completed` est signalé. L'attribut `Completed` (ayant pour valeur le nom de la fonction de traitement) peut être spécifié dans une balise `Storyboard` ou une balise `DoubleAnimation` (plus précisément une des trois balises `xyzAnimation`, voir tableau 9-2).

Tableau 9-2 – Les différentes balises `xyzAnimation`

Nom de la balise	Description
<code>DoubleAnimation</code>	Permet d'animer une propriété de type décimal (petit rappel : les valeurs telles que <code>Width</code> , etc. sont de type décimal, double même et non entier).
<code>PointAnimation</code>	Permet d'animer une propriété de type <code>Point</code> .
<code>ColorAnimation</code>	Permet d'animer une couleur (plus précisément, un <code>SolidColorBrush</code> dans une propriété <code>Fill</code> ou <code>GradientStop</code>).

L'exemple de code suivant permet de réaliser une animation sur couleur (un rectangle est peint de bleu en jaune de manière perpétuelle en trois secondes) :

```
<Storyboard x:Name="anim" RepeatBehavior="Forever" >
  <ColorAnimation Storyboard.TargetName="rcCol"
                  Storyboard.TargetProperty="Color" Duration="0:0:3"
                  From="Blue" To="Yellow" AutoReverse="True" />
</Storyboard>
.....
<Rectangle x:Name="rc" Width="200" Height="150"
           Canvas.Left="100" Canvas.Top="100" >
  <Rectangle.Fill>
    <SolidColorBrush x:Name="rcCol" />
  </Rectangle.Fill>
</Rectangle>
```

On aurait pu se passer de nommer la balise `SolidColorBrush`, à condition d'écrire `TargetName` et `TargetProperty` de la manière suivante :

```
<ColorAnimation ....
  Storyboard.TargetName="rc"
  Storyboard.TargetProperty="(Rectangle.Fill).(SolidColorBrush.Color)" />
```

La dernière ligne de code se lit : « dans la propriété `Fill` du `Rectangle`, prendre la propriété `Color` du `SolidColorBrush` ». L'attribut `Fill` doit alors être présent dans la balise `Rectangle`.

L'attribut `TargetName` aurait pu être placé dans la balise `Storyboard` puisque toutes les balises d'animation (ici, une seule) se rapportent à la même variable `TargetName`.

Le tableau 9-3 présente les propriétés spécifiées dans les balises `xyzAnimation` (`xyz` pour `Double`, `Point` ou `Color`) mais aussi dans la balise `Storyboard` (dans ce cas, les propriétés sont applicables à l'ensemble des animations du story-board).

Tableau 9-3 – Les propriétés des balises `xyzAnimation` et `Storyboard`

Nom de la propriété	Description
<code>AutoReverse</code>	Si <code>AutoReverse</code> vaut <code>true</code> , l'animation est jouée en sens inverse quand elle arrive à la fin. La durée de l'animation correspond en fait au double de la durée mentionnée dans <code>Duration</code> .
<code>BeginTime</code>	Spécifie le délai avant de démarrer une animation (vous pouvez ainsi démarrer une animation aussitôt qu'une autre se termine ou arrive à un certain point). Pour le format, voir <code>Duration</code> .
<code>Duration</code>	Durée de l'animation, au format <code>hh:mm:ss.mmm</code> (par défaut, une seconde, les millisecondes étant optionnelles). Du point de vue programmation, il s'agit d'un objet <code>TimeSpan</code> .
<code>FillBehavior</code>	Permet d'indiquer ce qu'il faut faire quand l'animation se termine. <code>FillBehavior</code> peut contenir l'une des deux valeurs suivantes de l'énumération <code>FillBehavior</code> : <code>HoldEnd</code> (valeur par défaut qui correspond à « le dernier état de l'animation reste affiché ») ou <code>Stop</code> (aucun affichage).
<code>RepeatBehavior</code>	Nombre de répétitions de l'animation. L'attribut <code>RepeatBehavior</code> peut contenir : <code>Forever</code> ou <code>nx</code> , où <code>n</code> désigne le nombre de répétitions. Par défaut, l'animation n'est jouée qu'une seule fois (la fonction de traitement de l'événement <code>Completed</code> étant exécutée lorsque l'animation se termine). Avec <code>Forever</code> , l'animation est jouée en permanence.
<code>Storyboard.TargetName</code>	Nom (propriété <code>x:Name</code>) de l'élément animé.
<code>Storyboard.TargetProperty</code>	Propriété qui est animée. Une propriété attachée comme <code>Canvas.Left</code> doit être spécifiée entre parenthèses.
<code>From</code>	Valeur initiale de la propriété animée. En l'absence de clause <code>From</code> , l'animation commence avec la valeur courante de la propriété.
<code>To</code>	Valeur finale de la propriété.
<code>By</code>	Au lieu de spécifier une valeur <code>To</code> , vous pouvez spécifier un incrément (<code>To</code> prenant alors la valeur <code>From + By</code>).

Le fait que certains attributs puissent être spécifiés à la fois dans une balise `Storyboard` et dans une balise comme `DoubleAnimation` conduit parfois à une certaine confusion. Afin d'éclaircir les problèmes que vous pourriez éventuellement rencontrer, nous allons animer deux rectangles, `rc1` et `rc2`, par simple déplacement horizontal.

L'exemple de code suivant correspond à deux animations jouées en parallèle sur 3 secondes :

```
<Storyboard x:Name="stb" >
  <DoubleAnimation Storyboard.TargetName="rc1"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300" Duration="0:0:3" />
  <DoubleAnimation Storyboard.TargetName="rc2"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300" Duration="0:0:3" />
</Storyboard>
```

Le code suivant correspond à deux animations jouées en décalage (la seconde démarre une seconde et demie après la première) mais se terminant en même temps :

```
<Storyboard x:Name="stb" >
  <DoubleAnimation Storyboard.TargetName="rc1"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300" Duration="0:0:3" />
  <DoubleAnimation Storyboard.TargetName="rc2"
    Storyboard.TargetProperty="(Canvas.Left)"
    BeginTime="0:0:1.5"
    From="1" To="300" Duration="0:0:1.5" />
</Storyboard>
```

L'exemple suivant permet de répéter l'animation. Le rectangle se déplace vers la droite en 3 secondes et revient à sa position initiale, également en 3 secondes, et ceci indéfiniment :

```
<Storyboard x:Name="stb" >
  <DoubleAnimation Storyboard.TargetName="rc1"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300" Duration="0:0:3"
    AutoReverse="True"
    RepeatBehavior="Forever" />
</Storyboard>
```

Le code suivant permet de spécifier une durée dans Storyboard (l'animation s'arrête au bout de 10 secondes) :

```
<Storyboard x:Name="stb" Duration="0:0:10" >
  <DoubleAnimation Storyboard.TargetName="rc1"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300" Duration="0:0:3"
    AutoReverse="True"
    RepeatBehavior="Forever" />
</Storyboard>
```

Enfin, le code suivant permet de spécifier une durée et des répétitions dans Storyboard (le rectangle est déplacé – vers la droite puis vers la gauche – durant 10 secondes, l'animation s'arrête ensuite pendant 4 secondes ; le cycle se répète ainsi 9 fois encore) :

```
<Storyboard x:Name="stb"
  Duration="0:0:10" RepeatBehavior="10x" >
  <DoubleAnimation Storyboard.TargetName="rc1"
    Storyboard.TargetProperty="(Canvas.Left)"
    From="1" To="300"
    Duration="0:0:3" AutoReverse="True" />
</Storyboard>
```

Les animations par key frames

Jusqu'ici, pour chaque animation, nous donnions une position de départ (attribut optionnel *From*), une position d'arrivée (attribut *To*) et une durée. Avec les animations par *key frames*, il est possible de spécifier les différentes étapes par lesquelles doit passer l'animation.

L'exemple de code suivant permet de faire varier la largeur d'un rectangle rouge :

```
<Rectangle x:Name="rc" Width="200" Height="100" Fill="Red" >
  <Rectangle.Resources>
    <Storyboard x:Name="stb" >
      <DoubleAnimationUsingKeyFrames Storyboard.TargetName="rc"
                                   Storyboard.TargetProperty="Width" >
        <LinearDoubleKeyFrame KeyTime="0:0:0" Value="100" />
        <LinearDoubleKeyFrame KeyTime="0:0:3" Value="400" />
        <LinearDoubleKeyFrame KeyTime="0:0:5" Value="50" />
      </DoubleAnimationUsingKeyFrames>
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

Après l'exécution de `stb.Begin()`, l'animation passe par les étapes suivantes :

- la largeur du rectangle est de 100 pixels au temps 0 (démarrage de l'animation) ;
- elle passe progressivement à 400 pixels en 3 secondes ;
- elle revient à 50 pixels en 2 secondes.

Avec une animation de type `LinearDoubleKeyFrame`, les mouvements ne sont pas accélérés mais continus. Ainsi, lorsque la largeur augmente progressivement de 100 à 400 pixels en 3 secondes : elle passe à 200 au bout d'une seconde, à 300 au bout de 2 secondes et finalement à 400 au bout de 3 secondes.

L'animation pourrait être également de type `DiscreteDoubleKeyFrame`. Pour expliquer la différence entre ces deux types, transformons l'animation précédente selon le code suivant :

```
<Rectangle x:Name="rc" Width="200" Height="100" Fill="Red" >
  <Rectangle.Resources>
    <Storyboard x:Name="stb" >
      <DoubleAnimationUsingKeyFrames Storyboard.TargetName="rc"
                                   Storyboard.TargetProperty="Width" >
        <DiscreteDoubleKeyFrame KeyTime="0:0:0" Value="100" />
        <DiscreteDoubleKeyFrame KeyTime="0:0:3" Value="400" />
        <DiscreteDoubleKeyFrame KeyTime="0:0:5" Value="50" />
      </DoubleAnimationUsingKeyFrames>
    </Storyboard>
  </Rectangle.Resources>
</Rectangle>
```

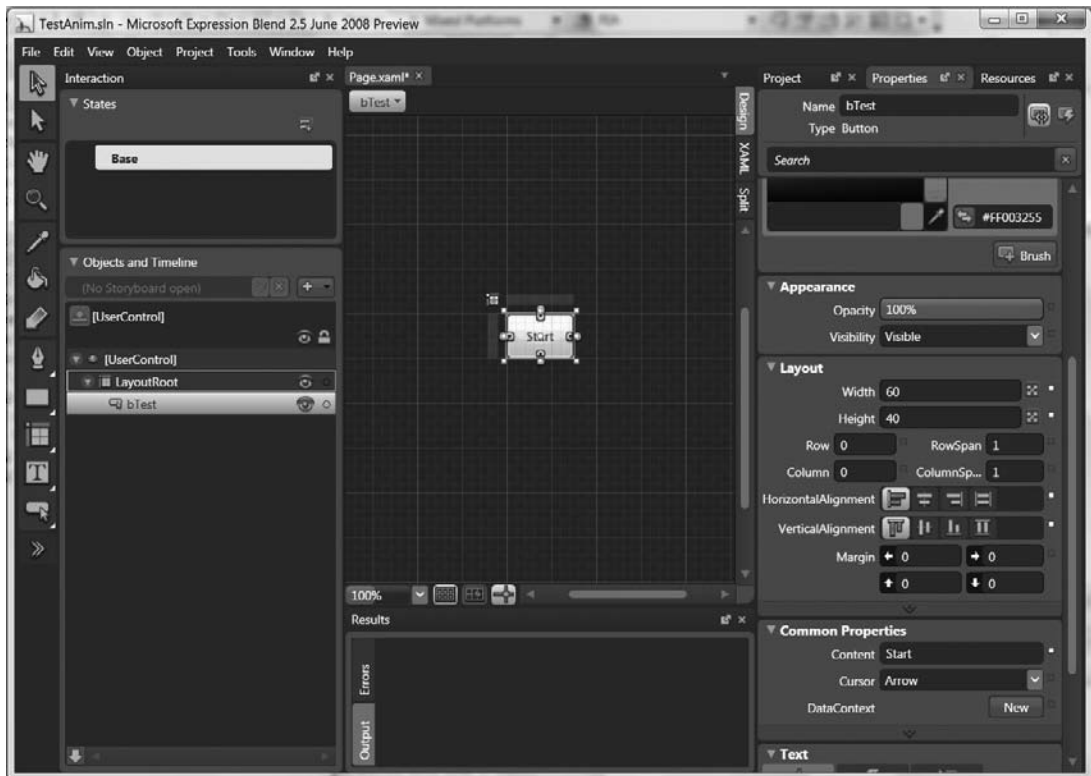
Au début de l'animation, la largeur du rectangle était de 100 pixels. Au bout de 3 secondes, elle passe brusquement à 400 et, 2 secondes plus tard, tout aussi brusquement à 50.

Avec les animations de type `SplineDoubleKeyFrame`, il est possible d'accélérer ou de ralentir à l'approche des étapes. Voyons cela avec Expression Blend.

Les animations avec Expression Blend

Prenons l'exemple suivant (figure 9-10) : un bouton Start est placé dans le coin supérieur gauche de la grille. Un clic sur ce bouton lancera l'animation qui consistera en un déplacement de balle (décrivant une ellipse). Depuis Visual Studio, passez à Expression Blend en cliquant droit sur le nom du fichier XAML>Ouvrir dans Expression Blend.

Figure 9-10



L'espace de travail étant assez restreint, réorganisez les différents panneaux au moyen de la touche F6 ou du menu Window>Active WorkSpace (figure 9-11).

Vous allez à présent animer une balle. Pour cela, il convient tout d'abord de la créer. Si l'outil Ellipse n'apparaît pas dans la boîte d'outils, cliquez sur l'outil Rectangle jusqu'à l'apparition du panneau Rectangle, ellipse et ligne. Sélectionnez alors l'outil Ellipse (il apparaît désormais dans la boîte d'outils) et cliquez sur la surface de travail. Positionnez correctement l'ellipse et donnez-lui la taille adéquate (appuyer sur la touche Maj tout en déplaçant le curseur de la souris pour obtenir un cercle). Spécifiez ensuite un fond rouge pour la balle (figure 9-12). De la routine déjà...

Figure 9-11

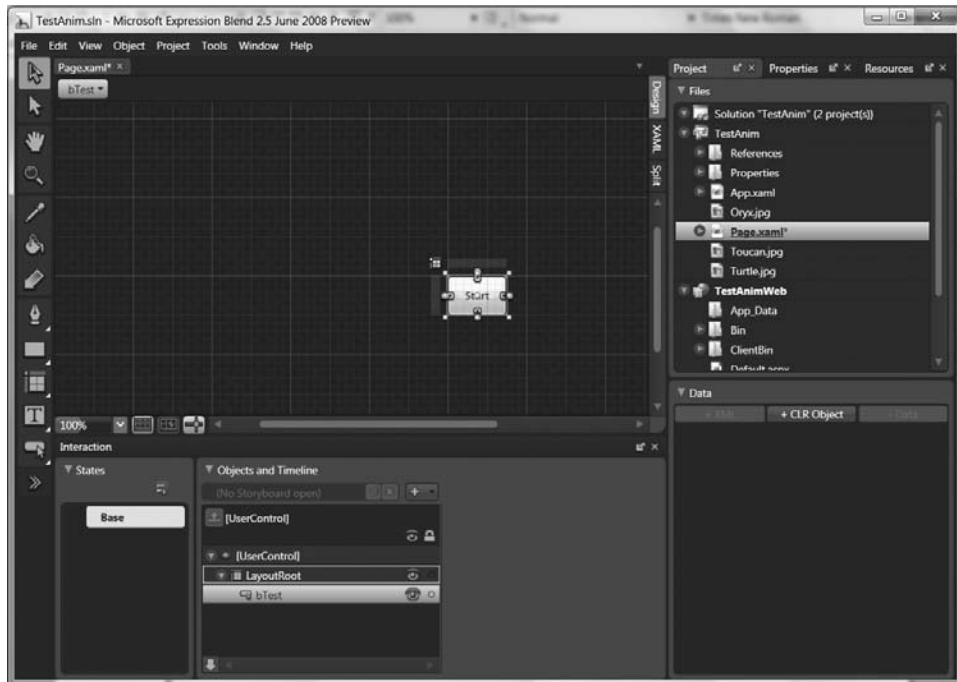
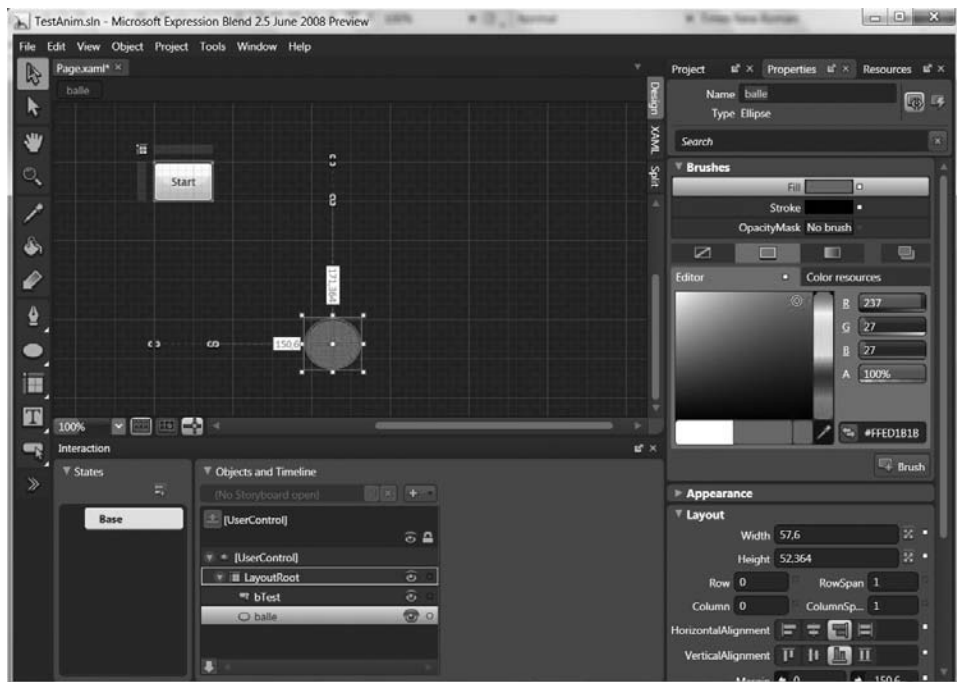
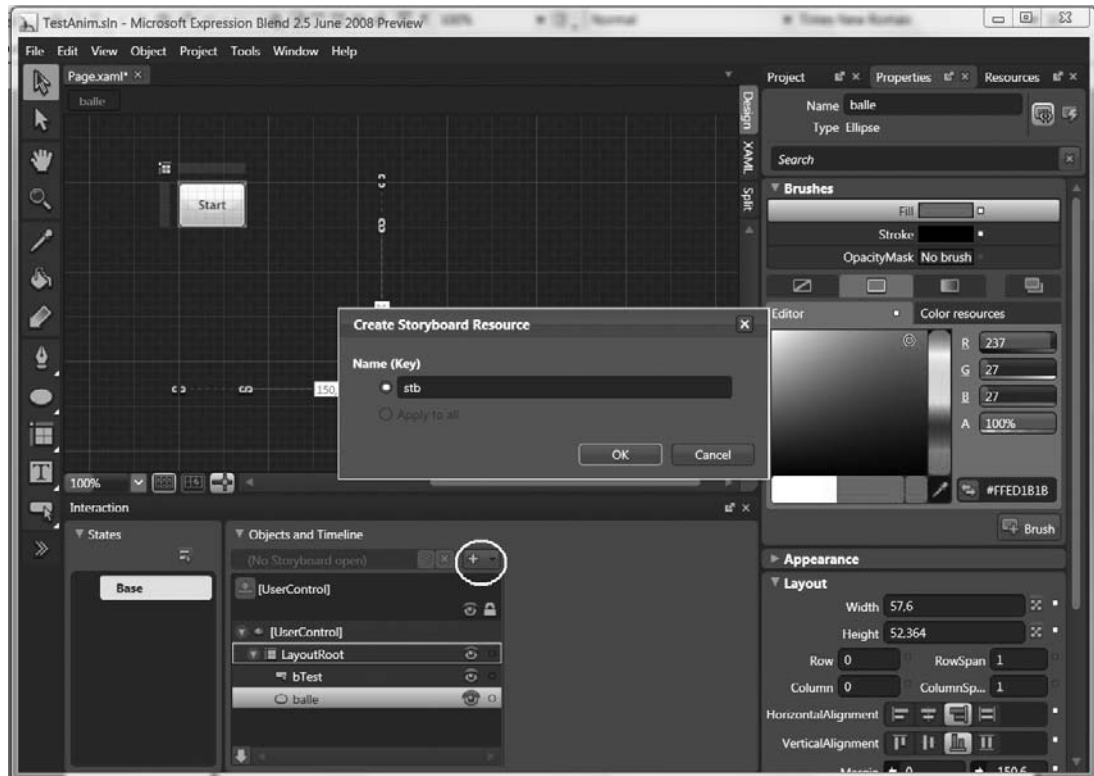


Figure 9-12



Vous allez à présent créer un story-board. Pour cela, cliquez sur le signe + du panneau Objects and Timeline (figure 9-13).

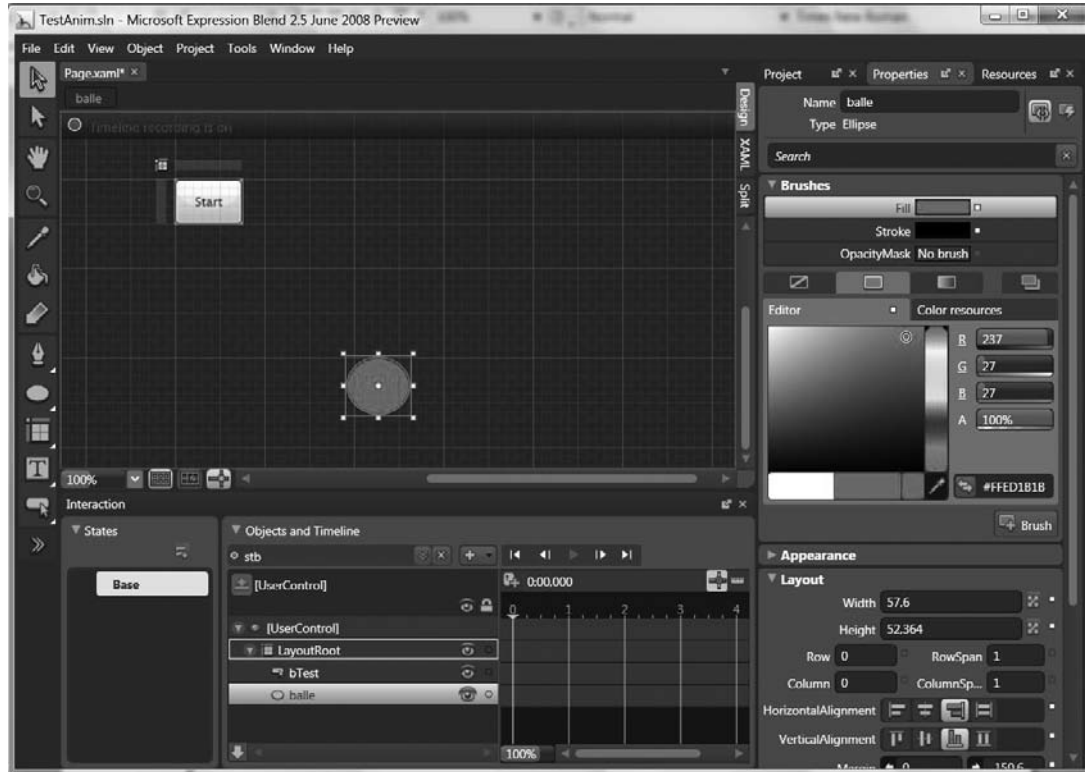
Figure 9-13



Expression Blend vous demande alors de spécifier un nom (attribut `x:Name`) pour le story-board (ici, `stb`).

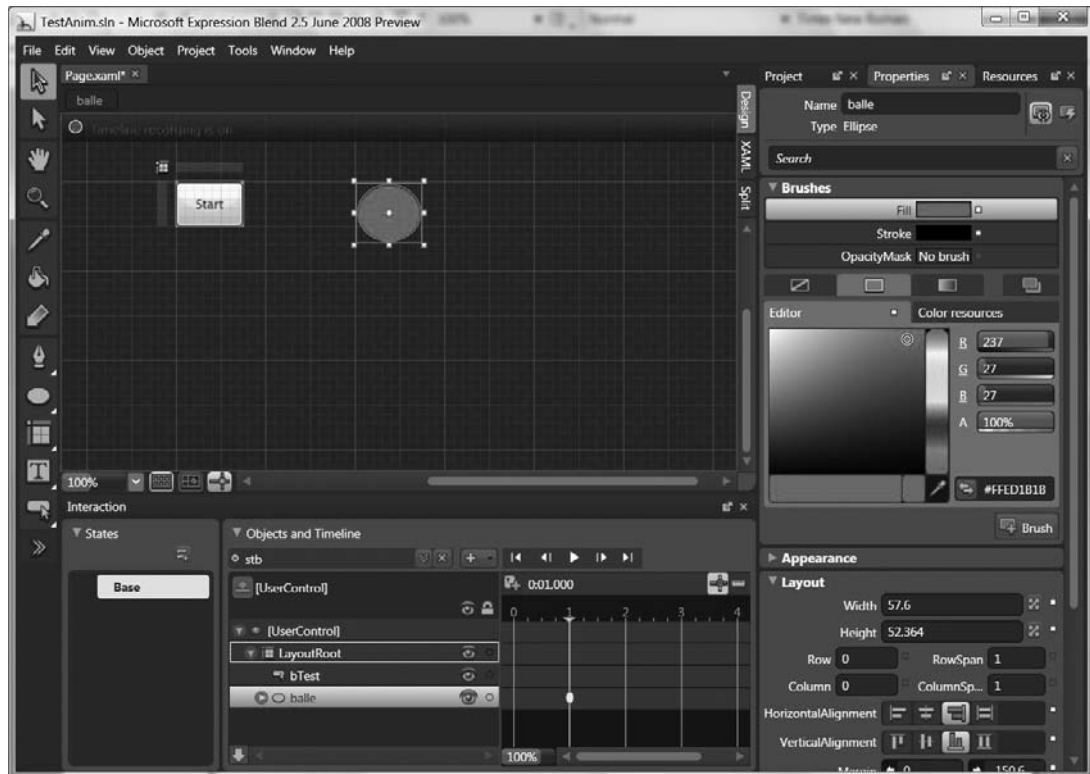
Le panneau Objects and Timeline change alors complètement d'aspect et comporte à droite des lignes verticales, dites de temps (figure 9-14). Par défaut, les lignes de temps sont placées à une, deux, trois, etc., secondes mais rien ne vous empêche d'en ajouter d'autres, à intervalles plus rapprochés.

Figure 9-14



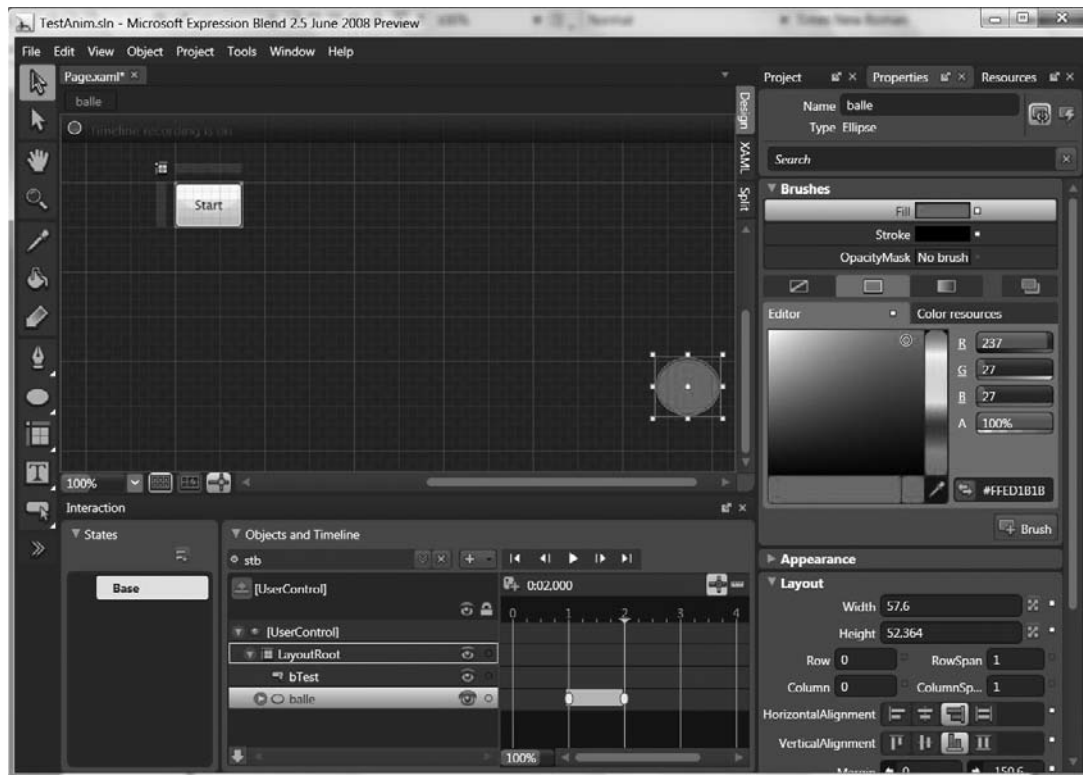
Cliquez sur la ligne de temps placée à une seconde. Un petit cercle rouge apparaît alors à gauche du nom interne de l'animation (ici, stb) dans le panneau Objects and Timeline ; il indique qu'Expression Blend est prêt à enregistrer vos opérations (animation de la première seconde). Déplacez la balle rouge en haut de la grille, puis cliquez sur la ligne de temps placée à deux secondes (figure 9-15).

Figure 9-15



Déplacez ensuite la balle vers un autre point (ici, plus bas et à droite) pour spécifier l'animation à la deuxième seconde (figure 9-16).

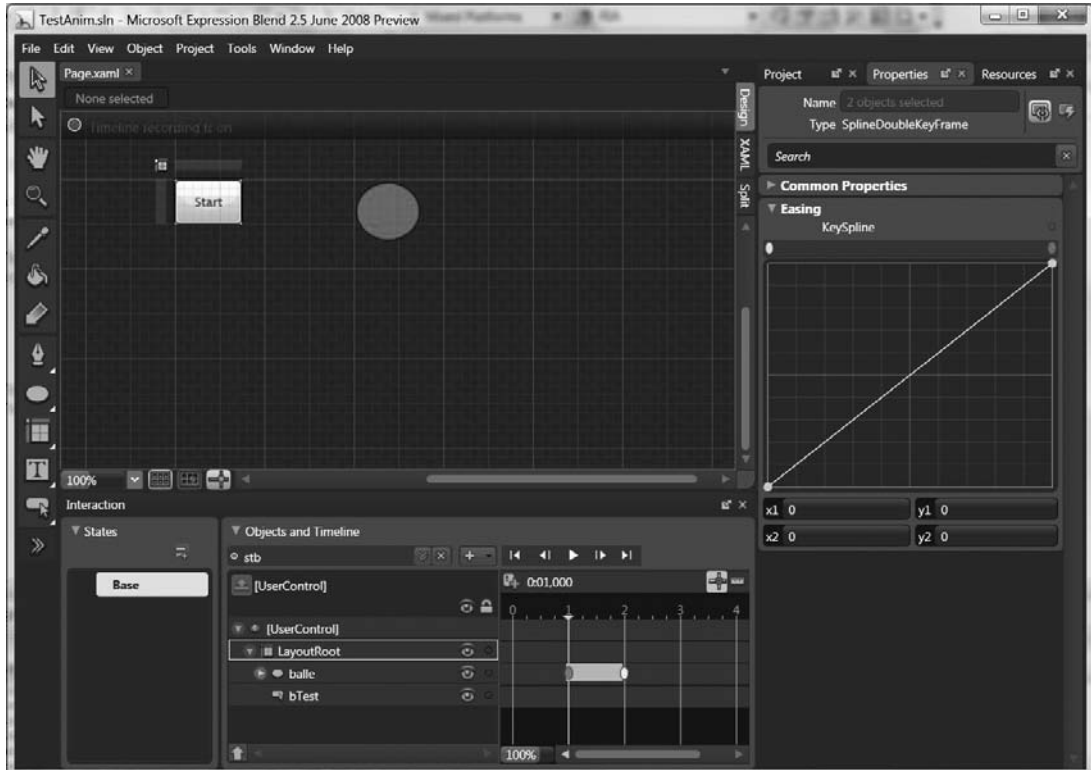
Figure 9-16



Il est déjà possible de visualiser l'animation en cliquant sur la ligne de temps placée à zéro seconde (ou à n'importe quel moment de l'animation), puis sur le bouton Play (symbolisé par un triangle pointant vers la droite). La balle monte dans la première seconde puis descend en oblique vers la droite dans la deuxième seconde.

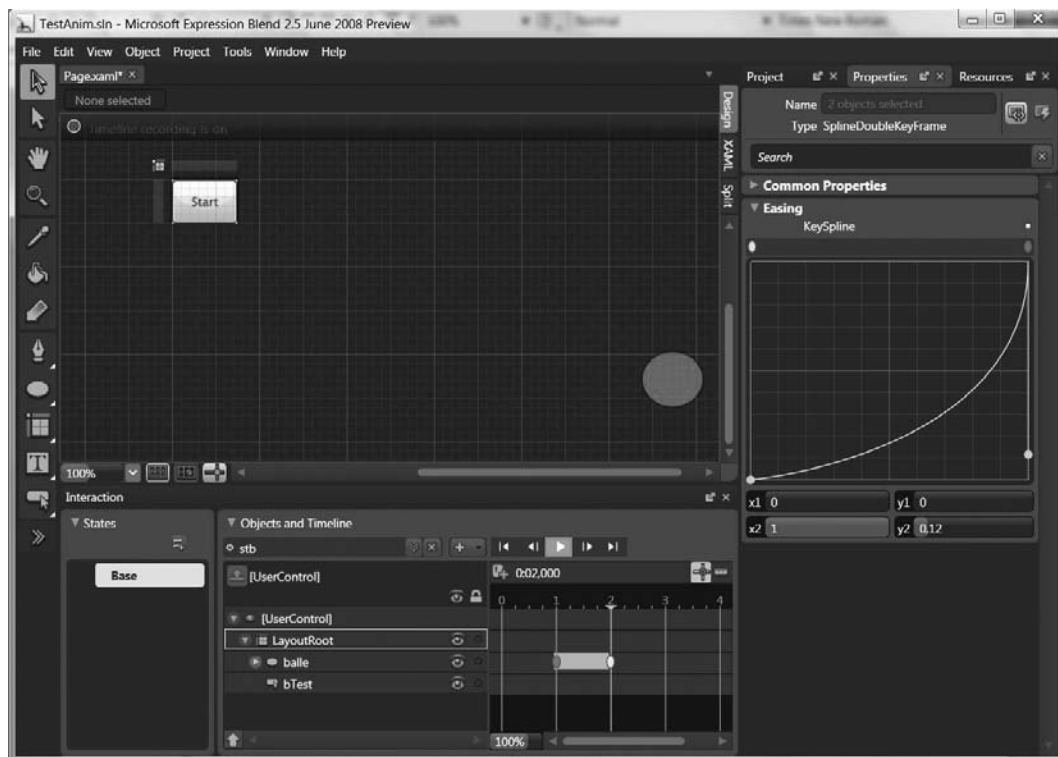
Pour le moment, le mouvement est continu et régulier (comme pour l'animation de type `LinearDoubleKeyFrame`) mais il est possible de forcer des accélérations ou des ralentis aux approches. Pour cela, cliquez au centre d'une ligne de temps pour faire apparaître le panneau Easing (figure 9-17).

Figure 9-17



Dans ce panneau, cliquez sur la courbe (allant du coin inférieur gauche au coin supérieur droit) pour la déformer. La figure 9-18 présente les modifications effectuées pour que la balle parte lentement mais accélère à l'approche du point d'arrivée.

Figure 9-18

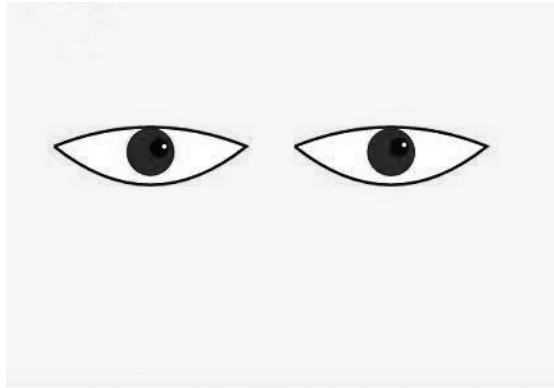


Programmes d'accompagnement

Exemple 1 :

Les yeux suivent les mouvements de la souris (figure 9-19).

Figure 9-19



Exemple 2 :

Une loupe est déplacée au moyen de la souris et permet d'afficher (en agrandissant cinq fois) plus de détails sur une partie de l'image de fond (figure 9-20).

Figure 9-20



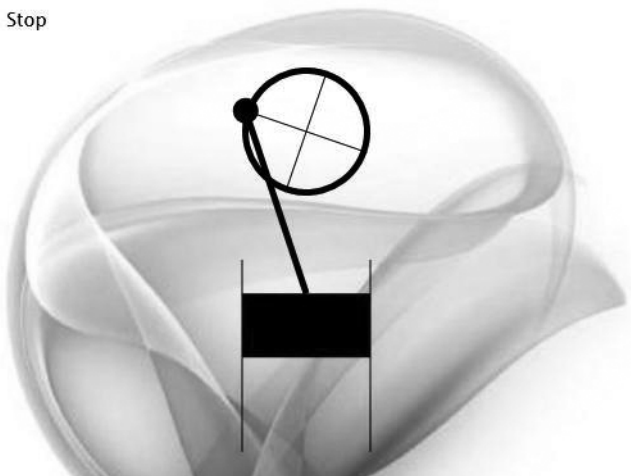
Exemple 3 :

Animation simulant le mouvement d'un piston (figure 9-21).

Figure 9-21

Piston en action

Stop

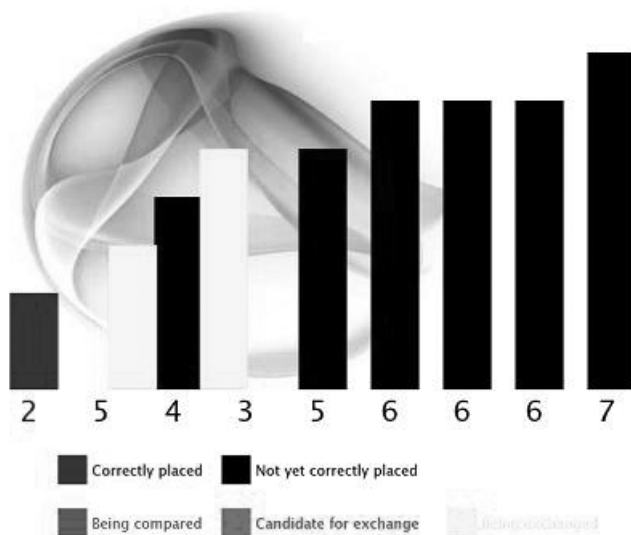
**Exemple 4 :**

Animation montrant la progression d'un tri (figure 9-22).

Figure 9-22

Generate numbers

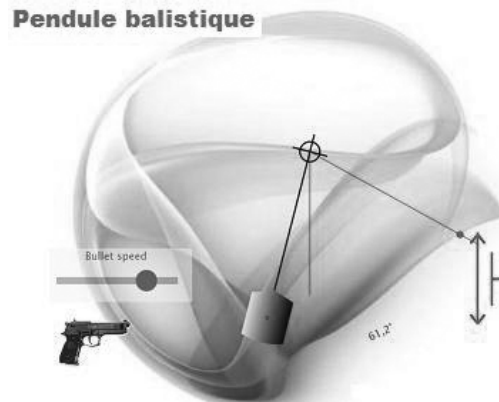
Sort



Exemple 5 :

Animation montrant le mouvement d'un pendule balistique en fonction de la vitesse de la balle (figure 9-23).

Figure 9-23



10

Les liaisons de données

Dans ce chapitre, nous allons nous intéresser à la liaison de données, c'est-à-dire à la connexion automatique entre une source de données et un composant d'affichage, appelé cible (*target* en anglais).

Nous présenterons les concepts de base avec une zone d'édition `TextBox` et une zone d'affichage `TextBlock`. Nous passerons ensuite à la boîte de liste et finalement à la grille de données avec communication bidirectionnelle entre la grille et la source de données (toute modification dans la grille étant répercutée dans la source de données).

Tout au long de ce chapitre, les données proviendront de sources de données déjà en mémoire dans l'application Silverlight. Au chapitre 13, nous verrons comment obtenir des données en provenance de sources de données extérieures, notamment de services Web.

Les liaisons de données avec `TextBlock` et `TextBox`

Comme nous l'avons vu précédemment, le contenu des balises `TextBlock` et `TextBox` est spécifié dans l'attribut `Text`. Dans l'exemple de code suivant, (le contenu de la zone d'édition est initialisé lors du chargement de la page) :

```
<TextBox x:Name="ze" Text="Victor Hugo" ..... />
```

Nous allons faire en sorte que le libellé ne soit plus codé « en dur » dans la balise mais qu'il provienne d'une source de données. Nous montrerons d'abord comment effectuer une liaison simple (qui n'est réalisée qu'une seule fois), puis une liaison suivie (le contenu de la zone affichée change si la source de données est modifiée) et finalement une liaison bidirectionnelle (toute modification de la zone d'édition étant alors automatiquement répercutée dans la source de données).

Partons d'un exemple très simple, que nous allons progressivement compliquer. Dans la classe `Page` de l'application Silverlight, il convient tout d'abord de définir une propriété en lecture seule :

```
public string Nom { get{return "Emile Zola";}}
```

Cela fait, la zone d'édition `ze` doit ensuite être liée à cette source de données. Pour cela, la balise `TextBox` doit devenir :

```
<TextBox x:Name="ze" Text="{Binding Nom}" ..... />
```

Mais cela ne suffit pas. Il faut encore indiquer à la zone d'édition quel est son « contexte de données », autrement dit l'objet contenant la propriété `Nom`. Ici, il s'agit de l'objet « page de l'application Silverlight ». Dans le fichier `Page.xaml.cs`, le mot réservé `this` fait référence à cet objet (en VB, le mot réservé `Me` fait référence à cet objet dans le fichier `Page.xaml.vb`) :

```
ze.DataContext = this;
```

Cette instruction est généralement exécutée dans la fonction qui traite l'événement `Loaded` adressé au conteneur mais cette liaison de pourrait être effectuée après, à n'importe quel moment.

Dans un cas aussi simple, nous avons surtout compliqué les choses ! Rapprochons-nous progressivement des cas réels et, surtout, travaillons plus proprement en séparant les données de leur présentation, ce qui est d'ailleurs le but principal du mécanisme de liaison de données. La présentation des données sera ainsi traitée distinctement des opérations effectuées sur celles-ci.

Les données proviendront quasiment toujours d'un objet contenant diverses informations (il s'agira même souvent d'un service Web qui rapatrie un tel objet d'un serveur distant). Il convient donc de créer une classe `Produit` très simple puisque limitée à la propriété `Nom`. Pour cela, effectuez un clic droit sur le nom du projet (partie Silverlight) dans l'Explorateur de solutions et sélectionnez `Ajouter>Nouvel élément...>Classe`. Soit `Produit.cs` (ou `Produit.vb`) le fichier contenant cette classe :

```
public class Produit
{
    public string Nom {get; set;}
}
```

En C#, il s'agit de la forme simplifiée qui devait autrefois s'écrire comme suit (une propriété ressemble à un champ public et s'utilise comme tel mais les instructions dans le `get` sont automatiquement exécutées pour obtenir sa valeur tandis que les instructions dans le `set` sont automatiquement exécutées pour modifier son contenu) :

```
public class Produit
{
    private string nom;
    public string Nom
    {
        get {return nom;}
        set {nom=value;}
    }
}
```

Dans la fonction traitant l'événement `Loaded` du conteneur, par exemple, un produit est ensuite créé puis associé à la zone d'édition. Le code correspondant suivant est conforme à la nouvelle syntaxe, introduite en version 3, qui permet d'initialiser un objet par initialisation de ses propriétés et ainsi d'éviter de devoir créer un constructeur :

```
Produit prod; // variable de la page
.....
prod = new Produit() {Nom="Cirage"};
ze.DataContext = prod; // on spécifie le contexte de la zone d'édition
```

Cirage est maintenant affiché dans la zone d'édition au démarrage de l'application.

La liaison de données n'est ici effectuée qu'une seule fois, après l'assignation d'un `Produit` dans `ze.DataContext`. Il n'y a donc pas (pour le moment) de suivi entre la source de données et la présentation de cette donnée dans la fenêtre du navigateur. Une modification de l'objet `prod` n'a aucune répercussion sur la zone d'édition.

Pour qu'il y ait suivi de la liaison, il faut modifier la classe `Produit` et implémenter l'interface `INotifyPropertyChanged` (une interface indique les propriétés et fonctions qu'une classe doit implémenter). En mentionnant qu'une classe implémente une interface, on force le programmeur à écrire les fonctions spécifiées dans l'interface (en respectant la signature de ces fonctions). Ces fonctions, à leur tour, vont provoquer l'exécution de fonctions du système, qui, elles, provoquent l'effet désiré.

Dans la classe `Produit` modifiée, il faut :

- déclarer une variable d'un type bien déterminé, le type « événement » ;
- appeler la fonction associée à cette variable en cas de modification de la propriété.

Dans la pratique, pour que la classe `Produit` implémente l'interface `INotifyPropertyChanged`, il faut la modifier comme suit :

```
using System.ComponentModel;
.....
public class Produit : INotifyPropertyChanged
{
    private string nom;
    public event PropertyChangedEventHandler PropertyChanged;
    public string Nom
    {
        get {return nom;}
        set
        {
            nom = value;
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs("Nom"));
        }
    }
}
```


Analysons la classe `Produit` ainsi modifiée. Un objet `Produit` présente la propriété `Nom`. Le programme peut demander le `Nom` d'un `Produit` particulier. Ici, le `Nom` présenté au monde extérieur est simplement le contenu du champ privé `nom` car les opérations mentionnées dans le `get` se résument à un `return`. L'utilisateur d'un objet `Produit` ignore tout du fonctionnement interne de la classe `Produit` et ne « voit » que ses propriétés et fonctions publiques. Dans son fonctionnement interne, la classe `Produit` peut effectuer des opérations complexes (spécifiées dans le `get`) pour rendre le `Nom` du `Produit`.

Un `Produit` présente également un champ de type « événement » (*event* en anglais). Ce champ contient initialement la valeur `null` mais le run-time Silverlight vient y spécifier l'adresse d'une de ses fonctions au moment où la liaison est effectuée. Pour ceux qui ont pratiqué le langage C et ses pointeurs, `PropertyChanged` peut être assimilé à un pointeur sur une fonction, autrement dit une variable qui contient l'adresse d'une fonction. Il devient ainsi possible d'exécuter la fonction pointée via le pointeur. `PropertyChangedEventHandler` donne des informations quant à la signature de la fonction pointée.

Lorsque l'application Silverlight demande le `Nom` d'un `Produit`, les instructions du `get` sont exécutées et le code de la classe renvoie le contenu du champ privé `nom` (ce code exécute pour cela les instructions du `get`).

Lorsque cette application modifie le `Nom` d'un `Produit`, la nouvelle valeur (symbolisée par `value`) est copiée dans le champ privé `nom` (ici, c'est tout simple mais on pourrait trouver de nombreuses instructions dans la clause `set`). Mais ce n'est pas tout : si le champ de type « événement » qu'est `PropertyChanged` a été initialisé (il l'est par le run-time Silverlight, au moment de la liaison), il est différent de `null` et `PropertyChanged(...)` a pour effet d'exécuter la fonction pointée. C'est cette fonction qui vient mettre à jour la cible (ici, la zone d'édition `ze`).

Le mécanisme peut paraître intimidant en première lecture mais il s'avère d'une rare élégance et d'une grande efficacité. Grâce à lui, toute modification de la source de données (ici, le champ `Nom` d'un objet `Produit`) est désormais automatiquement répercutée dans la zone d'édition `ze`. Pour le vérifier, exécutez le code suivant :

```
prod.Nom = "Rhubarbe";
```

On peut à présent affirmer que la zone d'édition `ze` (la cible) et la source de données `prod` sont liées de manière permanente mais dans une seule direction (*OneWay*), soit de la source à la cible.

Il est possible d'empêcher ce suivi automatique. Il suffit pour cela de réclamer une liaison *OneTime* (qui n'est donc exécutée qu'une seule fois, lors de l'assignation du `DataContext`), ce qui se fait en modifiant la balise :

```
<TextBox x:Name="ze" Text="{Binding Nom, Mode=OneTime}" ..... />
```

Pour que la liaison devienne bidirectionnelle, autrement dit pour que toute modification de la zone d'édition soit répercutée dans la source de données, il suffit d'imposer le mode *TwoWay* :

```
<TextBox x:Name="ze" Text="{Binding Nom, Mode=TwoWay}" ..... />
```

Si l'on dispose d'une liste de produits, il suffit de modifier le DataContext de la zone d'édition et de le faire correspondre à un produit particulier dans la liste pour modifier automatiquement la liaison. En fait, il suffit de modifier le DataContext du conteneur (généralement une grille) pour modifier automatiquement le DataContext de tous les éléments UI enfants, ce qui simplifie encore les choses.

En VB, la classe et la fonction traitant l'événement Loaded s'écrivent :

```
Imports System.ComponentModel
Public Class Produit
    Implements INotifyPropertyChanged
    Private mNom As String
    Public Property Nom() As String
        Get
            Return mNom
        End Get
        Set(ByVal value As String)
            mNom = value
            RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs("Nom"))
        End Set
    End Property
    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged
End Class
.....
Private prod As Produit
Private Sub LayoutRoot_Loaded(ByVal sender As System.Object, _
                                ByVal e As System.Windows.RoutedEventArgs)
    prod = New Produit() With {.Nom = "Cirage"}
    ze.DataContext = prod
End Sub
```

Les boîtes de liste

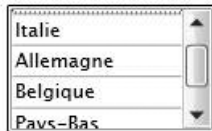
Les propriétés des boîtes de liste

Une boîte de liste permet d'afficher différents articles sur une seule colonne (voir figure 10-1), parmi lesquels l'utilisateur peut en sélectionner un. Dans sa forme la plus simple, une boîte de liste et ses données (ici, des chaînes de caractères) sont spécifiées comme suit :

```
<ListBox ..... >
<ListBoxItem Content="France" />
<ListBoxItem Content="Italie" />
<ListBoxItem Content="Allemagne" />
<ListBoxItem Content="Belgique" />
<ListBoxItem Content="Pays-Bas" />
<ListBoxItem Content="Luxembourg" />
</ListBox>
```

où doit être remplacé par des attributs tels que Width, Height, Margin, Canvas.Left, Grid.Row, etc.).

Figure 10-1



Nous ne reviendrons pas sur les attributs de taille (Width, Height, MinWidth, MaxWidth, MinHeight et MaxHeight), de positionnement (Margin, Padding, HorizontalAlignment, VerticalAlignment, Canvas.Left et Canvas.Right si le conteneur est un canevas) ou de présentation (Opacity, Style, Cursor, etc.), propriétés qui sont maintenant bien connues ou seront bientôt étudiées (dans le cas de Style).

Si la boîte de liste est un objet `ListBox` (classe dérivée de `UIElement`), les articles eux-mêmes sont des objets de la classe `ListBoxItem`, elle aussi dérivée de `UIElement`. C'est pour cette raison que les articles peuvent prendre n'importe quelle forme visuelle et ne sont pas du tout limités à des libellés sous forme de chaînes de caractères.

Dans les balises `ListBoxItem` (pour chaque article donc), il est possible de spécifier des attributs comme `Background` et `Foreground` ainsi que ceux de la famille `Font`. Les articles peuvent être des objets tels que des boutons ou des images et peuvent même être tout à fait différents d'un article à l'autre.

L'exemple de code suivant correspondant à la boîte de liste représentée à la figure 10-2 :

```
<ListBox ..... >
  <TextBlock Text="Italie" />
  <Image Source="France.jpg" Margin="0,5"/>
  <Button Content="Allemagne" />
</ListBox>
```

Figure 10-2



Le premier article est ici un libellé, le deuxième une image et le troisième un bouton.

Dans la mesure où les libellés (par défaut) proviennent de balises `Content`, il est aisé de modifier fondamentalement l'apparence des articles (voir chapitre 15).

Voyons maintenant comment remplir dynamiquement la boîte de liste et comment détecter des sélections, avant de passer à la personnalisation plus avancée de la boîte de liste.

Remplir et modifier le contenu d'une boîte de liste

Des articles peuvent être ajoutés ou supprimés en cours d'exécution de programme. Par exemple (sans oublier que l'argument pourrait être d'un autre type, comme une image ou un bouton) :

```
lbPays.Items.Add("Danemark");
```

Le tableau 10-1 présente les fonctions pouvant être appliquées à la propriété `Items`, qui représente une collection (de plusieurs `object`, donc de n'importe quel type).

Tableau 10-1 – Les opérations susceptibles d'être effectuées sur une collection

Nom de la fonction	Description
<code>Add(object)</code>	Ajoute un article en fin de liste. L'argument est bien de type <code>object</code> , ce qui permet d'insérer n'importe quel objet.
<code>Clear()</code>	Vide la collection.
<code>Count</code>	Propriété qui renvoie le nombre d'articles présents dans la boîte de liste.
<code>Insert(int pos, object)</code>	Insère un article à une position donnée par <code>pos</code> (0 pour la tête de la boîte de liste).
<code>Remove(object)</code>	Supprime l'objet passé en argument.
<code>RemoveAt(int)</code>	Supprime l'article occupant la position passée en argument.

Lors d'ajouts, même de chaînes de caractères, les articles ne sont pas triés. Si une présentation ordonnée s'avère nécessaire, il vous appartient d'effectuer le tri avant les ajouts. Par exemple (même s'il aurait été ici plus simple et plus efficace d'utiliser la propriété `ItemsSource`) :

```
string[] ts = {"Portugal", "Suède", "Autriche"};
.....
Array.Sort(ts);
for (int i=0; i<ts.Length; i++) lbPays.Items.Add(ts[i]);
```

Ce qui s'écrit en VB :

```
Dim ts() As String = {"Portugal", "Suède", "Autriche"}
.....
Array.Sort(ts)
For i As Integer = 0 To ts.Length - 1
    lbPays.Items.Add(ts(i))
Next i
```

Des articles déjà plus élaborés (ici, avec une couleur d’affichage) peuvent être insérés de cette manière :

```
ListBoxItem liBleu = new ListBoxItem();
liBleu.Content = "Bleu"; liBleu.Foreground = new SolidColorBrush(Colors.Blue);
lb.Items.Add(liBleu);

ListBoxItem liRouge = new ListBoxItem();
liRouge.Content = "Rouge";
liRouge.Foreground = new SolidColorBrush(Colors.Red);
lb.Items.Add(liRouge);
```

Au moment de l’ajout, la boîte de liste garde une référence à l’article passé en argument à Add. Toute modification de l’article (par exemple, liRouge), indépendamment de la boîte de liste, a donc une répercussion sur celui-ci. Il est ainsi possible de modifier le libellé de liRouge ou sa couleur d’affichage, même après insertion dans la boîte de liste.

Les articles peuvent provenir directement d’une collection. Pour cela, il suffit d’utiliser la propriété ItemsSource, qui correspond à une collection de données (tableau, liste générique, etc.) :

```
lbPays.ItemsSource = ts;
```

Retrouver l’article sélectionné

L’événement SelectionChanged est signalé quand l’utilisateur sélectionne un article. Une sélection est généralement opérée par un simple clic mais elle peut aussi l’être au moyen des touches de direction du clavier lorsque la boîte de liste a le focus d’entrée (dans ce cas, l’événement est signalé de manière répétitive, pour des articles différents, tant que l’utilisateur garde une touche de direction enfoncée) :

```
<ListBox x:Name="lbPays" SelectionChanged="lbPays_SelectionChanged" ..... >
.....
</ListBox>
```

En C#, la fonction de traitement s’écrit :

```
private void lbPays_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    .....
}
```

et en VB :

```
Private Sub lbPays_SelectionChanged(ByVal sender As System.Object, _
    ByVal e As System.Windows.Controls.SelectionChangedEventArgs)
    .....
End Sub
```

On pourrait s’attendre à ce que l’argument e, de type SelectionChangedEventArgs, donne accès à l’article sélectionné. Ce sont en fait des propriétés de la boîte de liste qui vont fournir ces informations.

La propriété `SelectedIndex` d'une boîte de liste contient le numéro d'ordre de l'article sélectionné (0 pour le premier et `-1` si aucun article n'est sélectionné) :

```
■ n = lbPays.SelectedIndex;
```

Si la boîte de liste ne contient que des libellés, on retrouve le libellé de l'article sélectionné en écrivant le code C# suivant :

```
■ ListBoxItem lbi = lbPays.Items[lbPays.SelectedIndex] as ListBoxItem;  
sPays = lbi.Content.ToString();
```

ou

```
■ ListBoxItem lbi = lbPays.SelectedItem as ListBoxItem;  
sPays = lbi.Content.ToString();
```

ce qui donne en VB :

```
■ Dim lbi As ListBoxItem = lbPays.Items(lb.SelectedIndex)  
sPays = lbi.Content
```

ou

```
■ Dim lbi As ListBoxItem = lbPays.SelectedItem  
sPays = lbi.Content
```

où `lbi` prend la valeur `null` (Nothing en VB) si aucun article n'est sélectionné.

Cas des données provenant d'une liste d'objets

Avec `ItemsSource`, les données peuvent provenir d'une collection de données. Nous allons à présent compliquer la source de données en créant une classe ainsi qu'une liste d'objets. Le code correspondant en C# s'écrit :

```
■ public class Pers  
{  
    public string Nom { get; set; }  
    public string Prénom {get; set;}  
    public int AN { get; set; }  
}  
.....  
Pers[] tabPers = { new Pers(){Nom = "Hugo", Prénom="Victor", AN = 1802},  
                  new Pers(){Nom = "Picasso", Prénom = "Pablo", AN = 1881},  
                  new Pers(){Nom = "Ravel", Prénom = "Maurice", AN = 1875},  
                  };
```

En VB, la déclaration de la classe est nettement plus verbeuse (ce l'était presque tout autant en C# avant l'introduction de la syntaxe très raccourcie des propriétés) :

```
■ Public Class Pers  
    Private mNom As String  
    Private mPrénom As String  
    Private mAN As Integer
```

```

Property Nom() As String
    Get
        Return mNom
    End Get
    Set(ByVal value As String)
        mNom = value
    End Set
End Property

Property Prénom() As String
    Get
        Return mPrénom
    End Get
    Set(ByVal value As String)
        mPrénom = value
    End Set
End Property

Property AN() As Integer
    Get
        Return mAN
    End Get
    Set(ByVal value As Integer)
        mAN = value
    End Set
End Property
End Class

```

Tandis que l'initialisation du tableau s'écrit :

```

Private tabPers() As Pers = _
{
    New Pers() {Nom="Hugo", Prénom="Victor", AN=1802}, _
    New Pers() {Nom="Picasso", Prénom="Pablo", AN=1881}, _
    New Pers() {Nom="Ravel", Prénom="Maurice", AN=1875}
}

```

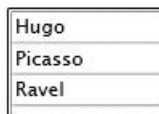
Nous allons afficher les noms dans la boîte de liste `lbNoms` (figure 10-3). Pour cela, il faut indiquer à la boîte de liste quel champ de `Pers` doit être retenu pour l'affichage, ce qui est indiqué dans la propriété `DisplayMemberPath` de la boîte de liste :

```

lbNoms.DisplayMemberPath = "Nom";
lbNoms.ItemsSource = tabPers;

```

Figure 10-3



Aucune barre de défilement n'est ici affichée à droite de la boîte de liste car celle-ci ne contient pas suffisamment d'articles.

L'attribut `DisplayMemberPath` est ici initialisé dynamiquement (par programme donc) mais aurait pu être spécifié dans le XAML, en attribut de la balise `ListBox`.

En l'absence de propriété `DisplayMemberPath` initialisée, on retrouve le nom de la classe dans tous les articles de la boîte de liste. Ceci est tout à fait normal car, en l'absence de cette information, le run-time Silverlight applique la fonction `ToString()` à chaque article. Or, par défaut, la fonction `ToString()` d'une classe (que l'on finit par trouver loin dans la hiérarchie, souvent au niveau `object`) renvoie le nom de la classe. Il suffit donc de redéfinir la fonction `ToString()` dans la classe `Pers` pour modifier ce comportement (voir figure 10-4), ce qui s'écrit en C# :

```
public class Pers
{
    ....
    public override string ToString()
    {
        return Prénom + " " + Nom + " (" + AN + ")";
    }
}
```

Figure 10-4

Victor Hugo (1802)
Pablo Picasso (1881)
Maurice Ravel (1875)

L'équivalent en VB de cette fonction `ToString` redéfinie est :

```
Public Overrides Function ToString() As String
    Return Prénom & " " & Nom & " (" & AN & ")"
End Function
```

La personnalisation des boîtes de liste

Avec les *templates*, il devient possible de modifier fondamentalement la présentation de la boîte de liste (voir la section « Les templates » du chapitre 15).

Dans la balise `ListBox.ItemTemplate`, il convient d'indiquer comment chaque article doit être affiché. Dans le code suivant, chaque article est constitué d'un `StackPanel` à orientation horizontale et composé de trois compartiments (limités chacun à une chaîne de caractères, dont une d'un seul caractère) :

```
<ListBox x:Name="lbNoms" ..... >
<ListBox.ItemTemplate>
<DataTemplate>
<StackPanel Orientation="Horizontal" >
<TextBlock Text="{Binding Prénom}" />
```



```

        <TextBlock Text=" " />
        <TextBlock Text="{Binding Nom}" />
    </StackPanel>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>

```

La liaison entre la boîte de liste et le tableau de Pers est effectuée via la propriété `ItemsSource`. La boîte de liste contient donc autant d'articles que d'objets Pers dans le tableau.

Nous allons maintenant ajouter une image. Pour cela, supposons que la classe Pays contienne deux champs, à savoir Image (nom de l'image du pays) et Nom (nom du pays) :

```

public class Pays
{
    public string Image {get; set; }
    public string Nom {get; set;}
}
.....
List<Pays> tabPays;
.....
tabPays = new List<Pays>();
tabPays.Add(new Pays() { Image = "France.jpg", Nom = "France" });
.....
lbPays.ItemsSource = tabPays;

```

Ce qui s'écrit en VB :

```

Public Class Pays
    Private mImage As String
    Public Property Image() As String
    Get
        Return mImage
    End Get
    Set(ByVal value As String)
        mImage = value
    End Set
End Property
    Private mNom As String
    Public Property Nom() As String
    Get
        Return mNom
    End Get
    Set(ByVal value As String)
        mNom = value
    End Set
End Property
End Class
.....
Private tabPays As List(Of Pays)
.....
tabPays = New List(Of Pays)()

```

```
tabPays.Add(New Pays() With {.Image = "France.jpg", .Nom = "France"})  
.....  
lbPays.ItemsSource = tabPays
```

Passons maintenant au XAML qui permet de décrire la présentation des données. Pour chaque article, nous affichons l'image du pays et son nom (figure 10-5) :

```
<ListBox x:Name="lbPays" ..... >  
  <ListBox.ItemTemplate>  
    <DataTemplate>  
      <StackPanel Orientation="Horizontal" Margin="10" >  
        <Image Source="{Binding Image}" />  
        <TextBlock Text="{Binding Nom}"  
          VerticalAlignment="Center" Margin="10"/>  
      </StackPanel>  
    </DataTemplate>  
  </ListBox.ItemTemplate>  
</ListBox>
```

Figure 10-5



La grille de données

Présentation générale

La grille de données constitue l'un des composants les plus utiles et les plus prisés. Les données sont présentées dans un tableau, à la manière d'un tableur. Même si la grille est prévue pour vous permettre de l'améliorer et de l'adapter à vos besoins, elle n'égale cependant pas un logiciel aussi abouti que Microsoft Excel. On regrettera notamment (par rapport à la grille d'ASP.NET) l'absence de pagination, ce qui permet de représenter les données en pages plutôt qu'en une longue (et parfois interminable) suite d'articles.

Dans la mesure où la grille de données (composant `DataGrid`) n'est pas un composant « léger » et que les programmes Silverlight n'en font pas tous l'usage, son code n'est inclus ni dans le run-time Silverlight ni dans la DLL de code automatiquement greffée au fichier XAP de l'application (cette dernière contenant le code compilé de l'application).

Le code de la grille de données se trouve dans `System.Windows.Controls.Data`, qu'il s'agit de charger séparément dans le fichier XAP. Cette opération est heureusement effectuée

automatiquement dès que l'on insère le composant `DataGrid` dans une page Silverlight. En effet, lorsque vous ajoutez un composant `DataGrid` par glisser-déposer à partir de la boîte d'outils de Visual Studio (onglet Silverlight XAML Controls), le code suivant est automatiquement ajouté au fichier `Page.xaml.cs` :

```
<UserControl
    xmlns:my="clr-namespace:System.Windows.Controls;
        assembly=System.Windows.Controls.Data"
```

Cela qui signifie que :


- le composant `DataGrid` injecté dans le projet devra être préfixé (dans ses balises) de `my` (vous pouvez choisir un autre préfixe) ;
- le code de la grille (dans la DLL `System.Windows.Controls.Data`) sera injecté dans le fichier XAP transmis au navigateur du client, ce qui accroît la taille de ce fichier d'une centaine de Ko.

Commençons avec une grille dans sa forme la plus simple, avec génération automatique des colonnes en fonction de la source de données (à chaque propriété de la source de données correspond une colonne). L'attribut `AutoGenerateColumns` valant `true` par défaut, nous pouvons l'omettre.

L'association entre la source de données (ici, le tableau d'objets `Pers` de la section précédente « Cas des données provenant d'une liste d'objets » avec ses trois propriétés `Nom`, `Prénom` et `AN`, figure 10-6) et la grille est effectuée en cours d'exécution, en initialisant la propriété `ItemsSource`, ce qui remplit la grille de données :

```
<my:DataGrid x:Name="dg" ..... />
.....
dg.ItemsSource = tabPers;
```

Figure 10-6



Nom	Prénom	AN
Hugo	Victor	1802
Picasso	Pablo	1881
Ravel	Maurice	1875
Courbet	Gustave	1819
Zola	Emile	1840
Monet	Claude	1840
Racine	Jean	1639

Au lieu d'initialiser `dg.ItemsSource` comme nous venons de le faire, nous aurions pu écrire :

```
<my:DataGrid ItemsSource="{Binding}" ..... />
.....
dg.DataContext = tabPers;
```

Une modification de la source de données (`tabPers`) n'a aucune influence sur la grille, pour cela, il faudrait que la classe `Pers` implémente l'interface `INotifyPropertyChanged`. Nous corrigerons cela bientôt.

Pour que le code ressemble davantage à celui de cas pratiques, avec accès à des données externes via des services Web, il faut écrire une fonction `GetData` qui renvoie une collection (ici, une fonction pour le moins triviale) :

```
Pers[] GetData() { return tabPers; }
```

ou (on opère ainsi sur une liste plutôt que sur un tableau de taille fixe) :

```
List<Pers> GetData() { return tabPers.ToList(); }
```

ou encore (`ObservableCollection` étant une classe qui implémente l'interface `INotifyPropertyChanged`, ce qui présente l'avantage d'assurer le suivi de la liaison de données (voir la section précédente « Les liaisons de données avec `TextBlock` et `TextBox` ») :

```
using System.Collections.ObjectModel;
.....
ObservableCollection<Pers> GetData()
{
    ObservableCollection<Pers> o = new ObservableCollection<Pers>();
    foreach (Pers p in tabPers) o.Add(p);
    return o;
}
```

ce qui s'écrit en VB :

```
Imports System.Collections.ObjectModel
.....
Private Function GetData() As ObservableCollection(Of Pers)
    Dim o As ObservableCollection(Of Pers) = New ObservableCollection(Of Pers)()
    For Each p As Pers In tabPers
        o.Add(p)
    Next p
    Return o
End Function
```

L'assignation dans `ItemsSource` (ou dans le contexte de données) devient dès lors :

```
dg.ItemsSource = GetData();
```

À ce stade, la grille est tout à fait standard, avec une rangée (en haut) des en-têtes (*headers* en anglais) de colonnes et une colonne (à gauche) qui sert à marquer la sélection. Si l'utilisateur clique sur un en-tête de colonne, la grille est alors triée selon le champ de cette colonne (tri alternativement croissant et décroissant). Même si l'apparence de la grille est déjà fort acceptable, nous allons en améliorer progressivement la présentation ainsi que les fonctionnalités.

Modifications simples de présentation de la grille

Pour commencer, nous allons nous intéresser à des modifications simples, puis nous verrons comment modifier fondamentalement l'apparence des cellules avec la technique des templates.

L'attribut `HeadersVisibility` permet d'afficher ou de supprimer des en-têtes, tant de colonnes que de rangées. Le tableau 10-2 présente les différentes valeurs de cet attribut.

Tableau 10-2 – Les différents valeurs de l'attribut `HeadersVisibility`

Nom de la valeur	Description
All	Les deux en-têtes sont affichés : en-tête de colonnes (rangée supérieure, avec les libellés de colonnes) et en-tête de rangées (colonne de gauche, avec marque de sélection).
Column	Seuls les en-têtes de colonnes (avec les libellés de colonnes) sont affichés.
None	Aucun en-tête de colonnes ou de rangées n'est affiché.
Row	Seuls les en-têtes de rangées sont affichés (dans la colonne de gauche).

Il est possible de ne pas afficher les lignes de séparation entre les rangées et les colonnes en initialisant à `false` l'attribut `GridlinesVisibility` dont les différentes valeurs possibles sont : `None` (aucune ligne de séparation), `All` (lignes de séparation tant horizontales que verticales), `Horizontal` (seules les lignes de séparation entre les rangées sont affichées) et `Vertical` (seules les lignes de séparation entre les colonnes sont affichées).

Le tableau 10-3 présente les différents attributs « simples » de présentation de grille.

Tableau 10-3 – Les différents attributs « simples » de présentation de grille

Nom de l'attribut	Description
<code>AlternatingRowBackground</code> <code>RowBackground</code>	Couleur (et plus précisément, pinceau) de coloriage du fond des rangées paires et impaires. Par défaut, les lignes paires et impaires ont déjà des couleurs de fond différentes, en vue d'assurer une meilleure lisibilité. Ces deux propriétés permettent d'imposer une autre couleur de fond. <code>RowBackground</code> s'applique aux rangées impaires et <code>AlternatingRowBackground</code> aux rangées paires.
<code>CanUserResizeColumns</code>	Indique (<code>true</code> ou <code>false</code>) si l'utilisateur peut redimensionner les colonnes. Par défaut, la valeur vaut <code>true</code> .
<code>ColumnHeadersHeight</code>	Hauteur de la rangée des en-têtes de colonnes (celles avec les libellés de colonnes).
<code>ColumnWidth</code>	Largeur par défaut de toutes les colonnes.
<code>SelectionMode</code>	Indique si l'utilisateur peut sélectionner une seule rangée (valeur <code>SingleFullRow</code>) ou plusieurs (valeur <code>ExtendedFullRow</code> , avec sélection multiple à l'aide des touches Maj et Ctrl), ce qui est le cas par défaut.

Les styles, qui rappellent les CSS (*Cascading Style Sheets*), seront étudiés au chapitre 15. Disons pour le moment qu'il est possible de spécifier un nom de style dans les attributs `Style` (style général de la grille), `RowStyle` (style général des rangées) ainsi que `RowHeaderStyle` et `ColumnHeaderStyle` (style des en-têtes).

Par défaut, les données sont rendues dans la grille comme des chaînes de caractères, sauf les propriétés de type booléen dans la source de données qui sont rendues par des cases à cocher.

Assurer le suivi des données entre la source et la grille

Comme nous l'avons vu précédemment, il faut implémenter l'interface `INotifyPropertyChanged` dans la classe des objets de la source (ici, `Pers`) pour assurer le suivi des données entre la source et la cible (autrement dit, faire en sorte qu'une modification dans la source soit automatiquement répercutée dans la cible). Ceci n'est valable que si la source de données est de type `ObservableCollection` (voir la section précédente « Présentation générale » plus haut). Si ce n'est pas le cas, il faut modifier la classe comme indiqué à la fin de la section précédente « Les liaisons de données avec `TextBlock` et `TextBox` ».

Désormais, si l'on exécute le code suivant :

```
tabPers[0].Nom = "Jules";
```

Cela a un effet immédiat sur la grille : la cellule dans la colonne `Nom` de la première rangée est modifiée.

Définir ses propres colonnes

Dans les exemples précédents, les colonnes étaient construites automatiquement, en fonction des propriétés déclarées dans la source de données. Cependant, il est possible de définir ses propres colonnes et de les associer (éventuellement) à des propriétés dans la source de données. Dans le code suivant, deux colonnes ont été définies (le champ `AN` ne sera donc pas repris dans la grille, l'attribut `AutoGenerateColumns` passant à `false`).

```
<my:DataGrid AutoGenerateColumns="False" ..... >
  <my:DataGrid.Columns>
    <my:DataGridTextColumn Header="NOM" DisplayMemberBinding="{Binding Nom}" />
    <my:DataGridTextColumn Header="PRENOM"
                          DisplayMemberBinding="{Binding Prénom}" />
  </my:DataGrid.Columns>
</my:DataGrid>
```

Pour une colonne, il est possible de spécifier un style pour l'en-tête (attribut `HeaderStyle`, voir la section « Les styles » du chapitre 15) ou un style pour les cellules (attribut `CellStyle`).

L'attribut `DisplayMemberBinding` permet d'associer une propriété de la source de données à une colonne.

Il est possible de définir deux types de colonne : `DataGridTextColumn` (pour un rendu sous forme de libellé) et `DataGridCheckBoxColumn` (pour un rendu sous forme de cases à cocher pour les booléens).

Si l'attribut `IsReadOnly` d'une colonne vaut `true`, l'utilisateur ne peut pas modifier les cellules de cette colonne, même si cette fonctionnalité est implémentée (ce que nous expliquerons bientôt). D'autres attributs de contrôle sont `CanUserReorder` (avec `False`, l'utilisateur ne peut pas déplacer cette colonne) et `CanUserResize` (avec `False`, l'utilisateur ne peut pas redimensionner cette colonne). La valeur par défaut de ces attributs est `True`.

Les templates de colonnes

Passons maintenant à la vitesse supérieure et voyons comment personnaliser une colonne (plus précisément, la représentation des cellules d'une colonne), ce qui est possible grâce à la balise `DataGridTemplateColumn`. Dans l'exemple qui suit, une cellule est représentée par une grille avec deux cellules en première rangée (pour Prénom et Nom) et une seule en seconde rangée (pour AN) :

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" RowHeight="50"
    AutoGenerateColumns="False" >
    <my:DataGrid.Columns>
    <my:DataGridTemplateColumn Header="Nos célébrités" >
    <DataGridTemplateColumn.CellTemplate>
    <DataTemplate>
    <Grid>
    <Grid.RowDefinitions>
    <RowDefinition /><RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
    <ColumnDefinition /><ColumnDefinition />
    </Grid.ColumnDefinitions>
    <TextBlock Text="{Binding Prénom}"
        Grid.Row="0" Grid.Column="0" Margin="5, 0" />
    <TextBlock Text="{Binding Nom}" Foreground="Red"
        Grid.Row="0" Grid.Column="1" />
    <TextBlock Text="{Binding AN}" Grid.ColumnSpan="2" TextAlignment="Center"
        Grid.Row="1" Grid.Column="0" />
    </Grid>
    </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
    </my:DataGridTemplateColumn>
    </my:DataGrid.Columns>
</my:DataGrid>
```

La figure 10-7 présente le résultat obtenu. La grille présente autant de colonnes que de balises `my:DataGridTextColumn`, `my:DataGridCheckBoxColumn` et `my:DataGridTemplateColumn` (encore faut-il que `AutoGenerateColumns` ait pour valeur `False`).

Figure 10-7

Nos célébrités	
Victor	Hugo 1802
Pablo	Picasso 1881
Maurice	Ravel 1875
Gustave	Courbet 1819

Éditer le contenu d'une colonne

Il est possible de spécifier la représentation d'une cellule lorsque celle-ci entre en mode « édition ». Pour cela, il suffit d'ajouter dans la balise `my:DataGridTemplateColumn` de la colonne de cette cellule :

```
<my:DataGridTemplateColumn.CellEditingTemplate>
  <DataTemplate>
    <TextBox Text="{Binding Nom, Mode=TwoWay}" />
  </DataTemplate>
</my:DataGridTemplateColumn.CellEditingTemplate>
```

Une zone d'édition s'affiche alors (et contient le nom pour cette rangée) suite à un clic sur une cellule. L'utilisateur peut annuler l'opération en appuyant sur la touche Echap (même si le champ avait été partiellement modifié) ou la confirmer au moyen de la touche Entrée ou en cliquant sur une autre cellule. Nous verrons plus loin comment contrôler ces opérations (et ne pas autoriser des modifications intempestives) en traitant l'événement `CommittingEdit`.

Déterminer la ou les rangées sélectionnées

Pour détecter les rangées sélectionnées, il suffit d'analyser `dg.SelectedItem` ou `dg.SelectedItems` (en fonction de la valeur de l'attribut `SelectionMode`, correspondant à une sélection simple ou à une sélection multiple).

Dans le cas d'une sélection multiple, `dg.SelectedItems.Count` contient le nombre d'articles sélectionnés ainsi que les caractéristiques de chaque article :

```
foreach (Pers p in dg.SelectedItems)
{
    ..... // nom dans p.Nom
}
```

ce qui s'écrit en VB :

```
For Each p As Pers In dg.SelectedItems
    ..... ' nom dans p.Nom
Next p
```

L'événement `LoadingRow`

Cet événement est signalé à la grille au moment où Silverlight prépare une rangée (travail en mémoire, avant affichage). Cela nous donne l'occasion de représenter la rangée d'une manière particulière, en fonction de son contenu.

Par exemple, pour afficher sur fond rouge les rangées pour lesquelles l'année de naissance est postérieure à 1850 :

```
<my:DataGrid LoadingRow="dg_LoadingRow" ..... >
.....
void dg_LoadingRow(object sender, DataGridRowEventArgs e)
```



```
{
    int N = e.Row.GetIndex(); // numéro de rangée en train d'être préparée
    if (tabPers[N].AN > 1850) e.Row.Background = new SolidColorBrush(Colors.Red);
}
```

L'événement `CommittingEdit`

La fonction suivante est exécutée quand la grille de données traite l'événement `CommittingEdit`, c'est-à-dire quand l'utilisateur confirme une modification de cellule (en appuyant sur la touche Entrée ou en cliquant sur une autre cellule) mais avant l'introduction du contenu de la cellule modifiée (à ce moment, généralement, dans une zone d'édition) dans la source de données. Par programme, il est ainsi possible de refuser la prise en compte de la modification.

```
private void dg_CommittingEdit(object sender, DataGridEndingEditEventArgs e)
{
    .....
}
```

Dans cette fonction, `e.Row.GetIndex()` et `e.Column.Header` font respectivement référence à la rangée (numéro de rangée) et à la colonne concernées. `e.EditingElement` (de type `FrameworkElement`, donc à convertir généralement en un `TextBox`) contient la nouvelle valeur (résultat de la modification effectuée par l'utilisateur). En faisant passer `e.Cancel` à `true`, on refuse que la modification soit répercutée dans la source de données :

```
TextBox te = e.EditingElement as TextBox;
if (te.Text == "???" ) e.Cancel=true;
```

En VB, on écrirait :

```
Dim te As TextBox = TryCast(e.EditingElement, TextBox)
If te.Text = "???" Then
    e.Cancel=True
End If
```

L'accès aux fichiers

Le stockage isolé avec IsolatedStorage

La zone de stockage isolé

Pour des raisons de sécurité, une application Silverlight n'a pas accès au système de fichiers de l'utilisateur. Néanmoins, elle peut accéder, sur la machine de l'utilisateur, à une zone du disque, dédiée à chaque application Silverlight qui en fait la demande. Cet espace disque, appelé stockage isolé (*isolated storage* en anglais) est par défaut de 1 Mo par application Silverlight, mais il est possible d'augmenter cette taille si l'application en fait la demande. Cette extension d'espace disque n'est cependant accordée que si l'utilisateur l'autorise explicitement.

À la manière des cookies, bien connus des programmeurs Web, une zone de stockage isolé est propre à une application Silverlight et persiste (sur le disque donc) au-delà de la session de travail avec cette application. Elle permet de conserver des informations propres à l'utilisateur (par exemple, ses préférences) mais également d'autres données qui permettraient d'éviter les allers-retours au serveur (ce qui a un impact favorable sur les performances). Les accès à cet espace disque sont indépendants du navigateur, contrairement aux cookies. Malgré une analogie de départ, la technique Silverlight est incomparablement supérieure aux cookies.

Un programme Silverlight émanant du site <http://www.aaa.com> peut créer des répertoires, des fichiers, les supprimer ou encore écrire et lire dans les fichiers localisés de la zone de stockage isolé qui lui a été accordée. Il ne peut accéder ni au système de fichiers de l'utilisateur ni aux zones de stockage isolé allouées aux autres programmes Silverlight. Une application Silverlight ignore tout du chemin qui mène, sur le disque de l'utilisateur, à la zone de stockage isolé.

Les classes relatives à cette mémoire isolée font partie de l'espace de noms `System.IO.IsolatedStorage`, qu'il convient donc de mentionner dans le programme C# (directive `using`) ou VB (directive `Imports`).

Stocker simplement des informations élémentaires

La technique la plus simple pour stocker (de manière permanente) puis retrouver (éventuellement longtemps après) des informations propres à l'application Silverlight et à l'utilisateur (puisque l'on est sur sa machine) consiste à utiliser la classe `IsolatedStorageSettings` qui joue le rôle de dictionnaire. En C#, cela donne :

```
using System.IO.IsolatedStorage;
....
IsolatedStorageSettings appSettings
    = IsolatedStorageSettings.ApplicationSettings;
appSettings["Agent"] = "Longtarin";
appSettings["ID"] = 15;      // ou : appSettings.Add("ID", 15);
appSettings.Save();
```

En VB, les crochets doivent être remplacés par des parenthèses.

Les objets du dictionnaire `appSettings` sont de type `object` (ce qui permet de stocker n'importe quoi) et la clé est de type `string`. Pour vérifier si la clé `Nom` existe, il convient d'écrire le code suivant :

```
if (appSettings.Contains("Nom")) ....
```

Et pour retrouver les valeurs associées à des clés :

```
IsolatedStorageSettings appSettings
    = IsolatedStorageSettings.ApplicationSettings;
string agent = appSettings["Agent"].ToString();
int id = (int)appSettings["ID"];
```

En VB, cela donne :

```
Imports System.IO.IsolatedStorage
....
Dim appSettings As IsolatedStorageSettings _
    = IsolatedStorageSettings.ApplicationSettings
appSettings("Agent") = "Longtarin"
appSettings("ID") = 15
appSettings.Save()
....
Dim n As Integer = appSettings("Année")
```

Le système de fichiers du stockage isolé

Une autre technique, plus élaborée, consiste à réclamer l'accès à cette mémoire disque isolée :

```
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
```

À partir de l'objet `IsolatedStorageFile`, il est possible :

- de déterminer l'espace disponible (pour l'application Silverlight qui réclame cette information) ;
- de créer et de supprimer des répertoires ;
- de créer et de supprimer des fichiers ;
- de vérifier si un fichier ou un répertoire existe ;
- d'obtenir les noms des répertoires et des fichiers déjà présents ;
- de tenter d'augmenter l'espace disque alloué à l'application.

Il est bien entendu également possible de lire et d'écrire dans ces fichiers, comme on le ferait pour n'importe quel fichier en programmation Windows.

Voyons comment effectuer ces opérations par programme.

Le tableau 11-1 présenté les différentes propriétés et méthodes de la classe `IsolatedStorageFile`.

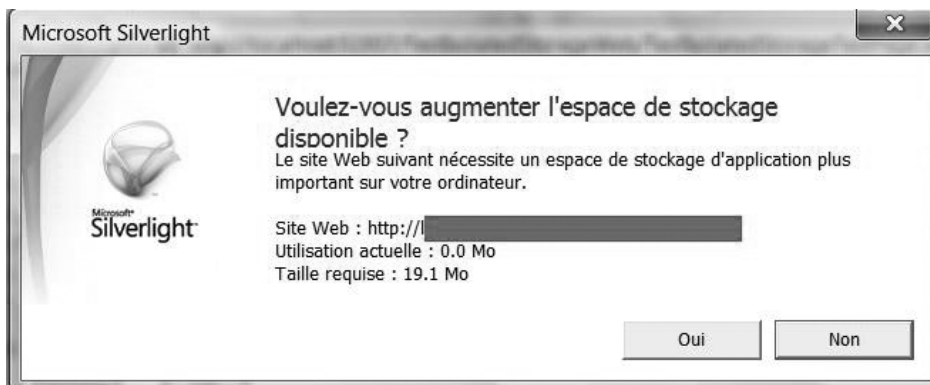
Tableau 11-1 – Les différentes propriétés et méthodes de la classe `IsolatedStorageFile`

Nom de la propriété/méthode	Description
<code>AvailableFreeSpace</code>	Propriété qui donne l'espace disque (en octets) encore disponible dans la zone de stockage isolé allouée à l'application Silverlight (entier au format <code>long</code>).
<code>CreateDirectory(dir)</code>	Crée un répertoire dont le nom est passé en argument. Si un répertoire porte déjà ce nom, il n'est pas modifié.
<code>CreateFile(fich)</code>	Crée un fichier dont le nom est spécifié en argument. Renvoie un objet <code>IsolatedStorageFileStream</code> qui donne accès au contenu de ce fichier. L'opération échoue si le fichier existe déjà.
<code>DeleteDirectory(dir)</code>	Supprime le répertoire dont le nom est passé en argument.
<code>DeleteFile(fich)</code>	Supprime le fichier dont le nom est passé en argument.
<code>DirectoryExists(dir)</code>	Renvoie <code>true</code> si le répertoire existe.
<code>FileExists(fich)</code>	Renvoie <code>true</code> si le fichier existe.
<code>GetDirectoryNames()</code>	Renvoie un tableau de chaînes de caractères contenant les noms des répertoires déjà créés.
<code>GetDirectoryName(pat)</code>	Même chose mais avec une chaîne de recherche, y compris les caractères génériques <code>*</code> et <code>?</code> .
<code>GetFileNames()</code>	Renvoie un tableau contenant les noms des fichiers.
<code>GetFileNames(pat)</code>	Même chose mais avec possibilité de spécifier une chaîne de recherche (par exemple, <code>"F*.*"</code> pour se limiter aux fichiers dont le nom commence par <code>F</code> , quelle que soit l'extension).

Tableau 11-1 – Les différentes propriétés et méthodes de la classe `IsolatedStorageFile` (suite)

Nom de la propriété/méthode	Description												
<code>OpenFile(fich, FileMode, FileAccess)</code>	<p>Ouvre le fichier dont le nom est spécifié en premier argument et renvoie un objet <code>IsolatedStorageFileStream</code> donnant accès au contenu du fichier. <code>OpenFile</code> renvoie null (<code>Nothing</code> en VB) en cas d'échec.</p> <p>Le deuxième argument indique comment Silverlight doit réagir en cas de conflit de nom. <code>FileMode</code> peut contenir l'une des valeurs suivantes de l'énumération <code>FileMode</code> (voir exemple plus loin) :</p> <table border="1"> <tr> <td><code>CreateNew</code></td><td>Un nouveau fichier doit être créé mais si un fichier existant porte déjà ce nom, une exception est générée.</td></tr> <tr> <td><code>Create</code></td><td>Si un fichier porte ce nom, il est écrasé.</td></tr> <tr> <td><code>Open</code></td><td>Un fichier existant doit être ouvert, sinon une exception est générée.</td></tr> <tr> <td><code>OpenOrCreate</code></td><td>Le fichier, s'il existe, doit être ouvert, sinon il faut en créer un nouveau.</td></tr> <tr> <td><code>Truncate</code></td><td>Un nouveau fichier doit être créé mais si un fichier existe, il est d'abord vidé de son contenu.</td></tr> <tr> <td><code>Append</code></td><td>Si le fichier existe, il doit y avoir positionnement à la fin du fichier de manière à pouvoir y ajouter de nouvelles données.</td></tr> </table> <p>Le troisième argument est optionnel. L'argument <code>FileAccess</code> peut être l'une des valeurs suivantes de l'énumération <code>FileAccess</code> : <code>Read</code>, <code>Write</code> ou <code>ReadWrite</code>.</p>	<code>CreateNew</code>	Un nouveau fichier doit être créé mais si un fichier existant porte déjà ce nom, une exception est générée.	<code>Create</code>	Si un fichier porte ce nom, il est écrasé.	<code>Open</code>	Un fichier existant doit être ouvert, sinon une exception est générée.	<code>OpenOrCreate</code>	Le fichier, s'il existe, doit être ouvert, sinon il faut en créer un nouveau.	<code>Truncate</code>	Un nouveau fichier doit être créé mais si un fichier existe, il est d'abord vidé de son contenu.	<code>Append</code>	Si le fichier existe, il doit y avoir positionnement à la fin du fichier de manière à pouvoir y ajouter de nouvelles données.
<code>CreateNew</code>	Un nouveau fichier doit être créé mais si un fichier existant porte déjà ce nom, une exception est générée.												
<code>Create</code>	Si un fichier porte ce nom, il est écrasé.												
<code>Open</code>	Un fichier existant doit être ouvert, sinon une exception est générée.												
<code>OpenOrCreate</code>	Le fichier, s'il existe, doit être ouvert, sinon il faut en créer un nouveau.												
<code>Truncate</code>	Un nouveau fichier doit être créé mais si un fichier existe, il est d'abord vidé de son contenu.												
<code>Append</code>	Si le fichier existe, il doit y avoir positionnement à la fin du fichier de manière à pouvoir y ajouter de nouvelles données.												
<code>IncreaseQuota(quota)</code>	<p>Effectue une tentative d'augmentation de la taille de l'espace disque susceptible d'être alloué à l'application Silverlight. Le quota réclamé (exprimé en octets) doit être supérieur à celui déjà accordé (sinon, une exception est générée). Une boîte de dialogue est affichée, qui demande à l'utilisateur s'il accepte cette augmentation de la taille de l'espace de stockage isolé (figure 11-1). La fonction renvoie <code>true</code> si l'opération s'est avérée possible. Un programme ne peut que réclamer une augmentation, il ne peut réduire la taille de la zone de stockage isolé qui lui a été dévolue.</p>												

Figure 11-1



Pour connaître les noms des fichiers déjà présents dans l'espace de stockage isolé alloué à l'application Silverlight (et uniquement à cette application), il convient d'écrire le code suivant :

```
using System.IO.IsolatedStorage;
.....
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
string[] ts = isf.GetFileNames();
foreach (string s in ts) .....
```

Le nombre de fichiers présents est donné par `ts.Length`. Dans la boucle `foreach`, on retrouve chaque nom de fichier dans `s`.

Pour lire ou écrire dans le fichier, il faut utiliser les fonctions de la classe `IsolatedStorageFileStream`, comme on le fait en programmation .NET pour n'importe quel fichier, à l'aide des classes dérivées de `FileStream`.

Le tableau 11-2 présente les différentes propriétés et méthodes de la classe `IsolatedStorageFileStream`.

Tableau 11-2 – Les différentes propriétés et méthodes de la classe `IsolatedStorageFileStream`

Nom de la propriété/méthode	Description
<code>Read(byte[], from, n)</code>	Lit <code>n</code> octets à partir du déplacement <code>from</code> dans le flux (le déplacement étant exprimé en nombre d'octets à partir du début du fichier). La fonction <code>Read</code> renvoie le nombre d'octets lus, ce qui permet de détecter la fin du fichier (quand cette valeur est inférieure à <code>n</code>). Les données ainsi lues sont stockées dans le tableau d'octets passé en premier argument.
<code>Write(byte[], from, n)</code>	Écrit <code>n</code> octets dans le fichier, à partir du déplacement <code>from</code> dans le fichier. Les données à écrire proviennent du tableau d'octets passé en premier argument.
<code>Length</code>	Taille du fichier, en octets et sous forme d'un entier de type <code>long</code> .
<code>Position</code>	Position courante dans le fichier, par rapport au début de celui-ci.
<code>Close()</code>	Ferme le flux.

Pour créer un fichier (en supposant qu'aucun fichier existant ne porte ce nom, ce dont le programme devrait s'assurer) et y écrire une chaîne de caractères, il convient de saisir le code suivant :

```
using System.Text                // pour UTF8Encoding
.....
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
IsolatedStorageFileStream fs = isf.CreateFile("Fich.dat");
string s = "Fichier créé le " + DateTime.Now.ToShortDateString()
          + " à " + DateTime.Now.ToLongTimeString();
// conversion de string en byte[]
UTF8Encoding enc = new UTF8Encoding();
fs.Write(enc.GetBytes(s), 0, s.Length);
fs.Close();
```

La classe UTF8Encoding fait partie de l'espace de noms System.Text, qu'il faut donc ajouter en tête du fichier C# ou VB (directive using ou Imports selon le langage).

Voici le code à écrire en C# pour lire le contenu du fichier :

```
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
IsolatedStorageFileStream fs = isf.OpenFile("Fich.dat", System.IO.FileMode.Open);
int N = (int)fs.Length;           // taille du fichier
byte[] tb = new byte[N];         // tableau de réception
fs.Read(tb, 0, N);
// convertir tb en une chaîne de caractères
UTF8Encoding enc = new UTF8Encoding();
string s = enc.GetString(tb, 0, tb.Length);
```

Ce qui donne en VB :

```
Private Sub bEcrire_Click(ByVal sender As System.Object, _
    ByVal e As System.Windows.RoutedEventArgs)
    Dim isf As IsolatedStorageFile = _
        IsolatedStorageFile.GetUserStoreForApplication()
    Dim fs As IsolatedStorageFileStream = isf.CreateFile("Fichvb.dat")
    Dim s As String = "Fichier créé le " & DateTime.Now.ToShortDateString()
        & " à " & DateTime.Now.ToLongTimeString()
    Dim enc As New UTF8Encoding
    fs.Write(enc.GetBytes(s), 0, s.Length)
    fs.Close()
End Sub

Private Sub bLire_Click(ByVal sender As System.Object, _
    ByVal e As System.Windows.RoutedEventArgs)
    Dim isf As IsolatedStorageFile
        = IsolatedStorageFile.GetUserStoreForApplication()
    Dim fs As IsolatedStorageFileStream = isf.OpenFile("Fichvb.dat", _
        System.IO.FileMode.Open)
    Dim N As Integer = CInt(Fix(fs.Length)) ' taille du fichier
    Dim tb(N - 1) As Byte ' tableau de réception
    fs.Read(tb, 0, N)
    ' convertir tb en une chaîne de caractères
    Dim enc As New UTF8Encoding()
    Dim s As String = enc.GetString(tb, 0, tb.Length)
End Sub
```

Dans les extraits de code suivants, des lignes de texte vont être écrites puis lues dans un fichier situé dans l'espace de stockage isolé :

```
using System.IO.IsolatedStorage;
using System.IO;
```

Pour écrire deux lignes de texte dans le fichier App.dat :

```
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
IsolatedStorageFileStream isfs
    = new IsolatedStorageFileStream("App.dat", FileMode.Create, isf);
StreamWriter sw = new StreamWriter(isfs);
sw.WriteLine("Première ligne"); sw.WriteLine("Deuxième ligne");
sw.Close(); isfs.Close();
```

Et pour lire ultérieurement ces deux lignes :

```
IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication();
IsolatedStorageFileStream isfs = isf.OpenFile("App.dat", FileMode.Open);
StreamReader sr = new StreamReader(isfs);
string s1 = sr.ReadLine();
string s2 = sr.ReadLine();
sr.Close(); isfs.Close();
```

où s1 contient Première ligne et s2, Deuxième ligne. sr.ReadLine() renvoie null quand plus aucune ligne n'est présente dans le fichier.

Localisation de la zone de stockage isolé

Pour windows, la zone de stockage isolé se trouve sur le disque (l'application Silverlight n'a aucun moyen de le savoir ni de tirer parti de cette information) :

- pour XP, dans le répertoire C:\Documents and Settings\utilisateur\Local Settings\Application\Data\Microsoft\Silverlight\is ;
- pour Vista, dans le répertoire utilisateur\AppData\LocalLow\Microsoft\Silverlight\is (figure 11-2).

Figure 11-2



Le répertoire mentionné ci-dessus (et qui dépend du système d'exploitation) contient deux sous-répertoires aux noms en apparence cryptés, qui contiennent à leur tour un répertoire nommé *g* ainsi qu'un répertoire *s*. Le répertoire *g* contient des sous-répertoires aux noms cryptés (sur une trentaine de caractères) qui dépendent du site d'origine de l'application Silverlight (autant de répertoires que de sites d'origine). Par ailleurs, chacun de ces répertoires contient quatre fichiers, notamment le fichier *id.dat* dans lequel figure, en clair, le nom du site d'origine. Le répertoire *s*, quant à lui, contient également des sous-répertoires aux noms cryptés (un par application Silverlight) qui incluent :

- un fichier *Group.dat* renfermant le nom du répertoire (sous *g*) propre à l'application Silverlight ;
- les données en clair (répertoires et fichiers) de l'application Silverlight.

Au moment de la sortie de Silverlight 2 bêta 2, seul l'outil *storeAdm* (introduit avec la version 2 du framework .NET en 2005) permet, en mode console, d'administrer la zone de stockage isolé.

Lire des fichiers distants

Une application Silverlight peut ouvrir et lire un fichier se trouvant sur le serveur grâce au composant *WebClient* (voir chapitre 13), capable d'effectuer une lecture asynchrone du fichier (autrement dit, sans bloquer l'application Silverlight durant la lecture). Si le fichier *Fich.txt* se trouve dans le répertoire du fichier XAP de l'application Silverlight (sous-répertoire *ClientBin*), il convient d'écrire le code suivant (voir les explications au chapitre 15) :

```
using System.Net;
using System.IO;
.....
WebClient client = new WebClient();
client.OpenReadCompleted
    += new OpenReadCompletedEventHandler(client_OpenReadCompleted);
client.OpenReadAsync(new Uri("Fich.txt", UriKind.Relative));
.....
void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    StreamReader sr = new StreamReader(e.Result);
    string s = sr.ReadLine();
    while (s != null)
    {
        ..... // contenu de la ligne dans s
        s = sr.ReadLine();
    }
}
```

Dans cet extrait de code, nous avons négligé de tester le bon achèvement de l'opération afin de n'introduire qu'un seul nouveau concept à la fois. En cas d'erreur (par exemple,

l'absence du fichier sur le serveur), une exception est générée dans la fonction de traitement de l'événement `OpenReadCompleted`.

Il faudrait donc écrire :

```
void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    try
    {
        .....          // Ok, tout s'est bien passé
    }
    catch (Exception exc)
    {
        .....          // traiter l'erreur (message d'erreur dans exc.Message)
    }
}
```

Si le fichier contient des lettres accentuées, veillez à le sauvegarder au format UTF-8.

Lire des fichiers locaux

Comme nous l'avons vu pour les images (voir la section « Lire une image à partir du système de fichiers local » du chapitre 7), une application Silverlight peut accéder à des fichiers locaux à condition que l'utilisateur mène explicitement (via une boîte de dialogue) à ce fichier :

```
using System.IO;
using System.Text;
.....
OpenFileDialog ofd = new OpenFileDialog();
ofd.Filter = "Fichiers de texte|*.txt";
if (ofd.ShowDialog() == DialogResult.OK)
{
    Stream str = ofd.SelectedFile.OpenRead();
    int N = (int)str.Length;          // taille du fichier de texte
    byte[] tb = new byte[N];
    str.Read(tb, 0, N);              // lire tout le contenu du fichier
    // amener le contenu du fichier local dans une variable de type string
    s = UTF8Encoding.UTF8.GetString(tb, 0, N);
    str.Close();
}
```

ou, plus simplement dans le cas de fichiers de texte (`ReadLine` étant applicable à `sr`) :

```
StreamReader sr = ofd.SelectedFile.OpenText();
s = sr.ReadToEnd();
sr.Close();
```

Ici aussi, l'application Silverlight ignore tout du chemin qui mène au fichier de texte sélectionné par l'utilisateur.

Les fichiers en ressources

Comme nous l'avons vu pour les images (voir la section « Les images en ressources » du chapitre 7), un fichier (ici, un fichier de texte) peut être incorporé en ressource. Pour cela, sélectionnez le menu Ajouter>Élément existant... dans la fenêtre du projet, partie Silverlight et localisez le fichier souhaité. Spécifiez ensuite les propriétés de ce nouvel élément dans le projet et forcez la valeur `Resource` dans le champ `Action` de génération. Le fichier est ainsi incorporé en ressource et sera greffé au fichier XAP après compilation (plus précisément à la DLL contenant le code de l'application, cette DLL étant elle-même greffée au fichier XAP).

Pour lire le contenu du fichier en cours d'exécution de programme (ici, le fichier de texte `SLRes.txt` composé de deux lignes et inclus en ressource de l'application `SLProg`), il convient d'écrire le code suivant :

```
using System.Windows.Resources;
using System.IO;
.....
StreamResourceInfo sri = Application.GetResourceStream(
    new Uri("SLProg;component/SLRes.txt", UriKind.Relative));
StreamReader sr = new StreamReader(sri.Stream);
string s1 = sr.ReadLine();
string s2 = sr.ReadLine();
```

12

Accès XML avec Linq

Les fichiers XML sont des fichiers de texte jouant un rôle considérable en informatique. Ils constituent en effet un moyen privilégié pour passer des informations d'un programme à un autre ou d'une machine à une autre, quelle que soit la plate-forme sur laquelle s'exécutent ces programmes. Ainsi, pour les services Web (qui consistent à faire exécuter des opérations sur une autre machine, voir chapitre 13), la communication se fait communément par de longues chaînes de caractères au format XML échangées entre programmes serveurs et programmes clients.

Pour expliquer de quelle manière est lu et décortiqué un fichier XML (ou une chaîne de caractères au format XML) dans un programme Silverlight, partons de l'exemple suivant (le fichier `Pers.xml`) :

```
<?xml version="1.0" encoding="utf-8" ?>
<Personnages>
  <Personnage>
    <Nom>Talon</Nom>
    <Prenom>Achille</Prenom>
    <Pere VN="Michel Regnier" DN="1931">Greg</Pere>
    <Creation>1963</Creation>
  </Personnage>
  <Personnage>
    <Nom>Petitpas</Nom>
    <Prenom>Prudence</Prenom>
    <Pere DN="1922">Maurice Maréchal</Pere>
    <Creation>1957</Creation>
  </Personnage>
  <Personnage>
    <Nom>Titeuf</Nom>
    <Pere VN="Philippe Chappuis" DN="1967">Zep</Pere>
    <Creation>1992</Creation>
  </Personnage>
</Personnages>
```

Ce fichier XML est certes limité à trois personnages (ce qui est suffisant pour cet exposé, davantage de données n'apporteraient rien) mais, surtout, on constate que des balises et des attributs ne sont pas présents de manière uniforme, ce qui correspond à un cas pratique trop souvent passé sous silence.

Pour décortiquer ce XML, nous utiliserons la nouvelle technologie introduite par Microsoft, à savoir Linq (*Language Integrated Query*), ici appliquée au XML, autrement dit Linq to XML.

Linq est un langage intégré à C# et VB qui permet de retrouver (mais aussi de mettre à jour) des informations en provenance de sources de données aussi disparates que des tables en mémoire, des bases de données ou des fichiers XML. Il est basé sur la syntaxe SQL, bien connue des programmeurs. Avant Linq, il fallait préparer des chaînes de caractères contenant des commandes SQL, exécuter ces commandes et amener les résultats dans des variables en mémoire. Tout cela se faisait sans que le compilateur puisse effectuer des vérifications et des conversions automatiques pourtant bien nécessaires. Pour le XML, les choses étaient encore fort différentes. Linq intègre tout cela dans C# et VB et nous fait bénéficier de l'aide contextuelle, si pratique.

Pour pouvoir utiliser Linq to XML dans un programme Silverlight, il faut réclamer l'inclusion de bibliothèques propres à cette technologie (il est en effet inutile de charger les programmes qui n'en feraient pas usage). Pour cela, effectuez un clic droit sur la partie Silverlight du projet dans l'Explorateur de solutions, sélectionnez Ajouter une référence...>onglet .NET et ajoutez System.Xml.Linq (par double-clic ou sélection suivie d'un clic sur le bouton OK).

La bibliothèque System.Xml.Linq.dll sera ainsi incluse dans le fichier d'extension .xap transmis au navigateur, ce qui accroît la taille de ce fichier d'une cinquantaine de Ko.

Passons maintenant à l'écriture du programme Silverlight qui décortiquera ce fichier XML.

En tête du fichier nommé par défaut Page.xaml.cs ou Page.xaml.vb selon le langage choisi, il faut ensuite ajouter :

```
using System.Xml.Linq; // en C#  
Imports System.Xml.Linq ' en VB
```

Nous allons d'abord envisager le cas d'un fichier XML inclus dans le projet. Dans l'Explorateur de solutions (partie Silverlight du projet), sélectionnez Ajouter>Élément existant... et localisez le fichier Pers.xml sur votre ordinateur. Celui-ci sera alors copié dans le répertoire du projet et, après compilation de l'application, il sera inclus dans le fichier .xap transmis au navigateur.

Nous expliquerons à la section « Application à la lecture d'un fichier XML » du chapitre 13 comment lire des fichiers XML non inclus en ressources et se trouvant sur le serveur.

Chargement du fichier XML

Pour charger un fichier XML et commencer à le décortiquer, nous avons deux possibilités :

- soit créer une variable de type `XDocument` et charger le fichier ;
- soit créer une variable de type `XElement` et charger le fichier.

Analysons ces deux possibilités en mettant en évidence ce qui les distingue (tableau 12-1).

Tableau 12-1 – Les deux possibilités de chargement d'un fichier XML

Première possibilité (C#)	Seconde possibilité (C#)
<code>XDocument xml;</code> <code>xml = XDocument.Load("Pers.xml");</code>	<code>XElement xml;</code> <code>xml = XElement.Load("Pers.xml");</code>
Première possibilité (VB)	Seconde possibilité (VB)
<code>Dim xml as XDocument</code> <code>xml = XDocument.Load("Pers.xml")</code>	<code>Dim xml as XElement</code> <code>xml = XElement.Load("Pers.xml")</code>

La différence se situe ici :

- avec la variable de type `XDocument`, il faut d'abord passer par la balise extérieure qu'est `Personnages` ;
- avec la variable de type `XElement`, on accède directement aux balises `Personnage`, en court-circuitant la balise `Personnages`.

Les balises XML peuvent se trouver dans une chaîne de caractères, ce qui est le cas lorsque les données proviennent d'un service Web. Si le XML est construit en mémoire, remplacez les apostrophes doubles (") du XML par des apostrophes simples ('). Si le XML provient d'un service Web, prenez soin de remplacer `<` par le caractère `<` et `>` par le caractère `>` (voir l'exemple de la section « Application à la lecture d'un fichier XML » du chapitre 13). Il faut alors exécuter la fonction `Parse` de la classe `XDocument` ou de la classe `XElement` (remplacez par les balises `Personnage`) :

```
s = "<?xml version='1.0' encoding='utf-8' ?><Personnages> ..... </Personnages>";  
xml = XElement.Parse(s);
```

L'exception `XmlException` est générée en cas d'erreur de balise XML dans `s`. Cette erreur peut être interceptée dans un `try / catch`.

Quelle que soit la technique utilisée, la fonction `Elements` donne accès à une collection de balises (de nœuds ou d'éléments, si vous préférez). Le tableau 12-2 présente les codes C# et VB à écrire pour connaître le nombre de balises `Personnage` (ici, trois), sachant que la fonction `Count` renvoie le nombre d'éléments dans une collection.

Tableau 12-2 – Calcul du nombre de balises Personnage

C#
Première possibilité
<pre>XDocument xml; xml = XDocument.Load("Pers.xml"); n = xml.Elements("Personnages").Elements("Personnage").Count();</pre>
Seconde possibilité
<pre>XElement xml; xml = XElement.Load("Pers.xml"); n = xml.Elements("Personnage").Count();</pre>
VB
Première possibilité
<pre>Dim xml as XDocument xml = XDocument.Load("Pers.xml") n = xml.Elements("Personnages").Elements("Personnage").Count()</pre>
Seconde possibilité
<pre>Dim xml as XElement xml = XElement.Load("Pers.xml") n = xml.Elements("Personnages").Elements("Personnage").Count()</pre>

Les espaces de noms

Il arrive souvent que la balise externe (ici, Personnages) fasse mention d’un espace de noms. Par exemple :

```
<?xml version="1.0" encoding="utf-8" ?>  
<Personnages xmlns="http://www.moi.fr" >  
.....
```

Dans ce cas, les arguments des fonctions faisant référence à des noms de balises doivent être écrits différemment :

```
XNamespace ns = "http://www.moi.fr";  
int n = xml.Elements(ns+"Personnage").Count();
```

Cas pratiques

Pour partir à la découverte de Linq, envisageons des cas pratiques de recherche dans le fichier XML. Pour ne pas alourdir inutilement l’exposé, nous partirons toujours d’une variable de type XElement. Les exemples pratiques suivants utilisent le langage Linq.

Retrouver les noms des personnages

Pour cela, il convient de créer et d'initialiser la variable `xml` de type `XElement`. À noter que le nom de cette variable, donnant accès directement à la collection de balises `Personnage`, est bien entendu sans importance. Linq va ensuite nous permettre de construire la liste des personnes (`s` est ici de type « chaîne de caractères » qui finalement contiendra les noms des personnages, séparés par un tiret) :

```
var listePers = from p in xml.Elements("Personnage") select p;  
foreach (var el in listePers) s += el.Element("Nom").Value + " - ";
```

Analysons d'abord la première ligne de cet extrait de code (la syntaxe VB sera présentée par la suite) :

- `xml`, qui est de type `XElement`, donne directement accès aux balises situées sous la balise la plus extérieure ;
- `xml.Elements("Personnage")` donne accès à la collection de balises `Personnage` ;
- cette collection est balayée avec `from p in xml.Elements("Personnage")`. Il s'agit là d'une syntaxe Linq, intégrée au C#, que nous retrouverons telle quelle en VB. Chaque élément de cette collection s'appelle `p` et est retenu (avec `select p`) pour figurer dans la liste.

Nous avons ainsi transformé une collection de balises en une collection d'objets, plus facilement manipulables par programme.

Cette première ligne en C# mérite de plus amples commentaires. Tout d'abord, quel est le véritable type de `listePers` ? Avec `var`, nous laissons au compilateur le soin de déterminer l'information. `var` n'a rien à voir avec les antiques (et dangereux à l'usage) variants du Basic (type indéterminé, sujet à tout moment à n'importe quel changement). Ici, `listePers` désigne une variable d'un type bien déterminé et ne pourra jamais, après initialisation, contenir une valeur d'un autre type. `listePers` est ici de type « énumération générique de `Xelement` » (il suffit de laisser un instant le curseur sur `var` pour s'en rendre compte).

Utiliser `var` plutôt qu'un type assez compliqué à formuler est d'une remarquable facilité pour le programmeur et n'a aucune conséquence sur la rigueur et la qualité du code généré (le compilateur effectue le travail à notre place et c'est d'ailleurs pour cela que `var` est utilisée). `listePers.Count()` donnerait le nombre d'éléments dans la liste (ici, 3).

Analysons maintenant la deuxième ligne de code (une boucle `foreach`, ce qui n'a rien de nouveau) :

- `el` correspond à un personnage (ensemble des balises `Nom`, `Prenom`, etc., relatives à un personnage) mais, ici aussi, on laisse au compilateur le soin de déterminer le type exact (qui est en fait `System.Xml.Linq.XElement`) ;
- `el.Element("Nom")` contient, pour le premier personnage, `<Nom>Talon</Nom>` ;
- `el.Element("Nom").Value` contient `Talon`.

À la suite des trois passages dans la boucle `foreach`, `el.Element("Nom").Value` est passé successivement par les valeurs Talon, Petitpas et Titeuf.

Notez la différence entre :

- `Elements`, qui donne accès à une collection de balises (les balises `Personnage`, par exemple) ;
- `Element`, qui donne accès à une balise particulière (la balise `Nom` d'un personnage particulier, par exemple).

La syntaxe VB est en tout point semblable :

```
Dim listePers = From p In xml.Elements("Personnage") Select p
For Each el In listePers
    s += el.Element("Nom").ToString() & " - "
Next
```

Ici, les balises `<Nom>` et `</Nom>` sont reprises dans `s` ; elles ne le seraient pas avec `el.Element("Nom").Value`.

Retrouver les prénoms des personnages

Certains personnages n'ont pas de balise `Prenom`. Pour tester si une telle balise existe, il convient d'écrire le code suivant dans la boucle `foreach` :

```
var o = el.Element("Prenom");
if (o == null) ..... .. la balise n'existe pas ..
else ..... .. la balise existe ..
```

Quand la balise `Prenom` existe (`o` est alors différent de `null` ou `Nothing` en VB), on peut utiliser `o.Value` pour retrouver le prénom.

Détecter si une balise contient une ou plusieurs balises

Pour détecter si une balise (celle qui est en train d'être examinée dans le `foreach`) contient une ou plusieurs balises (par exemple, si une balise `Pere` dans une balise `Personnage` particulière contient elle-même une balise), il convient d'écrire le code suivant :

```
var o = el.Element("Pere");
if (o.HasElements) ..... .. une ou plusieurs balises enfants ..
else ..... .. pas de balise enfant pour cet élément ..
```

On retrouve les noms des balises enfants de la balise courante en écrivant (on passe en revue toutes les balises enfants s'il y en a) :

```
if (o.HasElements)
    foreach (var x in o.Elements())
        ..... .. nom de la balise dans x.Name ..
```

Retrouver les attributs d'une balise

Pour tester si une balise (la balise `Pere` en train d'être examinée dans la boucle des personnages, par exemple) contient un ou plusieurs attributs, il convient d'écrire le code suivant :

```
if (el.Element("Pere").HasAttributes) .....
```

Et pour retrouver un attribut (VN pour « véritable nom », par exemple), il faut écrire le code suivant (sans oublier de tester si cet attribut existe bien dans la balise `Pere` particulière) :

```
var attVN = el.Element("Pere").Attribute("VN");
if (attVN != null) ..... .. l'attribut existe bien ..
```

Comme on peut s'y attendre, on retrouve alors la valeur de l'attribut dans `attVN.Value`.

Et maintenant la même chose en VB, mais sans créer de variable intermédiaire (on retient dans `s` le nom de l'auteur s'il s'agit du véritable nom et le pseudonyme suivi, entre parenthèses, du véritable nom s'il s'agit d'un nom d'auteur) :

```
Dim listePers = From p In xml.Elements("Personnage") Select p
For Each el In listePers
    If el.Element("Pere").Attribute("VN") Is Nothing Then
        s += el.Element("Pere").Value & " - "
    Else : s += el.Element("Pere").Value & " (" _
                & el.Element("Pere").Attribute("VN").Value & ")" & " - "
    End If
Next
```

Amélioration du select

Jusqu'ici, dans chaque itération de la boucle `from` (dans la partie Linq de nos instructions), nous avons retenu l'ensemble de la balise `Personnage` (à cause du simple `select p`). Cependant, il est possible de ne retenir qu'un ou plusieurs champs de cet élément :

```
var listeNoms = from p in xml.Elements("Personnage")
                select p.Element("Nom").Value;
foreach (var s in listeNoms) ..... .. nom dans s.Value ..
```

Ici, nous n'avons rencontré aucun problème car chaque personnage a un nom mais il n'en serait pas de même pour les prénoms, pour lesquels il faudrait alors écrire (`null` est inséré dans la liste des prénoms pour chaque personnage qui n'a pas de prénom) :

```
var listePrénoms = from p in xml.Elements("Personnage")
                    select p.Element("Prenom");
foreach (var s in listePrénoms)
    if (s != null) ..... .. prénom dans s.Value ..
    else ..... .. pas de prénom pour ce personnage ..
```

Convertir le résultat d'une recherche en un tableau ou une liste

Le résultat d'une opération Linq peut être copié dans un tableau ou une liste (autrement dit, un conteneur .NET) :

```
var listeNoms = from p in xml.Elements("Personnage")
                select p.Element("Nom").Value;
List<string> liste = listeNoms.ToList();
```

Le nombre d'articles dans la liste est alors donné par `liste.Count` (sans parenthèses ici). Le deuxième nom dans cette liste est donné par `liste[1]`.

Le résultat de l'opération Linq peut être copié dans un tableau, ici un tableau de chaînes de caractères (tableau qu'il ne sera pas possible d'agrandir par la suite, contrairement à la liste générique). En C#, cela donne :

```
var listeNoms = from p in xml.Elements("Personnage")
                select p.Element("Nom").Value;
string[] ts = listeNoms.ToArray(); // nom du deuxième personnage dans ts[1]
```

Et en VB :

```
Dim listeNoms = From p In xml.Elements("Personnage") _
                Select p.Element("Nom").Value
Dim ts As String() = listeNoms.ToArray() ' nom du deuxième personnage dans ts(1)
```

Création d'objets d'une classe à partir de balises

Nous allons maintenant créer une classe contenant les principales informations relatives à un personnage (informations retenues sous forme de propriétés). Pour créer une nouvelle classe dans le programme Silverlight, cliquez droit sur la partie Silverlight du projet dans l'Explorateur de solutions, sélectionnez **Ajouter>Nouvel élément...>Classe** et nommez le fichier de cette nouvelle classe `Pers.cs`.

```
public class Pers
{
    public string Nom { get; set; }
    public string Prénom { get; set; }
    public int AnnéeCréation { get; set; }
}
```

La classe `Pers` contient trois propriétés. Avec la syntaxe plus condensée introduite en C# version 3, les champs privés qui correspondent à ces propriétés sont automatiquement créés par le compilateur, ainsi que les fonctions associées en `get` et/ou `set` aux propriétés.

Nous allons à présent passer en revue les personnages et créer une liste d'objets `Pers`. Pour chaque balise `Personnage`, il convient de créer un nouvel objet `Pers`, dont les trois propriétés sont initialisées à partir du contenu des balises (certains contenus peuvent être `null`) :

```
var listePers = from p in xml.Elements("Personnage")
                select new Pers
                {
                    Nom = (string)p.Element("Nom"),
                    Prénom = (string)p.Element("Prenom"),
                    AnnéeCréation = (int)p.Element("Creation")
                };
```

La relation est ici directe entre les champs d'un objet `Pers` et les balises situées sous la balise `Personnage`, mais elle pourrait être plus complexe (voir l'exemple ci-dessous). Nous balayons maintenant `listePers` (liste d'objets `Pers`) et, pour chaque objet `e1` (de type `Pers`), nous accédons à ses différentes propriétés :

```
foreach (var e1 in listePers)
{
    .....      .. nom dans e1.Nom ..
    .....      .. prénom dans e1.Prénom (éventuellement null) ..
    .....      .. année de création dans e1.AnnéeCréation ..
}
```

Pour les balises `Personnage` sans prénom, `e1.Prénom` vaut `null` (cela ne pose pas de problème puisque `e1.Prénom` désigne une référence à une chaîne de caractères et que `null` dans une référence signifie « pas de chaîne »).

Mais que se passerait-il si une balise `Personnage` ne contenait pas de balise `Creation`, qui compte une valeur de type « entier » et ne peut donc contenir une valeur `null` (`Nothing` en VB), contrairement aux objets et chaînes de caractères ? Il faudrait en tenir compte (sous peine de voir le run-time Silverlight générer une exception) et écrire :

```
var listePers = from p in xml.Elements("Personnage")
                select new Pers
                {
                    Nom = (string)p.Element("Nom"),
                    Prénom = (string)p.Element("Prenom"),
                    AnnéeCréation = p.Element("Creation") != null ?
                                   (int)p.Element("Creation") : -1
                };

```

Cela signifie que nous balayons les balises `Personnage` et que, pour chacune d'elles, la propriété `AnnéeCréation` de l'objet `Pers` ainsi créé prend la valeur suivante : si la balise `Creation` est présente (`p.Element("Creation")` est alors différent de `null`), ce champ prend la valeur donnée par `p.Element("Creation")` mais convertie en un entier ; dans le cas contraire, ce champ prend la valeur `-1` (valeur choisie par convention pour signaler une absence de valeur).

Les contraintes et les tris

Linq, à l'instar du langage SQL, permet de spécifier des contraintes et des tris. Ainsi, pour ne retenir que les personnages créés après 1960 et les trier par ordre croissant d'ancienneté, on écrit :

```
var listePers = from p in xml.Elements("Personnage")
                where (int)p.Element("Creation") > 1960
                orderby (int)p.Element("Creation") ascending
                select new Pers
                {
                    Nom = (string)p.Element("Nom"),
                    AnnéeCréation = (int)p.Element("Creation")
                };

```

```
foreach (var el in listePers)
{
    .....          .. nom dans el.Nom ..
    .....          .. année de création dans el.AnnéeCréation ..
}
```

La partie Linq de notre programme pourrait faire référence à des variables du programme (Linq est en effet parfaitement intégré à C# et VB). Ainsi, la requête précédente pourrait s'écrire en C# :

```
int N = 1960;
var listePers = from p in xml.Elements("Personnage")
                where (int)p.Element("Creation") > N
                orderby (int)p.Element("Creation") ascending
                select new Pers
                {
                    Nom = (string)p.Element("Nom"),
                    AnnéeCréation = (int)p.Element("Creation")
                };
```

Et en VB :

```
Dim N As Integer = 1960
Dim listePers = From p In xml.Elements("Personnage") _
                where CInt(Fix(p.Element("Creation"))) > N _
                orderby (Integer)p.Element("Creation") ascending _
                select New Pers With _
                { _
                    .Nom = CStr(p.Element("Nom")), _
                    .AnnéeCréation = CInt(Fix(p.Element("Creation")))_
                }
```

13

Accès à distance aux données

Les accès distants

La plupart des applications Web (banques, agences de voyages, commerces, etc.) affichent des données qui proviennent en général d'une base de données, laquelle peut se trouver sur une machine particulière que l'on appelle le serveur de bases de données. Il est évidemment hors de question d'installer ou de déporter toute cette base de données (et même une partie de celle-ci) sur la machine du client.

En programmation serveur, le programme ASP.NET ou PHP (qui s'exécute sur le serveur) analyse la requête d'un client (et il en traite un grand nombre, venant de partout dans le monde), accède à la base de données, en extrait des données, les formate dans une page HTML (souvent en tenant compte des particularités et des droits d'accès du client) et envoie celle-ci au client.

Une application Silverlight 2 (qui s'exécute sur la machine du client et ne traite donc qu'un seul client) réclame des données d'un serveur. Celui-ci (bien entendu après les vérifications d'usage) les extrait de la base de données et les envoie à l'application Silverlight, généralement au format XML. L'application Silverlight décortique ensuite le XML et se charge de la présentation des données, éventuellement en y intégrant des effets d'animation (impossibles à réaliser en programmation serveur).

Dans ce chapitre, nous allons étudier une technique (basée sur l'objet `WebClient`) permettant de lire un fichier se trouvant sur une machine distante (image, fichier XML, etc.) ou d'obtenir une sélection de données opérée sur la base de données, laquelle se trouve sur un serveur et n'est pas directement accessible par les applications (Silverlight ou autres).

À cet effet, l'application Silverlight réclame l'exécution d'un service qui se trouve (dans une fonction, sous forme de code) quelque part sur un site serveur Internet. On dit que l'application Silverlight appelle un service Web. La technique n'est pas propre aux applications Silverlight car elle est plus générale et peut être adoptée par les applications Windows, les applications ASP.NET qui s'exécutent sur le serveur ainsi que celles, s'il en reste, qui s'exécutent en mode console.

Comment une application Silverlight qui s'exécute sur la machine d'un client peut-elle réclamer l'exécution d'une fonction qui se trouve quelque part sur le Web ? Tout simplement en effectuant une requête, semblable à une requête permettant d'obtenir une page Web mais avec l'extension `.asmx`. Elle passe ensuite des arguments à cette fonction distante en formatant des données en attributs de l'URL (sous la forme `?xyz=...`). Le serveur renvoie alors la réponse en la formatant dans du XML, avec des noms de balises définis dans le protocole des services Web.

L'objet WebClient

Application à la lecture d'une image

L'objet `WebClient` permet de réclamer le contenu d'un fichier (ou ce que le serveur veut bien vous envoyer) de manière asynchrone afin de ne pas bloquer l'application Silverlight durant l'opération (qui prend un « certain temps » puisqu'il y a accès au Web). Ainsi, l'utilisateur n'a pas le sentiment perturbant que l'application « ne répond plus », même temporairement. D'où vient ce « un certain temps » ? Même si les choses se passent aujourd'hui sur des lignes de transmission à grande vitesse et sur des serveurs très performants, lancer une requête sur le Web, attendre son tour sur le serveur dans la file d'attente des multiples requêtes, accéder aux données sur le disque serveur et renvoyer le tout au client prend « un certain temps ». Il est donc impératif que l'application Silverlight ne paraisse pas bloquée durant ce laps de temps.

Considérons le composant Image suivant qui, pour le moment, affiche l'image `VG1.jpg`, qui se trouve en ressource et est donc injectée dans le fichier XAP de l'application Silverlight :

```
<Image x:Name="img" Source="VG1.jpg" ..... />
```

Nous allons charger dans ce composant l'image `VG2.jpg` qui a été copiée dans le répertoire `ClientBin` de l'application Web. Cette image ne se trouve donc pas en ressource et n'a pas été greffée au fichier XAP (mettre une image en ressource n'a d'intérêt que si l'on est certain qu'elle sera affichée dans la page).

Pour réclamer et lire ce fichier image situé sur le serveur, il faut :

- créer un objet `WebClient` ;
- indiquer l'emplacement (sous la forme d'une URL) de l'image sur le serveur (ici, un emplacement relatif, par rapport à l'emplacement du fichier XAP) ;
- lancer l'opération (qui sera exécutée parallèlement, donc de manière asynchrone, à l'application Silverlight) ;

- indiquer la fonction à exécuter lorsque l'opération est terminée, ce qui est nécessaire puisque les choses se passent de manière asynchrone (c'est en effet dans cette fonction que nous allons récupérer les dizaines de milliers d'octets constituant l'image et permettant de la reconstruire).

Un objet `WebClient` peut réclamer du serveur :

- une chaîne de caractères avec la fonction `DownloadStringAsync` de la classe `WebClient`, convenant notamment pour lire le contenu d'un fichier XML (puisque celui-ci est constitué de lignes de texte) ;
- un flux d'octets (*stream*) avec `OpenReadAsync`, convenant pour les images.

Dans notre cas (lecture d'un fichier image), nous allons évidemment utiliser `OpenReadAsync` (la réponse est loin d'être une simple chaîne de caractères) et convertir la suite d'octets ainsi reçue en un objet `BitmapImage` (ce que nous avons déjà fait à la section « Lire une image à partir du système de fichiers local » du chapitre 7). Voici le code C# correspondant :

```
using System.Net;                                // pour WebClient
using System.Windows.Media.Imaging;              // pour BitmapImage
.....
WebClient client = new WebClient();
client.OpenReadCompleted
    += new OpenReadCompletedEventHandler(client_OpenReadCompleted);
client.OpenReadAsync(new Uri("VG2.jpg", UriKind.Relative));
.....
void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
{
    try
    {
        BitmapImage image = new BitmapImage();
        image.SetSource(e.Result);
        img.Source = image;
    }
    catch (Exception exc)
    {
        ..... // erreur lors de l'accès, avec message d'erreur dans exc.Message
    }
}
```

`OpenReadAsync` lance l'exécution. L'URL du fichier image est passée en argument (seul argument ou premier de deux), sous forme d'un objet `Uri`.

Un second argument, de type `object` et qui peut donc être n'importe quoi, peut être ajouté à la fonction `OpenReadAsync`. On retrouve cet argument tel quel dans `e.UserState` dans la fonction traitant l'événement `OpenReadCompleted`. Il permet de lancer plusieurs requêtes asynchrones avec la même fonction de traitement.

La fonction `client_OpenReadAsyncCompleted` est automatiquement exécutée aussitôt après réception du dernier octet des données (ou plus tôt, en cas d'erreur). Dans cette fonction

de traitement, les erreurs (fichier non trouvé ou au mauvais format) doivent être interceptées dans un try / catch.

Si tout s'est bien passé, `e.Result` fait référence au flux d'octets téléchargé depuis le serveur. Dans notre cas, un objet `BitmapImage` est créé à partir de tous ces octets ainsi reçus (un octet transmis pour chaque octet dans le fichier image).

La même chose en VB :

```
Imports System.Net                                ' pour WebClient
Imports System.Windows.Media.Imaging              ' pour BitmapImage
.....
Dim client = New WebClient()
client.OpenReadAsync(New Uri("VG2.jpg", UriKind.Relative), 0)
AddHandler client.OpenReadCompleted, AddressOf client_OpenReadCompleted
.....
Private Sub client_OpenReadCompleted(ByVal sender As Object, _
                                     ByVal e As OpenReadCompletedEventArgs)

    Try
        Dim image As New BitmapImage()
        image.SetSource(e.Result)
        img.Source = image
    Catch exc As Exception
        .....                                ' message d'erreur dans exc.Message
    End Try
End Sub
```

Application à la lecture d'un fichier XML

Nous allons maintenant lire un fichier XML se trouvant sur le serveur, dans le répertoire `ClientBin` de l'application. Nous disposons du fichier XML suivant qui a été copié sur le serveur (si vous utilisez le Bloc-notes pour le créer, n'oubliez pas de l'enregistrer au format UTF8, sous peine de perdre les lettres accentuées) :

```
<?xml version="1.0" encoding="utf-8" ?>
<Artistes>
  <Artiste>
    <Nom>Brel</Nom>
    <AN>1929</AN>
  </Artiste>
  <Artiste>
    <Nom>Brassens</Nom>
    <AN>1921</AN>
  </Artiste>
  <Artiste>
    <Nom>Ferré</Nom>
    <AN>1916</AN>
  </Artiste>
</Artistes>
```

Pour lire le contenu de ce fichier XML, nous utilisons la même technique que précédemment, mais cette fois avec `DownloadStringAsync` puisque nous lisons de simples lignes de texte :

```

using System.Net;
.....
WebClient client = new WebClient();
client.DownloadStringCompleted
    += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
client.DownloadStringAsync(new Uri("Artistes.xml", UriKind.Relative));
.....
void client_DownloadStringCompleted(object sender,
                                   DownloadStringCompletedEventArgs e)
{
    try
    {
        string s = e.Result;
        .....                .. contenu du fichier XML dans s ..
    }
    catch (Exception exc)
    {
        .....
    }
}

```

Si tout s'est bien passé (pas d'entrée dans le catch), le contenu du fichier XML se retrouve dans `e.Result`. Comme pour `OpenReadAsync`, il est possible de passer à `DownloadStringAsync` un deuxième argument, de type `object`. On retrouve la valeur inchangée de cet argument, dans `e.UserState`, dans la fonction de traitement de l'événement `DownloadStringCompleted`.

Il nous reste à décortiquer ce contenu (voir chapitre 12) :

```

using System.Xml.Linq;
.....
XElement xml = XElement.Parse(s);
var listeArtistes = from p in xml.Elements("Artiste") select p;
foreach (var p in listeArtistes)
{
    .. nom dans p.Element("Nom").Value ..
    .. année de naissance dans p.Element("AN").Value ..
}

```

La même chose en VB :

```

Imports System.Xml.Linq
.....
Dim client = New WebClient()
client.DownloadStringAsync(New Uri("Artistes.xml", UriKind.Relative), 0)
AddHandler client.DownloadStringCompleted, _
    AddressOf client_DownloadStringCompleted
.....
Private Sub client_DownloadStringCompleted(ByVal sender As Object, _
                                           ByVal e As DownloadStringCompletedEventArgs)
    Try
        Dim s As String = e.Result
    
```

```
Dim xml As XElement = XElement.Parse(s)
Dim listeArtistes = From p In xml.Elements("Artiste") _
                    Select p
For Each p In listeArtistes
    .. nom dans p.Element("Nom").Value ..
    .. année de naissance dans p.Element("AN").Value
Next p
Catch exc As Exception
    .....
End Try
End Sub
```

Application à un service Web météo

Le site <http://www.webservicex.net> fournit un certain nombre de services Web, notamment un service météo pour un grand nombre de villes du monde entier. Avec la technique des services Web, l'application Silverlight appelle une fonction qui se trouve quelque part sur Internet.

La page <http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=48> fournit des informations sur le service Web météo, mettant à disposition de ses clients (service gratuit) deux fonctions :

- `GetCitiesByCountry` qui indique les villes d'un pays pour lesquelles des informations d'ordre météorologique sont disponibles (plus d'une centaine de villes pour la France) ;
- `GetWeather` qui donne la météo pour une ville particulière.

Ces deux fonctions s'exécutent quelque part sur un serveur Internet. Peu importe où, peu importe le langage dans lequel elles sont écrites et peu importe la représentation des données en mémoire (ainsi un nombre entier peut avoir une représentation binaire différente sur le serveur et sur la machine du client, car le nombre d'octets et leur ordre dépendent du microprocesseur et du système d'exploitation). Ces fonctions sont appelées en spécifiant une URL, comme pour la requête d'une page Web (mais avec l'extension `.asmx`). Les données sont renvoyées au format XML, indépendamment des représentations en mémoire des données puisées au format texte.

La figure 13-1 représente la page Web donnant les informations relatives au service Web météo.

Avant même d'écrire la moindre ligne de code, il est possible de tester ces services Web et d'examiner la réponse, qui est au format XML. Cette possibilité n'est pas propre à ce service Web : il s'agit d'une fonctionnalité offerte par tous les services Web et qui devient automatiquement disponible dès que, sur le serveur, une fonction est marquée comme « service Web » (et est donc susceptible d'être appelée par un programme – pas seulement Silverlight – s'exécutant sur une machine connectée à Internet).

Toujours en interactif, testons la fonction Web `GetCitiesByCountry`. Pour cela, saisissez un nom de pays (en anglais) dans le champ `CountryName` et appelez la fonction distante en cliquant sur le bouton `Invoke` (figure 13-2).

Figure 13-1

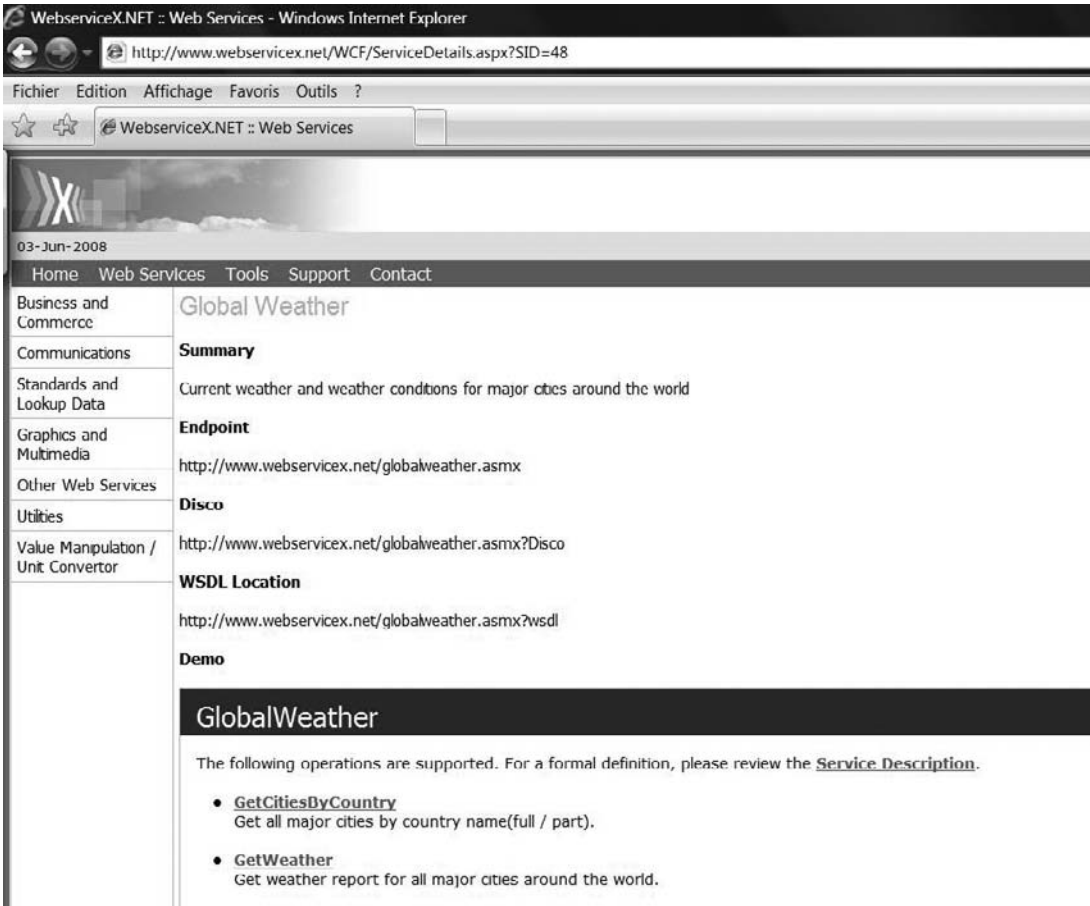


Figure 13-2



Après avoir testé la fonction `GetCitiesByCountry` en interactif sur Internet, le paquet de données suivant est reçu pour la France (seul un fragment, avec les deux premières villes, est présenté ici) :

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://www.webserviceX.NET">
  <NewDataSet>
    <Table>
      <Country>France</Country>
      <City>Le Touquet</City>
    </Table>
    <Table>
      <Country>France</Country>
      <City>Agen</City>
    </Table>
    . . . . .
  </NewDataSet>
</string>
```

Nous allons maintenant tester, toujours en interactif, la fonction `GetWeather` afin d'examiner le XML de réponse. Cette fonction admet un nom de ville et un nom de pays en arguments. Considérons ici que ces arguments sont Paris (non repris dans la centaine de villes françaises) et France. Le XML reçu est alors le suivant (figure 13-3).

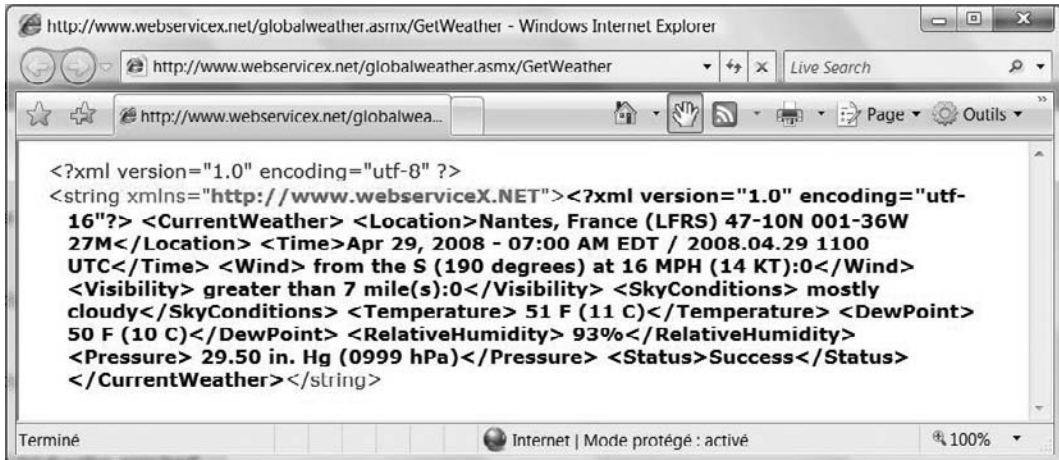
Figure 13-3



Dans le cas d'une ville reprise dans la centaine de villes françaises (ici Nantes), le XML aurait été le suivant (voir figure 13-4).

Voyons comment retrouver ces informations par programme et en premier lieu la liste des villes. Dans la réponse XML, il faut tenir compte du fait que les symboles `<` et `>` ont été remplacés par `<` et `>` afin qu'ils ne soient pas interprétés par le navigateur comme les caractères de début et de fin de balise. Pour résoudre ce problème, il suffit d'effectuer un remplacement et de tenir compte de l'espace de noms (attribut `xmlns` de la balise `string`) mentionné dans la réponse XML. Pour décortiquer la réponse, `Linq` sera utilisé (voir chapitre 12).

Figure 13-4



Voici le code C# correspondant :

```
WebClient client = new WebClient();
client.DownloadStringCompleted
    += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
string url = "http://www.webserviceX.net/GlobalWeather.aspx/"
    + "GetCitiesByCountry?CountryName=France";
client.DownloadStringAsync(new Uri(url), 0);
.....
List<string> listeVilles;
void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
{
    try
    {
        string s = e.Result;
        s = s.Replace("&lt;", "<");
        s = s.Replace("&gt;", ">");
        XDocument xml = XDocument.Parse(s);
        XNamespace xmlns = "http://www.webserviceX.NET";
        var q = from p in xml.Elements(xmlns + "string")
                .Elements(xmlns + "NewDataSet")
                .Elements(xmlns + "Table")
                select p.Element(xmlns + "City").Value ;
        listeVilles = q.ToList<string>();
    }
    catch (Exception exc)
    {
        ..... // traiter l'erreur
    }
}
```

Ce qui s'écrit en VB :

```
Imports System.Xml.Linq
Imports System.Net
.....
Dim client = New WebClient()
client.DownloadStringAsync( _
    New Uri("http://www.webServiceX.net/GlobalWeather.asmx/"
        & "GetCitiesByCountry?CountryName=France"), 0)
AddHandler client.DownloadStringCompleted, _
    AddressOf client_DownloadStringCompleted
.....
Dim listeVilles As List(Of String)
Private Sub client_DownloadStringCompleted(ByVal sender As Object, _
    ByVal e As DownloadStringCompletedEventArgs)

    Try
        Dim s As String = e.Result
        s = s.Replace("<", "<")
        s = s.Replace(">", ">")
        Dim xml As XDocument = XDocument.Parse(s)
        Dim xmlns As XNamespace = "http://www.webserviceX.NET"
        Dim q = From p In xml.Elements(xmlns + "string").Elements(xmlns
            & "NewDataSet").Elements(xmlns + "Table") _
            Select p.Element(xmlns + "City").Value
        listeVilles = q.ToList()
    Catch exc As Exception
        ..... ' traiter l'erreur
    End Try
End Sub
```

La deuxième ville est donnée par `listeVilles[1]` en C# et `listeVilles(1)` en VB.

Pour tester si le pays passé en argument (à la fin de l'URL, en valeur de `CountryName`) existe, il est possible d'écrire (d'autres solutions existent) :

```
if (xml.Elements(xmlns + "string").First().Value == "")
    s = "Pas de météo pour ce pays";
```

Passons maintenant à la météo pour une ville, information donnée par la fonction `GetWeather`, que nous allons d'abord tester en interactif (figure 13-5).

Figure 13-5

GlobalWeather

Click [here](#) for a complete list of operations.

GetWeather

Get weather report for all major cities around the world.

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
CityName:	<input type="text"/>
CountryName:	<input type="text"/>

Pour une ville particulière d'un pays, par exemple Calvi, la réponse XML est :

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://www.webserviceX.NET">
  <?xml version="1.0" encoding="utf-16"?>
    <CurrentWeather>
      <Location>Calvi, France (LFKC) 42-32N 008-48E 58M</Location>
      <Time>May 01, 2008 - 01:30 PM EDT / 2008.05.01 1730 UTC</Time>
      <Wind> from the W (260 degrees) at 7 MPH (6 KT) (direction variable):0</Wind>
      <Visibility> greater than 7 mile(s):0</Visibility>
      <Temperature> 66 F (19 C)</Temperature>
      <DewPoint> 30 F (-1 C)</DewPoint>
      <RelativeHumidity> 25%</RelativeHumidity>
      <Pressure> 30.00 in. Hg (1016 hPa)</Pressure>
      <Status>Success</Status>
    </CurrentWeather>
  </string>
```

Par programme, pour être informé de la température à Calvi, il suffit d'écrire :

```
WebClient client = new WebClient();
client.DownloadStringCompleted
    += new DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
string url = "http://www.webserviceX.net/GlobalWeather.asmx/"
    + "GetWeather?CityName=Calvi&CountryName=France";
client.DownloadStringAsync(new Uri(url), 0);
```

Et dans la fonction de traitement (une balise intempestive a été éliminée dans la réponse et la température exprimée en degrés Celsius se trouve entre parenthèses dans la balise Temperature) :

```
string s = e.Result;
s = s.Replace("&lt;", "<");
s = s.Replace("&gt;", ">");
s = s.Replace("<?xml version=\"1.0\" encoding=\"utf-16\"?>", "");
XElement xml = XElement.Parse(s);
XNamespace ns = "http://www.webserviceX.NET";
string temp = xml.Elements(ns + "CurrentWeather")
    .Elements(ns+"Temperature").First().Value;
// éliminer la valeur Fahrenheit (degrés Celsius entre parenthèses)
int n1 = temp.IndexOf("(");
int n2 = temp.IndexOf(")");
temp = temp.Substring(n1 + 1, n2 - n1 - 2);
```

temp contient désormais la température en degrés Celsius mais sous la forme d'une chaîne de caractères. Pour la convertir en un nombre entier, il suffit d'utiliser la fonction Parse de la classe Int32 :

```
int température = Int32.Parse(temp);
```


Application au service Web Flickr

Flickr de Yahoo! est un site de partage de photos bien connu des internautes. À condition d'être inscrit (ce qui est gratuit), n'importe qui peut déposer ou visionner des photos, du moins celles en accès public.

Flickr offre un service Web de téléchargement de photos répondant à certains critères, service Web que nous allons exploiter ici.

Pour pouvoir accéder à Flickr et à son service Web, il faut tout d'abord s'inscrire sur le site <http://www.flickr.com> et demander une adresse de courrier électronique Yahoo! (une adresse plus usuelle peut être fournie pour recevoir le courrier). Flickr vous communique alors un identifiant et un mot de passe, lequel n'est pas nécessaire pour visualiser des photos mises en partage. L'identifiant peut être communiqué sans danger.

La première étape consiste à réclamer la liste des photos répondant à un critère (une chaîne de caractères, ici dans `sCrit`) :

```
string clé = .. votre identifiant Flickr ici ..
string url = "http://api.flickr.com/services/rest/"
           + "?method=flickr.photos.search&api_key="
           + clé + "&text=" + sCrit;
WebClient client = new WebClient();
client.DownloadStringCompleted
    += new DownloadStringCompletedEventHandler(client.DownloadStringCompleted);
client.DownloadStringAsync(new Uri(url), 0);
```

Flickr renvoie ensuite un fichier XML au format suivant (les ... remplacent des valeurs qui n'ont aucun intérêt ici) :

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  <photos page="1" pages="..." perpage="100" total="..." >
    <photo id="..." owner="..." secret="..." server="..." farm="3"
           title="..." ispublic="1" isfriend="0" isfamily="0" />
    <photo id="..." owner="..." secret="..." server="..." farm="3"
           title="..." ispublic="1" isfriend="0" isfamily="0" />
    .....
  </photos>
</rsp>
```

L'attribut `stat` de la balise `rsp` contient `ok` si la requête est acceptée (ce qui ne signifie pas que Flickr a trouvé des photos qui correspondent au critère) et `fail` en cas d'erreur.

Dans la fonction traitant l'événement `DownloadStringCompleted`, la liste des titres associés aux douze premières photos est obtenue au moyen du code suivant :

```
var liste = from p in xml.Element("rsp").Element("photos").Elements("photo")
            select p ;
liste = liste.Take(12);
.. nombre de photos donné par liste.Count(), soit ici 12 ou moins ..
```

```
foreach (var ph in liste)           // on passe en revue les photos
{
    .. titre d'une photo dans ph.Attribute("title").Value ..
}
```

Les autres attributs n'ont pas de signification pour nous mais servent à construire la chaîne de caractères donnant accès à la photo (voir l'avant-dernière instruction). La *énième* photo (en supposant qu'elle existe) est obtenue par :

```
string sFarm = liste.ElementAt(N).Attribute("farm").Value;
string sServer = liste.ElementAt(N).Attribute("server").Value;
string sId = liste.ElementAt(N).Attribute("id").Value;
string sSecret = liste.ElementAt(N).Attribute("secret").Value;
string sUrl = "http://farm" + sFarm + ".static.flickr.com/" + sServer
              + "/" + sId + "_" + sSecret + ".jpg";
ImageSource imgs = new System.Windows.Media.Imaging.BitmapImage(new Uri(sUrl));
img.SetValue(Image.SourceProperty, imgs);
```

Le composant `Image`, dont le nom interne est `img`, contient désormais la *énième* image en provenance de Flickr.

Les contrôles utilisateurs

Création d'un contrôle utilisateur

Les contrôles utilisateurs (*user controls* en anglais) jouent le même rôle que les fonctions en programmation : ils regroupent, en une seule entité, un ou plusieurs composants et offrent de ce fait des fonctionnalités particulières. L'utilisateur d'un tel type de contrôle (un programmeur ou un graphiste) doit uniquement en connaître les fonctionnalités, sans se préoccuper des détails d'implémentation.

Pour illustrer la création d'un contrôle utilisateur, nous allons créer une nouvelle application (SLProg) qui accueillera un tel contrôle. Celui-ci consistera en un bouton qui « swingue » lors du survol de la souris et qui s'agrandit temporairement au moment du clic. Ce contrôle utilisateur ne contient ici qu'un seul composant (un bouton) mais il pourrait en contenir plusieurs, qui formeraient alors une « boîte noire ».

Dans cette application Silverlight, nous insérons un contrôle utilisateur (vierge pour le moment mais qui sera complété par la suite). Pour cela, sélectionnez le menu Ajouter> Nouvel élément...>Contrôle utilisateur Silverlight à partir de la fenêtre du projet (partie Silverlight). Modifiez le nom proposé par défaut (SilverlightControl1.xaml) en BoutonQuiSvingue. Deux fichiers sont ainsi créés (en plus de Page.xaml et Page.xaml.cs, toujours présents dans le projet, pour la page Silverlight qui va accueillir le contrôle utilisateur) : BoutonQuiSvingue.xaml et BoutonQuiSvingue.xaml.cs (ou son équivalent en VB).

Le fichier BoutonQuiSvingue.xaml contient à ce stade :

```
<UserControl x:Class="SLProg.BoutonQuiSvingue"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
    </Grid>
</UserControl>
```

Le fichier de code BoutonQuiSwingue.xaml.cs contient, quant à lui, le code suivant (la version VB suit le même schéma) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SLProg
{
    public partial class BoutonQuiSwingue : UserControl
    {
        public BoutonQuiSwingue()
        {
            InitializeComponent();
        }
    }
}
```

On retrouve l'espace de noms du programme d'accueil (ici, SLProg) dans le contrôle utilisateur. Dans la mesure où l'objectif est ici d'utiliser ce contrôle dans n'importe quelle application Silverlight, il convient de transformer namespace SLProg en namespace glBoutonQuiSwingue, ce qui implique une modification similaire dans le XAML du bouton personnalisé (nous avons suivi l'usage voulant que l'on préfixe le nom d'une marque, ici avec des initiales, pour rappeler l'auteur ou l'éditeur du composant personnalisé...) :

```
<UserControl x:Class="glBoutonQuiSwingue.BoutonQuiSwingue"
```

Comme ce n'est pas à nous mais bien à l'utilisateur de décider de la taille du bouton « swingueur », supprimez les deux attributs Width et Height dans le XAML.

Dans BoutonQuiSwingue.xaml, ajoutez un bouton sans aucune particularité pour le moment :

```
<Grid x:Name="LayoutRoot" Background="White">
    <Button x:Name="b" Content="Swingueur" />
</Grid>
```

Nous pouvons déjà ajouter notre contrôle utilisateur (même dans son état inachevé) dans la page hôte qu'est Page.xaml (page dont le conteneur est une grille, que l'on suppose déjà découpée en rangées et en colonnes). Pour cela, il convient tout d'abord d'ajouter le code suivant dans l'en-tête du fichier Page.xaml, en attribut de la balise UserControl :

```
xmlns:gl="clr-namespace:glBoutonQuiSwingue"
```

Ce code indique que les composants personnalisés mentionnés dans l'espace de noms `g1BoutonQuiSwingue` seront utilisés et que toute référence à ces contrôles personnalisés se fera via le préfixe `g1`.

On s'épargne souvent bien des soucis en contrôlant le plus tôt possible, à chaque stade du développement, que tout est correct. Une compilation (menu Générer>Régénérer la solution) nous rassure déjà.

Le composant personnalisé est ensuite inséré dans la page hôte qu'est `Page.xaml` :

```
<g1:BoutonQuiSwingue Grid.Row="1" Grid.Column="1" />
```

Un bouton, encore tout à fait classique, est affiché dans la cellule en (1, 1). Son libellé est `Swingueur`, il provient de l'attribut `Content` du bouton `b` enfoui dans le contrôle utilisateur. Ce bouton s'adapte automatiquement à la taille de la cellule.

Le bouton `swingueur` peut être amélioré en traitant l'événement `MouseEnter`. Le fichier `BoutonQuiSwingue.xaml` devient alors :

```
<Button x:Name="b" Content="Swingueur" MouseEnter="b_MouseEnter" >

</Button>
```

tandis que la fonction `b_MouseEnter` est générée par Visual Studio dans le fichier `BoutonQuiSwingue.xaml.cs`.

Nous complétons le fichier `BoutonQuiSwingue.xaml` en y ajoutant une transformation `RotateTransform` ainsi qu'une animation (`stbRotate`) d'une durée totale de deux secondes :

```
<Grid x:Name="LayoutRoot" Background="White" >
  <Button x:Name="b" Content="Swingueur" RenderTransformOrigin="0.5, 0.5"
    MouseEnter="b_MouseEnter" >
    <Button.Resources>
      <Storyboard x:Name="stbRotate" Storyboard.TargetName="rotBouton"
        Storyboard.TargetProperty="Angle" >
        <DoubleAnimationUsingKeyFrames >
          <LinearDoubleKeyFrame KeyTime="0:0:0" Value="0" />
          <LinearDoubleKeyFrame KeyTime="0:0:0.5" Value="-30" />
          <LinearDoubleKeyFrame KeyTime="0:0:2" Value="30" />
          <LinearDoubleKeyFrame KeyTime="0:0:2.5" Value="0" />
        </DoubleAnimationUsingKeyFrames>
      </Storyboard>
    </Button.Resources>
    <Button.RenderTransform>
      <RotateTransform x:Name="rotBouton" />
    </Button.RenderTransform>
  </Button>
</Grid>
```

Dans la fonction traitant l'événement `MouseEnter` adressé au bouton swingueur, il suffit de lancer l'animation :

```
private void b_MouseEnter(object sender, MouseEventArgs e)
{
    stbRotate.Begin();
}
```

Dès que la souris entre dans la surface du bouton, ce dernier enclenche un mouvement de balancier (de -30 à $+30$ degrés). Un fond blanc apparaît lors du mouvement, ce qui pourrait être corrigé en ajoutant une balise `Rectangle` (sans mention de `Width` et de `Height` pour une adaptation automatique de taille mais avec un attribut `Fill`) comme première balise à l'intérieur de `Grid` dans le fichier `BoutonQuiSwingue.xaml`.

Figure 14-1



Le libellé est toujours `Swingueur`. Améliorons cela en créant une propriété dans la classe `BoutonQuiSwingue` :

```
public partial class BoutonQuiSwingue : UserControl
{
    .....
    public string Libellé
    {
        get { return b.Content.ToString(); }
        set { b.Content = value; }
    }
}
```

L'utilisateur du bouton swingueur voit la propriété `Libellé` qui apparaît dans l'aide contextuelle lors de la construction d'une balise `g1:BoutonQuiSwingue` et n'a pas à se préoccuper des détails d'implémentation. En revanche, le programmeur du contrôle personnalisé sait que `Libellé` est lié à la propriété `Content` (ici, limitée à du texte) du bouton `b` « caché » dans le bouton swingueur.

Dans la page hôte, les balises des boutons swingueurs deviennent par exemple (remplacez par d'autres attributs comme `FontFamily`, `FontSize`, `Foreground`, etc.) :

```
<g1:BoutonQuiSwingue Libellé="Mon premier bouton swingueur" ..... />
<g1:BoutonQuiSwingue ..... />
```

Le libellé du deuxième bouton swingueur est resté `Swingueur`, sa valeur par défaut.

Traitement d'événements liés à un contrôle utilisateur

Traisons maintenant l'événement `Click` adressé au bouton `b` dont la taille augmente au moment du clic grâce à une animation (`stbScale`) d'une durée de deux secondes (car nous avons fait passer `AutoReverse` à `true`).

Ces effets sont ici bien évidemment exagérés et trop longs mais c'est pour mieux les mettre en évidence et surtout montrer la simultanéité de ces effets lorsque plusieurs boutons swingueurs sont placés dans une page Silverlight (ici, sans reprendre en intégralité le code de l'animation `stbRotate`) :

```
<Button x:Name="b" Content="Swingueur" RenderTransformOrigin="0.5, 0.5"
        MouseEnter="b_MouseEnter" Click="b_Click" >
  <Button.Resources>
    <Storyboard x:Name="stbRotate" ..... >
      .....
    </Storyboard>
    <Storyboard x:Name="stbScale" Storyboard.TargetName="scaleBouton" >
      <DoubleAnimation Storyboard.TargetProperty="ScaleX"
        From="1" To="1.3" Duration="0:0:1" AutoReverse="True" />
      <DoubleAnimation Storyboard.TargetProperty="ScaleY"
        From="1" To="1.3" Duration="0:0:1" AutoReverse="True" />
    </Storyboard>
  </Button.Resources>
  <Button.RenderTransform>
    <TransformGroup>
      <RotateTransform x:Name="rotBouton" />
      <ScaleTransform x:Name="scaleBouton" />
    </TransformGroup>
  </Button.RenderTransform>
</Button>
```

La fonction de traitement de l'événement `Click` adressé au bouton `b` enfoui dans le bouton swingueur devient alors :

```
private void b_Click(object sender, RoutedEventArgs e)
{
    stbRotate.Stop();
    stbScale.Begin();
}
```

À ce stade, un effet visuel est créé au moment du clic mais le programme utilisateur n'est toujours pas informé (l'événement `Click` n'est même pas connu dans la balise `gl:Bouton-QuiSwingue`).

Pour résoudre ce problème, il convient de déclarer dans la classe du bouton swingueur une variable de type « événement » (*event* en anglais). Appelons `Click` cette variable, ce qui donne `Click` comme nom d'événement. Tant mieux pour l'utilisateur qu'on puisse garder le même nom. Celui-ci est en effet habitué au fait qu'un bouton (qu'il soit swingueur ou non) présente l'événement `Click` et peu importe pour lui qu'en interne cela ait demandé un traitement spécial. Tant que cet événement n'est pas traité dans la balise

gl:BoutonQuiSwingue dans la page hôte, cette variable contient la valeur null. Quand cet événement est traité (attribut Click avec nom de fonction en valeur), cette variable de la classe BoutonQuiSwingue prend comme valeur l'« adresse de la fonction de traitement ». L'appel Click(...) dans la fonction b_Click a pour effet d'appeler la fonction en question (avec des arguments qui sont l'objet sender concerné ainsi qu'un objet fournissant éventuellement des informations complémentaires sur l'événement).

Pour cela, le code suivant est ajouté dans la classe BoutonQuiSwingue (dans le fichier BoutonQuiSwingue.xaml.cs) :

```
public event EventHandler Click;

private void b_Click(object sender, RoutedEventArgs e)
{
    stbRotate.Stop();
    stbScale.Begin();
    if (Click != null) Click(this, EventArgs.Empty);
}
```

L'événement Click peut maintenant être traité dans la balise gl:BoutonQuiSwingue dans la page hôte :

```
<gl:BoutonQuiSwingue x:Name=bMonBQS" Libellé="Mon premier bouton swingueur" ....
    Click="bMonBQS_Click" />
```

Et cette fonction de traitement est :

```
private void bMonBQS_Click(object sender, EventArgs e)
{
}
}
```

Utilisation d'un contrôle utilisateur

Notre contrôle utilisateur est constitué de deux fichiers : BoutonQuiSwingue.xaml et BoutonQuiSwingue.xaml.cs. Pour l'utiliser dans une autre application Silverlight, il faut disposer de ces deux fichiers quelque part sur la machine de développement. On ajoute alors le fichier XAML au projet de l'application (partie Silverlight). Pour cela, sélectionnez Ajouter>Élément existant... et localisez le fichier BoutonQuiSwingue.xaml. Le fichier de code est automatiquement ajouté.

Il faut encore ajouter une directive xmlns dans la balise UserControl en tête du fichier XAML de la page hôte, qui est Page.xaml par défaut (pour rappel, glBoutonQuiSwingue est le nom donné à l'espace de noms mentionné dans le contrôle utilisateur) :

```
<UserControl x:Class=.....
    xmlns=.....
    xmlns:gl="clr-namespace:glBoutonQuiSwingue" >
    .....
</UserControl>
```

Contrôle utilisateur DLL

Cette solution présente l'inconvénient de devoir fournir les fichiers .xaml et .cs. Pour diverses raisons, on préférera souvent fournir un seul fichier compilé (et plus précisément, sous la forme d'une DLL). Pour cela, il faut créer une « Silverlight Class Library » et non un projet d'application Silverlight comme nous l'avons toujours fait jusqu'à présent. Pour créer cette Silverlight Class Library, sélectionnez Fichier>Nouveau>Silverlight>Bibliothèque de classes Silverlight dans Visual Studio. Nommez ce projet BoutonQuiSwingue.

Visual Studio crée automatiquement une classe nommée `Class1`. Supprimez-la du projet en cliquant droit sur son nom dans l'Explorateur de solutions et en sélectionnant Supprimer. Cette classe sera remplacée par un contrôle utilisateur. Pour cela, sélectionnez Ajouter>Élément existant... dans l'Explorateur de solutions et localisez le fichier `BoutonQuiSwingue.xaml` dans le répertoire du projet précédemment créé. Compilez ensuite via le menu Générer>Régénérer la solution, ce qui crée le fichier `BoutonQuiSwingue.dll` dans le sous-répertoire `Bin/Debug`. Il suffit maintenant de fournir ce fichier DLL à l'utilisateur.

Quand l'utilisateur crée une application Silverlight, il lui suffit de faire référence à cette DLL en allant dans l'Explorateur de solutions, menu Ajouter référence, et rechercher le fichier `BoutonQuiSwingue.dll`. Il faut alors ajouter à la balise `UserControl` l'attribut :

```
xmlns:gl="clr-namespace:glBoutonQuiSwingue;assembly=BoutonQuiSwingue"
```

Un tel bouton sera alors inséré dans `Page.xaml` avec (par exemple) :

```
<gl:BoutonQuiSwingue x:Name = "bBQS" Libellé="Mon bouton qui swingue"
    FontFamily="Verdana" Foreground="Red" Click="bBQS_Click"
    ClickGrid.Row="1" Grid.Column="1" />
```

La DLL `BoutonQuiSwingue.dll` sera automatiquement greffée au fichier XAP envoyé au navigateur.

Les styles et les templates

Les styles

Pour l'uniformité de la page Silverlight, il est généralement souhaitable que les composants présentent des caractéristiques d'affichage communes. L'exemple de code suivant correspond à deux zones d'affichage en rouge et en police Verdana de taille 20 :

```
<TextBlock Foreground="Red" FontFamily="Verdana" FontSize="20" ..... />  
<TextBlock Foreground="Red" FontFamily="Verdana" FontSize="20" ..... />
```

Dans ce cas, plutôt que de répéter les mêmes attributs d'un composant à l'autre, il est préférable de recourir aux styles. Ainsi, si la présentation doit être personnalisée par la suite, il suffira de modifier le style (en un seul emplacement) plutôt que de nombreux attributs en de nombreux emplacements du fichier XAML (avec le risque d'en oublier lors d'un changement ou d'effectuer des modifications intempestives).

Les styles sont placés en ressources, dans une balise `xyz.Resources`, où `xyz` doit être remplacé par `Grid`, `Canvas` ou un nom de balise. Les styles deviennent ainsi communs :

- aux composants d'un conteneur si le style est défini en ressource du conteneur dans la balise `UserControl` ;
- à tous les composants de l'application si le style est défini en ressource de l'application.

Moins utile (tout au moins dans ce chapitre) mais néanmoins possible : un style peut être spécifié dans la balise d'un composant particulier, par exemple une balise `Button`, mais toujours en ressource et alors sans attribut `x:Key`.

Voyons comment spécifier des styles (ici, un seul) dans le conteneur qu'est la grille (seules les zones d'affichage mentionnées dans la grille pourront faire référence au style `StyleRouge`, à cause de l'attribut `TargetType`) :

```

<Grid ..... >
  <Grid.Resources>
    <Style x:Key="StyleRouge" TargetType="TextBlock" >
      <Setter Property="Foreground" Value="Red" />
      <Setter Property="Background" Value="Beige" />
      <Setter Property="FontFamily" Value="Verdana" />
      <Setter Property="FontSize" Value="20" />
    </Style>
  </Grid.Resources>
  .....
</Grid>

```

On donne un nom à chaque style (attribut `x:Key`) et on indique, ce qui est indispensable, quel type de composant est concerné (attribut `TargetType`). Chaque valeur d'attribut est mentionnée dans une balise `Setter`, avec le nom de l'attribut en attribut de `Property` et sa valeur en attribut de `Value`. Plusieurs balises `Style` peuvent être spécifiées.

Les deux balises `TextBlock` précédentes deviennent dès lors :

```

<TextBlock Style="{StaticResource StyleRouge}" ..... />
<TextBlock Style="{StaticResource StyleRouge}" ..... />

```

Des attributs, même repris dans un style, peuvent être spécifiés dans une balise. Dans ce cas, c'est l'attribut de la balise qui prime sur l'attribut de style. Par exemple (la zone d'affichage a ici un fond jaune) :

```

<TextBlock Style="{StaticResource StyleRouge}" Background="Yellow" ..... />

```

Les styles peuvent être spécifiés (de la même manière) en ressources de l'application, dans le fichier `App.xaml` du projet. Ils s'appliquent alors à tous les composants de tous les conteneurs de l'application :

```

<Application xmlns="http://schemas.microsoft.com/client/2007"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SLProg.App" >
  <Application.Resources>
    <Style x:Key="StyleRouge" TargetType="TextBlock" >
      <Setter Property="Foreground" Value="Red" />
      .....
    </Style>
  </Application.Resources>
</Application>

```

L'attribut `Value` (dans une balise `Setter`) peut devenir à son tour une balise. Pour illustrer cette possibilité, compliquons une propriété (ici, `Background`) en forçant un dégradé :

```

<Style x:Key="StyleRectDegr" TargetType="Rectangle" >
  <Setter Property="Fill" >
    <Setter.Value>
      <LinearGradientBrush StartPoint="0, 0.5" EndPoint="1, 0.5" >
        <GradientStop Offset="0" Color="Black" />
        <GradientStop Offset="1" Color="White" />
      </LinearGradientBrush>
    </Setter.Value>
  </Setter>
</Style>

```

```

        </LinearGradientBrush>
    </Setter.Value>
</Setter>
</Style>

```

Pinceaux et couleurs en ressources

D'autres éléments que les styles peuvent être placés en ressources, par exemple, un pinceau :

```

<Grid.Resources>
    <SolidColorBrush x:Key="PinceauRouge" Color="Red" />
</Grid.Resources>

```

Un élément UI peut dès lors faire usage du pinceau `PinceauRouge` :

```

<TextBlock Foreground="{StaticResource PinceauRouge}" ..... />

```

Avec un exemple aussi simple, nous avons surtout compliqué les choses... Prenons un exemple plus probant. Un pinceau plus élaboré peut être créé, avec référence à des couleurs en ressources :

```

<Grid.Resources>
    <Color x:Key="Rouge">Red</Color>
    <Color x:Key="Bleu">#FF0000FF</Color>
    <LinearGradientBrush x:Key="PinceauDegr" >
        <GradientStop Offset="0" Color="{StaticResource Rouge}" />
        <GradientStop Offset="1" Color="{StaticResource Bleu}" />
    </LinearGradientBrush>
</Grid.Resources>
.....
<Rectangle Fill="{StaticResource PinceauDegr}" ..... />

```

Les templates

Nous allons maintenant apprendre à modifier fondamentalement l'apparence de composants mais sans changer quoi que ce soit au programme. En d'autres termes : la tâche peut être confiée à un graphiste sans risque de devoir retravailler ensuite le programme. Ce sera surtout impressionnant lorsque nous passerons à Expression Blend pour personnaliser un contrôle avec Visual State Manager.

Partons d'un bouton simple, à l'apparence déjà fort acceptable (figure 15-1) :

```

<Button Content="GO !"..... />

```

Figure 15-1



La propriété `Content` est ensuite redéfinie afin de forcer l’affichage d’une image (ici, une image `.png` en ressource dont certaines parties sont transparentes, figure 15-2) :

```
<Button .....>
  <Button.Content>
    <Image Source="Sport.png" />
  </Button.Content>
</Button>
```

Figure 15-2



Dans la balise `Content`, il est possible d’insérer n’importe quel type de conteneur, par exemple un `StackPanel` à orientation verticale (figure 15-3) :

```
<Button ..... >
  <Button.Content>
    <StackPanel>
      <Image Source="Sport.png" />
      <TextBlock Text="Sport" HorizontalAlignment="Center" />
    </StackPanel>
  </Button.Content>
</Button>
```

Figure 15-3



Nous allons maintenant modifier bien plus fondamentalement l’apparence du bouton, en lui donnant une forme originale (ici, une ellipse pour ne pas compliquer inutilement l’exposé mais la forme pourrait être bien plus élaborée à l’aide d’un `Path`, voir la section « Le `Path` » du chapitre 8).

Pour cela, il faut redéfinir l’attribut `Template` du bouton, qui doit contenir une balise `ControlTemplate` :

```
<Button ..... >
  <Button.Template>
    <ControlTemplate>
      <Ellipse Fill="Gray" />
    </ControlTemplate>
  </Button.Template>
</Button>
```

Figure 15-4



Le bouton est maintenant limité à une ellipse grise (figure 15-4), ce qui est certes peu attrayant mais nous n'en sommes encore qu'au début. Si l'utilisateur clique en dehors de l'ellipse, cela n'a aucun effet. En revanche, s'il clique dessus, un événement `Click` est généré (rien de changé donc par rapport au fonctionnement initial), bien qu'aucun effet visuel ne signale à l'utilisateur que le clic a été pris en compte. Nous réglerons cela bientôt.

Plaçons maintenant un texte à l'intérieur de l'ellipse, ce qui est conforme à la pratique (figure 15-5) :

```
<Button ..... >
  <Button.Template>
    <ControlTemplate>
      <Grid>
        <Ellipse Fill="Gray" />
        <TextBlock Text="GO" Foreground="White"
          HorizontalAlignment="Center"
          VerticalAlignment="Center" />
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```

Figure 15-5



Améliorons encore l'aspect esthétique du bouton en introduisant un dégradé radial, ce qui donne l'impression que le bouton est éclairé (figure 15-6) :

```
<Button ..... >
  <Button.Template>
    <ControlTemplate>
      <Grid>
        <Ellipse >
          <Ellipse.Fill>
            <RadialGradientBrush GradientOrigin="0.3, 0.3" >
              <GradientStop Offset="0" Color="White" />
              <GradientStop Offset="1" Color="Gray" />
            </RadialGradientBrush>
          </Ellipse.Fill>
        </Ellipse>
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```



```

        <TextBlock Text="GO" Foreground="White"
                FontWeight="Bold" FontFamily="Verdana"
                HorizontalAlignment="Center"
                VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>
</Button.Template>
</Button>

```

Figure 15-6



L'ellipse s'agrandit si l'utilisateur place des valeurs plus élevées dans les attributs `Width` et `Height` de la balise `Button` ou (en l'absence de ces attributs `Width` et `Height`) s'il agrandit la fenêtre du navigateur, ce qui augmente d'autant la cellule de la grille incorporant le bouton en forme d'ellipse. La police de caractères reste néanmoins inchangée.

Toutes ses caractéristiques d'affichage sont ensuite placées dans un style, ce qui n'est pas indispensable à ce stade mais de bonne pratique (cela serait indispensable pour que la nouvelle apparence s'applique à plusieurs boutons) :

```

<Application.Resources>
<Style x:Key="styleBoutonOval" TargetType="Button" >
    <Setter Property="Template" >
        <Setter.Value>
            <ControlTemplate>
                <Grid>
                    <Ellipse >
                        <Ellipse.Fill>
                            <RadialGradientBrush GradientOrigin="0.3, 0.3" >
                                <GradientStop Offset="0" Color="White" />
                                <GradientStop Offset="1" Color="Gray" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <TextBlock Text="GO" Foreground="Orange" FontWeight="Bold"
                            FontFamily="Verdana"
                            HorizontalAlignment="Center" VerticalAlignment="Center" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
</Application.Resources>

```

La balise du bouton ovale se résume maintenant à :

```
<Button Style="{StaticResource styleBoutonOval}" ..... />
```

À ce stade, certaines choses sont encore assez fixes : le libellé du bouton est codé « en dur » dans le style (inchangeable et donc le même pour tous les boutons) mais aussi la police, la couleur d’affichage du libellé, etc. Il est évidemment indispensable de pouvoir prendre en compte les desiderata des utilisateurs (ici, des programmeurs ou des graphistes qui ont inséré la balise `Button` dans le XAML). Pour cela, il convient d’utiliser les `TemplateBinding`.

Un `TemplateBinding` permet de faire référence à des attributs de la balise du composant intégrant le nouveau style. L’utilisateur de la balise (ici, `Button`) spécifie alors des valeurs pour certains attributs (de la balise `Button`) mais il n’a pas à se préoccuper des détails d’implantation de la nouvelle apparence du bouton.

La balise :

```
<TextBlock Foreground="{TemplateBinding Foreground}" ..... />
```

qui fait partie de l’implémentation du bouton signifie que cette zone d’affichage trouve la valeur de son attribut `Foreground` dans l’attribut `Foreground` du bouton.

Dans l’extrait de code suivant :

- l’attribut `Background` de la balise `Button` indique comment sera peint le fond de l’ellipse (attribut `Fill` de l’ellipse dans la nouvelle apparence) ;
- l’attribut `Foreground` de la balise `Button` donne la couleur d’affichage du libellé (attribut `Foreground` du `TextBlock` caché dans le bouton) ;
- même chose pour `FontSize`.

```
<Style x:Key="styleBoutonOval" TargetType="Button" >
  <Setter Property="Template" >
    <Setter.Value>
      <ControlTemplate TargetType="Button">
        <Grid>
          <Ellipse Fill="{TemplateBinding Background}" />
          <TextBlock Text="GO"
                    Foreground="{TemplateBinding Foreground}"
                    FontSize="{TemplateBinding FontSize}"
                    FontWeight="Bold"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center" />
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

La balise Button devient alors (par exemple) :

```
<Button .....
    Background="Gray" Foreground="White"
    FontSize="30" Style="{StaticResource styleBoutonOval}"
/>
```

Figure 15-7



Le libellé du bouton (attribut Text du TextBlock enfoui dans l'unique cellule de la grille) ne peut pas encore être personnalisé. Pour corriger ce problème, il convient de remplacer la balise TextBlock par une balise ContentPresenter qui indique comment le libellé (dans le cas le plus simple car il pourrait s'agir de la balise Content du bouton) doit être affiché :

```
<Style x:Key="styleBoutonOval" TargetType="Button" >
    <Setter Property="Template">
        <Setter.Value >
            <ControlTemplate TargetType="Button">
                <Grid>
                    <Ellipse >
                        <Ellipse.Fill>
                            <RadialGradientBrush GradientOrigin="0.3, 0.3" >
                                <GradientStop Offset="0" Color="White" />
                                <GradientStop Offset="1" Color="Gray" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <ContentPresenter Content="{TemplateBinding Content}"
                        Foreground="{TemplateBinding Foreground}"
                        FontSize="{TemplateBinding FontSize}"
                        FontWeight="Bold" HorizontalAlignment="Center"
                        VerticalAlignment="Center" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

La balise du bouton peut maintenant devenir (figure 15-8) :

```
<Button .....
    Content="GO !"
    Background="Gray" Foreground="White"
    FontSize="30" Style="{StaticResource styleBoutonOval}"
/>
```

Figure 15-8



À noter qu'elle pourrait contenir une balise `Content` bien plus élaborée, qui serait reprise telle quelle dans le bouton.

Donner un feedback visuel

À ce stade, aucun signal visuel n'est encore fourni quand l'utilisateur clique sur le bouton ou lorsque la souris le survole. Voyons à présent comment fournir ces signaux visuels.

Afin de nous concentrer sur la manière de réaliser les transitions (par exemple, la transition de l'état « normal » à l'état « survol »), partons d'un style de bouton encore plus simple que celui de l'exemple précédent (ici, un bouton rectangulaire sans fioriture, figure 15-9) :

```
<Button Style="{StaticResource styleBouton}"
        Content="Bouton template" FontSize="20" Foreground="White" ..... />
dont le style est :
<Style TargetType="Button" x:Key="styleBouton">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Grid x:Name="RootElement">
                    <Rectangle Fill="DeepSkyBlue" />
                    <Rectangle x:Name="FocusVisualElement"
                               Fill="Yellow" />
                    <ContentPresenter
                        Content="{TemplateBinding Content}"
                        Foreground="{TemplateBinding Foreground}"
                        FontFamily="Verdana"
                        FontSize="{TemplateBinding FontSize}"
                        HorizontalAlignment="Center"
                        VerticalAlignment="Center" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Figure 15-9



Nous avons défini deux rectangles (de manière générale deux éléments UI) correspondant à deux états du bouton. Les noms internes (attribut `x:Name`) attribués à ces rectangles doivent être :

- `RootElement` pour la représentation de l'état « normal » ;
- `FocusVisualElement` lorsque le bouton reçoit le focus.

Le bouton est ainsi uniformément bleu à l'état normal et jaune lorsqu'il a le focus (la touche Entrée du clavier a alors le même effet qu'un clic sur le bouton).

Nous allons ajouter des transitions et améliorer encore l'apparence du bouton (difficile en effet d'y résister...). Pour cela, des bords arrondis sont spécifiés pour le bouton à l'état « normal » ainsi qu'un effet de transparence (figure 15-10) obtenu à l'aide de deux couches (avec dégradé de transparence sur le second rectangle, voir la section « Le masque d'opacité » du chapitre 4) :

```
<Rectangle x:Name="Normal1"
    Fill="DeepSkyBlue" RadiusY="5" RadiusX="5" />
<Rectangle x:Name="Normal2"
    RadiusY="5" RadiusX="5" Stroke="Black">
    <Rectangle.Fill>
    <LinearGradientBrush
        StartPoint="0.5,0" EndPoint="0.5,1">
    <LinearGradientBrush.GradientStops>
        <GradientStop Color="#D0FFFFFF" Offset="0" />
        <GradientStop Color="#90FFFFFF" Offset="0.5" />
        <GradientStop Color="#60FFFFFF" Offset="0.5" />
        <GradientStop Color="#90FFFFFF" Offset="1" />
    </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

Figure 15-10



Nous allons à présent ajouter les transitions. Pour cela, il faut définir des story-boards aux noms bien précis et à la signification évidente (il s'agit en effet de créer des animations mais rien n'empêche qu'elles soient instantanées) :

```
<Grid x:Name="RootElement">
    <Grid.Resources>
        <Storyboard x:Key="MouseOver State">
            ....
        </Storyboard>
        <Storyboard x:Key="Normal State">
```

```

.....
</Storyboard>
<Storyboard x:Key="Pressed State">
.....
</Storyboard>
<Storyboard x:Key="Disabled State">
.....
</Storyboard>
</Grid.Resources>
</Grid>

```

Intéressons-nous d'abord à l'état « survol » qu'est `MouseOver` (quand la souris entre dans la surface du bouton) :

```

<Storyboard x:Key="MouseOver State">
  <ColorAnimation Duration="0:0:0.3"
    Storyboard.TargetName="Normal1"
    Storyboard.TargetProperty=
      "(Rectangle.Fill).(SolidColorBrush.Color)"
    To="Silver"
  />
</Storyboard>

```

La transition spécifiée dans cet extrait de code permet de modifier la couleur du premier rectangle en trois dixièmes de seconde. À noter qu'elle pourrait être instantanée en spécifiant 0 dans `Duration`.

La ligne `TargetProperty`, même si elle paraît un peu plus compliquée, nous évite de devoir expliciter l'attribut `Fill` du rectangle `Normal1` et de devoir donner un nom à la couleur. Cette ligne peut donc se lire : « dans l'attribut `Fill` du rectangle, un `SolidColorBrush` et plus précisément son attribut `Color` ».

Modifions maintenant l'apparence du bouton qui a le focus : selon l'usage, on trace une ligne en pointillés autour du libellé (figure 15-11) :

```

<Rectangle x:Name="FocusVisualElement"
  RadiusY="5" RadiusX="5" Visibility="Collapsed"
  Stroke="White" StrokeThickness="2"
  StrokeDashArray="2 1" Margin="10" />

```

Figure 15-11



Au moment du clic sur le bouton (état « pressé »), nous épaississons sa bordure (figure 15-12) :

```

<Storyboard x:Key="Pressed State">
  <DoubleAnimation Duration="0:0:0"
    Storyboard.TargetName="Normal2"

```

```
Storyboard.TargetProperty="StrokeThickness"  
To="5" />  
</Storyboard>
```

Figure 15-12



Nous venons de voir comment réaliser des boutons personnalisés avec signaux visuels lors de changements d'état. Il n'y a plus qu'à faire preuve de créativité...

Les styles des composants Silverlight de Microsoft

La créativité se nourrissant des meilleurs exemples, il peut être enrichissant de jeter un coup d'œil aux styles créés par Microsoft pour ses composants Silverlight. Ceux-ci sont incorporés dans la DLL contenant le code des composants Silverlight. Il ne reste donc plus qu'à jouer au petit explorateur...

Lutz Roeder a développé pour cela Reflector for .NET, un fantastique outil de décompilation et d'exploration de programmes, y compris les DLL. Cet outil peut être téléchargé gratuitement à partir du site <http://www.aisto.com/roeder/dotnet>.

Lancez Reflector en lui demandant d'analyser des DLL Silverlight. Les styles qui nous intéressent sont stockés en ressources dans `System.Windows.Controls`, soit dans `System.Windows.Controls.g.resources`. Reflector signale que les styles proviennent de `generic.xaml` et vous propose d'enregistrer ce fichier sur le disque.

Analysons maintenant le fichier `generic.xaml` ainsi créé. Il contient les balises du bouton dans `<Style Target="Button" >`, ainsi que les styles par défaut pour `CheckBox`, `ContentControl`, `HyperlinkButton`, `ListBox`, `ListBoxItem`, `RadioButton`, `RepeatButton`, `ScrollBar`, `ScrollViewer`, `ToggleButton` et `ToolTip`.

Modifier n'importe quel contrôle avec Expression Blend

Modification de l'apparence

En passant à Expression Blend, un graphiste peut modifier n'importe quel élément de n'importe quel contrôle : son apparence mais aussi les transitions lors de changements d'état (par exemple, un passage à l'état `MouseOver` ou l'indication d'un clic). Ceci est rendu possible grâce au Visual State Manager d'Expression Blend.

Pour illustrer cette fonctionnalité, partons d'un projet contenant une barre de défilement (figure 15-13), dont aucun attribut ne permet de modifier fondamentalement l'apparence :

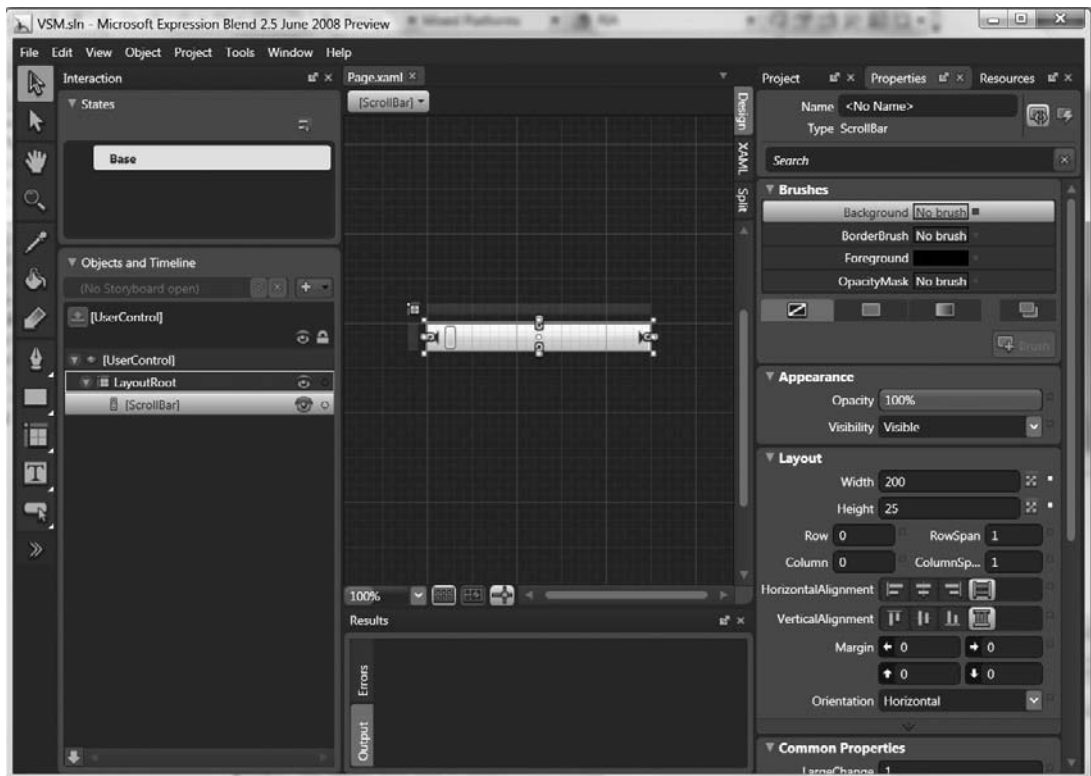
```
<ScrollBar Width="200" Height="20" />
```

Figure 15-13



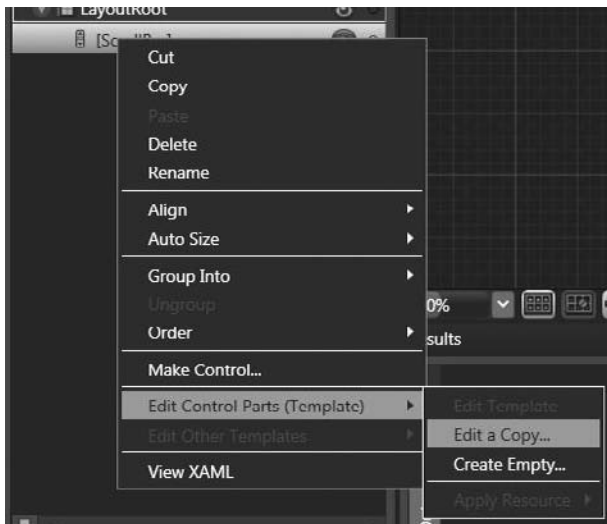
Nous allons modifier le curseur (*thumb* en anglais), sachant que la technique est applicable à n'importe quelle partie constitutive de n'importe quel contrôle. Pour cela, passez tout d'abord à Expression Blend en effectuant un clic droit sur *Page.xaml* dans l'Explorateur de solutions de Visual Studio et en sélectionnant *Open in Expression Blend*.

Figure 15-14



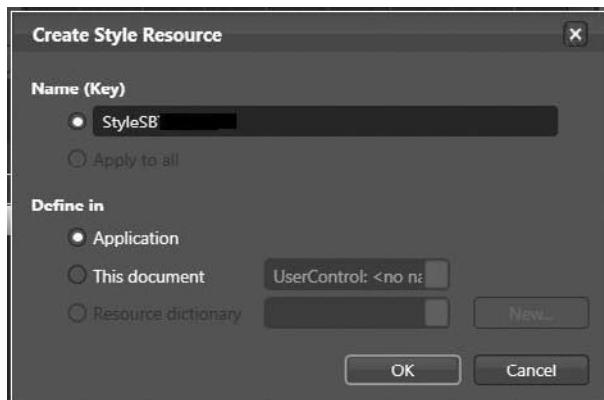
Sélectionnez ensuite le contrôle à modifier (figure 15-14), effectuez un clic droit dessus et choisissez Edit Control Parts (Template)>Edit a copy... (figure 15-15). Par la suite, il sera possible d'appliquer la modification à d'autres barres de défilement.

Figure 15-15



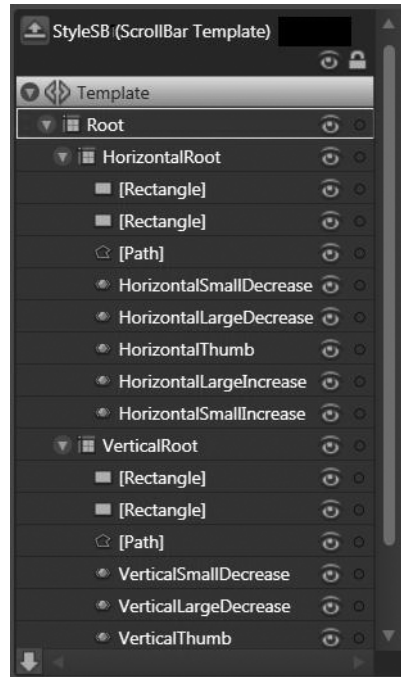
Dans la mesure où il est impossible de modifier les styles utilisés par défaut par Silverlight, il convient de travailler sur une copie de ceux-ci. Expression Blend vous propose d'attribuer un nom (ici, StyleSB) au style qui va être créé suite aux modifications et de spécifier s'il devra être créé dans le fichier App.xaml ou Page.xaml (figure 15-16).

Figure 15-16



Expression Blend affiche alors les éléments de base (surtout des rectangles) qui participent à la création de la barre de défilement (figure 15-17).

Figure 15-17



Pour cet exemple, l'élément `HorizontalThumb` va être modifié : sélectionnez-le agrandissez-le (figure 15-18).

Pour le colorier, il faut aller plus en avant dans le `HorizontalThumb`, qui est composé de plusieurs éléments. Pour les découvrir, effectuez un clic droit sur l'élément `HorizontalThumb` et sélectionnez `Edit Control Parts (Template)>Edit Template` (figure 15-19).

Figure 15-18

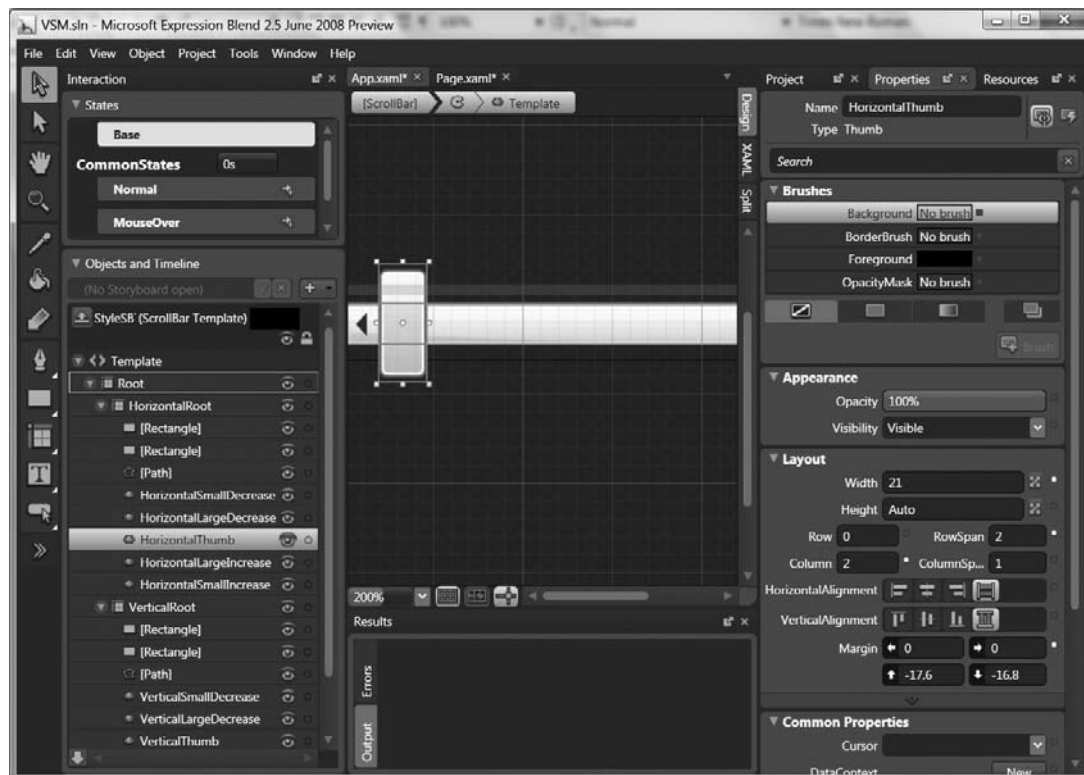
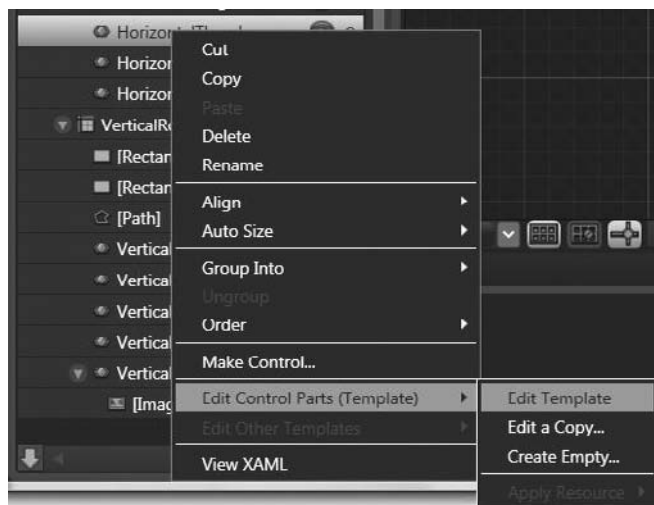
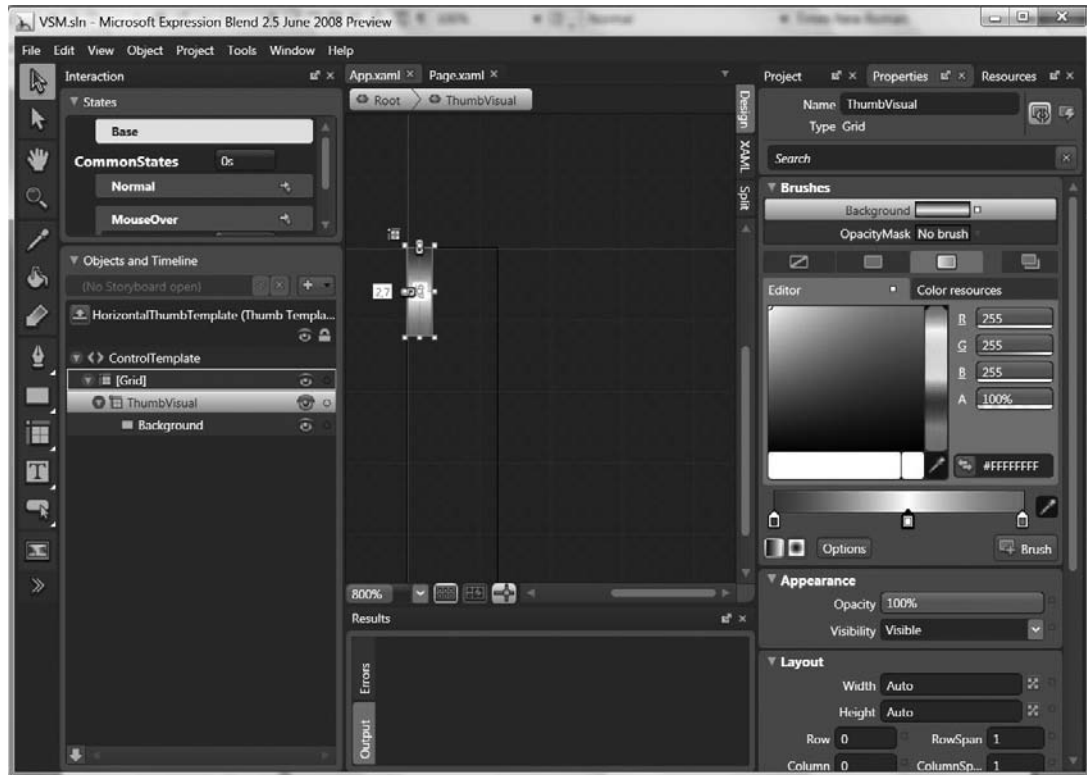


Figure 15-19



Le curseur `HorizontalThumb` est créé à partir d'une grille qu'il est possible de sélectionner et de colorier (ici, un dégradé du bleu au rouge avec le blanc comme couleur intermédiaire), comme nous l'avons vu au chapitre 3 (figure 15-20).

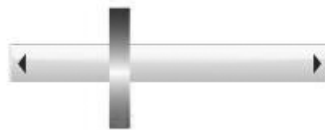
Figure 15-20



Après sauvegarde dans Expression Blend, retournez dans Visual Studio. Celui-ci détecte la modification effectuée dans Expression Blend et vous demande s'il faut en tenir compte. Le fichier `App.xaml` a été automatiquement modifié pour y ajouter un style de barre de défilement.

Après compilation, la barre de défilement présente un curseur rectangulaire et peint avec le dégradé spécifié précédemment (figure 15-21).

Figure 15-21



Modification des transitions

Il est possible, avec Expression Blend, de modifier les animations (éventuellement instantanées) lors du passage d'un état à un autre (par exemple, lors de l'entrée de la souris dans la surface du composant ou au moment du clic, pour donner une indication visuelle à l'utilisateur).

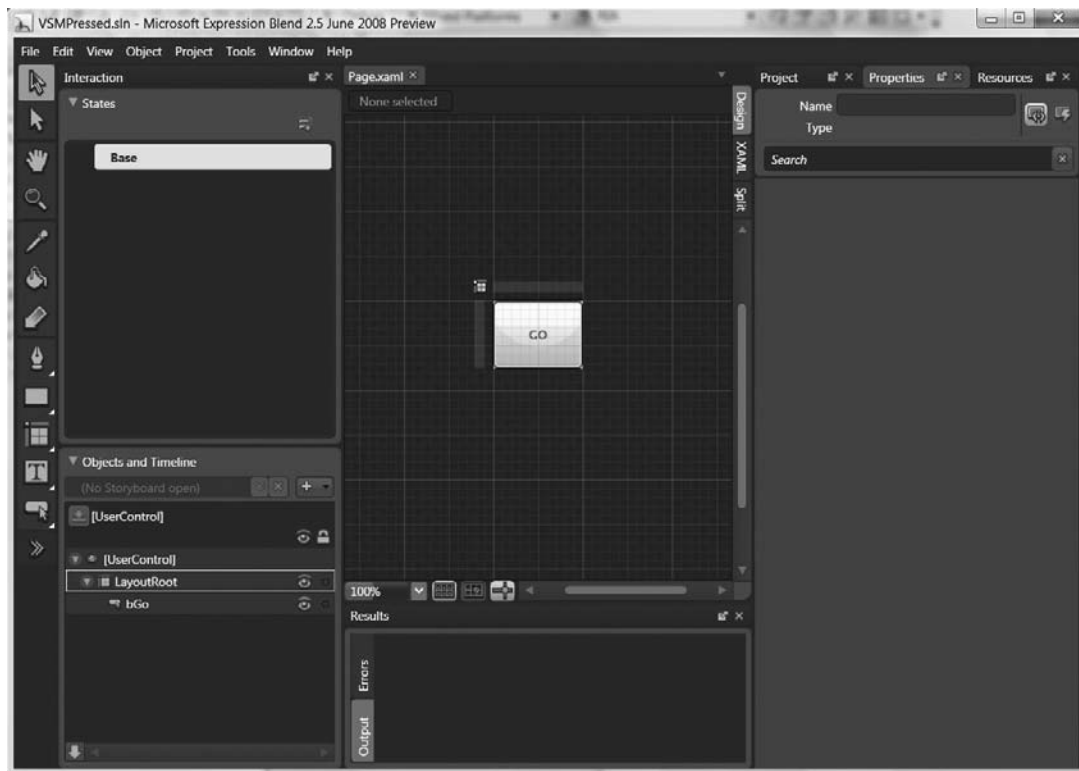
Pour illustrer le sujet, nous allons réaliser l'animation suivante : lorsque l'utilisateur clique sur un bouton (passage à l'état Pressed), celui devient plus petit puis reprend aussitôt son état normal.

Partons du bouton suivant :

```
<Button x:Name="bGo" Content="GO" Width="80" Height="60" />
```

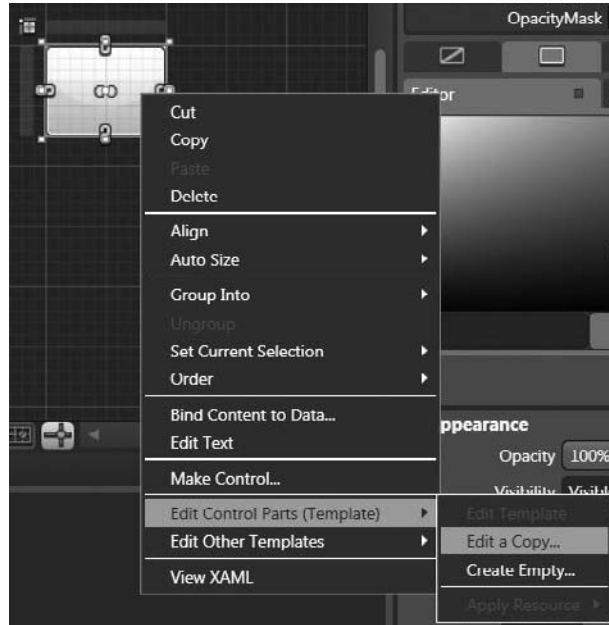
et passons directement à Expression Blend en cliquant droit sur le fichier `Page.xaml` et en sélectionnant Ouvrir dans Expression Blend (figure 15-22).

Figure 15-22



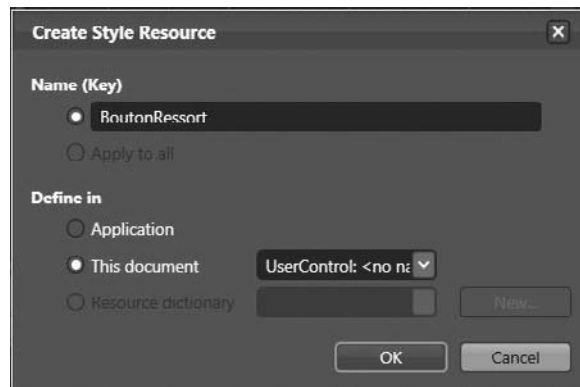
Créez ensuite un style pour ce bouton, basé sur le style général (existant) des boutons. Pour cela, effectuez un clic droit sur le bouton et sélectionnez Edit Control Parts (Template)>Edit a Copy... (figure 15-23).

Figure 15-23



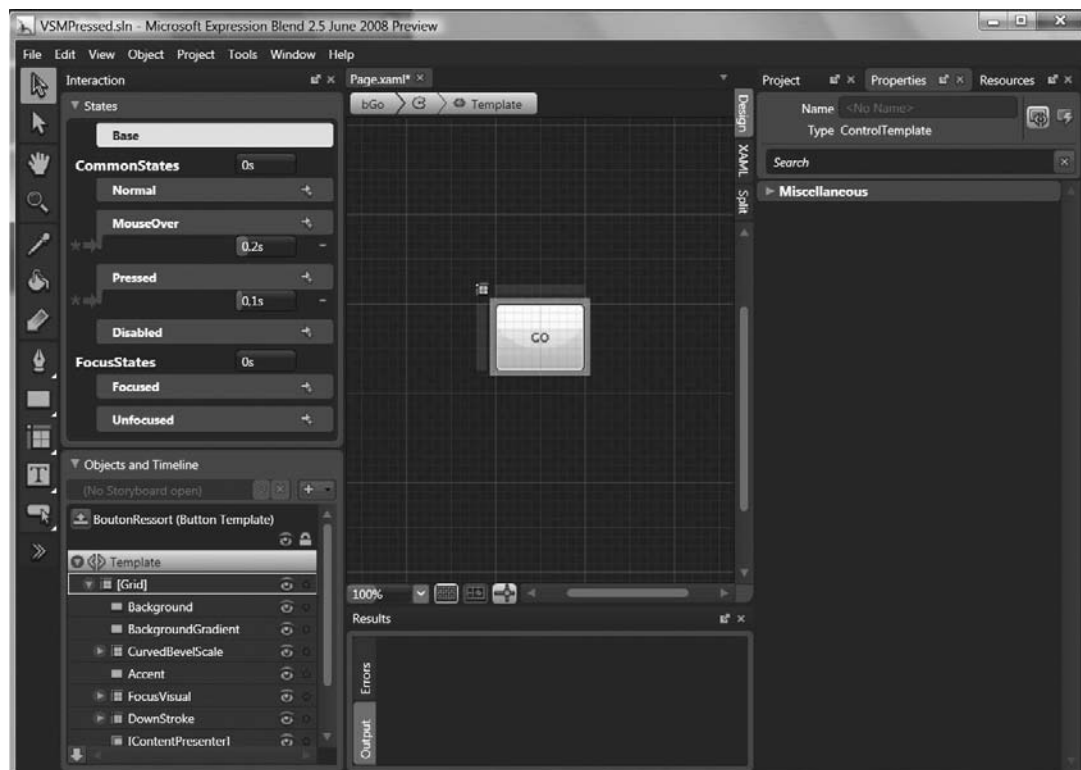
Attribuez un nom à ce style (BoutonRessort, figure 15-24).

Figure 15-24



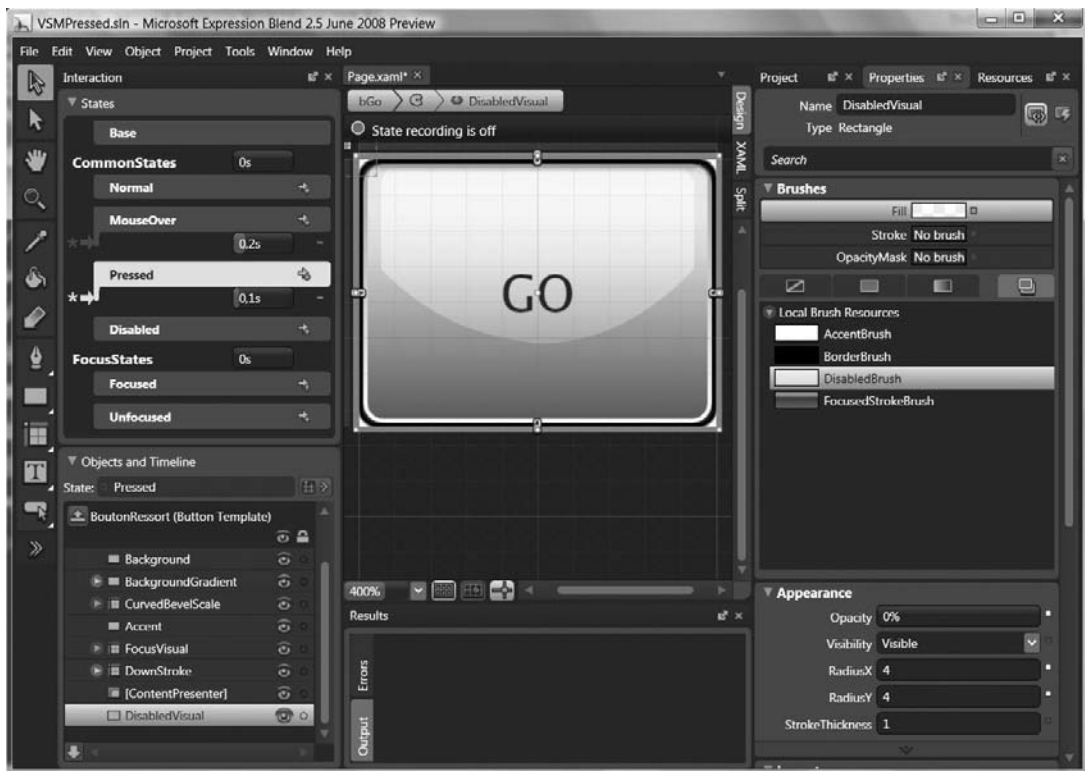
Expression Blend est maintenant prêt à créer un nouveau style. Nous pourrions modifier chaque élément constitutif du bouton (ils apparaissent dans le panneau Objects and Timeline) mais aussi les animations lors de changements d'état (dans le panneau Interaction, figure 15-25) :

Figure 15-25



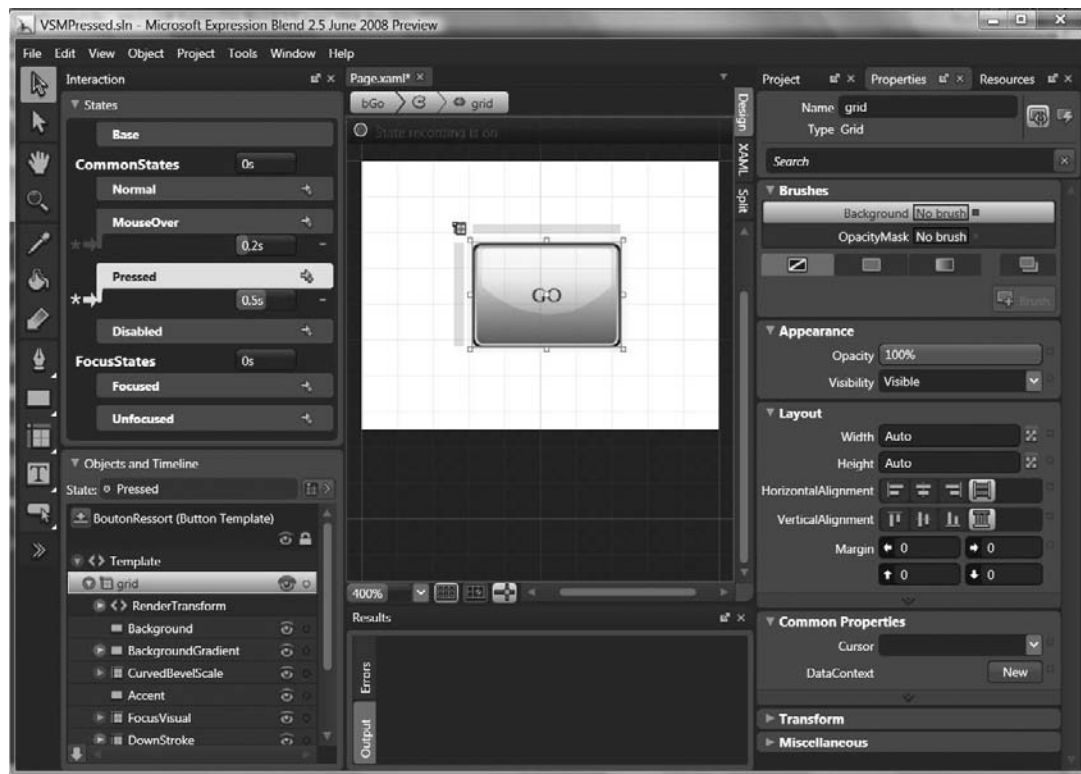
Nous allons modifier l'animation qui indique à l'utilisateur que le bouton a bien été pressé (état Pressed). Pour travailler plus confortablement, agrandissez (à 400 %, figure 15-26) la surface de travail (tout se passe en effet dans le bouton).

Figure 15-26



Au moment du clic, le bouton deviendra nettement plus petit. Comme c'est tout le bouton qui est concerné, et non un élément particulier de celui-ci, il convient de s'assurer que le conteneur d'implémentation du bouton (qui est une grille) est sélectionné (article *Grid* situé juste sous *Template* dans le panneau *Objects and Timeline*) avant de choisir l'état *Pressed* dans le panneau *Interaction* (figure 15-27).

Figure 15-27



Un bouton rond et rouge (étiqueté *State recording is on* dans la partie supérieure de la surface de travail) s'allume alors, indiquant qu'Expression Blend est prêt à enregistrer des modifications. Réduisez la taille du bouton en déplaçant le coin supérieur gauche puis le coin inférieur droit (afin que le bouton rétrécisse mais en gardant son centre fixe). Vous pouvez également modifier la durée de l'animation en cliquant et en déplaçant la souris sur l'indication de durée, sous la barre *Pressed*.

Une fois l'opération terminée, cliquez sur le bouton rond et rouge, qui devient gris, maintenant étiqueté *State recording is off*.

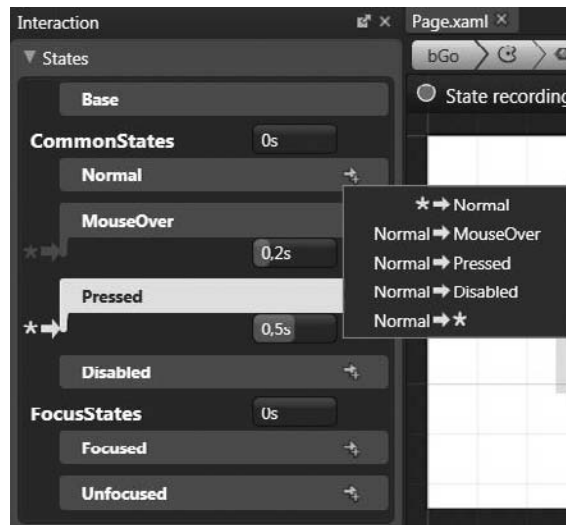
Expression Blend a créé un nouveau style (BoutonRessort) et la balise du bouton est devenue :

```
<Button x:Name="bGo" Content="GO" Width="80" Height="60"  
        Style="{StaticResource BoutonRessort}" />
```

Enregistrez le tout (menu File>Save All) et retournez à Visual Studio pour tester le nouveau bouton (on pourrait le faire sous Expression Blend mais les programmeurs ne s'y sentent pas à l'aise et ne font confiance qu'à Visual Studio...).

Il est possible de spécifier des animations pour des transitions bien particulières (figure 15-28).

Figure 15-28



Interaction Silverlight/HTML

Blocs Silverlight dans une page Web

Une page Web n'est pas nécessairement entièrement Silverlight. Elle peut en effet être constituée d'éléments HTML traditionnels (boutons, textes et zones d'édition, boîtes de liste, etc., mais à la mode HTML, ce qui paraît déjà tristounet) ainsi que de blocs Silverlight. Introduire un ou plusieurs blocs Silverlight dans une page HTML (ou PHP ou ASP.NET) permet de l'améliorer avant, éventuellement, de la réécrire plus tard en application Silverlight.

Apprenons donc à créer une page HTML en incorporant un élément Silverlight. Pour cela, nous allons créer une nouvelle application Silverlight (appelons-la SLProg) et examiner ce qui est généré par Visual Studio dans le fichier d'extension .html (fichier SLProg-TestPage.html par défaut, dans le répertoire SLProg_Web). Le code suivant correspond à la partie body de ce fichier .html :

```
<body>
<!-- Runtime errors from Silverlight will be displayed here.
This will contain debugging information and should be removed or hidden when
debugging is completed -->
<div id='errorLocation' style="font-size: small;color: Gray;"></div>
<div id="silverlightControlHost">
  <object data="data:application/x-silverlight,"
    type="application/x-silverlight-2-b1" width="100%" height="100%">
    <param name="source" value="ClientBin/HtmlPlusSL.xap"/>
    <param name="onerror" value="onSilverlightError" />
    <param name="background" value="white" />
    <a href="http://go.microsoft.com/fwlink/?LinkId=108182"
      style="text-decoration: none;">
      
```

```

    </a>
  </object>
  <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
</div>
</body>

```

Dans le fichier .aspx (fichier nommé SLProgTestPage.aspx par défaut, dans le répertoire SLProg_Web), les choses paraissent plus simples puisque la partie Silverlight se résume à la balise `asp:Silverlight`. La mécanique sous-jacente apparaîtra cependant plus clairement avec les pages HTML.

Pour rappel, pour rendre des pages accessibles sur Internet, il faut déployer soit le fichier .html, soit le fichier .aspx sur le serveur chez l'hébergeur (sans oublier, dans le cas d'un déploiement Silverlight, de créer un sous-répertoire `ClientBin` et d'y copier également le fichier XAP). La solution ASPX n'est possible que si l'hébergeur supporte les hébergements ASP.NET.

Nous allons tout d'abord modifier le fichier HTML en ajoutant une balise `table` de trois lignes, avec trois cellules en deuxième ligne. Dans la deuxième de ces trois cellules (balise `td`), on insère un élément Silverlight. Il suffit pour cela de déplacer les balises :

```

<div id="silverlightControlHost">
  .....
</div>

```

dans cette cellule de grille :

```

<body >
  .....
  <table width="100%" border="1" >
<tr>
<td colspan="3">Première ligne</td>
</tr>
<tr>
<td>Ligne 2, colonne 1</td>
<td style="width:400px;height:300px">
  <div id="silverlightControlHost">
    <object data="data:application/x-silverlight,"
      type="application/x-silverlight-2-b1" width="100%" height="100%">
      <param name="source" value="ClientBin/HtmlPlusSL.xap"/>
      <param name="onerror" value="onSilverlightError" />
      <param name="background" value="white" />
      <a href="http://go.microsoft.com/fwlink/?LinkId=108182"
        style="text-decoration: none;">
        
      </a>
    </object>
    <iframe style='visibility:hidden;height:0;width:0;border:0px'></iframe>
  </div>
</td>
<td >Ligne 2, colonne 3</td>
</tr>

```

```

<tr>
  <td colspan="3">Troisième ligne</td>
</tr>
</table>
</body>

```

Il reste à écrire (dans `Page.xaml`) une « application » Silverlight, ici relativement simple (texte Silverlight qui tourne en permanence en superposition du logo Silverlight). Cet élément Silverlight est inséré en deuxième cellule de la deuxième rangée du tableau :

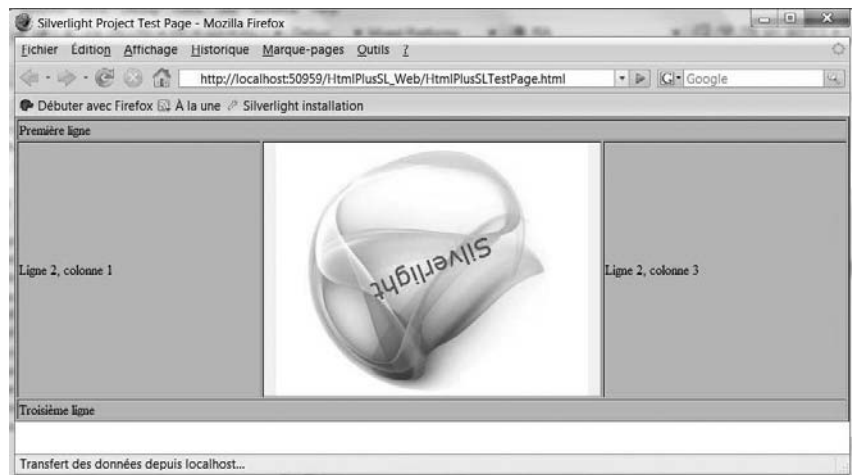
```

<Grid x:Name="LayoutRoot" Background="Beige" >
  <Grid.Triggers>
    <EventTrigger RoutedEvent="TextBlock.Loaded" >
      <BeginStoryboard >
        <Storyboard>
          <DoubleAnimation Storyboard.TargetName="rotSilverlight"
                           Storyboard.TargetProperty="Angle"
                           From="0" To="360" Duration="0:0:10"
                           RepeatBehavior="Forever"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Grid.Triggers>
  <Image Source="SilverlightLogo.jpg" />
  <TextBlock Text="Silverlight"
             FontFamily="Verdana" FontSize="30" Foreground="Red"
             HorizontalAlignment="Center" VerticalAlignment="Center"
             RenderTransformOrigin="0.5, 0.5">
    <TextBlock.RenderTransform>
      <RotateTransform x:Name="rotSilverlight" />
    </TextBlock.RenderTransform>
  </TextBlock>
</Grid>

```

La figure 16-1 illustre le résultat obtenu.

Figure 16-1



Accès aux éléments HTML depuis Silverlight

Depuis quelques années déjà, il est possible d'accéder aux éléments HTML et de les manipuler à partir du JavaScript injecté dans la page. La technique utilisée à cet effet est connue sous le nom de DOM (*Document Object Model*) et est standardisée par le comité de standardisation W3C. Tous les navigateurs aujourd'hui sur le marché ont adopté ce modèle.

Il est possible d'accéder à ces éléments HTML et de les manipuler à partir du code Silverlight (code écrit en C# ou VB). Cette technique s'avère plus puissante sans être plus compliquée (grâce aux classes .NET) et nettement plus efficace que JavaScript. En effet, JavaScript interprète du texte à l'état brut tandis que le run-time Silverlight opère sur du code compilé par le compilateur C# ou VB.

Voyons quelles sont les différentes étapes à réaliser pour arriver à ce résultat.

En tête du programme (dans le fichier nommé par défaut `Page.xaml.cs` ou `Page.xaml.vb`), vous devez tout d'abord saisir les directives suivantes :

- `using System.Windows.Browser;` en C# ;
- `Imports System.Windows.Browser` en VB.

Avant de voir comment modifier les éléments HTML, présentons `HtmlPage`, qui fournit essentiellement des informations sur le navigateur et permet de naviguer vers une autre page Web (tableau 16-1).

Tableau 16-1 – Les objets statiques de `HtmlPage`

Nom de l'objet	Description
BrowserInformation	Contient des informations sur le navigateur, avec les champs suivants :
	BrowserVersion Information spécifique au navigateur.
	Name Nom du navigateur (par exemple, Microsoft Internet Explorer ou Netscape).
	Platform Nom de la plate-forme (par exemple, Win32 si l'utilisateur est sous Windows).
	UserAgent Contient (parmi d'autres choses) une chaîne telle que MSIE ou Firefox, ainsi qu'un numéro de version.
Navigate	Fonction qui permet une redirection sur une autre page du site ou un autre site Web. Par exemple : <code>HtmlPage.Navigate("http://www.microsoft.com");</code>

Pour vérifier si une chaîne contient une autre chaîne (par exemple, la chaîne MSIE ou Mac, il est possible d'écrire :

```
if (HtmlPage.BrowserInformation.UserAgent.Contains("MSIE")) ...
```

ou :

```
int n = HtmlPage.BrowserInformation.UserAgent.IndexOf("MSIE");
```

où `n` est égal à la position de MSIE dans `UserAgent` ou à -1 si la chaîne recherchée n'est pas présente.

HtmlPage donne surtout accès à l'objet « document » (de type HtmlDocument), qui est le cœur du DOM et qui permet d'accéder aux éléments HTML de la page Web :

```
HtmlDocument doc = HtmlPage.Document;
```

Voyons maintenant, à travers des exemples pratiques, ce que nous pouvons faire avec cette variable doc.

Modifier les attributs de la page

En partant de cette variable doc, il est possible de modifier le titre de la page (qui peut être une page Web, une page partiellement Silverlight ou une page entièrement Silverlight). Pour cela, il faut modifier la balise title dans la section head de la page HTML :

```
doc.SetProperty("title", "Nouveau titre");
```

Accéder aux éléments HTML

Comme recommandé par le comité de normalisation W3C, un élément HTML (vu par le code Silverlight comme un objet HtmlElement) est référencé comme suit (xyz devant être remplacé par la valeur de l'attribut id de l'élément HTML) :

```
HtmlElement elem = doc.GetElementById("xyz");
```

Voyons comment accéder à des éléments HTML spécifiques et comment les manipuler à partir du code Silverlight, en utilisant les fonctions GetAttribute, SetAttribute ainsi que GetStyleAttribute et SetStyleAttribute (pour les attributs de style).

Pour modifier le libellé d'une balise span ou changer sa couleur de fond et celle d'affichage, saisissez le code suivant :

```
<span id="zaMsg">Hello</span>
.....
HtmlElement elem = doc.GetElementById("zaMsg");
elem.SetAttribute("innerHTML", "Nouveau");
elem.SetStyleAttribute("background", "green");
elem.SetStyleAttribute("color", "white");
```

Pour lire et initialiser une zone d'édition HTML (balise input) :

```
<input id="zeNom" type="text" />
.....
HtmlElement elem = doc.GetElementById("zeNom");
string s = elem.GetAttribute("value"); // contenu de la zone d'édition dans s
elem.SetAttribute("value", "Votre nom ici"); // modification du contenu
```

Pour modifier une image (rendue par une balise HTML et non une balise Silverlight) :

```

.....
HtmlElement elem = doc.GetElementById("img");
elem.SetAttribute("src", "Vous.jpg");
```


Pour lire le numéro d'ordre de l'article sélectionné dans une boîte de liste :

```
<select id="villes">
  <option>New York</option>
  <option>Paris</option>
  <option>London</option>
</select>
.....
HtmlElement elem = doc.GetElementById("villes");
string s = elem.GetAttribute("selectedIndex");
```

où `s` contient, sous la forme d'une chaîne de caractères, le numéro d'ordre (0 pour le premier) de l'article sélectionné. Il reste à convertir `s` en un entier :

```
int n = Int32.Parse(s);
```

Pour modifier le libellé d'un bouton :

```
<input id="bGo" type="button" value="GO !" />
.....
HtmlElement elem = doc.GetElementById("bGo");
elem.SetAttribute("value", "En avant !");
```

Nous allons maintenant construire dynamiquement un tableau HTML qui sera inséré dans la première cellule de la deuxième rangée (balise `td` avec `cell1R1C0` comme id) et occupera toute la largeur de cette cellule :

```
HtmlDocument doc = HtmlPage.Document;
HtmlElement table = doc.CreateElement("table");
table.SetAttribute("width", "100%");
table.SetStyleAttribute("background-color", "Beige");
table.SetAttribute("border", "1");

// création de la première rangée
HtmlElement li0 = doc.CreateElement("tr");
HtmlElement colPays = doc.CreateElement("td"); // création de la première cellule
colPays.SetAttribute("innerText", "Pays"); // initialisation de son contenu
li0.AppendChild(colPays); // accrocher la cellule colPays à la ligne li0

HtmlElement colPop = doc.CreateElement("td"); // création de la deuxième cellule
colPop.SetAttribute("innerText", "Population"); li0.AppendChild(colPop);
table.AppendChild(li0); // accrocher la ligne

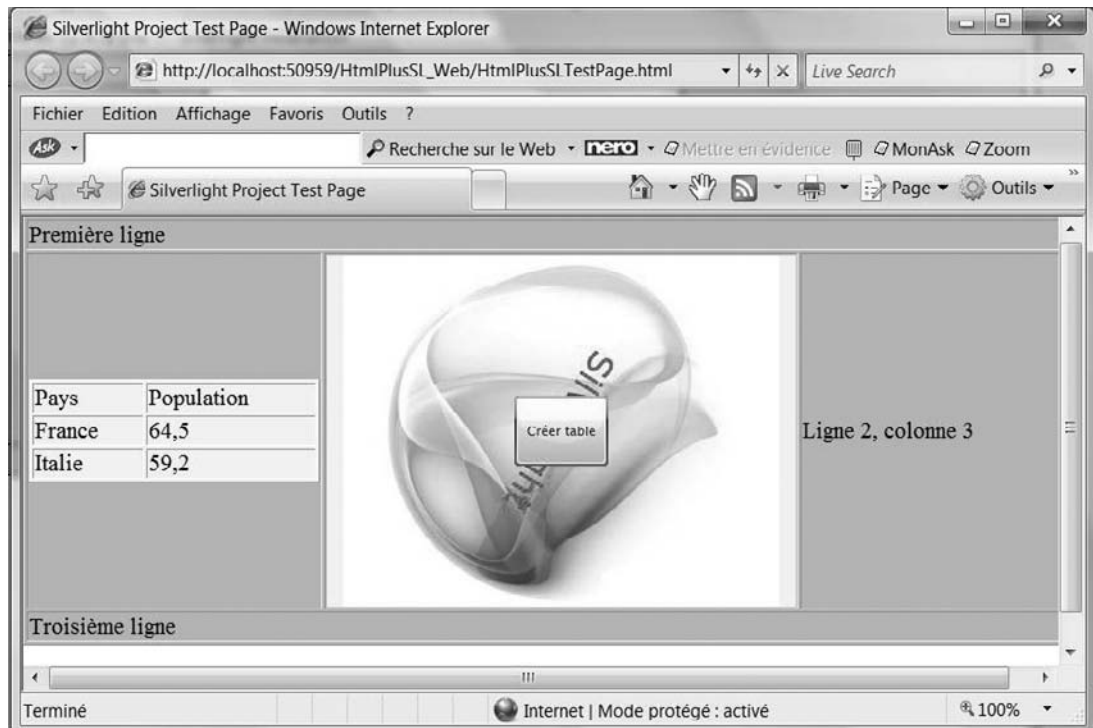
// création de la deuxième rangée
HtmlElement li1 = doc.CreateElement("tr");
HtmlElement colFrance = doc.CreateElement("td"); // création de la première cellule
colFrance.SetAttribute("innerText", "France"); li1.AppendChild(colFrance);
HtmlElement colFrancePop = doc.CreateElement("td"); // création de la deuxième cellule
colFrancePop.SetAttribute("innerText", 64.5.ToString());
li1.AppendChild(colFrancePop);
table.AppendChild(li1);
```

```
// création de la troisième rangée
HtmlElement li2 = doc.CreateElement("tr");
HtmlElement colItalie = doc.CreateElement("td");
colItalie.SetAttribute("innerText", "Italie"); li2.AppendChild(colItalie);
HtmlElement colItaliePop = doc.CreateElement("td");
colItaliePop.SetAttribute("innerText", 59.2.ToString());
li2.AppendChild(colItaliePop);
table.AppendChild(li2);

// accroch la table dans la cellule cellR1C0
doc.GetElementById("cellR1C0").SetAttribute("innerHTML",
table.GetAttribute("outerHTML"));
```

La figure 16-2 illustre le résultat obtenu dans le navigateur (le tableau HTML créé dynamiquement apparaît en première cellule de la deuxième rangée).

Figure 16-2



Il suffit ensuite de jouer sur les styles (fonction `SetStyleAttribute`) pour améliorer la présentation du tableau.

Attacher des événements

Depuis longtemps, il est possible de traiter en JavaScript des événements déclenchés par la page Web écrite en HTML. On peut maintenant faire en sorte que ces événements soient traités par du code C# ou VB.

Par exemple, pour répondre à un clic de bouton (bouton HTML, événement `onclick`) avec du code C#, il convient tout d'abord de signaler que le clic sur le bouton HTML va être traité (avec `AttachEvent`, généralement dans la fonction de Silverlight qui traite l'événement `Loaded`) :

```
<input id="bGo" type="button" value="GO !" />
.....
HtmlElement elem = doc.GetElementById("bGo");
elem.AttachEvent("onclick", new EventHandler<HtmlEventArgs>(bGo_onClick));
```

Il faut ensuite écrire la fonction C# qui va être automatiquement appelée lors d'un clic sur le bouton HTML :

```
private void bGo_onClick(object sender, HtmlEventArgs e)
{
    .....
}
```

Dans la fonction de traitement, `e.SourceElement` fait référence à l'élément HTML qui est à l'origine de l'événement (ce qui présente de l'intérêt dans le cas où une même fonction traite le clic pour plusieurs boutons).

Autre exemple : pour répondre à un changement de sélection dans une boîte de liste HTML (événement `onchange`) et copier l'article nouvellement sélectionné dans la variable `sSel`, il convient d'écrire :

```
<select id="villes">
    <option value="New York">New York</option>
    <option value="Paris">Paris</option>
    <option value="London">London</option>
</select>
.....
HtmlElement elem = doc.GetElementById("villes");
elem.AttachEvent("onchange",
    new EventHandler<HtmlEventArgs>(villes_onChange));
.....
private void villes_onChange(object sender, HtmlEventArgs e)
{
    HtmlElement elem = e.SourceElement;
    sSel = elem.GetAttribute("value");
}
```

Appeler une fonction JavaScript

Il est possible d'appeler une fonction JavaScript à partir de C# ou VB. Prenons par exemple la fonction JavaScript suivante :

```
<script type="text/javascript" >
    function f(msg)
    {
        alert(msg);
    }
</script>
```

Pour appeler la fonction `f` à partir de code managé (C# ou VB), il faut réclamer un objet `ScriptObject` relatif à `f`. La fonction JavaScript est alors appelée en exécutant `InvokeSelf` appliqué à l'objet `ScriptObject` :

```
ScriptObject jsF = (ScriptObject)HtmlPage.Window.GetProperty("f");
jsF.InvokeSelf("Blabla");
```

Une fonction JavaScript retournant une valeur numérique (par exemple, le résultat de n'importe quelle opération arithmétique) est supposée retourner une valeur de type `Double` (la notion de type n'est pas aussi précise en JavaScript qu'en C# ou VB).

Appeler une fonction C# à partir de JavaScript

Il est possible d'appeler une fonction C# à partir du JavaScript en créant une classe qui contiendra les fonctions susceptibles d'être appelées à partir du JavaScript. Pour cela, sélectionnez **Ajouter>Nouvel élément...>Classe** dans l'Explorateur de solutions, partie Silverlight. Appelons `Calcul` la classe ainsi créée, laquelle doit être marquée `ScriptableType` et les fonctions (ici, une seule) `ScriptableMember` :

```
using System.Windows.Browser;
.....
[ScriptableType]
public class Calcul
{
    [ScriptableMember]
    public int Somme(int a, int b) { return a + b; }
}
```

Dans la fonction qui traite l'événement `Loaded`, un objet de cette classe est ensuite créé puis enregistré pour JavaScript. Lors de cet enregistrement, on donne un nom (ici `CalculSL2`) par lequel le code JavaScript pourra avoir accès à la classe :

```
using System.Windows.Browser;
.....
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
    Calcul calc = new Calcul();
    HtmlPage.RegisterScriptableObject("CalculSL2", calc);
}
```

Passons au fichier HTML et insérons (dans la balise `body`) deux zones d'édition ainsi qu'un bouton. La fonction JavaScript qui traite l'événement `onclick` sur le bouton appellera la fonction `Calcul` avec, en arguments, les valeurs numériques lues dans les deux zones d'édition :

```
<body>
  N1 : <input type="text" id="ze1" /> <br /><br />
  N2 : <input type="text" id="ze2" /> <br /><br />
  <input type="button" id="bSomme" value="Somme"
    onclick="CalculSomme()" />
```

Nous donnons un nom (ici `SL2`) à la balise `object` qui permet de télécharger le code lié à l'application Silverlight (ici, essentiellement le code de la classe `Calcul`) :

```
<object id="SL2" d
```

Il nous reste à écrire le code JavaScript qui appelle la fonction C# (le `1*` dans les deux premières lignes de la fonction sert à transformer la chaîne de caractères en un format numérique) :

```
<script type="text/javascript" >
  function CalculSomme()
  {
    var N1 = 1*document.getElementById("ze1").value;
    var N2 = 1*document.getElementById("ze2").value;
    var objSL2 = document.getElementById("SL2").content;
    var res = objSL2.CalculSL2.Somme(N1, N2);
    alert(res);
  }
</script>
```

Étant donné que le fichier HTML a été modifié, c'est celui-ci qu'il faut faire exécuter. Pour cela, effectuez un clic droit sur le nom du fichier HTML dans l'Explorateur de solutions (partie Web) et sélectionnez **Afficher** dans le navigateur ou **Naviguer avec**. Il est également possible de définir la page HTML comme page de démarrage.

Si on donne la valeur 0 aux attributs `Width` et `Height` du `UserControl` dans `Page.xaml`, rien n'indique à l'utilisateur que les opérations sont en fait effectuées par du code C# compilé.

Animation Flash dans une page Silverlight

Les animations Flash ont le mérite d'exister depuis plusieurs années et d'être très répandues. Nous allons voir ici comment insérer une animation Flash dans une cellule de grille. Pour cet exemple, l'animation Flash a pour nom `FeuArtifice.swf` et sa taille est de 400×300 pixels.

Il faut d'abord ajouter une ligne (avec l'attribut `isWindowless` à `true`) dans la balise `object` (ici, dans le fichier HTML) :

```
<object data="data:application/x-silverlight,"
  type="application/x-silverlight-2-b2" width="100%" height="100%">
  ....
  <param name="isWindowless" value="true" />
  ....
</object>
```

Nous désirons que le composant Flash apparaisse dans la cellule en deuxième colonne de la deuxième rangée de la grille. Il convient donc de l'insérer en superposition d'un canevas (de la taille de l'animation Flash) occupant cette cellule :

```
<Grid x:Name="LayoutRoot" Background="Yellow" Loaded="LayoutRoot_Loaded" >
  <Grid.RowDefinitions>
    <RowDefinition Height="300"/><RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/><ColumnDefinition Width="400"/><ColumnDefinition/>
  </Grid.ColumnDefinitions>
  .....
  <Canvas x:Name="can" Width="400" Height="300" Grid.Row="1" Grid.Column="1" />
  .....
</Grid>
```

Dans le fichier HTML, il faut ensuite ajouter la balise du composant Flash, laquelle dépend du navigateur. Ceux qui utilisent Flash ont recours à divers trucs et astuces pour résoudre ce problème mais ce n'est pas le propos de cet ouvrage.

Pour Internet Explorer, la balise est :

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
  width="400" height="300">
  <param name="movie" value="FeuArtifice.swf" />
</object>
```

Pour Firefox, elle s'écrit :

```
<object type="application/x-shockwave-flash" data="FeuArtifice.swf"
  width="400" height="300"/>
```

Cette balise est insérée dans une balise div :

```
<div id="flash" style="position:absolute; z-index:2; width:400px; height:300px;
  left:10px; top:10px">
  .....
</div>
```

On pourrait s'arrêter là si le composant Flash devait toujours être affiché à partir du point (10, 10) dans la grille. Mais dans la mesure où il doit se superposer à une cellule de la grille, dont la position dépend de la taille de la fenêtre du navigateur, il faut être capable de repositionner le composant Flash quand cette fenêtre change de taille.

Pour cela, nous traitons l'événement `SizeChanged` adressé à la grille (il faut donc aussi traiter l'événement `Loaded`) :

```
private void LayoutRoot_Loaded(object sender, RoutedEventArgs e)
{
  LayoutRoot.SizeChanged += new SizeChangedEventHandler(LayoutRoot_SizeChanged);
}
```

Dans cette fonction `LayoutRoot_SizeChanged`, il faut ensuite déterminer la position du canevas `can` et repositionner le composant `Flash` en utilisant les fonctions étudiées au début de ce chapitre :

```
GeneralTransform gt = can.TransformToVisual(LayoutRoot);
Point offset = gt.Transform(new Point(0, 0));
int X = (int)offset.X;
int Y = (int)offset.Y;
HtmlDocument doc = HtmlPage.Document;
HtmlElement elem = doc.GetElementById("flash");
elem.SetStyleAttribute("left", X + "px");
elem.SetStyleAttribute("top", Y + "px");
```

Exécutez l'application `Silverlight` en effectuant un clic droit sur le fichier `HTML` dans l'Explorateur de solutions (partie `Web`) et en sélectionnant `Naviguer avec`, ou directement à partir de l'explorateur de fichiers.

Annexe

C#, VB et Visual Studio pour le développement Silverlight 2

Les applications Silverlight 2 peuvent être programmées en C# ou VB.NET, les deux principaux langages de l'environnement .NET de Microsoft. On retrouve ces deux langages, avec les mêmes outils et quasiment les mêmes classes, pour le développement d'applications Windows ou ASP.NET (programmation Web côté serveur). Les applications Silverlight 2 peuvent aussi être programmées en IronPython et IronRuby mais la communauté des développeurs Silverlight en ces deux langages est sans commune mesure avec celle des développeurs en C# et VB.

VB et surtout C# ont la réputation d'être des langages complexes, difficiles à maîtriser et par conséquent réservés à une prétendue « élite ». Même si certains ne négligent pas leurs efforts pour entretenir cette légende, écrire en C# ou en VB des applications Silverlight d'un niveau acceptable est à la portée d'un bien plus grand nombre qu'on ne le pense. Il n'est en effet pas nécessaire de maîtriser les techniques avancées de ces langages pour atteindre un niveau acceptable. Toutefois, ceci ne signifie pas qu'il s'agit de langages faciles ne réclamant aucun effort d'apprentissage. La récompense est cependant grande, tant les perspectives sont énormes pour ceux qui feraient le petit effort éventuellement nécessaire pour se former à C# ou VB.

C# et VB sont des langages dits typés et orientés objet, largement utilisés en milieu professionnel. C#, comme Java d'ailleurs, a beaucoup pris au C++, donc aussi au C mais en visant la facilité d'utilisation (source d'efficacité) et en éliminant les constructions dangereuses et inutilement compliquées du C et du C++ (par exemple, les pointeurs ne sont plus utilisés que dans des cas bien particuliers d'optimisation). Visual Studio y est pour beaucoup dans cette facilité, grâce notamment à son aide contextuelle.

Il serait regrettable que certains passent à côté d'un outil puissant et convivial au service du développement d'applications Web novatrices et spectaculaires par crainte injustifiée de ces outils. Se mettre au C# ou à VB, tout en restant raisonnable dans ses ambitions, au moins au début, est à la portée de ceux qui ont quelque peu pratiqué un langage, quel qu'il soit.

Cette annexe a pour seul but de vous mettre, si nécessaire, le pied à l'étrier mais ne remplace en rien un ouvrage complet consacré à C# ou VB (les ouvrages de formation à C# et VB dans le cadre de la programmation Windows peuvent très bien convenir). Nous espérons donc que vous poursuivrez, si nécessaire, cette étude en consultant des ouvrages spécialisés.

C# et VB offrent les mêmes possibilités et opter pour l'un ou l'autre de ces deux langages relève d'un choix personnel. À noter toutefois que la majorité des articles (et sur Silverlight 2 en particulier) sont aujourd'hui écrits en C#.

C# et VB sont des langages typés. Cela veut tout simplement dire que les variables utilisées doivent avoir été déclarées au moment d'écrire le programme, avec un nom et un type. Autrement dit, en déclarant une variable, on indique le genre de données (entier, valeur décimale, chaîne de caractères, etc.) que celle-ci va contenir. Au premier abord, cela peut paraître contraignant par rapport à des langages qui n'imposent ni de prédéclarer les variables ni de leur assigner un type bien précis. Dans la pratique cependant, on se félicite que VB soit maintenant devenu aussi typé que C#. Cela impose une rigueur bien nécessaire dans l'écriture d'un programme et permet au compilateur de détecter des problèmes (assurés ou potentiels) mais aussi de générer un code plus efficace, en évitant d'incessantes déterminations de type et conversions en cours d'exécution de programme.

C# et VB sont orientés objet, ce qui signifie qu'on travaille avec des boîtes noires, appelées classes. L'utilisateur d'une telle boîte noire (un programmeur) doit uniquement en connaître l'interface, c'est-à-dire ses fonctionnalités et la manière de l'utiliser. Le fonctionnement interne de cette boîte noire est du ressort de son concepteur (un autre programmeur ou le même). Le fonctionnement interne peut ainsi être modifié (par exemple, en vue d'une optimisation) sans répercussion sur l'interface, donc sans modification des programmes qui utilisent cette classe. Néanmoins, une recompilation s'avère souvent nécessaire.

Une classe regroupe des fonctions et des variables qui ont un lien entre elles. Certaines de ces fonctions et de ces variables sont dites publiques et font dès lors partie de l'interface : elles peuvent être utilisées par ceux qui intègrent la classe dans leurs programmes. D'autres sont privées et liées au fonctionnement interne de la classe (son implémentation) : elles ne peuvent donc être utilisées que dans le cadre de la modification de cette classe.

Partons à la découverte, certes en survol, de ces deux langages, en se limitant à ce qui est vraiment essentiel pour écrire des applications Silverlight 2.

Voyons tout d'abord comment insérer des commentaires. En C#, plusieurs lignes peuvent être mises en commentaires avec `/*` et `*/` qui permettent de délimiter la zone de commentaires. Les caractères `//` (en C#) et `'` (en VB) permettent d'indiquer que c'est le reste de la ligne qui est en commentaire. En C#, cela donne donc :

```
// ceci est un commentaire sur une ligne
/* une ou plusieurs lignes
   de commentaires */
```

et en VB :

```
' ceci est un commentaire
```

C# fait une distinction entre majuscules et minuscules et il importe donc d'être attentif à la casse (ainsi, `n` et `N` constituent des variables différentes). VB ne fait pas cette distinction mais Visual Studio corrigera automatiquement votre code si nécessaire. Par exemple, si une variable a été déclarée avec comme nom `za`, vous pouvez saisir `ZA`, `zA` ou `Za` dans le code, Visual Studio le corrigera en forçant la casse d'origine.

En VB, on écrit une instruction par ligne. Si, généralement pour des questions de présentation, la fin d'une instruction doit se poursuivre à la ligne suivante, il faut terminer la première ligne par le caractère de soulignement (`_`). Deux ou plusieurs instructions peuvent néanmoins être écrites sur une même ligne à condition de les séparer par le caractère : (deux-points).

C# et VB opèrent sur des données (contenu des variables) qui ont la même représentation en mémoire, ce qui assure l'interopérabilité entre ces langages. Voyons quels sont ces types de données.

- Une variable de type `bool` peut contenir une valeur booléenne : `true` ou `false`, correspondant à vrai ou faux.
- Une variable de type `byte` est codée sur 8 bits et peut contenir un caractère compris dans les 255 du code ANSI. C# et VB travaillent naturellement avec des lettres codées sur 16 bits (type `char`).
- Les variables de type `short`, `int` et `long` peuvent contenir des valeurs entières et sont codées respectivement sur 16, 32 et 64 bits, ce qui détermine les valeurs limites dans une variable de l'un de ces types.
- Une variable de type `char` peut contenir une lettre (ou un chiffre ou encore un signe de ponctuation) codée sur 16 bits, donc conforme à la norme Unicode qui tient compte des alphabets autres que celui en usage en Occident. Une chaîne de caractères consiste en une succession de variables de type `char`. Contrairement à ce que doivent encore faire les programmeurs en C, il est inutile de se préoccuper (avec les zéros de fin de chaîne) de la manière dont la chaîne est codée en mémoire. Une chaîne de caractères est de type `string` et le reste relève de la cuisine interne, cachée dans l'implémentation de la classe `String` correspondant au type `string`.
- Les variables de type `float` et `double` contiennent des valeurs décimales et sont codées respectivement sur 32 et 64 bits. Silverlight privilégie le type `double`, qui donne plus de précision dans la représentation en mémoire du nombre décimal (étant donné la manière de coder les nombres, une valeur comme 0.1 n'est qu'approchée, certes très approchée mais approchée quand même et plus approchée en format `double` qu'en format `float`).

Le tableau A-1 présente la correspondance des types entre C# et VB (il existe également d'autres types mais ceux mentionnés ici couvrent la majorité des besoins).

Tableau A-1 – Correspondance des types de données entre C# et VB

C#	VB
bool	Boolean
byte	Byte
char	Char
short	Short
int	Integer
long	Long
float	Single
double	Double
string	String

Une variable doit être déclarée et son type spécifié dès la phase de développement du programme. Le compilateur (qui traduit les instructions C# ou VB en code binaire) sanctionne toute utilisation d'une valeur d'un autre type dans une variable, sauf s'il y a compatibilité sans perte de précision (par exemple, l'entier 5 peut être converti sans problème en 5.0, c'est-à-dire en une variable de type double).

Une variable de type `string` ne pourra jamais contenir une valeur décimale. Mais il existe des mécanismes permettant de convertir le contenu d'une variable en une autre, d'un autre type. Ainsi, "12" désigne la chaîne de caractères formée des lettres 1 et 2, mais 12 représente la valeur numérique (entière) douze. Cela peut paraître équivalent aux yeux de beaucoup (et même dans certains langages) mais "12" et 12 sont représentés de manière très différente par les ordinateurs.

Passons aux déclarations de variables. Les lettres accentuées sont acceptées dans le nom des variables. Notez la différence dans la manière de spécifier la valeur d'un `char` (tableau A-2).

Tableau A-2 – Déclaration de variable et éventuelle initialisation en C# et VB

C#	VB
<code>int N;</code>	<code>Dim N As Integer</code>
<code>double a, b;</code>	<code>Dim a, b As Double</code>
<code>bool bOK = true;</code>	<code>Dim bOK As Boolean = True</code>
<code>string sNom = "Moi";</code>	<code>Dim sNom As String = "Moi"</code>
<code>char cLettre = 'A';</code>	<code>Dim cLettre As Char = "A"c</code>
<code>int Année = 1789;</code>	<code>Dim Année As Integer = 1789</code>
<code>double pi = 3.14159;</code>	<code>Dim pi As Double = 3.14159</code>

Les opérations arithmétiques et logiques (ET et OU) sont effectuées à l’aide d’opérateurs communs à la plupart des langages. À noter toutefois que :

- « égal » s’écrit == en C# et = en VB ;
- « différent » s’écrit != en C# et <> en VB.

En C#, le caractère / permet d’effectuer une division (comme en VB) mais il s’agit d’une division entière si les deux opérandes sont des entiers tandis que VB effectue une division réelle (voir les exemples suivants). Le reste d’une division est donné par le caractère % en C# et Mod en VB. Le tableau A-3 présente les différents opérateurs utilisés en C# et VB.

Tableau A-3 – Les différents opérateurs utilisés en C# et VB

C#	VB
== < > <= >= !=	= < > <= >= <>
+ - * /	+ - * /
%	Mod
= += -= *= /=	= += -= *=
++ --	++ --
&&	And Or
int a=7, b=2, c; double d, e; c = a / b; // c contient 3 d = a/b; // d contient 3.0 e = (double)a / b; // e contient 3.5 a += 3; // a passe à 10 a++; // a passe à 11	Dim a As Integer=7, b As Integer=2, _ c As Integer Dim d As Double c = a / b ' c contient 4 d = a / b ' d contient 3.5 a += 3 ' a passe à 10 a += 1 ' a passe à 11
c prend la valeur 3 car la division entière est effectuée, avec perte de la partie décimale. d vaut 3.0 pour la même raison. e vaut 3.5 car une division réelle a été forcée par le mécanisme du transtypage (casting en anglais).	c vaut 4 à cause de l'arrondi (supérieur à partir de .5).

Les exemples suivants (tableau A-4) illustrent la manière d'écrire les alternatives.

Tableau A-4 – Écriture d'alternatives en C# et VB

C#	VB
if (a > 10) b = 0;	If a > 10 Then b = 0
Une seule instruction à exécuter quand la condition est vraie, a et b doivent avoir été déclarées au préalable. En C#, les parenthèses sont nécessaires autour de la condition.	
if (a > 10) b = 0; else b = 1;	If a > 10 Then b = 0 Else b = 1 End If
Une seule instruction à exécuter dans les deux cas (condition vraie et condition fausse). Les accolades sont facultatives en C# quand une branche d'alternative ne contient qu'une seule instruction.	
if (a > 10) { }	If a > 10 Then End If
Plusieurs instructions à exécuter quand la condition est vraie et aucune quand la condition est fausse. Les accolades sont nécessaires en C#. remplace une ou plusieurs instructions (une instruction par ligne en VB, à moins de les séparer par le caractère :) En C#, une instruction se termine toujours par le caractère ; (point-virgule).	
if (a > 10) { } else { }	If a > 10 Then Else End If
Plusieurs instructions à exécuter quand la condition est vraie et plusieurs dans le cas « sinon ».	
switch (n) { case 0 : b = 10; break; case 1 : b = 100; break; default : b = 1000; }	Select Case n Case 0 b = 10 Case 1 b = 100 Case Else B = 1000 End Select
Instruction « selon que ». Si n vaut 0, b prend la valeur 10 et le programme sort du switch. Si n vaut 1, b prend la valeur 100 et on sort du switch. Dans tous les autres cas, b prend la valeur 1000. n et b doivent avoir été déclarées au préalable.	

Passons maintenant aux boucles (tableau A-5).

Tableau A-5 – Exemples de boucles en C# et VB

C#	VB
<pre>for (int i=0; i<10; i++) { }</pre>	<pre>For i As Integer = 0 To 9 Next i</pre>
La variable <code>i</code> n'existe qu'à l'intérieur de la boucle <code>for</code> , pour contrôler l'avancement de celle-ci. <code>i</code> est d'abord initialisée à 0, puis incrémentée (par <code>i++</code>) à la suite de chaque passage dans la boucle. La condition est ensuite évaluée et le processus se répète tant que la condition est vraie (deuxième clause du <code>for</code> en C#, sans parenthèse dans ce cas autour de la condition).	
<pre>while (n < 10) { }</pre>	<pre>Do While n < 10 Loop</pre>
Boucle « tant que ». <code>n</code> doit avoir été déclarée et initialisée avant l'entrée dans la boucle <code>while</code> . Le programme exécute les instructions du <code>while</code> (<code>Do While</code> en VB) si la condition est vraie. Il vous appartient de modifier <code>n</code> dans le corps de la boucle pour modifier la condition. Quand l'exécution tombe sur l'accolade de fin (sur <code>Loop</code> en VB), il y a retour au <code>while/Do While</code> , avec examen de la condition. Si le corps de la boucle ne contient qu'une seule instruction, les accolades sont facultatives en C#, mais les parenthèses sont nécessaires autour de la condition.	
<pre>do { } while (n < 10);</pre>	<pre>Do_ Loop While n < 10</pre>
Boucle « faire tant que ». La condition est examinée à la fin d'un passage dans la boucle dont les instructions sont donc toujours exécutées au moins une fois. Si le corps de la boucle ne contient qu'une seule instruction, les accolades sont facultatives en C#, mais les parenthèses sont nécessaires autour de la condition.	
<pre>foreach (int n in tab) { }</pre>	<pre>For Each n As Integer In tab Next i</pre>
Balayage du tableau <code>tab</code> d'entiers. <code>n</code> passe successivement (lors de chaque passage dans la boucle) par chacune des valeurs du tableau <code>tab</code> .	

Voyons maintenant comment déclarer et initialiser des tableaux dans les deux langages (tableau A-6). Notez l’utilisation des crochets en C# et des parenthèses en VB.

Tableau A-6 – Déclaration et initialisation de tableaux en C# et VB

C#	VB
<pre>int[] tab; tab = new int[5]; tab[0] = 10;</pre>	<pre>Dim tab() As Integer tab = New Integer(4){} tab(0) = 10</pre>
La première ligne, quel que soit le langage, est une déclaration de tableau : on signale au compilateur qu'on va utiliser un tableau d'entiers. À ce stade, la taille du tableau n'est pas encore spécifiée. Celui-ci est véritablement créé à la deuxième ligne (de la mémoire est réservée à ce moment pour accueillir les cinq valeurs entières du tableau, opération appelée instanciation). La première cellule du tableau prend alors la valeur 10. L'indice d'accès aux cellules du tableau commence à 0 et s'étend jusqu'à 4.	
<pre>int[] tab = {1, 2, 3};</pre>	<pre>Dim tab() As Integer = {1, 2, 3}</pre>
On déclare et initialise un tableau de trois entiers. Le compilateur se base sur le nombre de valeurs d'initialisation pour déterminer la taille du tableau.	
<pre>string nom = "Moi";</pre>	<pre>Dim nom As String = "Moi"</pre>
Déclaration et initialisation d'une chaîne de caractères.	
<pre>int[3, 2] t = {{10, 11}, {20, 21}, {30, 31} }</pre>	<pre>Dim t(3, 2) As Integer = _ {{10, 11}, {20, 21}, {30, 31} }</pre>
Déclaration et initialisation d'un tableau de trois lignes, chacune comportant deux colonnes. L'accès à la cellule située dans la seconde colonne de la première ligne se fait par t[0, 1] en C# et t(0, 1) en VB.	

La tableau A-7 montre des exemples d’opérations effectuées sur des chaînes de caractères.

Tableau A-7 – Exemples d’opérations effectuées sur des chaînes de caractères en C# et VB

C#	VB
<pre>string s = "Hello"; s += " Silverlight"; s = s.SubString(1, 6); s = s.Insert(1, 'i'); n = s.Length;</pre>	<pre>Dim s As String = "Hello" s &= " Silverlight" s = s.SubString(1, 6) s = s.Insert(1, "i"c) n = s.Length</pre>
s est d'abord initialisée à "Hello". On lui ajoute ensuite (concaténation) une autre chaîne (s contient maintenant "Hello Silverlight"). s.SubString ne modifie pas la chaîne sur laquelle porte l'opération mais renvoie une chaîne de 6 caractères à partir du deuxième. s contient donc "ello S". On insère i en deuxième position. s devient donc "eillo S". Enfin, n prend la valeur 7 (nombre de caractères dans s).	

Voyons maintenant comment écrire et appeler des fonctions (tableau A-8).

Tableau A-8 – Déclaration et appel de fonctions en C# et VB

C#	VB
<pre>void f() { } f(); // appel de f</pre>	<pre>Private Sub f() End Sub f() ' appel de f</pre>
f est une fonction qui ne reçoit aucun argument et ne renvoie aucune valeur. La présentation étant libre en C#, elle est souvent adaptée à des styles personnels ou imposés par l'entreprise (position des accolades, retraits, etc.).	
<pre>double f(int a, int b) { return (double)a / b; } int y = 2; double x = f(7, y);</pre>	<pre>Function f(ByVal a As Integer, _ ByVal b As Integer) As Double Return a / b End Function Dim y As Integer = 2 Dim x As Double = f(7, y)</pre>
f est une fonction qui reçoit deux arguments de type entier par valeur : l'argument a prend la valeur 7 et l'argument b la valeur 2. Toute modification de b dans la fonction n'a aucune répercussion sur y. f renvoie une valeur de type double (ici, 3.5).	
<pre>void g() { int a=5; f(ref a); // a vaut 10 } void f(ref int x) { x = 10; }</pre>	<pre>Private Sub g() Dim a As Integer=5 f(a) ' a vaut 10 End Sub Private Sub f(ByRef x As Integer) x = 10 End Sub</pre>
Il y a ici passage d'argument par référence : l'argument x de f se confond avec la variable (ici, a) passée en argument. Toute modification de x dans f modifie donc a de g.	

C# et VB peuvent intercepter des erreurs dans un programme (division par zéro, accès à un tableau en dehors de ses bornes, etc.), ce qui évite un « plantage », souvent catastrophique. Pour cela, on lance l'exécution des instructions qui se trouvent dans la clause `try`. En cas de problème, le programme quitte immédiatement le bloc `try` (sans exécuter les instructions qui suivent dans ce bloc) et rentre dans la clause `catch`. On dit que l'erreur a été interceptée. Le tableau A-9 présente un exemple d'exceptions.

Tableau A-9 – Exemple d'exceptions en C# et VB

C#	VB
<pre>try { } catch (Exception exc) { // msg d'erreur dans exc.Message; }</pre>	<pre>Try Catch exc As Exception ' msg d'erreur dans exc.Message End Try</pre>

Les classes jouent un rôle fondamental dans les langages orientés objet. Une classe regroupe des fonctions ainsi que des variables en relation avec ces fonctions (tableau A-10).

Tableau A-10 – Exemples de classes en C# et VB

C#	VB
<pre>class Pers { string mNom, mPrénom; // champs privés public Pers(string aNom, string aPrénom) // constructeur { mNom = aNom; // initialisation des champs } privés mPrénom = aPrénom; } public string NomComple() // fonction publique { return mPrénom + " " + mNom; } }</pre>	<pre>Class Pers Private mNom, mPrénom As String Public Sub New(ByVal aNom As String, _ ByVal aPrénom As String) mNom = aNom mPrénom = aPrénom End Sub Public Function NomComple() As String Return mPrénom & " " & mNom End Function End Class</pre>
<p>Insérez la classe <code>Pers</code> dans <code>Page.xaml.cs</code>, au-dessus de la ligne <code>class Page</code>.</p>	<p>Insérez la classe avant la ligne <code>Class Page</code> dans <code>Page.xaml.cs</code></p>
<pre>Pers p = new Pers("Chaplin", "Charlie"); s = p.NomComple(); // s contient Charlie Chaplin</pre>	<pre>Private p As New Pers("Chaplin", "Charlie") s = p.NomComple() ' s contient Charlie Chaplin</pre>

Une classe peut contenir des propriétés. Une propriété ressemble à un champ public (et s'utilise aussi facilement) mais son utilisation implique l'exécution d'instructions : celles mentionnées dans la clause get pour la lecture de la propriété et celles mentionnées dans la clause set pour une modification. Le tableau A-11 présente des exemples de propriétés.

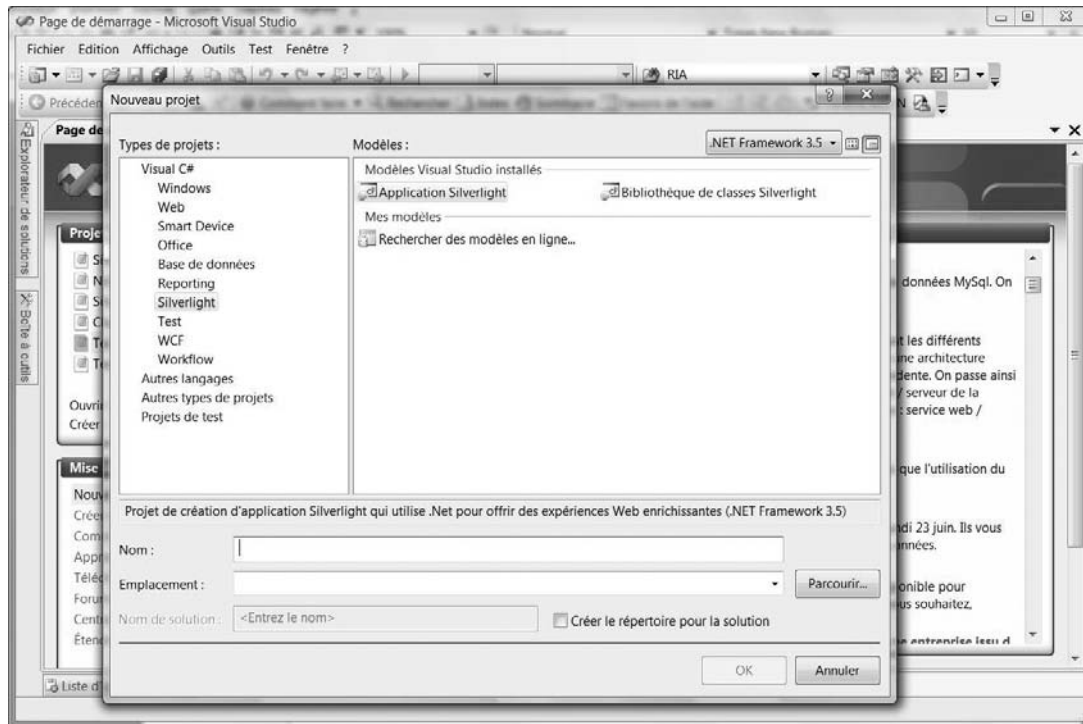
Tableau A-11 – Exemples de propriétés en C# et VB

C#	VB
<pre>class Pers { string mNom; int mAge; public Pers(string aNom, int aAge) { mNom = aNom; mAge = aAge; } public string NomAge { get { if (mAge>40) return mNom; else return mNom + " (" + mAge + ")"; } } }</pre>	<pre>Class Pers Private mNom As String Private mAge As Integer Public Sub New(ByVal aNom As String, ByVal aAge As Integer) mNom = aNom mAge = aAge End Sub Public Property NomAge() As String Get If mAge>40 Then Return mNom Else Return mNom & " (" & mAge + ")" End If End Get Set(ByVal value As String) End Set End Property End Class</pre>
Nous avons introduit ici la propriété NomAge. Pour obtenir NomAge d'une personne, le programme exécute les instructions de la clause get. Il n'y a pas de clause set ici, on ne peut donc pas écrire p.NomAge = ...;	Il n'y a pas de clause set mais Set/End Set doit être présent.
<pre>Pers p = new Pers("Elle", 30); s = p.NomAge; // s contient Elle (30)</pre>	<pre>Dim p As Pers p = New Pers("Elle", 50) s = p.NomAge ' s contient Elle</pre>

Pour apprendre un langage, rien ne vaut la pratique. Créons un projet Silverlight très simple. Pour cela, sélectionnez le menu Fichier>Nouveau>Projet, choisissez le langage souhaité, puis le menu Application Silverlight (figure A-1). Spécifiez le nom du répertoire dans lequel l'application sera créée (zone d'édition Emplacement) et attribuez un nom au projet.

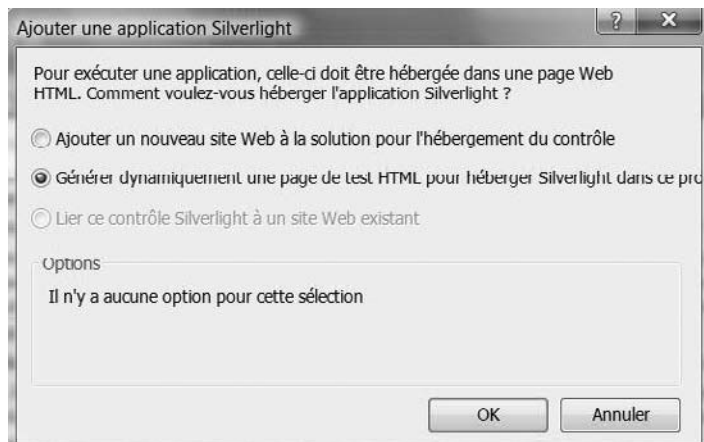
Un projet regroupe tous les fichiers nécessaires à une application. Une solution regroupe un ou plusieurs projets.

Figure A-1



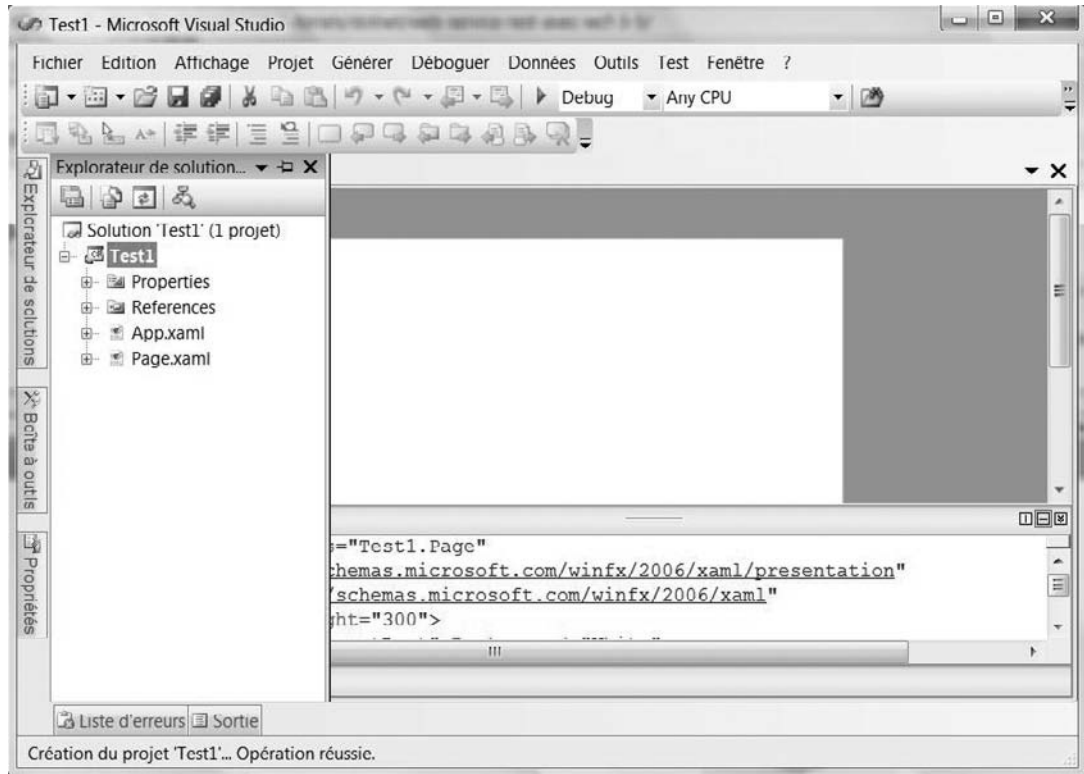
Pour un simple programme de familiarisation avec un langage, sélectionnez la plus simple des solutions, c'est-à-dire la création d'une page de test (figure A-2).

Figure A-2



La présentation de la page est décrite en XAML, dans le fichier `Page.xaml`. Double-cliquez sur ce fichier dans l'Explorateur de solutions pour l'éditer (figure A-3).

Figure A-3



On peut insérer dans la page un grand nombre de composants (figures A-4 et A-5) mais pour un apprentissage, nous nous contenterons ici d'une zone d'affichage (TextBlock), d'une zone d'édition (TextBox) et d'un bouton (Button).

Figure A-4

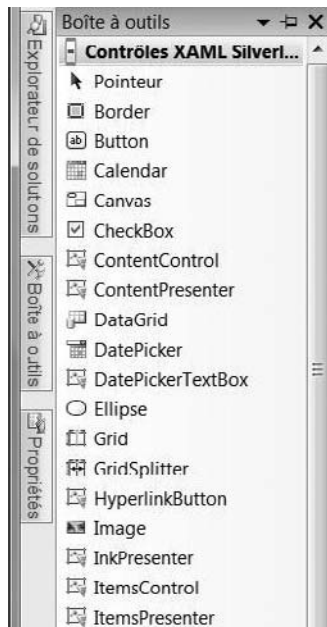
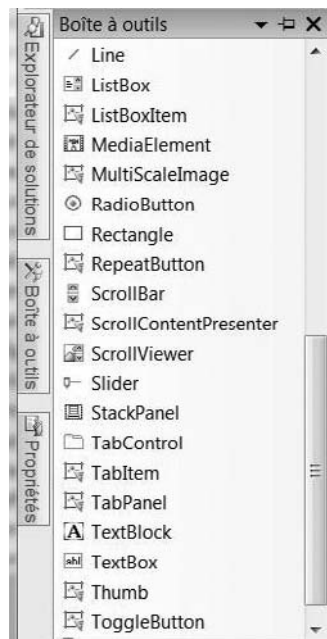
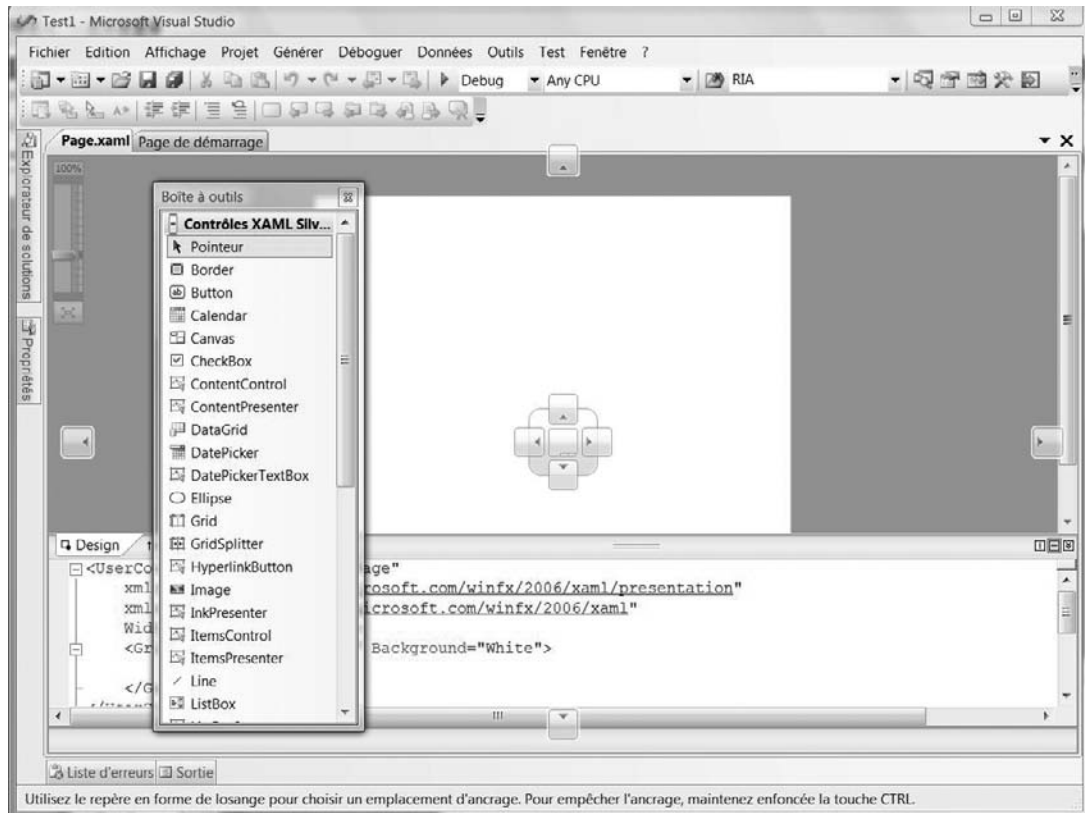


Figure A-5



Visual Studio est constitué de plusieurs fenêtres rétractables et déplaçables. Une fenêtre rétractable réagit au survol de la souris et sort de son encoche. Une fenêtre est rétractable si la punaise est en position horizontale (cliquez dessus pour inverser la position, figure A-6).

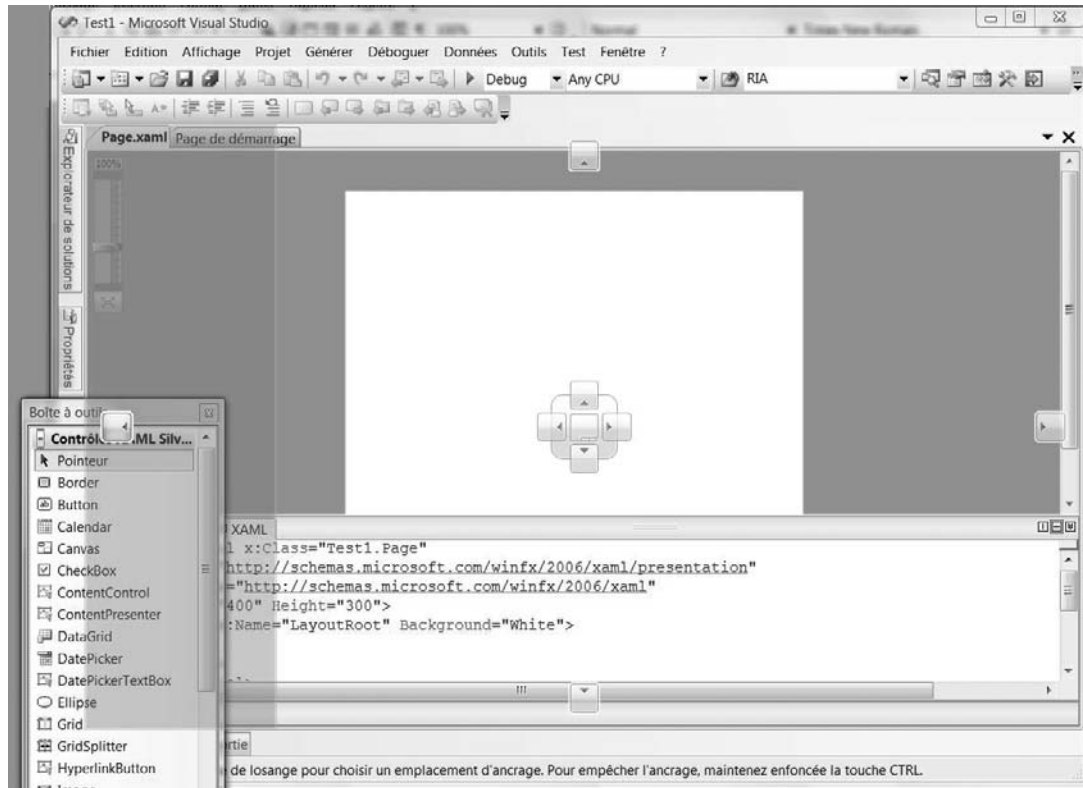
Figure A-6



Vous pouvez déplacer une fenêtre (non automatiquement rétractable) en cliquant sur sa barre de titre et en la déplaçant, bouton de la souris enfoncé.

Lors du déplacement de la fenêtre, Visual Studio affiche des zones d'ancrage privilégiées (figure A-7). Si vous faites glisser la fenêtre sur une telle zone, elle sera accolée au bord et pourra devenir automatiquement rétractable (en fonction de la position de la punaise).

Figure A-7



Pour écrire de petits programmes d'apprentissage, éditez le fichier Page.xaml et insérez les trois composants suivants :

```
<UserControl x:Class="Test1.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
    <Grid x:Name="LayoutRoot" Background="White">
        <TextBlock x:Name="za" Text="???" />
        <TextBox x:Name="ze" Width="150" Height="20"
            HorizontalAlignment="Left" VerticalAlignment="Top"
            Margin="10, 50" />
        <Button x:Name="bTest" Content="Test" Width="60" Height="40"
            HorizontalAlignment="Left" VerticalAlignment="Top"
            Margin="10, 100" Click="bTest_Click" />
    </Grid>
</UserControl>
```

Si vous travaillez en C#, insérez la fonction suivante dans `Page.xaml.cs` (après le constructeur de `Page`) :

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

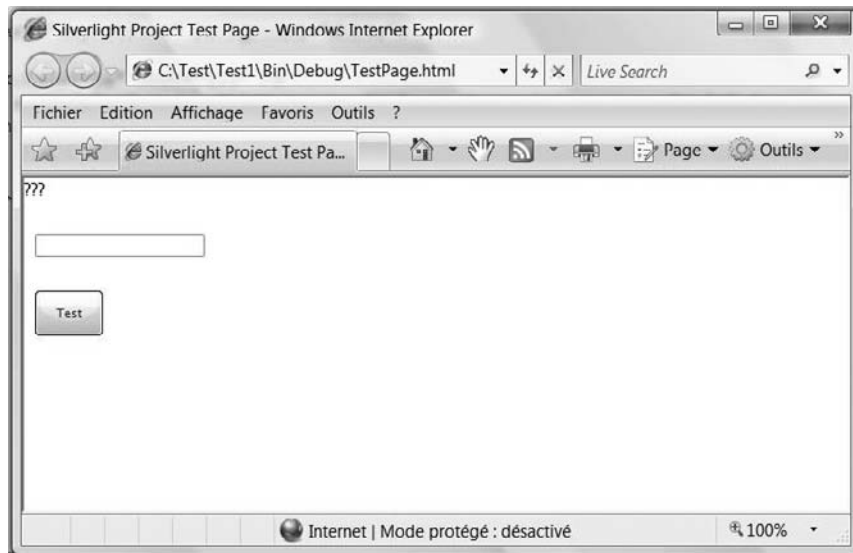
Si vous travaillez en VB, insérez la fonction suivante dans `Page.xaml.vb` (après le constructeur `New`) :

```
Private Sub bTest_Click(ByVal sender As System.Object, _
                        ByVal e As System.Windows.RoutedEventArgs)

End Sub
```

La figure A-8 présente le résultat obtenu dans le navigateur.

Figure A-8

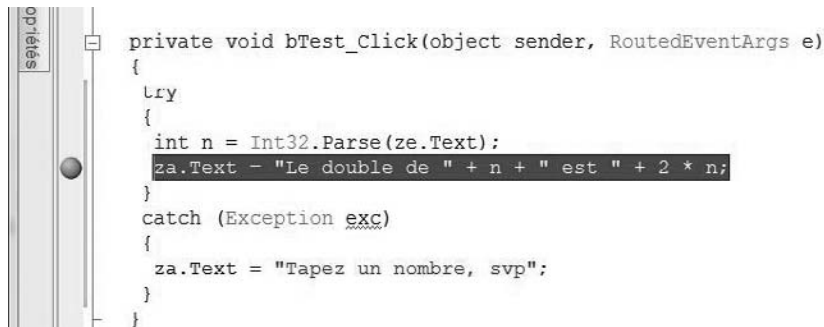


Insérez ensuite vos instructions dans la fonction `bTest_Click`. Par exemple :

C#
<code>za.Text = "Il est " + DateTime.Now.ToLongTimeString();</code>
<code>za.Text = "Vous avez tapé " + ze.Text;</code>
<code>int N = 20;</code> <code>za.Text = "Le double de " + N + " est " + 2*N;</code>
<pre>try { int n = Int32.Parse(ze.Text); // conversion de la chaîne de caractères en un entier za.Text = "Le double de " + n + " est " + 2 * n; } catch (Exception exc) { za.Text = "Tapez un nombre, svp"; }</pre>
VB
<code>za.Text = "Il est " & DateTime.Now.ToLongTimeString()</code>
<code>za.Text = "Vous avez tapé " & ze.Text;</code>
<pre>Try Dim n As Integer = Integer.Parse(ze.Text) ' conversion de la chaîne de caractères en un entier za.Text = "Le double de " & n & " est " & 2 * n Catch exc As Exception za.Text = "Tapez un nombre, svp" End Try</pre>

Pour déboguer un programme, cliquez simplement à gauche de l'instruction pour placer un point d'arrêt, puis exécutez-le en appuyant sur la touche F5 (ou en cliquant sur le petit triangle vert dans la barre des boutons). Le programme s'arrêtera à cette instruction (figure A-9).

Figure A-9



Vous pouvez alors visualiser le contenu des variables (il suffit d'immobiliser la souris sur le nom de la variable, figure A-10) :

Figure A-10

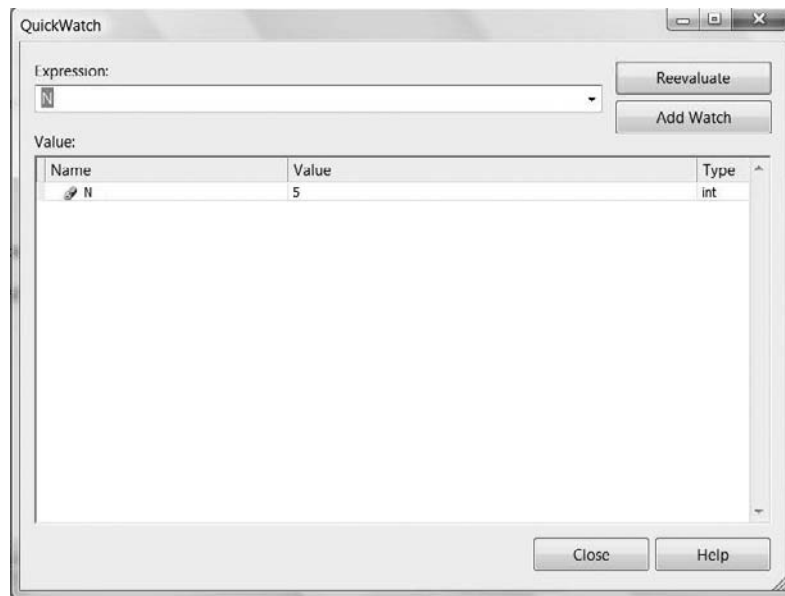


```
int N=5;
private void bTest_Click(object sender, RoutedEventArgs e)
{
    za.Text = "Clic sur bouton à " + DateTime.Now.ToLongTimeString();
}
```

The image shows a code editor with a mouse cursor hovering over the variable 'N' in the line 'private void bTest_Click(object sender, RoutedEventArgs e)'. A tooltip is visible, showing the variable 'N' and its value '5'.

Il est également possible de modifier le contenu des variables et de continuer l'exécution du programme (jusqu'à la fin ou jusqu'au prochain point d'arrêt, figure A-11).

Figure A-11



Index

A

- AcceptsReturn 104
- Action de génération 103, 138
- ActualHeight 56
- ActualWidth 56
- Add 132
- AddHandler 121
- Ajax 3
- alpha channel 69
- AlternatingRowBackground 224
- animation 191
- Apache 12
- ApplicationSettings 230
- aspx 17
- attribut 50
- AutoGenerateColumns 222
- AutoPlay 145
- AutoReverse 194
- AvailableFreeSpace 231

B

- Background 53, 67
- balise 50
- barre de défilement 112
- Begin 193
- BeginTime 194
- Bézier 167
- BezierSegment 167
- Binding 210
- BitmapImage 139
- boîte de liste 213
- Border 65
- BorderBrush 65
- BorderThickness 65
- bouton 105
 - gel 87
 - radio 108
- BrowserInformation 298

- Brush 67
- BufferingTime 145
- Button 105
- By 194

C

- Calendar 113
- calendrier 113
- canevas 48
- CanUserResizeColumns 224
- Canvas 48
- carrousel 9
- case à cocher 108
- CheckBox 108
- Checked 108
- Children 132
- CIL 15
- clavier 124
- Clear 132
- clic 123
- ClickMode 105
- ClientBin 15
- clipping 142
- ColorAnimation 193
- Colors 70
- Column 53
- ColumnDefinitions 53
- ColumnSpan 57
- ColumnWidth 224
- CommittingEdit 228
- conteneur 47
- Content 105
- ContentPresenter 278
- contrôle utilisateur 263
- ControlTemplate 275
- couleur 67
- Count 132
- courbe 163

- CreateDirectory 231
- CreateFile 231
- création dynamique 130
- curseur 110, 144
- Cursor 144

D

- Data 164
- DataContext 210
- DataGrid 221
- DataGridTemplateColumn 226
- DatePicker 114
- Déboguer 14
- Deep Zoom 146
- dégradé de couleurs 84
- DeleteDirectory 231
- DeleteFile 231
- déploiement 12, 16
- DirectoryExists 231
- DiscreteDoubleKeyFrame 196
- DisplayDate 114
- DisplayMemberBinding 225
- DisplayMemberPath 218
- DOM (Document Object Model)
2, 298
- DoubleAnimation 193
- DownloadStringAsync 251
- Duration 192, 194

E

- effet
 - de relief 100
 - miroir 183
- Element 244
- Elements 244
- ellipse 94
- EllipseGeometry 166
- EndPoint 73

événement 117, 302
Expression Blend 7, 80
Expression Design 7
Expression Media Encoder 7

F

feedback visuel 279
fichiers
 en ressource 238
 locaux 237
FileExists 231
Fill 67, 96, 140
FillBehavior 194
Filter 141
Firefox 3
Flash 2, 304
Flickr 9, 38, 260
FocusVisualElement 280
fonction de traitement 117
FontFamily 102
FontSize 102
FontStretch 102
FontStyle 102
FontWeight 102
Foreground 67
FrameworkElement 122
From 194
from 245

G

GeneralTransform 306
GeometryGroup 166
GetAttribute 299
GetDirectoryName 231
GetDirectoryNames 231
GetElementById 299
GetFileNames 231
GetPosition 122
GetStyleAttribute 299
GradientOrigin 75, 76
Grid 53
GridSplitter 61
grille 53
 de données 221
GroupName 109

H

Handled 122
Handles 121
HasAttributes 245
HasElements 244

HeadersVisibility 224
hébergeur 12
Height 58, 139
hexadécimal 69
HorizontalAlignment 52
HorizontalOffset 65
HorizontalScrollBarVisibility 64
horloge 126
HTML 295
HtmlElement 299
HtmlPage 298

I

IIS 12
image 137
Imports 13
IncreaseQuota 232
infobulle 106
INotifyPropertyChanged 211, 223
Insert 132
installation 7
Interaction 290
InvokeSelf 303
isolated storage 229
IsolatedStorageFile 230
IsThreeState 108
Items 215
ItemsSource 217
ItemTemplate 219

J

JavaScript 2, 303

K

Key 125
key frames 195
Keyboard 125

L

liaison de données 209
ligne 163
Line 163
LinearDoubleKeyFrame 196
LinearGradientBrush 73
LineBreak 99
LineGeometry 166
Linq 240
Linux 4
ListBox 213
ListBoxItem 214
Load 133

Loaded 118
LoadingRow 227

M

Mac 3
Margin 52
masque d'opacité 80
matrice 188
MatrixTransform 182
MaxHeight 56
Maximun 110
MaxWidth 56
MediaElement 22, 145
MediaEnded 23
MinHeight 56
Minimum 110
MinWidth 56
MouseEnter 119
MouseEventArgs 120
MouseLeave 119
MouseLeftButtonDown 119
MouseLeftButtonUp 119
MouseMove 119
MouseWheelHelper 151
mp3 145
MultiScaleImage 153
my 62

N

Navigate 298
Naviguer 21
Netscape 2

O

object 17
objet 50
ObservableCollection 225
Offset 73
ombre 90
Opacity 96
OpenFile 232
OpenFileDialog 140
OpenReadAsync 251
orderby 247
Orientation 50, 110

P

Pad 74
Padding 104
Page 13
page turner 33

- Path 164
- PathGeometry 166
- Pause 193
- pinceau 67
- PlatformKeyCode 125
- Play 146
- PointAnimation 193
- pointillés 174
- police de caractères 27, 102
- Polygon 163
- Polyline 163
- Position 145, 233
- programmation serveur 118
- projet 10
- PropertyChanged 212
- propriété 50

Q

- QuadraticBezierSegment 167

R

- RadialGradientBrush 75
- RadioButton 108
- RadiusX 76, 96
- RadiusY 76, 96
- Read 233
- rectangle 94
- RectangleGeometry 166
- Reflect 74
- Remove 132
- RemoveAt 132
- Repeat 74
- RepeatButton 106
- Resources 271
- ressource 27
- Result 252
- Resume 193
- RootElement 280
- RotateTransform 182
- Row 53
- RowBackground 224
- RowDefinitions 53
- RowSpan 57
- Run 99

S

- Safari 3
- ScaleTransform 182
- scRGB 69
- ScriptableMember 303
- ScriptableType 303

- ScriptObject 303
- ScrollBar 112
- ScrollViewer 64
- select 245
- SelectedDate 114
- SelectedText 104
- SelectionChanged 216
- SelectionMode 224
- service Web 254
- SetAttribute 299
- SetProperty 299
- SetStyleAttribute 299
- SetValue 131
- ShowDialog 141
- ShowGridLines 55
- ShowPreview 63
- Silverlight Tools 7
- SizeChanged 305
- SkewTransform 182
- Slider 110
- SolidColorBrush 72
- solution 10
- Source 122, 125
- souris 119
- SplineDoubleKeyFrame 196
- SpreadMethod 74
- sRGB 69
- StackPanel 50
- Start 126
- StartPoint 73
- StaticResource 272
- stockage isolé 229
- Stop 126, 193
- story-board 191
- Stretch 139
- Stroke 67, 172
- style 271

T

- TargetName 194
- TargetProperty 194
- TargetType 272
- template 273
- TemplateBinding 277
- Text 104
- TextBlock 98, 209
- TextBox 103, 209
- TextWrapping 98
- Threading 126
- Timeline 200
- timer 126
- TimeSpan 25, 126

- To 194
- ToggleButton 106
- tourneur de pages 34, 155
- transformation 181
- TransformGroup 182
- transition 288
- TranslateTransform 182
- transparence 69
- ttf 103

U

- Uniform 140
- UniformToFill 140
- user control 263
- using 13

V

- ValueChanged 110
- var 245
- VerticalAlignment 51
- VerticalOffset 65
- VerticalScrollBarVisibility 64
- vidéo 22
- Visibility 96
- Visual Studio 10
- Visual Web Developer 10

W

- WebClient 236, 250
- Width 58, 139
- wma 145
- wmv 145
- Write 233

X

- x:Key 272
- XAML 13
- XamlReader 133
- XAP 16, 28
- XDocument 241
- XElement 241
- XML 239

Z

- ZIndex 97

Silverlight 2



Créer des applications Web plus dynamiques

S'exécutant sur différentes plates-formes (Windows, Mac et bientôt Linux) et compatible avec les navigateurs les plus répandus (Internet Explorer, Firefox, Safari), Silverlight est une nouvelle technologie de Microsoft qui permet de développer des applications Internet riches, pouvant être déployées sur n'importe quel serveur, sans nécessiter de composants spécifiques ni de droits particuliers. Avec la version 2, il devient possible de concevoir des sites Web aussi conviviaux que des applications Windows, dotés d'une grande interactivité et d'effets visuels spectaculaires, notamment grâce aux techniques d'animation et à la technologie Deep Zoom.

Rédigé par l'un des spécialistes de Silverlight, cet ouvrage explique de manière pratique comment écrire ces applications Web d'un nouveau type en C# ou en Visual Basic, sachant que peu de connaissances sont requises, Visual Studio et l'outil graphique Expression Blend facilitant amplement la tâche du développeur. Le livre met l'accent sur la programmation des animations et l'intégration de médias (son, vidéos, etc.), mais également sur l'accès aux données via des services Web. Le code source des exemples du livre en C# et Visual Basic est disponible sur www.editions-eyrolles.com.

G. Leblanc

Spécialiste de C# et de .NET, **Gérard Leblanc** est l'auteur de plusieurs best-sellers sur la programmation Windows. Distingué MVP (*Most Valuable Professional*) par Microsoft pour sa contribution à la connaissance des logiciels de développement d'applications, il est particulièrement attentif aux besoins des entreprises avec lesquelles il entretient des contacts suivis.

Au sommaire

Installation de Silverlight • **Création d'une application Silverlight** • **Conteneurs de base et spécifiques** • **Couleurs et pinceaux** • Masque d'opacité • Pinceaux dans Expression Blend • Dégradés • **Première série de composants** • Rectangles et ellipses • Zones d'affichage (TextBlock) • Polices de caractères • Zones d'édition (TextBox) • Boutons • Cases à cocher (CheckBox) • Boutons radio (RadioButton) et hyperliens • Composant Slider • Barre de défilement (ScrollBar) • Calendrier (Calendar) • Composant DatePicker • **Du code dans les applications Silverlight** • Événements • Création dynamique d'objets • Création dynamique à partir du XAML • **Images, curseurs et vidéos** • Clipping d'image • Sons et films • Deep Zoom • Tourneur de pages • **Figures géométriques** • Line, Polyline et Polygon • Path • Courbes de Bézier • Stroke • Dessiner avec Expression Blend • **Transformations et animations** • **Liaisons de données** • Avec TextBlock et TextBox • Boîtes de liste • Grille de données • **Accès aux fichiers** • Stockage isolé avec IsolatedStorage • Lire des fichiers distants ou locaux • Fichiers en ressources • **Accès XML avec Linq** • Chargement du fichier XML • Espaces de noms • Cas pratiques • **Accès à distance aux données** • Objet WebClient • **Contrôles utilisateurs** • Traitement d'événements liés à un contrôle utilisateur • Création et utilisation d'un contrôle utilisateur • Contrôle utilisateur en DLL • **Styles et templates** • Styles des composants Silverlight de Microsoft • Modification de contrôles avec Expression Blend • **Interaction Silverlight/HTML** • Blocs Silverlight dans une page Web • Accès aux éléments HTML depuis Silverlight • Attacher des événements • Appeler une fonction JavaScript ou C# • Animation Flash dans une page Silverlight.

À qui s'adresse ce livre ?

- Aux programmeurs souhaitant réaliser des sites Web de nouvelle génération
- À tous les graphistes et designers ayant des notions de programmation



Sur le site www.editions-eyrolles.com

- Téléchargez le code source des exemples du livre.
- Consultez les mises à jour et compléments.
- Dialoguez avec l'auteur.