Les outils du développeur

Microsoft

MICROSOFT®

VISUAL BASIC® 2008

ÉTAPE PAR ÉTAPE

Michael Halvorson

Adapté de l'américain par
Fabrice Lemainque

Titre U.S. : MICROSOFT® VISUAL BASIC® 2008 STEP BY STEP
ISBN U.S. : 978-0-7356-2537-2

Édition et diffusion de la version française : Dunod
Distribution : Hachette Livre Distribution
Couverture : Agence SAMOA
Adapté de l'américain par Fabrice Lemainque
Mise en page : Arclemax
ISBN : 978-2-10-055193-4

Sommaire

Partie III Concevoir l'interface utilisateur

Annexe

Introduction

Je suis profondément ravi que vous ayez choisi ce livre pour acquérir les compétences et techniques de programmation avec Microsoft Visual Basic 2008. Si nous nous découvrons mutuellement dans ce paragraphe, il est probable que nous ne sommes pas si différents. Je travaille quotidiennement avec un ordinateur et passe un temps considérable à aider amis et collègues à se faciliter la vie à l'aide de nouveaux logiciels et de leurs techniques apparentées. Au fil des années, j'ai appris des douzaines d'applications informatiques, de langages et d'outils. J'éprouve un plaisir certain à les combiner pour résoudre des problèmes du monde réel. Il en va probablement de même pour vous, le spécialiste en informatique de votre bureau, école ou domicile. Ce pourquoi vous devez apprendre ou vous mettre à niveau avec Visual Basic 2008, un des plus puissants outils de développement actuel.

Microsoft Visual Basic 2008 Étape par Étape constitue une introduction exhaustive à la programmation avec le logiciel Microsoft Visual Basic 2008. J'ai conçu cet ouvrage pratique et concret en gardant à l'esprit un large éventail de niveaux de compétence. Grâce à lui, les nouveaux venus en programmation peuvent apprendre les bases du développement dans le contexte d'applications du monde réel et les programmeurs Visual Basic expérimentés peuvent maîtriser rapidement les plus importants outils et techniques de programmation qu'offre la nouvelle version 2008 de Visual Basic.

Pour compléter cette approche globale, la structure du livre repose sur quatre parties thématiques dans lesquelles se répartissent 20 chapitres et 53 exercices pas à pas, ainsi que des exemples de programmes. En utilisant ce livre, vous apprendrez rapidement à créer des applications Visual Basic 2008 pour le système d'exploitation Microsoft Windows et plusieurs navigateurs web. Et en plus, vous allez y prendre plaisir !

Qu'est-ce que Visual Basic 2008 ?

Visual Basic 2008 est un outil de développement que vous pouvez employer pour construire des applications logicielles qui permettront d'accomplir des tâches utiles de manière conviviale et avec un grand nombre d'options. Avec Visual Basic 2008, vous pouvez créer des applications pour le système d'exploitation Windows, le web, les appareils mobiles et bien d'autres environnements. L'avantage majeur de Visual Basic est qu'il a été conçu pour augmenter la productivité de votre travail de développement au quotidien – notamment si vous avez besoin d'exploiter les informations contenues dans des bases de données ou créer des solutions pour internet – mais ce n'est pas tout : une fois que vous serez familiarisé avec l'environnement de développement de Microsoft Visual Studio 2008, vous pourrez utiliser les mêmes outils pour écrire des programmes avec Microsoft Visual C++ 2008, Microsoft Visual C# 2008, Microsoft Visual Web Developper 2008, ainsi qu'avec des outils et compilateurs tiers.

Versions de Visual Basic .NET

Tout cela ne s'est pas fait en un jour. La première version de Visual Basic .NET (Microsoft Visual Basic .NET 2002) a été publiée en février 2002. La deuxième version (Microsoft Visual Basic .NET 2003) a fait l'objet d'une large diffusion en mars 2003. Vint ensuite Visual Basic 2005, à la fin de l'année 2005. À l'issue d'une longue période de développement et d'intégration, Microsoft publie Visual Basic 2008 au début de l'année 2008. Visual Basic 2008 est maintenant si bien intégré à Visual Studio qu'il est disponible sous forme de composant de la suite d'outils de programmation Visual Studio 2008, qui comprend les compilateurs Visual C#, Visual C++ et Visual Web Developper, ainsi que d'autres outils de développement Microsoft .NET.

Il reste que Visual Studio 2008 est commercialisé sous différentes configurations : Standard Edition, Professional Edition, Team Suite et Express Edition. J'ai écrit ce livre de façon qu'il soit compatible avec toutes les éditions de Visual Basic 2008 et Visual Studio 2008, mais en le focalisant plus particulièrement sur les outils et techniques disponibles dans Visual Studio Standard Edition et Visual Studio Professional Edition. Bien que Visual Basic 2008 soit, sous bien des aspects, comparable à Visual Basic .NET 2005, il subsiste des différences importantes et de nombreuses améliorations. C'est pourquoi je vous recommande de faire les exercices de ce livre en utilisant le logiciel Visual Basic 2008.

Remarque Le logiciel Visual Basic 2008 n'est pas livré avec cet ouvrage ! Mais vous trouverez sur le site compagnon de ce livre, à l'adresse *www.dunod.com*, les fichiers d'exercices, des exemples de bases de données et d'autres informations utiles que vous pourrez exploiter avec Visual Basic.

Mise à niveau depuis Microsoft Visual Basic 6.0

Avant Visual Basic .NET, le monde de la programmation se félicitait de disposer de Visual Basic 6, diffusé dix ans auparavant en septembre 1998. Visual Basic 6 est si célèbre que de nombreux programmeurs enthousiastes l'emploient encore, surtout hors de l'Europe et de l'Amérique du Nord, là où les mises à niveau matérielles peuvent être d'accès difficile. (Merci à tous les utilisateurs de Visual Basic 6 qui m'ont écrit depuis l'Afrique et l'Asie !). D'une certaine façon, je vous comprends : Visual Basic 6 était et reste d'emploi si facile, grâce à sa méthode de programmation simple et directe. Comme toutefois vous êtes nombreux à vous en être rendu compte, créer de réelles applications de niveau professionnel avec Visual Basic 6 n'avait rien d'évident. Je ressentais toujours un certain complexe de taille et de vitesse en discutant avec des amis qui vantaient leurs véloces et compacts programmes Visual C++. Pour écrire des applications Visual Basic 6 réellement complexes, il fallait généralement jongler avec de multiples problèmes.

Dix ans plus tard, Visual Basic 2008 a formidablement facilité l'écriture d'applications Windows ou fondées sur l'Internet de qualité professionnelle qui soutiennent la comparaison avec des applications Visual C++, Visual C# ou Java. La beauté de Visual Basic est qu'il est beaucoup plus facile à apprendre que les autres outils de programmation. Même s'il existe quelques écueils de vitesse, la mise à niveau de Visual Basic 6 vers Visual Basic 2008 est relativement simple. Visual Studio 2008 propose un assistant de mise à niveau qui débute la procédure de conversion pour vous. Vous constaterez que, dans la majorité des cas, les contrôles, instructions, méthodes et propriétés héritées que vous avez appris sont toujours présents dans Visual Basic 2008.

Ce livre comporte des remarques de mise à niveau pour les lecteurs qui effectuent une mise à niveau depuis Visual Basic 6. Ayant été auparavant un programmeur Visual Basic 6, je sais ce que l'on ressent lors d'une mise à niveau vers Visual Basic .NET. Lors de la lecture de ce livre, vous découvrirez ici et là des remarques sur les modifications de syntaxe et de paradigmes conceptuels, ainsi que comment employer ce que vous connaissez pour devenir un programmeur Visual Basic 2008 efficace. Croyez-moi, vous apprécierez de pouvoir placer cette compétence sur votre C.V. !

Voici un message pour tous les programmeurs : je vous conseille d'évaluer votre compétence globale en matière de développement et de ne pas vous concentrer sur les nouveaux dispositifs du langage de programmation que vous allez apprendre. Les compétences sous-jacentes, comme le travail avec des algorithmes, des structures de données, la programmation orientée objet et le débogage vous aident à écrire de meilleurs programmes. C'est pourquoi il est aussi important de pleinement comprendre la conception des interfaces utilisateur et les techniques de gestion de bases de données que de découvrir les nouvelles astuces d'un dispositif particulier dont vous avez entendu parler par la presse. Les développeurs Visual Basic 6 pourront tirer profit de tout ce qu'ils ont appris en matière de développement logiciel. Les outils ont évolué, mais les compétences sous-jacentes restent les mêmes.

Bien démarrer avec cet ouvrage

Ce livre est conçu pour vous aider à construire vos compétences dans un certain nombre de domaines importants. Il vous sera utile si vous êtes débutant en programmation, si vous avez décidé d'adopter ce nouveau langage de programmation ou si vous connaissez déjà Visual Basic 6 ou Visual Basic .NET 2005. Le tableau suivant vous aide à trouver le bon point de départ pour vous.

Si vous êtes	Suivez ces étapes
Nouveau venu en programmation	**1.** Installez les fichiers d'exercices comme décrit dans la section « Installer et utiliser les fichiers d'exercices » plus loin dans cette Introduction. **2.** Dotez-vous des compétences de base nécessaires à l'utilisation de Visual Basic 2008 en suivant dans l'ordre les chapitres 1 à 17. **3.** Lancez-vous ensuite dans la partie 4 « Programmer pour les bases de données et le web » en fonction de vos centres d'intérêt et votre expérience.
À l'aise avec Visual Basic .NET 2002, 2003 ou 2005	**1.** Installez les fichiers d'exercices comme décrit dans la section « Installer et utiliser les fichiers d'exercices ». **2.** Faites les chapitres 1 à 4 et sautez les chapitres 5 à 17 pour passer directement aux chapitres 18 à 20. **3.** Pour en savoir plus sur les apports de cette nouvelle version, lisez les chapitres 1, 4, 5, 7, 8, 13 et 18 à 20.
À l'aise avec Visual Basic 6	**1.** Installez les fichiers d'exercices comme décrit dans la section « Installer et utiliser les fichiers d'exercices ». **2.** Lisez attentivement les chapitres 1 à 4 pour apprendre les nouvelles caractéristiques de l'environnement de développement de Visual Studio 2008. **3.** Faites attention aux commentaires situés dans différents chapitres, qui signalent les différences importantes entre Visual Basic 6 et Visual Basic 2008. **4.** Lisez rapidement les chapitres 5 à 13 pour revoir les bases de la programmation pilotée par l'événement, de l'emploi des variables et de l'écriture de structures de décision. Concentrez-vous sur les chapitres 5, 6, 9 et 12. **5.** Lisez dans l'ordre les chapitres 14 à 20. Vous y apprendrez les nouveautés de Visual Basic 2008 en ce qui concerne la conception de l'interface utilisateur, la programmation de bases de données et la programmation pour le web.
En deuxième lecture de ce livre après avoir travaillé sur tous les chapitres	**1.** Servez-vous de l'index pour localiser les informations que vous recherchez sur des sujets spécifiques et de la table des matières pour trouver des informations sur des thèmes généraux. **2.** Servez-vous de l'index des nouveautés pour y consulter une liste des nouvelles fonctionnalités de Visual Basic 2008 et découvrir comment le code Visual Basic 6 peut être actualisé. **3.** Lisez la section « Rappel du chapitre » à la fin de chaque leçon pour revoir les principales tâches accomplies chaque fois. Vous les y retrouverez dans leur ordre d'apparition dans le chapitre.

Matériel et logiciels nécessaires

Pour réaliser les exercices de ce livre, vous devez posséder l'équipement suivant :

- Windows Vista ou Windows XP avec le service pack 2 ou Windows Server 2003 avec le service pack 1.
- Microsoft Visual Studio 2008 (Standard Edition, Professional Edition ou Team Suite).
- Les exigences matérielles recommandées sont un processeur d'une fréquence de 2,2 GHz, 1 024 Mo de mémoire RAM, un écran de résolution 1 280 x 1 024 et un disque dur à 7 200 tours/minute (pour Windows Vista, un processeur d'une fréquence de 2,4 GHz et 768 Mo de mémoire RAM).
- 1,22 Go d'espace libre sur le disque dur pour une installation minimale et 2 Go pour une installation complète.
- Un lecteur de CD-Rom ou de DVD-Rom.
- Une souris Microsoft ou compatible.

Remarque J'ai testé le contenu de ce livre et les fichiers d'exercices avec Visual Studio 2008 Standard et Visual Studio 2008 Professional. Vous pourriez observer quelques différences si vous employez d'autres versions de Visual Studio 2008 et notamment avec Visual Studio 2008 Express Edition, dans laquelle quelques fonctions ne sont pas disponibles. En outre, les copies d'écran de ce livre ont été réalisées avec Windows Vista. Si vous vous servez de Windows XP ou de Windows Server 2003, vous pourrez remarquer quelques différences en certaines situations.

Installer et utiliser les fichiers d'exercices

Les fichiers d'exercices sont téléchargeables sur la page consacrée à cet ouvrage sur le site *www.dunod.com*. Une fois les fichiers installés, lorsque vous apprendrez par exemple à afficher des tables de base de données dans un formulaire grâce au contrôle *DataGridView*, vous ouvrirez le fichier nécessaire à la réalisation de l'exercice (une base de données des élèves d'un établissement), puis vous pourrez utiliser les outils de programmation de base de données pour accéder à ladite base. Si vous vous servez des fichiers d'exercices, vous ne perdrez pas de temps à créer les fichiers nécessaires à leur réalisation. Ainsi, vous vous concentrerez sur l'apprentissage et la maîtrise des techniques de programmation avec Visual Basic 2008. Avec les fichiers et les instructions étape par étape proposées dans les chapitres, vous apprendrez en mettant « la main à la pâte », ce qui constitue la meilleure manière d'acquérir et conserver de nouvelles compétences.

Important Avant de vous lancer dans le téléchargement des fichiers d'exercices, assurez-vous que ce livre correspond bien à votre version du logiciel. Ce livre est consacré à Visual Studio 2008 et à Visual Basic 2008. Pour savoir quel logiciel équipe votre ordinateur, vérifiez l'emballage ou démarrez le logiciel, ouvrez un projet, puis cliquez sur À propos de Microsoft Visual Studio dans le menu Aide, en haut à droite de l'écran.

Installer les fichiers d'exercices

Pour installer les fichiers d'exercices sur votre disque dur, vous devez disposer d'environ 10 Mo d'espace disponible. Voici les étapes à accomplir :

1. Dans votre navigateur favori, tapez **www.dunod.com** pour vous rendre sur le site de téléchargement.
2. Cliquez sur la rubrique « Informatique » et localisez l'ouvrage que vous avez entre les mains par son titre en faisant défiler la liste vers le bas si nécessaire.
3. Dans le bandeau de gauche, cliquez sur le lien intitulé documents téléchargeables sous la rubrique « Compléments en ligne ».
4. Suivez les instructions de la page suivante, puis laissez-vous guider pour le téléchargement des fichiers.

Remarque Pour que les fichiers d'exercices fonctionnent selon la manière prévue, installez-les impérativement dans l'emplacement c:\vb08epe. Si vous préférez un autre emplacement d'installation, il vous faudra corriger manuellement les chemins d'accès de certains fichiers d'exercices afin de localiser des composants tels que les images et les fichiers de base de données afin de pouvoir les utiliser.

Utiliser les fichiers d'exercices

Chaque chapitre de ce livre explique quand et comment utiliser les fichiers d'exercices pour le chapitre correspondant. Lorsqu'il vous faut utiliser un fichier, le livre vous donne les instructions nécessaires à son ouverture. Les chapitres de ce livre sont construits à partir de scénarios qui simulent des projets de programmation réels. Cela vous permet d'appliquer à votre travail les connaissances que vous acquérez.

Remarque Visual Basic 2008 a recours à un nouveau format de fichier pour ses projets et solutions. Ainsi, il ne vous sera pas possible d'ouvrir les fichiers d'exercices de ce livre si vous avez installé une version plus ancienne de Visual Basic ou de Visual Studio. Pour savoir quelle version de Visual Basic ou Visual Studio équipe votre ordinateur, cliquez sur la commande À propos de... dans le menu d'aide de votre logiciel.

Il est possible de personnaliser et configurer Visual Studio de manière très poussée, afin d'ouvrir et enregistrer des projets et des solutions de différentes manières. De manière générale, ce livre part du principe que vous employez les réglages par défaut de Visual Studio. Pour en savoir plus sur les incidences de ces réglages de l'environnement de développement sur votre manière d'écrire des programmes et utiliser les fichiers d'exercices, lisez la section « Personnaliser les réglages de l'environnement de développement intégré pour réaliser les exercices pas à pas », dans le chapitre 1 « Explorer l'environnement de développement intégré de Visual Studio ». Pour ceux d'entre vous qui aiment connaître tous les détails, voici une liste des projets Visual Basic que vous trouverez sur le site compagnon de ce livre. Chaque projet se trouve dans son propre dossier et contient plusieurs fichiers. Voyez tout ce que vous allez faire !

Projet	Description
Chapitre 1	
Musique	Un programme tout simple qui vous souhaite la bienvenue dans la leçon et affiche une photo numérique.
Chapitre 2	
Bandit Manchot	Votre premier programme : un jeu qui simule le célèbre « bandit manchot » de Las Vegas, la machine à sous.
Chapitre 3	
Anniversaire	Un programme qui utilise le contrôle *DateTimePicker* pour sélectionner une date.
Cocher Case	Un programme qui vous montre comment utiliser le contrôle *CheckBox* et ses propriétés.
Bonjour	Un programme qui affiche « Bonjour ! » et présente les contrôles *Label* et *TextBox*.
Saisie	L'interface utilisateur d'un environnement d'ordonnancement graphique, assemblé à l'aide de plusieurs contrôles d'entrée d'une grande efficacité.
WebLink	Une démonstration du contrôle *LinkLabel*, qui ouvre un navigateur web dans votre application Visual Basic.
Chapitre 4	
Menu	Montre comment utiliser les contrôles Visual Studio de boîte de dialogue, de barre d'outils et de menus.
Chapitre 5	
Math Plus	Pour une utilisation avancée des opérateurs de division de nombres entiers, de division avec reste, de mise en exposant et de concaténation de chaîne.
Math de base	Pour une utilisation de base de l'addition, de la soustraction, de la multiplication et de la division.
Testeur de constante	Utiliser une constante pour détenir une entité mathématique fixe.
Types de données	Une démonstration des types de données fondamentaux de Visual Basic et de leur emploi avec des variables.
Math et Framework	Pour découvrir les classes du .NET Framework avec des méthodes mathématiques.
Input Box Test de variable	Reçoit une entrée avec la fonction *InputBox*. Déclare et utilise des variables pour stocker des informations.
Chapitre 6	
Select Case	Utilise une structure de décision *Select… Case* et un contrôle *ListBox* pour afficher un message de bienvenue en plusieurs langues.
Validation Utilisateur	Utilise la structure de décision *If…Then…Else* et un contrôle *MaskedTextBox* pour gérer un processus d'ouverture de session

Projet	Description
Chapitre 7	
Conversion Celsius	Convertit les températures de degrés Fahrenheit en Celsius à l'aide d'une boucle Do.
Horloge Numérique	Un programme d'horloge numérique qui illustre le fonctionnement du contrôle *Timer*.
Boucle For	Démontre l'utilisation d'une boucle *For…Next* pour afficher du texte dans un contrôle *TextBox* et de la fonction *Chr* pour créer un caractère de retour à la ligne.
Boucle For Icones	Utilise une variable compteur globale dans une procédure événementielle comme alternative aux boucles. Ce programme affiche également des images avec le contrôle *PictureBox*.
Mot de passe chronométré	Montre comment employer un contrôle Timer pour créer un programme d'ouverture de session avec une fonction d'expiration du délai de saisie du mot de passe.
Extrait Version Windows	Montre comment utiliser la nouvelle commande Insérer un extrait pour afficher la version de Windows actuellement en cours d'exécution sur l'ordinateur d'un utilisateur.
Chapitre 8	
Test de débogage	Simulation d'un problème de débogage, conçu pour être résolu avec les outils de débogage de Visual Studio.
Chapitre 9	
Erreur Disque	Un programme qui s'interrompt en cas de mauvaise utilisation du lecteur de CD/DVD. Ce projet est utilisé comme base d'un gestionnaire d'erreur Visual Basic.
Gestionnaire Disque	Projet avec un gestionnaire d'erreur de chargement de fichier ; il sert à illustrer l'utilisation de la syntaxe *Try…Catch*.
Chapitre 10	
SubTextBox	Une procédure *Sub* généraliste qui ajoute des éléments à une zone de liste.
Gagnant	Une variante du projet Bandit Manchot du chapitre 2, que vous allez améliorer en utilisant des variables publiques et une fonction qui calcule le taux de réussite au jeu.
Chapitre 11	
Tri de tableau	Un programme qui vous montre comment créer et manipuler d'importants tableaux d'entiers. Démontre les méthodes *Array.Sort* et *Array.Reverse* et l'emploi d'un contrôle *ProgressBar* pour donner à l'utilisateur un retour d'information visuel pendant les tris de longue durée.
Tableau dynamique	Calcule la température moyenne sur un nombre de jours donné à l'aide d'un tableau dynamique.
Tableau fixe	Calcule la température moyenne hebdomadaire à l'aide d'un tableau à longueur fixe.

Projet	Description
Chapitre 12	
Collection Controls	Utilise une boucle *For Each…Next* et la collection de contrôles Visual Studio pour déplacer des objets sur un formulaire.
Collection URL	Présente une collection définie par l'utilisateur contenant une liste d'adresses web (URL) récemment consultées par l'utilisateur.
Chapitre 13	
Prendre Note	Un petit utilitaire de prise de notes, qui montre comment utiliser la fonction *FileOpen* et les contrôles *TextBox*, *MenuStrip* et *SaveFileDialog*.
Tri de texte	Un éditeur de fichiers texte, avec une barre de menus qui démontre comment gérer les commandes Ouvrir, Fermer, Enregistrer sous, Insérer Date, Trier Texte et Quitter dans un programme. Contient un module *ShellSort* pour trier des tableaux. Ce module peut être ajouté à d'autres objets de programmation.
Navigateur texte	Affiche le contenu d'un fichier texte dans un programme Visual Basic. Démontre les commandes de menu, un gestionnaire d'erreur *Try…Catch* et les fonctions *FileOpen* et *LineInput* ; il sert de base aux autres programmes de ce chapitre.
Chapitre 14	
Ajouter des contrôles	Montre comment ajouter des contrôles à un formulaire Windows en cours d'exécution en faisant intervenir du code (et non pas le Concepteur).
Ancrer et aligner	Utilise les propriétés *Anchor* et *Dock* d'un formulaire pour aligner des objets en cours d'exécution.
Limites du bureau	Utilise les propriétés *StartPosition* et *DesktopBounds* pour positionner un formulaire Windows en cours d'exécution. Montre également le fonctionnement de la propriété *FormBorderStyle* de la structure *Rectangle* et de la méthode *ShowDialog*.
Aide Bandit Manchot	Le programme Bandit Manchot amélioré du chapitre 10, encore optimisé par l'ajout d'un deuxième formulaire pour afficher des informations d'aide.
Chapitre 15	
Dessiner des formes	Démontre les quelques méthodes graphiques parmi les plus utiles de l'espace de noms *System.Drawing*, à savoir *DrawEllipse*, *FillRectangle* et *DrawCurve*.
Icône animée	Anime une icône sur le formulaire, la faisant monter et descendre chaque fois que vous cliquez sur un bouton.
Formulaire transparent	Montre comment modifier la transparence d'un formulaire en utilisant l'objet *Me* et la propriété *Opacity*.
Zoomer	Simule l'agrandissement d'un objet sur un formulaire (dans ce cas, la planète Terre).

Projet	Description
Chapitre 16	
Héritage de formulaires	Utilise le Sélecteur d'héritage de Visual Studio pour créer un formulaire qui va hériter ses caractéristiques et fonctionnalités d'un autre formulaire.
Classe Personne	Montre comment créer de nouvelles classes, propriétés et méthodes dans un projet Visual Basic. La nouvelle classe *Personne* est un enregistrement d'employé avec les champs prénom, nom de famille et date de naissance. Elle contient une méthode qui calcule l'âge de l'employé.
Chapitre 17	
Fenêtre d'impression	Montre comment créer des boîtes de dialogue Aperçu avant impression et Mise en page.
Imprimer un fichier	Un projet qui prend en charge des tâches d'impression plus sophistiquées, et notamment l'impression d'un fichier texte de plusieurs pages avec des sauts de ligne. Contient du code que vous réutiliserez dans vos propres projets.
Imprimer une image	Imprime des images depuis un programme Visual Basic en utilisant un gestionnaire d'erreur, la méthode *Print* et la méthode *DrawImage*.
Imprimer du texte	Montre comment imprimer du texte simple dans un programme Visual Basic.
Chapitre 18	
Formulaire ADO	Montre comment utiliser ADO.NET pour établir une connexion à une base de données Microsoft Access et afficher des informations de cette base de données.
Chapitre 19	
Exemple DataGridView	Montre comment employer le contrôle *DataGridView* pour afficher plusieurs tables de données dans un formulaire. Montre également comment les barres de navigation, les ensembles de données (*datasets*) et les adaptateurs de tables sont interconnectés et liés aux objets d'un formulaire.
Chapitre 20	
Chap20	Montre comment exploiter Visual Web Developer et ASP.NET pour créer un calculateur de mensualités pour un prêt auto, lequel fonctionne dans un navigateur web, présente des informations d'aide et affiche des enregistrements de base de données.

Désinstallation des fichiers d'exercices

Pour désinstaller les fichiers d'exercice de Visual Basic 2008 Étape par étape de votre système, supprimez simplement le dossier vb08epe (normalement créé dans c:/) ainsi que tous ses sous-dossiers. Cela fait, vous pourrez supprimer manuellement tout autre projet Visual Basic créé par vous-même et éventuellement stocké ailleurs, si vous le souhaitez.

Conventions utilisées dans ce livre

Avant de commencer les exercices de ce livre, vous pouvez gagner du temps en comprenant comment je délivre les instructions et quels sont les éléments que j'emploie pour communiquer les informations concernant la programmation en Visual Basic.

Conventions de style

- Les noms de tous les éléments de programme (contrôles, objets, fonctions, méthodes, propriétés, etc.) figurent en italiques.
- Les exercices à suivre pas à pas sont présentés sous forme de listes numérotées (1, 2 et ainsi de suite). Une puce ronde (●) indique un exercice à une seule étape.
- Le texte à saisir apparaît en **gras**.
- À mesure que vous avancez dans les étapes à réaliser, vous trouverez des tableaux de propriétés que vous définirez dans Visual Studio. Les propriétés de texte apparaissent entre guillemets, **mais vous ne devez pas saisir les guillemets**.
- Le signe plus (+) placé entre deux noms de touches signifie que vous devez enfoncer ces touches de manière simultanée. Par exemple, « Appuyez sur ALT+TAB » signifie que vous devez maintenir enfoncée la touche ALT tout en appuyant sur la touche TAB.
- Les encadrés intitulés Astuce, Remarque ou Important vous donnent des informations complémentaires ou d'autres méthodes pour réaliser une étape. Lisez-les avant de poursuivre l'exercice en cours.

Autres caractéristiques

- Vous pourrez découvrir des techniques de programmation spéciales, des informations fondamentales ou des dispositifs apparentés à l'information étudiée dans les encadrés présents dans les chapitres. Ces encadrés soulignent fréquemment des terminologies complexes ou suggèrent des pistes de réflexion et d'exploration.
- Vous pourrez en apprendre plus sur des options ou techniques fondées sur ce que vous avez appris dans un chapitre en effectuant l'exercice Aller plus loin présent à la fin de ce chapitre.
- Un bref résumé de la façon d'effectuer les tâches apprises figure dans la section Rappel du chapitre située à la fin de chaque chapitre.

Liens utiles

Voici quelques liens qui vous permettront d'obtenir du support technique pour le logiciel Visual Studio 2008, ainsi que pour le contenu de ce livre.

Support technique Visual Studio 2008

Voici deux sites web que je vous recommande de consulter pour y trouver des réponses aux questions que vous vous posez sur Visual Studio 2008 :

- *http://msdn2.microsoft.com/fr-fr/vstudio/default.aspx*
  (la page d'accueil pour Visual Basic).
- *http://www.microsoft.com/france/msdn/communautes/default.mspx*
  (les communautés travaillant autour des produits et technologies Microsoft).

Ces deux sites vous permettront d'entrer en contact avec des développeurs Visual Basic, des personnes travaillant chez Microsoft, des blogs, des groupes de news, des webcasts, des chats et autres groupes d'utilisateurs intéressants. Pour de plus amples informations concernant ces sites et d'autres ressources, en ligne ou pas, lisez dans ce livre l'annexe « Où trouver d'autres informations ».

Contenu d'accompagnement en ligne

Vous trouverez le contenu d'accompagnement de ce livre en français sur la page consacrée à cet ouvrage sur le site *dunod.com*.

Vous trouverez le contenu d'accompagnement de ce livre en anglais à l'adresse suivante :

http://www.microsoft.com/mspress/companion/0-7356-2131-4/

Site web Microsoft Press

Le site web de Microsoft Press contient la description de la gamme complète des ouvrages publiés par Microsoft Press :

http://www.dunod.com/mspress

Support technique

Malgré tous les soins apportés à la réalisation de cet ouvrage et de son contenu d'accompagnement, il se peut que des erreurs nous aient échappé et nous vous invitons à nous en faire part (en anglais) à l'adresse suivante :

http://support.microsoft.com/kb/905036/

À mesure que nous collectons les éventuelles corrections, nous les ajoutons à l'article correspondant dans la Base de connaissances de Microsoft. En outre, Microsoft Press offre des informations (en anglais) de support pour ce livre et son contenu d'accompagnement à l'adresse suivante :

http://www.microsoft.com/learning/support/books/

Partie I

Démarrer avec Visual Basic 2008

Dans cette partie :

La partie 1 présente un aperçu des techniques de programmation fondamentales de Visual Basic 2008 et constitue une introduction aux outils et dispositifs que vous allez employer lors de la plupart de vos sessions de programmation en Visual Basic. Vous apprendrez à utiliser l'Environnement de développement intégré (EDI) de Visual Studio 2008, avec son exhaustive collection d'outils de programmation, de fenêtres et de commandes de menu. Vous recevrez des instructions étape par étape sur la façon de construire à partir de zéro et d'exécuter plusieurs programmes intéressants. C'est là que vous devez commencer si vous abordez la programmation Visual Basic ou effectuez une mise à niveau depuis une version antérieure.

Le chapitre 2 explique comment employer conjointement contrôles, formulaires, propriétés et code de programme pour créer un amusant jeu de machine à sous. Le chapitre 3 présente les contrôles les plus utiles de la Boîte à outils, qui aident à la présentation d'informations ou de choix à l'utilisateur, récupèrent des entrées, travaillent avec des dates et des heures et se connectent au Web. Le chapitre 4 se concentre sur l'ajout de menus, de barres d'outils et de boîtes de dialogue à des programmes Visual Basic afin de leur donner l'aspect d'une application commerciale Windows.

Chapitre 1

Explorer l'environnement de développement intégré de Visual Studio

À la fin de ce chapitre, vous saurez :

- Démarrer Microsoft Visual Studio 2008
- Utiliser l'environnement de développement intégré Visual Studio
- Ouvrir et exécuter un programme Microsoft Visual Basic
- Modifier les propriétés
- Déplacer, redimensionner, aligner et masquer automatiquement les fenêtres d'outils
- Utiliser le navigateur de l'environnement de développement intégré.
- Ouvrir un Navigateur Web dans Visual Studio
- Utiliser de nouvelles commandes d'aide et personnaliser l'aide
- Personnaliser les paramètres de l'environnement de développement intégré conformément aux instructions pas à pas de ce manuel
- Sauvegarder vos modifications et quitter Visual Studio

Êtes-vous prêt à débuter avec Microsoft Visual Basic 2008 ? Ce chapitre apporte les compétences nécessaires pour mettre en œuvre et faire fonctionner rapidement et efficacement l'environnement de développement intégré (EDI), l'espace dans lequel vous allez écrire vos programmes Visual Basic. Il est recommandé de lire ce chapitre, que vous soyez nouveau venu à la programmation en Visual Basic ou un utilisateur avancé des versions antérieures de Visual Basic ou de Visual Studio.

Dans ce chapitre, vous allez apprendre à démarrer Visual Studio 2008 et à employer l'EDI pour ouvrir et exécuter un programme simple. Vous découvrirez également les commandes de menu essentielles de Visual Studio et les procédures de programmation. Vous ouvrirez et exécuterez un programme Visual Basic simple intitulé Musique, modifierez un paramètre de programmation, ou *propriété*, et déplacerez, dimensionnerez, alignerez et masquerez automatiquement des fenêtres d'outils. Nous verrons aussi comment basculer entre outils et fichiers à l'aide du Navigateur de l'EDI, ouvrir un Navigateur Web dans Visual Studio, trouver davantage d'informations grâce à l'aide en ligne et personnaliser l'EDI pour répondre aux instructions étape par étape de ce livre. Enfin, vous quitterez l'environnement de développement après avoir sauvegardé les modifications.

L'environnement de développement Visual Studio

Bien que Visual Basic soit le langage de programmation étudié dans ce livre, l'environnement de développement utilisé pour écrire des programmes est appelé environnement de développement intégré de Microsoft Visual Studio (EDI). Visual Studio est un espace de programmation puissant et personnalisable, qui comporte tous les outils nécessaires pour créer rapidement et efficacement des programmes fiables destinés à Windows et au web. La plupart des caractéristiques de l'EDI Visual Studio s'appliquent aussi à Visual Basic, Visual C++, et Visual C#. Voici comment démarrer Visual Studio :

Important Si vous n'avez pas encore installé les fichiers d'exercices, suivez les étapes « Bien démarrer avec cet ouvrage » et « Installer et utiliser les fichiers d'exercices » de l'introduction. Il est recommandé de placer les fichiers du projet et les sous-dossiers associés dans le dossier c:\vb08epe.

Démarrer Visual Studio 2008

1. Dans la barre de menus Windows, cliquez sur Démarrer, Tous les programmes puis sur le dossier Microsoft Visual Studio 2008.

   La liste des dossiers et des icônes du dossier Microsoft Visual Studio 2008 apparaît.

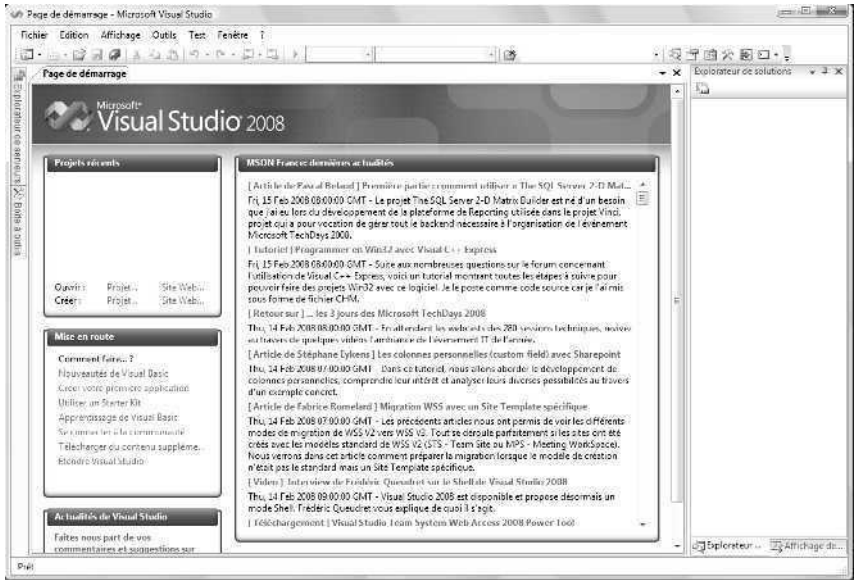
Remarque Pour suivre les étapes de cet ouvrage, vous devez avoir installé une version du logiciel Microsoft Visual Studio 2005. La plupart des procédures décrites fonctionnent avec les éditions Standard Edition, Professional Edition et Express Edition de Visual Studio 2008. Si vous avez accès aux outils Visual Studio 2008 Team Suite, vous pourrez suivre les procédures de cet ouvrage sans peine et vous aurez accès à d'intéressantes fonctions et capacités avancées. Toutefois, n'essayez pas d'utiliser cet ouvrage si vous possédez une version antérieure du logiciel Visual Basic. Dans ce cas, procurez-vous une version antérieure de cet ouvrage, comme *Microsoft Visual Basic 2005 Étape par étape* (qui décrit le logiciel Visual Basic) ou *Microsoft Visual Basic Professional 6.0 Étape par étape* (qui décrit le logiciel Visual Basic 6).

2. Cliquez sur l'icône Microsoft Visual Studio 2008.

   S'il s'agit du premier démarrage de Visual Studio, quelques minutes peuvent être nécessaires à la configuration de l'environnement. si vous êtes invité à spécifier les paramètres à employer, sélectionnez Paramètres de développement Visual Basic.

   Visual Studio démarre et l'environnement de développement apparaît à l'écran avec ses nombreux menus, outils et fenêtres de composants, parfois appelées *fenê-*

tres d'outils. La page de démarrage s'affiche également. Elle propose une série de liens, d'articles MSDN et d'options de projet. La page de démarrage est une source complète d'informations sur votre projet et de ressources au sein de la communauté de développement Visual Basic. C'est le chemin d'accès privilégié à de nouvelles informations sur Visual Studio après l'achat du logiciel.



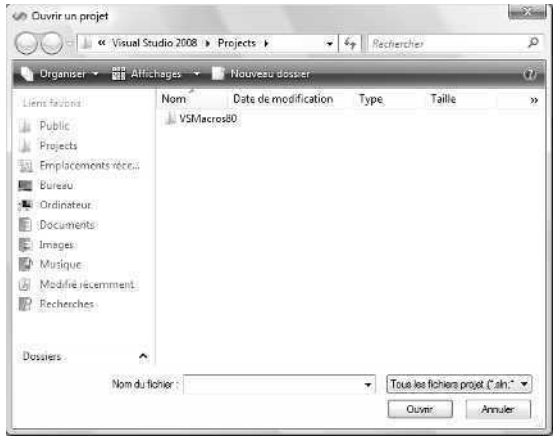
Au démarrage de Visual Studio, la plupart des programmeurs commencent par ouvrir un projet existant : une solution complète à poursuivre ou un projet de développement en cours.

Essayons d'ouvrir un projet existant que j'ai créé pour vous : le programme Musique.

Ouvrir un projet Visual Basic

1. Cliquez sur le lien Ouvrir Projet/Solution de la page de démarrage.

   La boîte de dialogue Ouvrir un projet et ses différentes options apparaissent à l'écran. Vous pouvez également afficher cette boîte de dialogue en cliquant sur les commandes Ouvrir, puis Projet du menu Fichier ou en appuyant sur CTRL+MAJ+O. Même si vous n'avez jamais utilisé Visual Studio auparavant, la boîte de dialogue Ouvrir un projet vous paraîtra simple, car elle ressemble à la boîte de dialogue Ouvrir de Microsoft Word ou Excel.



Astuce Dans la boîte de dialogue Ouvrir un projet, vous trouverez de nombreux raccourcis le long du bord gauche de la fenêtre. L'icône Projects est très utile : elle ouvre le dossier Projects situé dans le dossier Mes Documents\Visual Studio 2008 de votre système. Par défaut, Visual Studio sauvegarde vos projets dans ce dossier Projects et leur attribue à chacun un sous-dossier. L'aspect et le contenu des raccourcis dépendent de votre version de Windows, ainsi que de la façon dont vous avez configuré l'affichage des boîtes de dialogue. Les captures d'écran de ce livre sont réalisées sous Windows Vista.

2. Parcourez votre disque dur à la recherche du dossier c:\vb08epe.

Le dossier c:\vb08epe est l'emplacement par défaut de l'ensemble des fichiers d'exemples de cet ouvrage. Vous y trouverez les fichiers si vous avez suivi les instructions d'installation de la section « Installer et utiliser les fichiers d'exercices » de l'introduction. Si vous n'avez pas installé les fichiers d'exemples, fermez cette boîte de dialogue et installez-les, puis reprenez cette procédure.

3. Ouvrez le dossier chap01\Musique, puis double-cliquez sur le fichier Musique.

Si votre système affiche les extensions, le nom de ce fichier se termine par .sln. Visual Studio charge le formulaire, les propriétés et le code de la solution Musique. La page de démarrage est sans doute encore affichée, mais dans l'angle supérieur droit de l'écran, l'Explorateur de solutions liste quelques fichiers de la solution.

Dépannage Si un message d'erreur apparaît, indiquant que le format du projet que vous souhaitez ouvrir est nouveau, vous essayez peut-être de charger des fichiers Visual Basic 2008 dans un ancien logiciel Visual Basic .NET 2002, 2003 ou 2005. Il est impossible d'ouvrir les projets Visual Basic 2008 exemples avec les anciennes versions de Visual Basic. Pour savoir quelle version de Visual Basic ou Visual Studio équipe votre ordinateur, cliquez sur la commande À propos de... dans le menu d'aide.

Visual Studio propose une case à cocher particulière, appelée Toujours afficher la solution, qui permet de contrôler plusieurs options des solutions dans l'EDI. Vous la trouverez sur la page Général du nœud Projets et solutions dans la boîte de dialogue Options, que vous ouvrez en cliquant sur la commande Options du menu Outils. Lorsque la case est cochée (c'est en principe le cas par défaut), Visual Studio crée un sous-dossier pour chaque nouvelle solution et place le projet et ses fichiers dans un dossier séparé de la solution. Il propose, de surcroît, quelques options relatives aux solutions dans l'EDI, comme des commandes dans le menu Fichier et une entrée Solution dans l'Explorateur de solutions. Si vous préférez créer des dossiers séparés pour les solutions et voir les commandes et les paramètres de la solution, cochez cette case. Vous en apprendrez plus sur ces options à la fin du chapitre.

Projets et solutions

Dans Visual Studio, les programmes en cours de développement sont appelés *projets* ou *solutions* car ils contiennent plusieurs composants et non un seul fichier. Les programmes Visual Basic 2005 comportent un fichier projet (.vbproj) et un fichier solution (.sln). Si vous examinez ces fichiers à l'aide de l'Explorateur Windows, vous remarquerez que les icônes du fichier solution comportent un petit 9, indiquant leur version (dans le jargon interne à Microsoft, Visual Basic 2008 est désigné par VB9). Un fichier projet contient les données spécifiques à une tâche unique de programmation. Un fichier solution contient les données d'un ou de plusieurs projets. Les fichiers solution permettent de gérer plusieurs projets liés et sont similaires aux groupes de projets (.vbg) de Visual Basic 6.

Les exemples fournis avec cet ouvrage comportent généralement un projet unique pour chaque solution : ouvrir le fichier projet (.vbproj) équivaut à ouvrir le fichier solution (.sln). Si toutefois la solution comporte plusieurs projets, ouvrez le fichier solution. Le format des projets et des solutions de Visual Basic 2008 est nouveau, mais la terminologie de base reste la même que celle de Visual Basic .NET 2002, 2003 ou 2005.

Les outils de Visual Studio

Prenez le temps d'étudier l'EDI Visual Studio et d'identifier les outils de programmation et les fenêtres que vous serez amené à utiliser au cours de cette formation.

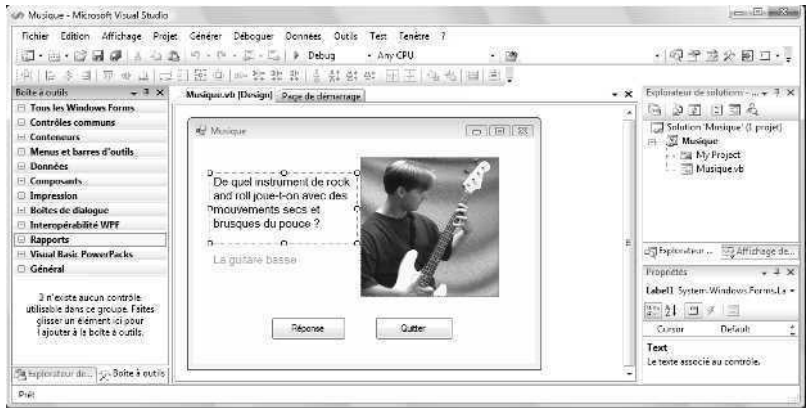
Si vous avez déjà écrit des programmes Visual Basic, vous reconnaîtrez de nombreux outils de programmation (mais probablement pas tous). Ces fonctionnalités sont les composants qui permettent de construire, d'organiser et de tester vos programmes Visual Basic. Certains des outils de programmation vous aident également à mieux connaître les ressources de votre système, dont les bases de données et les sites web auxquels vous pouvez vous connecter. Vous disposez également de quelques outils d'aide fort intéressants.

La *barre de menus* permet d'accéder à la plupart des commandes qui contrôlent l'environnement de développement.

Les menus et les commandes fonctionnent comme dans tous les programmes Windows et vous y accédez à l'aide du clavier ou de la souris. Sous la barre de menus, vous trouverez la *barre d'outils Standard*, un ensemble de raccourcis permettant d'exécuter les commandes et de contrôler l'EDI Visual Studio. Je suppose que vous avez déjà suffisamment employé Excel ou Word ou d'autres applications Windows pour que le concept de barre d'outils vous soit familier et que vous sachiez vous servir des commandes classiques comme Ouvrir, Enregistrer, Couper et Coller ! Vous serez toutefois probablement impressionné par le nombre et la diversité des barres d'outils proposées par Visual Studio pour des tâches de programmation. Vous apprendrez dans ce livre à employer plusieurs de ces barres d'outils. Vous pouvez consulter la liste complète des barres d'outils disponibles en cliquant droit sur n'importe quelle barre d'outils dans l'EDI.

La *barre des tâches* Windows longe le bas de l'écran. Elle sert à passer d'un composant Visual Studio à un autre ou à activer d'autres programmes Windows. Des icônes de barre des tâches peuvent apparaître pour Microsoft Internet Explorer, pour les anti-virus et les autres programmes installés sur votre système. La barre des tâches sera masquée dans la majorité des copies d'écran, pour montrer plus de l'EDI.

L'illustration suivante représente certains des outils et fenêtres de l'EDI Visual Studio. Cette illustration peut différer de votre propre environnement de développement. Vous apprendrez à connaître ces éléments (et à régler vos affichages) au cours du chapitre. Ce qui est montré ici est l'affichage « brut » obtenu lors du premier lancement du produit.



Les principaux outils visibles dans l'EDI Visual Studio sont le Concepteur, l'Explorateur de solutions, la fenêtre Propriétés, et la Boîte à outils. D'autres outils plus spécialisés tels que l'Explorateur de serveurs et l'Explorateur d'objets peuvent être visibles ou apparaître sous forme d'onglets dans l'EDI.

Les développeurs n'ont pas tous les mêmes préférences. Il est difficile de prévoir ce que vous allez voir si votre logiciel Visual Studio a déjà été utilisé. La figure précédente présente celle qui apparaît lorsque le logiciel vient d'être installé. Pour afficher un outil qui n'est pas visible, cliquez sur le menu Affichage et sélectionnez l'outil. Le menu Affichage s'étant étendu au cours des années, Microsoft a déplacé certains outils d'affichage dans un sous-menu appelé Autres fenêtres. Ce que vous recherchez s'y trouve probablement.

La taille et la forme exacte des outils et des fenêtres dépendent de la configuration de votre environnement de développement. Avec Visual Studio, vous pouvez aligner et attacher, ou *ancrer*, des fenêtres pour ne faire apparaître que les éléments que vous voulez voir. Vous pouvez également masquer partiellement des outils sous forme d'*onglets* longeant l'environnement de développement, puis passer rapidement d'un document à l'autre. Lorsque vous découvrez l'interface Visual Studio, il est difficile de savoir quels sont les outils importants pour démarrer et ceux que vous pourrez découvrir ultérieurement. L'environnement de développement sera probablement plus clair si vous paramétrez votre écran et le bureau Windows de manière à optimiser l'espace à l'écran, mais même dans ce cas, il se peut que vous manquiez de place.

Astuce La totalité des illustrations de cet ouvrage ont été réalisées avec une résolution d'écran de 1024 x 768, afin d'afficher clairement l'EDI. C'est ce que je préfère pour rédiger du code. Pour modifier la résolution d'écran, cliquez droit sur le bureau Windows et choisissez Personnaliser (avec Windows Vista) ou Propriétés (avec Windows XP).

La raison de cette complexité est simple : elle permet d'ajouter de nombreuses fonctions nouvelles et intéressantes à l'EDI tout en fournissant des mécanismes intelligents de gestion de l'encombrement. Il est ainsi possible d'ancrer les fenêtres, de les masquer automatiquement, de les laisser flotter, plus quelques autres états que je décrirai plus tard. Si vous débutez avec Visual Basic, la meilleure manière de gérer cette pléthore de fonctions est de masquer les outils que vous ne pensez pas utiliser souvent pour laisser de la place aux plus importants. Les outils essentiels pour démarrer la programmation dans Visual Basic (ceux que vous allez commencer à utiliser dès maintenant dans cet ouvrage) sont l'Explorateur de solutions, la fenêtre Propriétés, le Concepteur et la Boîte à outils. Ce n'est que plus tard dans ce livre que vous aurez recours aux fenêtres Explorateur de serveurs, Affichage de classes, Affichage des ressources, Explorateur d'objets et Déboguer.

Dans les exercices suivants, vous commencerez à vous servir des outils essentiels de l'EDI Visual Studio. Vous apprendrez également à afficher un Navigateur Web dans Visual Studio et à masquer les outils dont vous n'avez pas besoin dans l'immédiat.

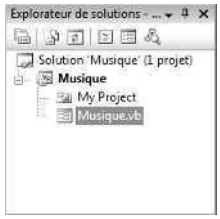
Le Concepteur

Si vous avez réalisé l'exercice précédent, le projet Musique est chargé dans l'environnement de développement Visual Studio. Toutefois, il est possible que l'interface utilisateur, ou *formulaire*, n'apparaisse pas dans Visual Studio. Les projets plus sophistiqués peuvent comporter plusieurs formulaires, mais ce petit programme n'en requiert qu'un. Pour afficher le formulaire du projet Musique dans l'EDI, utilisez l'Explorateur de solutions.

Afficher le Concepteur

1. Localisez la fenêtre Explorateur de solutions près de l'angle supérieur droit de l'environnement de développement. Si vous ne le voyez pas (il peut être masqué sous forme d'onglet dans un emplacement que vous ne voyez pas ou ne pas être visible), cliquez sur Explorateur de solutions dans le menu Affichage.

   Lorsque le projet Musique est chargé, l'Explorateur de solutions prend cette forme :



2. Dans l'Explorateur de solutions, cliquez sur le formulaire Musique.vb.

   La petite icône située à côté de tous les fichiers formulaire vous permet de les identifier facilement. Lorsque vous cliquez sur le fichier formulaire, Visual Studio le sélectionne dans l'Explorateur de solutions et des données concernant le fichier apparaissent dans la fenêtre Propriétés (si elle n'est pas masquée).

3. Cliquez sur le bouton Concepteur de vues dans l'Explorateur de solutions pour afficher l'interface utilisateur du programme.

   Le formulaire Musique est affiché dans le Concepteur :

   Notez qu'un onglet reste visible en haut du Concepteur pour la page de démarrage. Cliquez sur cet onglet pour afficher la page de démarrage, dans laquelle vous trouverez des articles et des liens et qui vous permettront d'ouvrir d'autres fichiers projet. Pour retourner au Concepteur, cliquez sur l'onglet Musique.vb [Design] au sommet du formulaire Musique.

   Exécutons maintenant un programme Visual Basic avec Visual Studio.

Astuce Si vous ne voyez pas les onglets Page de démarrage et Musique.vb [Design], il se peut que votre environnement de développement se trouve en mode d'affichage à fenêtres multiples et non dans des onglets. Pour modifier cette option, dans le menu Outils, cliquez sur Options. À gauche de la boîte de dialogue Options, développez la catégorie Environnement, puis cliquez sur Général. À droite, sous Disposition de la fenêtre, cliquez sur l'option Documents avec onglet puis sur OK. Au redémarrage de Visual Studio, les différentes fenêtres que vous ouvrirez possèdent des onglets : cliquez dessus pour passer de l'un à l'autre.

Exécuter un programme Visual Basic

Musique est un programme Visual Basic simple, conçu pour vous familiariser avec les outils de programmation de Visual Studio. Le formulaire que vous voyez a été personnalisé avec cinq objets (deux étiquettes, une image et deux boutons) et j'ai ajouté trois lignes de code pour que le programme pose une question simple et affiche la réponse appropriée. À l'écran, le programme « donne » la réponse car il fonctionne en mode conception, mais elle est masquée lorsque vous exécutez le programme. Vous apprendrez à créer des objets et à ajouter du code de programme dans le chapitre suivant. Pour l'instant, essayez d'exécuter le programme dans l'EDI.

Exécuter le programme Musique

1. Pour exécuter le programme Musique, cliquez sur le bouton Démarrer le débogage (flèche verte pointant à droite) de la barre d'outils Standard.

> **Astuce** Pour exécuter un programme dans l'environnement de développement Visual Studio, vous pouvez également appuyer sur F5 ou choisir la commande Démarrer le débogage du menu Déboguer.

Visual Studio charge et compile le projet dans un assembly (un ensemble structuré de modules, de données et de renseignements relatifs au programme), prépare le programme au test ou au débogage, puis (si la compilation a réussi) exécute le programme dans l'environnement de développement. Quand le programme est en cours, une icône apparaît dans la barre des tâches Windows. Après quelque temps, le formulaire Musique réapparaît, la photographie et la réponse étant cette fois masquées :

Le programme Musique pose sa question : De quel instrument de rock and roll joue-t-on avec des mouvements secs et brusques du pouce ?

2. Cliquez sur le bouton Réponse pour découvrir la solution.

   Le programme affiche la réponse (La guitare basse) sous la question, puis la photo d'un obscur joueur de basse de Seattle faisant la démonstration de la technique. Le programme test fonctionne.

3. Cliquez sur Quitter pour fermer le programme.

   Le formulaire se ferme et l'EDI Visual Studio redevient actif.

À propos des propriétés

Dans Visual Basic, chaque élément de l'interface utilisateur d'un programme, y compris le formulaire lui-même, possède un ensemble de propriétés configurables. Vous pouvez définir les propriétés lors de la conception à l'aide de la fenêtre Propriétés. Les propriétés peuvent également être référencées dans le code afin d'effectuer un travail utile lors de l'exécution du programme. Les éléments de l'interface utilisateur qui reçoivent des entrées de l'utilisateur ont souvent recours à des propriétés pour transmettre l'information au programme. Vous pourriez initialement éprouver des difficultés à appréhender le concept de propriété. Les considérer dans la vie courante peut faciliter les choses.

Prenez comme exemple une bicyclette. C'est un objet qui sert à se transporter d'un endroit à un autre. Puisqu'une bicyclette est un objet physique, elle possède plusieurs caractéristiques fondamentales. Elle possède une marque, une couleur, des vitesses, des freins et des roues et est construite d'une certaine façon (ce peut être une bicyclette de ville, un VTT ou un tandem). Selon la terminologie Visual Basic, ces caractéristiques sont les propriétés de la bicyclette. La plupart de ces propriétés sont définies lors de la construction de la bicyclette, mais d'autres, comme les pneus, la vitesse et des options tels des réflecteurs ou des rétroviseurs sont autant de propriétés susceptibles d'être modifiées alors que la bicyclette est employée. Elle peut même posséder des propriétés immatérielles (ou invisibles), comme sa date de fabrication, son propriétaire actuel ou un éventuel état de prêt ou de location. Au fil de votre travail avec Visual Basic, vous emploierez des propriétés d'objet de ces deux types, visible et invisible.

La fenêtre Propriétés

La fenêtre Propriétés sert à modifier les caractéristiques, ou *paramètres de propriété*, des éléments de l'interface utilisateur dans un formulaire. Un paramètre de propriété est la *qualité* de l'un des objets dans votre programme. Vous pouvez modifier les paramètres des propriétés à l'aide de la fenêtre Propriétés pendant la création de l'interface utilisateur, ou ajouter du code *via* l'Editeur de code pour modifier un ou plusieurs paramètres de propriété pendant l'exécution du programme. Il est, par exemple, possible de modifier la taille ou le type de police ainsi que l'alignement de la question affichée par le programme Musique. Visual Studio accepte toutes les polices installées sur votre système, comme dans Excel ou Word.

La fenêtre Propriétés comporte une liste d'objets qui énumère tous les éléments (objets) de l'interface utilisateur du formulaire. La fenêtre présente également les paramètres de propriété modifiables de chaque objet. Cliquez sur l'un ou l'autre des boutons pour afficher les propriétés dans l'ordre alphabétique ou par catégorie. Vous allez maintenant modifier la propriété de police de la première étiquette dans le programme Musique.

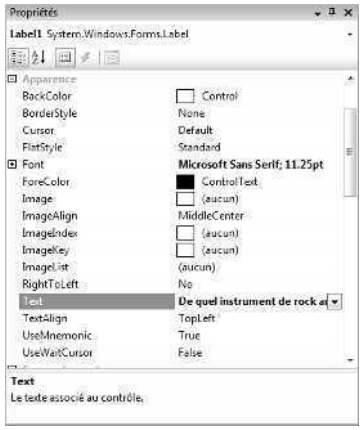
Modifier une propriété

1. Cliquez sur l'objet Label1 du formulaire. Label1 contient le texte « De quel instrument de rock and roll joue-t-on avec des mouvements secs et brusques du pouce ? ».

   Pour travailler sur un objet dans un formulaire, vous devez d'abord cliquer dessus. Lorsque vous sélectionnez un objet, les poignées de dimensionnement apparaissent. Les paramètres de propriétés de l'objet s'affichent dans la fenêtre Propriétés.

2. Dans la barre d'outils Standard, cliquez sur le bouton Fenêtre Propriétés.

   La fenêtre Propriétés peut être ou non visible dans Visual Studio, selon sa configuration et son utilisation dans votre système. En général, elle apparaît en dessous de l'Explorateur de solutions, sur le bord droit de l'environnement de développement. Si elle est visible, inutile de cliquer sur le bouton, activez simplement la fenêtre en cliquant dessus pour afficher une fenêtre similaire à celle de l'illustration suivante :



La fenêtre Propriétés affiche tous les paramètres de propriété du premier objet étiquette (*Label1*) du formulaire. Dans Visual Basic 2008, plus de 60 propriétés sont associées aux étiquettes. Les noms des propriétés apparaissent dans la colonne de gauche de la fenêtre et le paramètre actuel de chaque propriété figure dans la colonne de droite. Du fait du nombre important de propriétés (dont certaines sont rarement modifiées), Visual Studio les regroupe en catégories et les affiche en mode plan. Si un signe plus (+) figure à côté d'une catégorie, vous pouvez cliquer sur le titre de l'ensemble pour afficher toutes les propriétés de la catégorie. Si un signe moins (–) figure à côté d'une catégorie, toutes les propriétés sont visibles, mais vous pouvez masquer la liste sous le nom de la catégorie en cliquant sur le signe moins.

Astuce La fenêtre Propriétés possède deux boutons très pratiques que vous pouvez utiliser pour organiser les propriétés. Pour classer toutes les propriétés par ordre alphabétique et les classer dans quelques catégories, cliquez sur le bouton Alphabétique. Pour organiser la liste des propriétés en plusieurs catégories logiques, cliquez sur le bouton Catégories. Si vous débutez dans Visual Studio, je vous recommande cet affichage.

3. Faites défiler la fenêtre Propriétés vers le bas jusqu'à ce que la propriété *Font* soit visible.

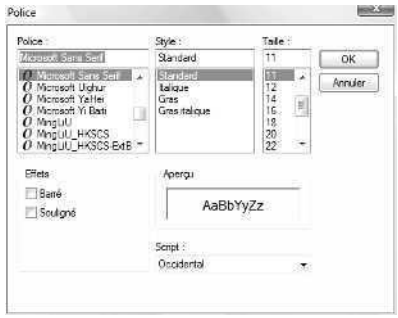
La fenêtre Propriétés défile comme une zone de liste normale. Si vous êtes en vue par catégorie, la propriété *Font* figure dans la catégorie Apparence.

4. Cliquez sur le nom de la propriété *Font* (dans la colonne de gauche).

Un aperçu de la police actuelle (Microsoft Sans Serif) s'affiche dans la colonne de droite et un bouton représentant des points de suspension apparaît près du nom de la police. Ce bouton indique qu'une boîte de dialogue permet de personnaliser le paramètre de propriété.

5. Cliquez sur le bouton équipé de points de suspension adjacent à la propriété *Font* dans la fenêtre Propriétés.

   Visual Studio affiche la boîte de dialogue Police, qui vous permet de donner de nouvelles caractéristiques de format au texte de l'étiquette sélectionnée dans votre formulaire. La boîte de dialogue Police comporte plusieurs options de format : chaque option sélectionnée modifie un paramètre de propriété.



6. Modifiez le style de la police de Normal à Italique. Cliquez sur OK pour confirmer.

   Visual Studio enregistre vos modifications et adapte les paramètres de propriété en conséquence. Vous pouvez vérifier les modifications en visualisant votre formulaire dans le Concepteur ou en agrandissant la catégorie Font de la fenêtre Propriétés.

   Modifiez maintenant un paramètre de propriété de l'objet *Label2* (l'étiquette qui contient le texte « La guitare basse »).

7. Dans le Concepteur, cliquez sur le deuxième objet étiquette (*Label2*).

   Lorsque vous sélectionnez l'objet, les poignées de dimensionnement apparaissent.

8. Cliquez sur la propriété *Font* dans la fenêtre Propriétés.

   L'objet *Label2* possède son propre ensemble de paramètres de propriétés. Les noms des propriétés sont les mêmes que ceux de l'objet *Label1*, mais les valeurs des paramètres de propriété sont différentes et l'objet *Label2* est indépendant sur le formulaire.

9. Cliquez sur le bouton équipé de points de suspension adjacent à la propriété *Font*, choisissez le style de police Gras, puis cliquez sur OK.

10. Faites défiler la fenêtre Propriétés, puis cliquez sur la propriété *ForeColor* dans la colonne de gauche.

11. Cliquez sur la flèche *ForeColor* dans la colonne de droite, cliquez sur l'onglet Personnaliser, puis sur la couleur pourpre.

    Sur le formulaire, le texte de l'objet *Label2* apparaît désormais en gras et en couleur pourpre.

Félicitations ! Vous venez d'apprendre à déterminer des propriétés dans un programme Visual Basic à l'aide de la fenêtre Propriétés de Visual Studio : c'est l'une des compétences essentielles pour devenir un programmeur Visual Basic.

Déplacer et redimensionner les outils de programmation

Étant donné les nombreux outils de programmation à intégrer à l'écran, l'environnement de développement peut devenir un espace très encombré. Pour vous donner un contrôle total de la forme et de la taille des éléments au sein de l'environnement de développement, Visual Studio permet de déplacer, dimensionner, ancrer et masquer automatiquement la plupart des éléments de l'interface que vous utilisez pour construire des programmes.

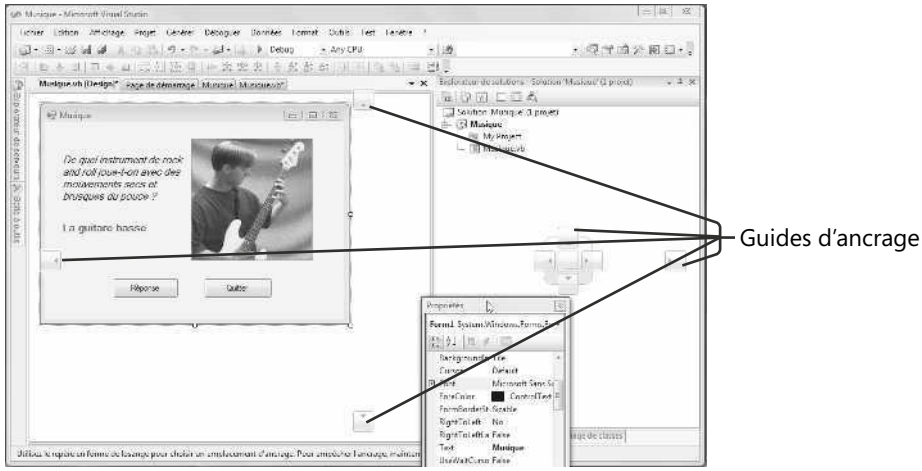
Pour déplacer une fenêtre d'outils, il suffit de cliquer sur sa barre de titre et de faire glisser l'objet vers un nouvel emplacement. Si vous alignez une fenêtre le long d'une autre, elle s'y attache ou *s'ancre*. Les fenêtres ancrables sont pratiques, car elles restent toujours visibles : elles ne sont pas masquées derrière d'autres fenêtres. Pour agrandir une fenêtre ancrable, faites simplement glisser l'une de ses bordures.

Pour fermer complètement une fenêtre, cliquez sur le bouton Fermer, dans son angle supérieur droit. Vous pourrez l'ouvrir ultérieurement, en cliquant sur la commande correspondante dans le menu Affichage.

Si vous ne souhaitez ni ancrer, ni fermer une fenêtre, essayez de la masquer automatiquement sur un bord de l'EDI en cliquant sur la petite punaise située à droite de sa barre de titre. Cette opération annule l'ancrage de la fenêtre et place le titre de l'outil dans un angle de l'environnement de développement, sur un onglet discret. Lorsque vous masquez automatiquement une fenêtre, notez que la fenêtre de l'outil reste visible tant que le pointeur de la souris se trouve dans la zone de la fenêtre. Lorsque vous déplacez la souris dans une autre partie de l'EDI, la fenêtre disparaît.

Pour activer une fenêtre que vous avez masquée automatiquement, cliquez sur l'onglet d'outils au coin de l'environnement de développement ou maintenez votre souris au-dessus de l'onglet. Une fenêtre auto-escamotable se reconnaît à la punaise dans la barre de titre, qui pointe vers le côté. En maintenant le pointeur de la souris au-dessus du titre, vous pouvez utiliser les outils du mode que j'appelle « coucou » (en d'autres termes, pour afficher rapidement une fenêtre auto-escamotée, cliquez sur son onglet, vérifiez ou définissez les données dont vous avez besoin, puis déplacez la souris pour faire disparaître la fenêtre). Si vous souhaitez que l'outil apparaisse en permanence, cliquez de nouveau sur la punaise Masquer automatiquement pour qu'elle pointe vers le bas et la fenêtre restera visible.

Visual Studio permet également d'afficher des fenêtres sous forme de fenêtres à onglets (fenêtres possédant des poignées d'attache qui sont partiellement masquées sous les autres fenêtres) et d'ancrer des fenêtres à l'aide de guides d'ancrage, comme ceci :



Les guides d'ancrage sont des icônes modifiables qui apparaissent dans l'EDI lorsque vous déplacez une fenêtre ou un outil de sa position d'ancrage vers un nouvel emplacement. Les guides d'ancrage sont liés à des zones rectangulaires ombrées de l'EDI : vous pouvez prévisualiser votre ancrage avant de le mettre en œuvre.

Pour maîtriser les techniques d'ancrage et de masquage automatique, il faut de l'expérience. Servez-vous des exercices suivants pour aiguiser vos compétences de gestion des fenêtres et mettez-les en pratique dans l'environnement de développement Visual Studio. Une fois que vous aurez terminé les exercices, configurez les outils Visual Studio à votre convenance.

Déplacer et redimensionner des fenêtres d'outils

Pour déplacer et redimensionner l'une des fenêtres de programmation dans Visual Studio, suivez ces étapes. Cet exercice illustre la manipulation de la fenêtre Propriétés, mais vous pouvez utiliser une autre fenêtre d'outils si vous le souhaitez.

Déplacer et redimensionner la fenêtre Propriétés

1. Si la fenêtre Propriétés n'est pas visible dans l'environnement de développement, cliquez sur le bouton Fenêtre Propriétés de la barre d'outils Standard.

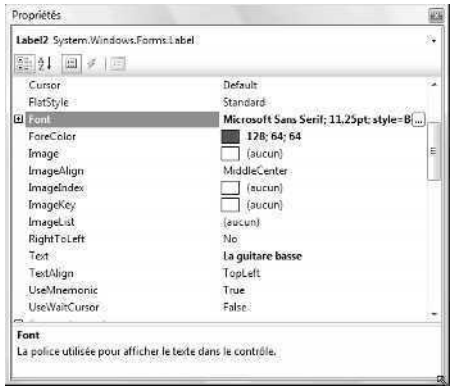
   La fenêtre Propriétés est activée dans l'EDI et sa barre de titre apparaît en surbrillance.

2. Double-cliquez sur la barre de titre de la fenêtre Propriétés pour afficher la fenêtre sous forme de fenêtre flottante (non ancrée).

3. À l'aide de la barre de titre de la fenêtre Propriétés, faites glisser la fenêtre vers un nouvel emplacement dans l'environnement de développement, mais ne l'ancrez pas (pas encore).

   Déplacer des fenêtres dans l'EDI Visual Studio assouplit votre gestion des outils et de l'apparence de votre environnement de développement. Vous allez maintenant redimensionner la fenêtre Propriétés pour afficher plusieurs paramètres de propriétés à la fois.

4. Déplacez la souris vers l'angle inférieur droit de la fenêtre Propriétés jusqu'à ce que le pointeur se transforme en poignée de dimensionnement. Faites glisser la bordure inférieure droite de la fenêtre vers le bas et vers la droite pour agrandir la fenêtre. Votre fenêtre Propriétés est désormais plus grande :



   Dans une fenêtre plus grande, vous pouvez travailler plus rapidement et plus clairement. N'hésitez pas à déplacer et redimensionner une fenêtre quand vous souhaitez voir davantage de contenu.

Ancrer des fenêtres d'outils

Si une fenêtre d'outils flotte dans l'environnement de développement, vous pouvez rétablir son ancrage initial en double-cliquant sur la barre de titre de la fenêtre (remarquez que vous avez utilisé la même technique pour rendre flottante une fenêtre ancrée dans l'exercice précédent). En double-cliquant sur une barre de titre, vous déclenchez une sorte de *bascule*, qui passe d'une position standard à une autre. Vous pouvez également attacher ou ancrer un outil flottant à un autre endroit. Cette opération permet de libérer de l'espace dans Visual Studio pour une tâche de programmation particulière, comme la création de l'interface utilisateur à l'aide du Concepteur. Essayez maintenant d'ancrer la fenêtre Propriétés à un autre endroit.

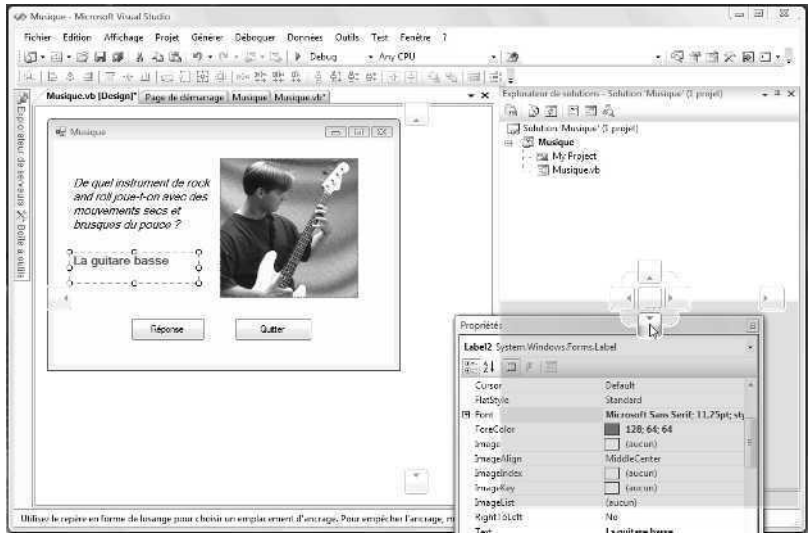
Ancrer la fenêtre Propriétés

1. Vérifiez que la fenêtre Propriétés (ou tout autre outil que vous souhaitez ancrer) flotte dans l'environnement de développement et n'est pas ancrée.

   Si vous avez effectué l'exercice précédent, la fenêtre Propriétés n'est pas ancrée.

2. Faites glisser la barre de titre de la fenêtre Propriétés vers le sommet, le bas, le bord droit ou gauche de l'environnement (à votre convenance), en prenant soin de faire glisser le pointeur de la souris sur l'un des guides d'ancrage (petites flèches) au bord de l'EDI Visual Studio ou sur le groupe de quatre guides d'ancrage situé au centre.

   Lorsque vous déplacez la souris au-dessus d'un guide d'ancrage, la fenêtre Propriétés se met en place et un rectangle ombré bleu indique l'apparence que prendra la fenêtre lorsque vous relâcherez le bouton de la souris. Notez que dans Visual Studio, il existe plusieurs emplacements d'ancrage valides pour les fenêtres d'outils. Il se peut que vous essayiez deux ou trois positions différentes avant de trouver celle qui vous convient le mieux. La fenêtre doit être placée à un endroit pratique et ne pas gêner l'accès à d'autres outils.



3. Relâchez le bouton de la souris pour ancrer la fenêtre Propriétés.

   La fenêtre se met en place.

> **Astuce** Pour basculer entre fenêtre ancrée, document avec onglet ou fenêtre flottante, cliquez droit sur la barre de titre de la fenêtre (ou sur l'onglet, s'il s'agit d'un document avec onglet), puis sélectionnez l'option souhaitée. Bien que la fenêtre Propriétés soit très maniable en tant que fenêtre ancrable, vous découvrirez qu'il est préférable d'afficher les fenêtres plus grandes sous forme de fenêtres à onglets.

4. Essayez plusieurs fois d'ancrer la fenêtre Propriétés à différents endroits pour vous habituer.

   La manipulation des fenêtres peut paraître confuse ; avec l'expérience, elle devient une habitude. En règle générale, vous préférerez créer des fenêtres suffisamment grandes pour afficher les informations dont vous avez besoin pendant que vous effectuez d'autres tâches plus importantes dans le Concepteur et l'Editeur de code.

Masquer des fenêtres d'outils

Pour masquer une fenêtre d'outils à l'arrière d'un onglet d'outils sur un bord de l'EDI, cliquez sur la punaise Masquer automatiquement située à droite de la barre de titre. Cliquez à nouveau dessus pour réactiver la fenêtre ancrée. Pour masquer automatiquement une fenêtre d'outils, vous pouvez aussi utiliser la commande Masquer automatiquement du menu Fenêtre (ou cliquer droit sur la barre de titre et sélectionner Masquer automatiquement). Essayez.

Utiliser la fonction Masquer automatiquement

1. Localisez la punaise Masquer automatiquement dans la barre de titre de la fenêtre Propriétés.

   La punaise est en position « basse » ou « enfoncée ». La fenêtre Propriétés est « épinglée » en position ouverte et le masquage automatique est désactivé.

2. Cliquez sur la punaise dans la barre de titre de la fenêtre Propriétés.

   La fenêtre Propriétés glisse en dehors de l'écran et elle est remplacée par un petit onglet nommé Propriétés. L'activation du masquage automatique permet de libérer de l'espace de travail dans Visual Studio. Toutefois, la fenêtre masquée reste rapidement accessible.

3. Maintenez le pointeur de la souris au-dessus de l'onglet Propriétés. Vous pouvez également cliquer sur l'onglet Propriétés si vous le souhaitez.

   La fenêtre Propriétés réapparaît immédiatement.

4. Cliquez n'importe où dans l'EDI : elle disparaît à nouveau.

5. Enfin, affichez à nouveau la fenêtre Propriétés puis cliquez sur la punaise de la barre de titre de la fenêtre.

La fenêtre Propriétés reprend sa position ancrée et vous pouvez l'utiliser sans craindre qu'elle ne glisse.

Prenez dès maintenant le temps de déplacer, redimensionner, ancrer et masquer automatiquement des fenêtres d'outils dans Visual Studio, afin de créer votre version de l'environnement de travail parfait. En parcourant cet ouvrage, vous pourrez ajuster régulièrement les paramètres de votre fenêtre pour adapter votre zone de travail aux nouveaux outils employés. Si besoin est, revenez à cette section et exercez-vous de nouveau.

Astuce Visual Studio 2008 vous permet désormais d'enregistrer les paramètres de votre fenêtre et de votre environnement de programmation et de les copier sur un deuxième ordinateur pour les partager avec les membres de votre équipe de programmation. Pour tester cette nouvelle fonction, cliquez sur la commande Paramètres d'importation et d'exportation du menu Outils et suivez les instructions de l'assistant pour exporter (enregistrer) ou importer (charger) des paramètres depuis un fichier.

Basculer entre fichiers et outils ouverts à l'aide du Navigateur EDI

Visual Studio 2008 possède un dispositif qui facilite encore plus le basculement entre les fichiers et les outils de programmation ouverts sur l'environnement de développement. Ce dispositif, nommé Navigateur EDI, permet de parcourir l'ensemble des fichiers et outils ouverts à l'aide de combinaison de touches d'accès rapide, comme vous pouvez parcourir les programmes ouverts depuis la barre des tâches Windows. Essayez maintenant.

Travail avec le Navigateur EDI

1. Maintenez enfoncée la touche CTRL et appuyez sur la touche TAB pour ouvrir le Navigateur EDI.

   Le navigateur EDI s'ouvre et affiche les fichiers et outils ouverts de l'EDI. Votre écran devrait ressembler à ceci :

2. Tout en maintenant enfoncée la touche CTRL, appuyez à plusieurs reprises sur TAB pour parcourir la liste des fichiers ouverts, jusqu'à ce que le fichier souhaité soit sélectionné.

Pour parcourir les fichiers en ordre inverse, maintenez enfoncées les touches CTRL+MAJ et appuyez sur TAB.

Vous pouvez également sélectionner un fichier ou un outil ouvert en cliquant sur son nom.

3. Lorsque vous en avez fini avec le Navigateur EDI, relâchez la touche CTRL.

   Le dernier élément sélectionné dans le Navigateur EDI devient l'élément actif.

Astuce Pour parcourir l'ensemble des outils ouverts sans ouvrir le Navigateur IDE, vous pouvez également appuyer sur ALT+F7. MAJ+ALT+F7 effectue le parcours dans l'ordre inverse.

Ouvrir un Navigateur Web dans Visual Studio

Une des fonctions pratiques de Visual Studio est la possibilité d'ouvrir un Navigateur Web simple dans l'environnement de développement. Le navigateur apparaît sous forme de document avec onglet dans l'EDI ; il prend peu de place et peut être ouvert immédiatement si besoin est. Vous pourriez ouvrir un Navigateur Web autonome comme Internet Explorer et le laisser ouvert dans la barre des tâches Windows, mais exécuter un Navigateur Web *dans* Visual Studio facilite la navigation et la copie de données dans Visual Studio. Essayez maintenant le Navigateur Web de Visual Studio.

Ouvrir le Navigateur Web de Visual Studio

1. Dans le menu Affichage, cliquez sur Autres fenêtres, puis sur la commande Navigateur Web.

   Le Navigateur Web s'affiche :

Par défaut, le navigateur est un document avec onglet, mais vous pouvez l'utiliser sous forme de fenêtre flottante ou de fenêtre ancrée en cliquant droit sur la barre de titre de la fenêtre puis en cliquant sur la commande Flottante ou Ancrable.

Astuce Vous pouvez modifier la page d'accueil par défaut de la fenêtre du Navigateur Web à l'aide des paramètres de la boîte de dialogue Options. Ouvrez celle-ci en cliquant sur Options dans le menu Outils. Développez Environnement, puis cliquez sur Navigateur Web. Modifiez le paramètre Page d'accueil en l'URL de la page que vous voulez voir afficher par défaut.

2. Testez le navigateur et son fonctionnement dans l'EDI.

   Bien qu'il soit plus simple qu'Internet Explorer ou un autre navigateur complet, il constitue un ajout très utile à l'ensemble des outils Visual Studio.

3. Lorsque vous avez terminé, cliquez sur le bouton Fermer, à droite de la barre de titre du Navigateur Web, pour fermer la fenêtre. Si la fenêtre de votre navigateur apparaît sous forme de document avec onglet, vous devrez peut-être la transformer en fenêtre flottante d'abord.
4. Si la barre d'outils Aide apparaît toujours dans l'EDI, effectuez dessus un clic droit, puis cliquez sur Aide dans le menu contextuel pour la retirer.

Obtenir de l'aide

Visual Studio comporte un centre de référence électronique appelé Documentation Microsoft Visual Studio 2008, que vous pouvez utiliser pour vous documenter sur l'EDI Visual Studio, le langage de programmation Visual Basic, les ressources du .NET Framework, les communautés en ligne spécialisées dans Visual Basic et Visual Studio et les autres outils de la suite Visual Studio. Prenez le temps de passer en revue ces nombreuses ressources avant de passer au chapitre 2, au cours duquel vous allez créer votre premier programme.

Deux sources d'aide : les fichiers d'aide locaux et le contenu en ligne

Il existe deux ressources d'aide dans Visual Studio :

- Les fichiers locaux d'aide installés avec Visual Studio 2008.
- L'aide en ligne (sur Internet) *via* MSDN Online, les groupes de news MSDN et un ensemble de sites web de développeurs parrainés par Microsoft appelé communauté Codezone. La communauté Codezone est particulièrement intéressante, car elle regroupe des développeurs professionnels qui utilisent Visual Studio et Visual Basic 2008 pour écrire des applications véritablement opérationnelles. Le contenu et les conseils qu'ils dispensent sont mis à jour en permanence et reflètent les ten-

dances, les préoccupations et les succès de la communauté de programmation Visual Basic.

Configurez dès maintenant votre système d'aide pour bénéficier de ressources d'aide locale et en ligne tout en découvrant Visual Basic.

Définir les options du système d'aide

1. Pour ouvrir le système d'aide, dans le menu Aide, cliquez sur Comment faire.

   Visual Studio prend l'aide en charge au moyen d'un outil HTML appelé Explorateur de documents Microsoft. Plusieurs commandes du menu Aide permettent d'ouvrir cet outil. Chaque commande ouvre et configure l'Explorateur de documents pour afficher un type d'information différent. Comment faire est l'un des meilleurs points de départ : il présente une liste hiérarchique de tâches de programmation standard, que vous pouvez utiliser pour trouver rapidement les informations dont vous avez besoin. Votre écran présente un résultat similaire à

2. Cliquez sur un ou plusieurs sujets de la liste Comment faire pour découvrir le type de données que vous obtenez.

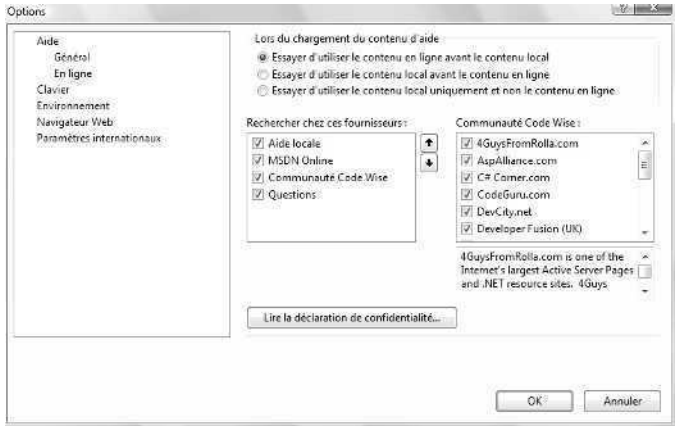
   Le système d'aide contient des centaines de descriptions techniques et de didacticiels (dont un grand nombre avec un exemple de code). Vous allez maintenant configurer l'aide pour qu'elle affiche seulement le contenu souhaité à l'ouverture.

3. Dans la barre de menus de l'Explorateur de documents, cliquez sur Outils, puis sur la commande Options.

   Vous découvrez les options de personnalisation permettant de configurer le fonctionnement du système d'aide et les ressources qu'il mobilise lorsqu'il recherche des informations.

4. Développez la catégorie Aide puis cliquez sur En ligne dans.

   Votre écran se présente ainsi :



   Je vous conseille de définir vos options en ligne comme sur cet écran. Pour commencer, sélectionnez le bouton d'option du haut pour utiliser d'abord l'aide en ligne (la mieux à jour), puis l'aide locale située sur votre disque. Si toutefois vous possédez une connexion Internet lente ou pas de connexion, contentez-vous uniquement des sources locales. Ensuite, assurez-vous que MSDN Online et la communauté Codezone sont sélectionnés, pour que Visual Studio télécharge les articles récents des développeurs Visual Basic à chaque nouvelle utilisation de la commande Rechercher. Si, après quelque temps, vous préférez une ou deux communautés Codezone aux autres, vous pouvez adapter l'ordre de recherche ou retirer des éléments de la liste.

5. Sélectionnez les options de configuration qui vous conviennent, puis cliquez sur OK pour les enregistrer.

   Vous pouvez à tout moment retourner dans le menu Options lorsque le système d'aide est ouvert. Essayez maintenant d'utiliser un autre dispositif astucieux : les Favoris de l'aide, qui fonctionnent comme la liste des Favoris Internet Explorer.

Créer une liste de Favoris dans l'Aide

1. Dans la barre d'outils de l'Explorateur de documents, cliquez sur le bouton Ajouter aux favoris de l'aide (situé après le bouton Favoris de l'aide, dont l'icône représente une page sur laquelle figure un signe plus (+)).

   Lorsque vous cliquez sur ce bouton, l'Explorateur de documents ajoute l'article actif à votre liste de documents d'aide préférés. Vos ressources d'aide favorites sont désormais organisées et à portée de main !

2. Cliquez sur l'onglet Rechercher en haut de la fenêtre Explorateur de documents.

   La fenêtre Rechercher s'ouvre sur un outil de recherche de texte dans vos ressources d'aide locales et en ligne.

3. Cliquez sur la flèche Langage (filtre de contenu) et supprimez les coches de tous les langages à l'exception de Visual Basic.

   Vous pouvez configurer le système d'aide pour limiter vos recherches aux langages, aux technologies ou aux sujets souhaités à l'aide des flèches de filtre. Vous démarrez avec Visual Studio : il est peut-être préférable de limiter votre recherche à Visual Basic pour l'instant.

4. Dans la zone Rechercher, tapez **données contrôles** puis appuyez sur Entrée.

   Visual Studio recherche la chaîne de texte « données contrôles » dans vos fichiers d'aide locaux et en ligne, dans les communautés MSDN, les groupes de discussion et les communautés Codezone. Faites particulièrement attention à la zone de liste Ordre de tri dans la fenêtre Recherche : elle permet de modifier l'affichage des articles identifiés.

5. Cliquez sur la source d'aide MSDN Online à droite de la fenêtre pour afficher les résultats de votre recherche en ligne

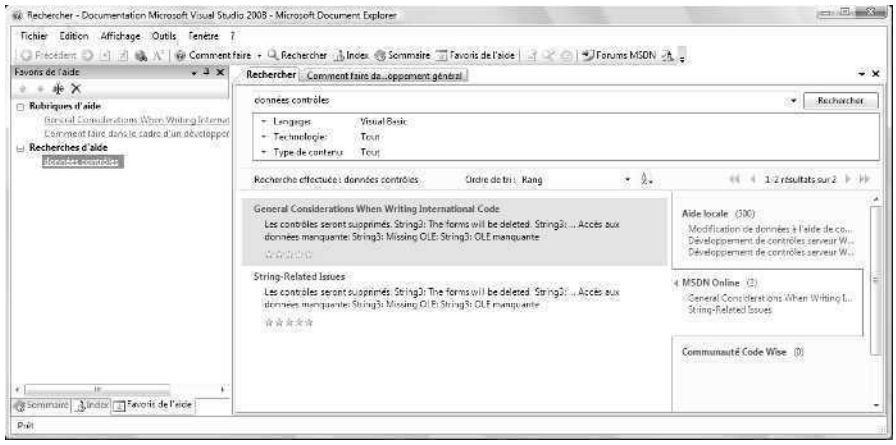
   Les informations d'aide en ligne affichées sont dynamiques : elles se modifient régulièrement afin de refléter toute nouvelle information publiée sur MSDN.

6. Enregistrez le premier élément (affiché en surbrillance) dans votre liste des favoris d'aide.

7. Cliquez sur l'onglet Rechercher puis cliquez sur le bouton Enregistrer la recherche dans la barre d'outils de l'Explorateur de documents.

Astuce Outre les articles d'aide, vous pouvez enregistrer dans votre liste de favoris des résultats de recherche importants pour vous.

Votre écran est similaire à l'illustration suivante : la fenêtre Favoris de l'aide contient maintenant les deux nouveaux favoris que vous avez ajoutés.



8. Cliquez sur le bouton Renommer dans la fenêtre Favoris de l'aide. Vous pouvez également effectuer un clic droit sur la recherche que vous avez sauvegardée, puis cliquer sur Renommer. L'Explorateur de documents sélectionne le nom de votre recherche et vous permet de le modifier pour qu'il corresponde mieux à la recherche. Cette étape est facultative, mais utile.
9. Tapez **Lier des données à des contrôles**, puis appuyez sur ENTRÉE.

   L'Explorateur de documents modifie le nom de votre recherche dans votre liste de favoris. J'ai choisi ce titre car il me semble plus clair que la chaîne de texte saisie initialement. Toutefois, vous pouvez donner un autre titre, plus adapté aux résultats de recherche obtenus.
10. Cliquez sur le bouton Comment faire dans la fenêtre Favoris de l'aide.

    Le premier article que vous avez enregistré apparaît dans l'Explorateur de documents. Vous allez maintenant effacer un favori, ce qui pourra s'avérer utile lorsque vous estimerez que votre liste d'articles d'aide favoris est devenue trop longue.
11. Cliquez sur le bouton Supprimer dans la fenêtre Favoris de l'aide.
12. Si vous êtes invité à confirmer votre intention d'effacer ce favori, cliquez sur Oui.

    L'article Comment faire est effacé de votre liste de favoris (mais pas du système d'aide).
13. Cliquez sur le bouton Fermer de la barre de titre de l'Explorateur de documents.

Il existe d'autres fonctions d'aide à découvrir, mais il est temps pour moi de résumer les commandes d'aide les plus importantes et pour vous d'écrire votre premier programme au cours du prochain chapitre.

Résumé des commandes d'Aide

Voici une petite compilation de commandes d'Aide utiles et de leur utilisation dans l'EDI Visual Studio :

Pour obtenir de l'aide	Faites ceci
Organisée par tâche de programmation	Dans le menu Aide de Visual Studio, cliquez sur Comment faire
Sur la fonction ou la commande que vous utilisez actuellement	Dans le menu Aide de Visual Studio, cliquez sur Aide dynamique.
Par sujet ou activité	Dans le menu Aide de Visual Studio, cliquez sur Sommaire.
Tout en travaillant dans l'Editeur de code	Cliquez sur le mot clé ou l'instruction du programme qui vous intéresse, puis appuyez sur F1.
Tout en travaillant dans une boîte de dialogue	Cliquez sur le bouton d'aide (point d'interrogation) dans les boîtes de dialogue sélectionnées (par exemple, la boîte de dialogue affichée lorsque vous sélectionnez la commande Options du menu Outils).
À la recherche d'un mot clé spécifique	Dans le menu Aide, cliquez sur Rechercher et tapez le terme recherché. Filtrez et organisez les résultats de recherche à l'aide de la zone de liste Ordre de tri.
Sur les sites web MSDN et Visual Studio indépendants	Dans le menu Aide, cliquez sur Forums MSDN.
Pour contacter Microsoft et obtenir un support produit	Dans le menu Aide, cliquez sur Support technique.

Personnaliser les paramètres de l'EDI pour réaliser les exercices pas à pas

Tout comme la fenêtre d'outils et le système d'aide, les paramètres du compilateur de l'EDI Visual Studio sont personnalisables. Il est important de passer en revue certains de ces paramètres, afin de configurer votre version de Visual Studio pour qu'elle soit compatible avec les exercices de programmation pas à pas qui suivent. Vous allez également apprendre à personnaliser Visual Studio de manière générale, pour organiser le logiciel à votre convenance à mesure que vous acquerrez de l'expérience de programmation.

Configurer l'EDI pour le développement en Visual Basic

Le premier paramètre que vous devez vérifier a été défini lors de l'installation de Visual Studio sur votre ordinateur. Pendant l'installation, Visual Studio vous a demandé de quelle manière vous souhaitiez configurer votre environnement de développement

général. Visual Studio est un outil de programmation plurifonctionnel : vous aviez plusieurs options (développement Visual Basic, Visual C++, Visual C#, web et même un environnement de programmation universel qui correspond à peu près à des versions antérieures de Visual Studio). Le choix effectué a configuré outre l'Editeur de code et les outils de développement les commandes de menus et les barres d'outils, ainsi que le contenu de plusieurs fenêtres d'outils. C'est pourquoi, si vous souhaitez utiliser cet ouvrage pour apprendre la programmation en Visual Basic, mais que vous avez configuré votre logiciel pour un autre langage, certaines commandes de menu et procédures décrites dans cet ouvrage ne correspondront pas exactement à la configuration actuelle de votre logiciel. L'emplacement de la commande Navigateur Web, mentionnée précédemment, en est un exemple. Heureusement, vous pouvez y remédier et modifier les paramètres de votre environnement à l'aide de la commande Paramètres d'importation et d'exportation du menu Outils. Suivez les prochaines étapes pour paramétrer votre environnement pour le développement en Visual Basic, qui est la configuration recommandée pour cet ouvrage.

Configurer l'EDI pour le développement en Visual Basic

1. Dans le menu Outils, cliquez sur Importation et exportation de paramètres.

   Vous pouvez utiliser l'assistant qui s'affiche pour enregistrer les paramètres de votre environnement afin de les utiliser sur un autre ordinateur, charger ceux d'un autre ordinateur ou réinitialiser les vôtres. C'est cette dernière option que vous allez choisir maintenant.

2. Cliquez sur Réinitialiser tous les paramètres, puis sur Suivant.

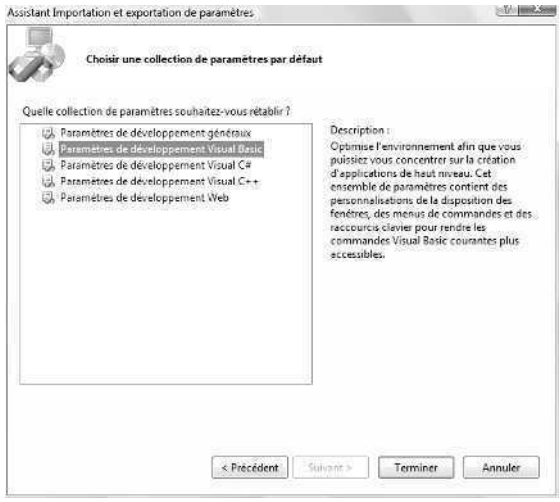
   Visual Studio vous demande si vous souhaitez enregistrer vos paramètres actuels dans un fichier avant de configurer l'EDI pour un autre type de programmation. Il est toujours bon de sauvegarder vos paramètres actuels, de manière à les retrouver si les nouveaux paramètres ne fonctionnent pas.

3. Vérifiez que le bouton Oui, enregistrer mes paramètres actuels est sélectionné et notez le nom du fichier et l'emplacement du dossier dans lequel Visual Studio va stocker les paramètres.

   Pour les restaurer, utilisez le même assistant et cliquez sur le bouton Importer les paramètres d'environnement sélectionnés.

4. Cliquez sur Suivant pour visualiser la liste des paramètres par défaut que vous pouvez utiliser dans Visual Studio.

   Selon les composants Visual Studio installés, vous verrez une liste de paramètres plus ou analogue à celle de l'illustration suivante :



5. Cliquez sur Paramètres de développement Visual Basic (s'il n'est pas déjà sélectionné), puis sur Terminer.

   L'assistant modifie les paramètres de votre EDI, les commandes des menus, les barres d'outils et les paramètres de quelques boîtes de dialogue, des fenêtres d'outils et de l'Editeur de code. Si la fenêtre d'aide d'un exercice précédent est restée ouverte, une alerte vous informe que le système d'Aide ne peut être mis à jour entièrement tant que vous n'avez pas fermé et redémarré l'aide.

   N'hésitez pas à répéter ce processus de personnalisation à chaque fois que vous souhaitez réinitialiser vos paramètres (par exemple, si vous regrettez de les avoir personnalisés) ou si vous souhaitez personnaliser Visual Studio pour un autre outil de programmation.

6. Cliquez sur Fermer pour fermer l'assistant.

Vérifier les paramètres du projet et du compilateur

Si vous venez de configurer les paramètres de votre environnement pour faire du développement avec Visual Basic, vous êtes prêt à débuter les exercices de programmation. Si vous n'avez pas réinitialisé vos paramètres (par exemple, si vous étiez déjà configuré pour le développement en Visual Basic et que vous utilisez Visual Studio 2008 depuis un certain temps, ou si vous partagez votre ordinateur avec d'autres programmeurs susceptibles d'avoir modifié les paramètres par défaut), suivez les prochaines étapes pour vérifier que les paramètres des projets, des solutions et du compilateur Visual Basic correspondent à ceux qui sont utilisés dans l'ouvrage.

Vérifier les paramètres du projet et du compilateur

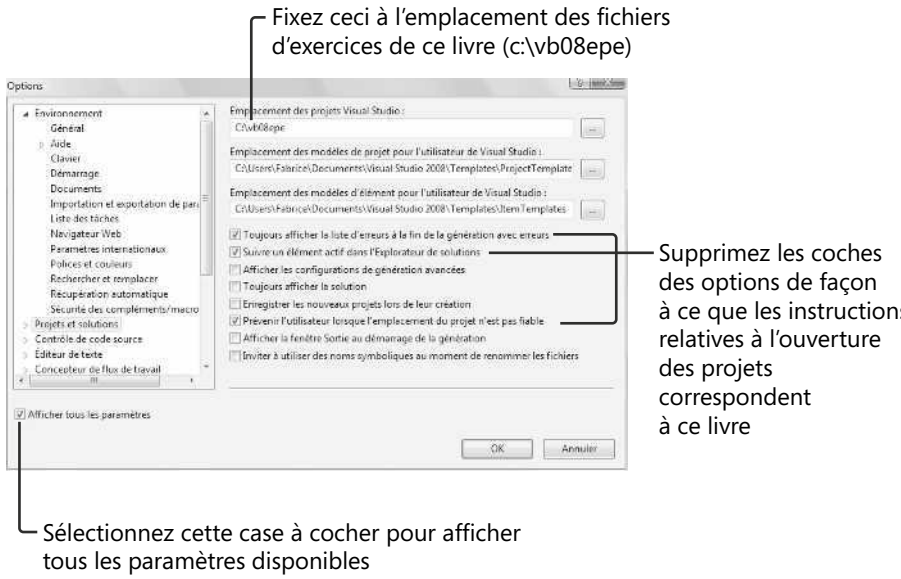
1. Cliquez sur la commande Options du menu Outils pour afficher la boîte de dialogue Options.

   La boîte de dialogue Options permet de personnaliser de nombreux paramètres de Visual Studio. Pour visualiser tous les paramètres modifiables, cochez la case Afficher tous les paramètres, dans l'angle inférieur gauche de la boîte de dialogue.

2. Développez la catégorie Projets et solutions puis cliquez sur la rubrique Général de la boîte de dialogue Options.

   Cet ensemble de cases à cocher et d'options configure les paramètres des projets et des solutions Visual Studio.

3. Ajustez les paramètres de votre logiciel à ceux qui sont utilisés dans cet ouvrage à l'aide la boîte de dialogue suivante :



Il est particulièrement recommandé de supprimer les coches des cases Toujours afficher la solution et Enregistrer les nouveaux projets lors de leur création. La première option affiche des commandes supplémentaires de solution dans l'EDI, ce qui est superflu pour les solutions ne comportant qu'un projet (c'est le cas de la plupart des programmes de cet ouvrage). La deuxième option (qui n'existait pas dans Visual Studio .NET 2003 et Visual Basic 6) reporte l'enregistrement de votre projet jusqu'à ce que vous cliquiez sur la commande Enregistrer tout du menu Fichier et déclariez un emplacement à cet effet. Cette fonction d'enregistrement « retardé » vous permet de créer un programme test, de le compiler et de le déboguer et même de l'exécuter sans l'enregistrer sur le disque. Une fonction très utile pour créer rapidement un programme de test que vous ne souhaitez pas conserver (en termes de

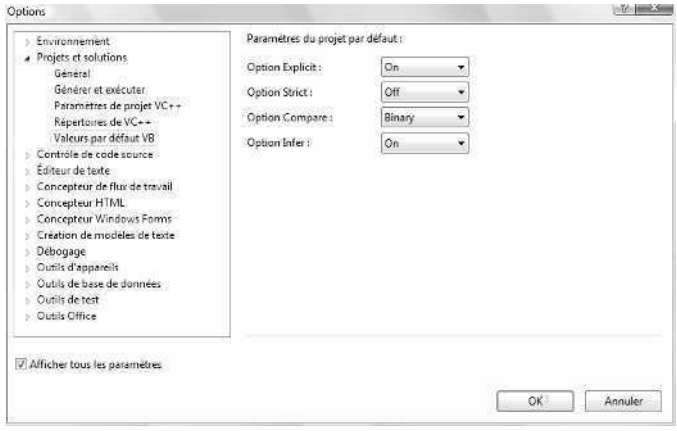
traitement de texte, cela revient au même que d'ouvrir un nouveau document Word, de saisir une adresse pour une étiquette, d'imprimer l'adresse puis de quitter Word sans enregistrer le fichier). Ce paramètre par défaut vous incite à sauvegarder les projets des exercices après les avoir créés, bien que vous puissiez également les sauvegarder à l'avance en cochant la case Enregistrer les nouveaux projets lors de leur création.

Vous observerez que j'ai sélectionné le dossier c:\vb08epe, emplacement par défaut des fichiers d'exercices de cet ouvrage, comme emplacement pour les projets Visual Studio. Je vous invite à enregistrer la plupart des projets que vous allez créer dans ce dossier et de les différencier du projet terminé en les faisant précéder des mots Mon ou Ma (comme par exemple Ma Musique).

Une fois que vous avez ajusté ces paramètres, il vous faut vérifier quatre paramètres du compilateur Visual Basic.

4. Dans la boîte de dialogue Options, cliquez sur l'élément Valeurs par défaut VB.

Visual Studio affiche une liste de quatre paramètres du compilateur : *Option Explicit*, *Option Strict*, *Option Compare* et *Option Infer*. Votre écran se présente de la manière suivante :



Bien que la description détaillée de ces paramètres dépasse le cadre de ce chapitre, vérifiez que *Option Explicit* est sur *On* et *Option Strict* sur *Off* (paramètres par défaut de programmation en Visual Basic dans Visual Studio). *Option Explicit On* est un réglage qui vous demande de déclarer une variable avant de l'utiliser dans un programme (une très bonne habitude de programmation que j'encourage). *Option Strict Off* permet de combiner des variables et des objets de différents types sous certaines conditions sans générer d'erreur de compilateur. Un nombre peut par exemple être assigné à un objet zone de texte sans provoquer d'erreur. Bien que cette pratique de programmation puisse être gênante, *Option Strict Off* est un paramètre très utile pour certains types de programmes de démonstration. Si vous ne maintenez pas ce paramètre, quelques projets afficheront des messages d'erreur lors de leur exécution.

Option Compare détermine la méthode de comparaison lors du tri et de la comparaison de chaînes. Vous trouverez plus d'informations sur la comparaison de chaînes et le tri de texte au Chapitre 13, « Explorer le traitement des fichiers texte et des chaînes »

Option Infer est un nouveau réglage de Visual Basic 2008. En fixant *Option Strict* à *Off* et *Option Infer* à *On*, vous pouvez déclarer des variables sans stipuler explicitement de type de données. Dans un tel cas, le compilateur Visual Basic va déduire (ou tenter de le deviner) le type de données d'après la première affectation effectuée pour la variable. Les concepteurs de Visual Basic autorisent ce type de déclaration dans l'espoir de préserver la mémoire vive de votre ordinateur. Vous en apprendrez plus sur ce dispositif au cours du Chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework ».

En règle générale, mieux vaut fixer *Option Infer* à *Off* afin d'éviter tout résultat inattendu dans l'emploi des variables au sein de vos programmes. J'ai fixé *Option Infer* à *Off* dans la plupart des projets exemples qui accompagnent ce livre.

5. N'hésitez pas à passer en revue les autres paramètres de la boîte de dialogue Options, relatifs à votre environnement de programmation et à Visual Studio. Lorsque vous avez terminé, cliquez sur OK pour la refermer.

Vous êtes prêt à quitter Visual Studio et à démarrer la programmation.

Aller plus loin : Quitter Visual Studio

Chaque chapitre de cet ouvrage se termine par une section intitulée « Aller plus loin », qui vous permet de découvrir une autre compétence relative au sujet traité. Après la rubrique « Aller plus loin », vous trouverez un tableau de Rappel qui récapitule les concepts importants vus dans chaque chapitre.

Lorsque vous avez fini d'utiliser Visual Studio, enregistrez tous les projets ouverts et fermez l'environnement de développement.

Quitter Visual Studio

1. Enregistrez toutes les modifications que vous avez apportées à votre programme en cliquant sur le bouton Enregistrer tout de la barre d'outils Standard.

   Comme vous l'avez appris dans la section précédente, le comportement par défaut de Visual Studio 2008 consiste à donner un nom à votre programme lorsque vous commencez un projet ou une solution, sans spécifier d'emplacement pour le fichier, et de ne sauvegarder le projet qu'une fois que vous cliquez sur le bouton Enregistrer tout ou la commande Enregistrer tout du menu Fichier. Vous avez modifié votre projet, vous devez donc enregistrer vos changements.

2. Dans le menu Fichier, cliquez sur la commande Quitter.

   Le programme Visual Studio se ferme. Il est temps de passer à votre premier programme avec la lecture du chapitre 2 !

Rappel du chapitre 1

Pour	Faites ceci
Démarrer Visual Studio	Cliquez sur Démarrer dans la barre des tâches, cliquez sur Tous les programmes, sur le dossier Microsoft Visual Studio 2008, puis cliquez sur l'icône du programme Microsoft Visual Studio 2008.
Ouvrir un projet existant	Démarrez Visual Studio. Dans le menu Fichier, cliquez sur Ouvrir un projet *ou* Dans la page de démarrage, cliquez sur Projet sous la section Projets récents.
Compiler et exécuter un programme	Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. *ou* Appuyez sur F5.
Définir des propriétés	Cliquez sur l'objet de formulaire dont voulez définir les propriétés. Dans la fenêtre Propriétés, cliquez sur le nom de la propriété dans la colonne de gauche, puis modifiez le paramètre de propriété dans la colonne de droite correspondante.
Redimensionner une fenêtre d'outils	Affichez l'outil sous forme de fenêtre flottante (s'il est ancré) et redimensionnez-le en faisant glisser ses bords.
Déplacer une fenêtre d'outils	Affichez l'outil sous forme de fenêtre flottante (s'il est ancré) et faites glisser sa barre de titre.
Ancrer une fenêtre d'outils	À l'aide du pointeur de la souris, faites glisser la barre de titre de la fenêtre au-dessus d'un guide d'ancrage pour prévisualiser le résultat, puis relâchez le bouton de la souris pour mettre l'outil en place.
Masquer automatiquement une fenêtre d'outils	Cliquez sur la punaise Masquer automatiquement située à droite de la barre de titre de la fenêtre d'outils. La fenêtre est masquée derrière un petit onglet au bord de l'environnement de développement jusqu'à ce que la souris passe au-dessus.
Désactiver le masquage automatique d'une fenêtre d'outils	Cliquez sur l'onglet de l'outil, puis sur la punaise Masquer automatiquement.
Basculer entre fichiers ouverts	Maintenez enfoncée la touche CTRL et appuyez sur TAB pour afficher le Navigateur EDI. La touche CTRL restant enfoncée, appuyez sur TAB pour parcourir l'ensemble des fichiers ouverts. Servez-vous des touches fléchées pour faire défiler la liste des fichiers et outils ouverts. Vous pouvez également cliquer sur un fichier ou un outil dans le Navigateur EDI pour basculer vers celui-ci.
Basculer entre outils ouverts.	Appuyez sur ALT+F7 pour naviguer en avant dans la liste des outils ouverts dans l'EDI. Appuyez sur ALT+MAJ+F7 pour parcourir la liste en sens inverse.

Pour	Faites ceci
Obtenir de l'aide	Démarrez le système d'aide (hébergé par l'Explorateur de documents de Microsoft) en cliquant une commande du menu Communauté ou Aide.
Personnaliser l'aide	Dans l'Explorateur de documents, cliquez sur la commande Options du menu Outils.
Configurer l'environnement de Visual Studio pour le développement en Visual Basic	Cliquez sur la commande Paramètres d'importation et d'exportation du menu Outils, cliquez sur Réinitialiser tous les paramètres puis sur le bouton Suivant. Cliquez sur Oui, enregistrer mes paramètres actuels, puis sur le bouton Suivant. Enfin, cliquez sur Paramètres de développement Visual Basic et sur le bouton Terminer, puis sur Fermer.
Personnaliser les paramètres de l'EDI	Dans le menu Outils, cliquez sur la commande Options, puis personnalisez les paramètres de Visual Studio par catégorie. Pour visualiser et personnaliser les paramètres du projet, cliquez sur l'élément Général dans la catégorie Projets et solutions. Pour visualiser et personnaliser les paramètres du compilateur, cliquez sur l'élément Valeurs par défaut VB dans la catégorie Projets et Solutions.
Quitter Visual Studio	Dans le menu Fichier, cliquez sur Quitter.

Chapitre 2

Écrire son premier programme

À la fin de ce chapitre, vous saurez :

- Créer l'interface utilisateur d'un nouveau programme
- Définir les propriétés de chaque objet de votre interface utilisateur
- Écrire du code
- Enregistrer et exécuter le programme
- Créer un fichier exécutable

Comme vous l'avez appris au chapitre 1 « Explorer l'environnement de développement intégré de Visual Studio », l'environnement de développement intégré (EDI) de Microsoft Visual Studio 2008 comporte plusieurs outils performants qui permettent d'exécuter et de gérer vos programmes. Visual Studio comporte également tout ce dont vous avez besoin pour créer de A à Z vos propres applications pour Windows et le web.

Dans ce chapitre, vous allez apprendre à créer une interface utilisateur simple, mais agréable, à l'aide des contrôles de la Boîte à Outils Visual Studio. Ensuite, vous découvrirez comment personnaliser le fonctionnement de ces contrôles à l'aide des paramètres de propriétés. Puis vous verrez comment définir les tâches de votre programme en écrivant du code. Enfin, vous apprendrez à enregistrer votre nouveau programme (une machine à sous de Las Vegas), à l'exécuter et à le compiler dans un fichier exécutable.

Bandit Manchot : votre premier programme Visual Basic

L'application Windows que vous allez créer s'appelle Bandit Manchot. Ce programme de jeu simule une machine à sous. Bandit Manchot possède une interface utilisateur simple, et peut être créé et compilé en quelques minutes avec Visual Basic. Voici à quoi ressemblera votre programme une fois terminé :

Étapes de programmation

L'interface utilisateur de Bandit Manchot comporte deux boutons, trois cases pour les numéros de la chance, une photo numérique indiquant les gains, et l'étiquette « Bandit Manchot ». J'ai élaboré ces éléments en créant sept objets dans le formulaire Bandit Manchot, puis en modifiant plusieurs propriétés de chaque objet. Après avoir conçu cette interface, j'ai ajouté du code de programme pour que les boutons Lancer et Arrêter répondent aux clics de l'utilisateur et affichent les nombres aléatoires. Pour recréer ce programme par vous-même, vous allez suivre trois étapes essentielles de programmation en Visual Basic : création de l'interface utilisateur, définition des propriétés et écriture du code.

Le tableau suivant résume ce processus.

Étape de programmation	Nombre d'éléments
1. Créer l'interface utilisateur.	7 objets
2. Définir les propriétés.	13 propriétés
3. Écrire le code.	2 objets

Créer l'interface utilisateur

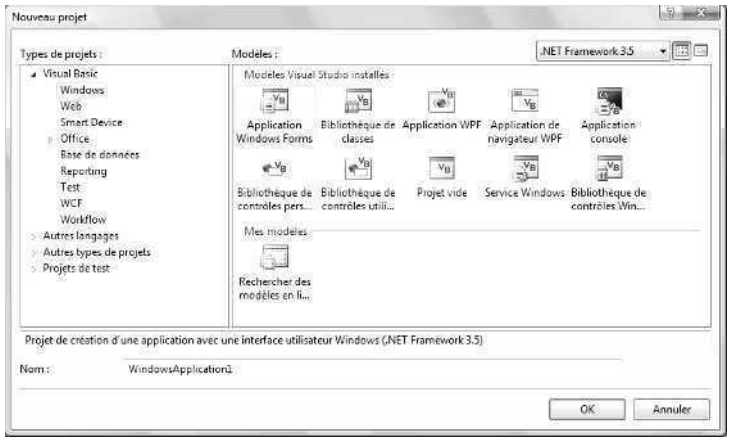
Dans cet exercice, vous allez commencer à élaborer Bandit Manchot en créant tout d'abord un nouveau projet, puis en utilisant les contrôles de la Boîte à outils pour construire l'interface utilisateur.

Créer un nouveau projet

1. Démarrez Visual Studio 2008.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

Astuce Vous pouvez aussi créer un nouveau projet de programmation à partir de la page de démarrage, en cliquant sur le lien bleu Projet situé à droite de Créer, en bas du volet Projets récents.

La boîte de dialogue Nouveau projet s'affiche.



La boîte de dialogue Nouveau projet permet d'accéder aux principaux types de projet permettant d'écrire des applications Windows. Si pendant l'installation, vous avez indiqué que vous étiez un programmeur Visual Basic, Visual Basic est votre principale option de développement (c'est le cas ici), mais les autres langages de Visual Studio (Visual C# et C++) sont toujours disponibles dans cette boîte de dialogue. Bien que, dans cet exercice, vous sélectionniez un projet d'application Windows de base, la boîte de dialogue constitue également une passerelle vers d'autres types de projets de développement, comme les applications web, les applications de console, les applications Smart Device (Framework compact .NET) ou les projets de déploiement de Visual Studio.

Dans le coin supérieur droit de la boîte de dialogue Nouveau projet, remarquez la zone de liste déroulante. C'est un nouveau dispositif de Visual Studio 2008 qui porte le nom de « ciblage multiple ». Cette zone de liste déroulante permet de spécifier la version de .NET Framework ciblée par votre application. Par exemple, en sélectionnant .NET Framework 3.5, les ordinateurs sur lesquels s'exécutera l'application devront posséder le .NET Framework 3.5 installé. Visual Studio n'affiche que les options compatibles avec la version de .NET Framework sélectionnée. Les applications créées avec Visual Basic 2005 ciblaient toutes le .NET Framework 2.0. Des programmes créés avec Visual Basic 2005 mis à niveau vers Visual Basic 2008 cibleront toujours le .NET Framework 2.0. Sauf en cas de nécessité précise, laissez dans cette zone de liste la valeur par défaut de .NET Framework 3.5. Vous en apprendrez plus sur le .NET Framework au Chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework ».

3. Dans la zone Modèles de la boîte de dialogue, cliquez sur l'icône Application Windows Forms, si elle n'est pas déjà sélectionnée.

   Visual Studio prépare l'environnement de développement pour la programmation d'une application Windows en Visual Basic.

4. Dans la zone de texte Nom, tapez **Mon Bandit Manchot**.

   Visual Studio intitule votre projet Mon Bandit Manchot. Vous spécifierez ultérieurement un emplacement de dossier. Le préfixe « Mon » permet d'éviter toute confusion entre cette nouvelle application et le projet Bandit Manchot que j'ai créé pour vous.

Astuce Si votre boîte de dialogue Nouveau projet comporte les zones de texte Emplacement et Solution, vous devez spécifier dès maintenant un emplacement de dossier et un nom de solution pour votre nouveau projet de programmation. La présence de ces zones de texte est contrôlée par une case à cocher située dans la boîte de dialogue Options du menu Outils, mais ce n'est pas le paramètre par défaut de Visual Basic 2008. Tout au long de cet ouvrage, vous apprendrez à enregistrer vos projets (ou à les supprimer) *après* avoir terminé l'exercice de programmation. Pour davantage de renseignements à propos de cette fonction d'« enregistrement retardé » et de la restauration des paramètres par défaut de Visual Studio, voir la section « Personnaliser les réglages de l'environnement de développement intégré pour réaliser les exercices pas à pas » dans le chapitre 1.

5. Cliquez sur OK pour créer le nouveau projet Visual Studio.

   Visual Studio nettoie l'espace de travail du nouveau projet de programmation et affiche le formulaire Windows vierge que vous allez utiliser pour créer votre interface utilisateur.

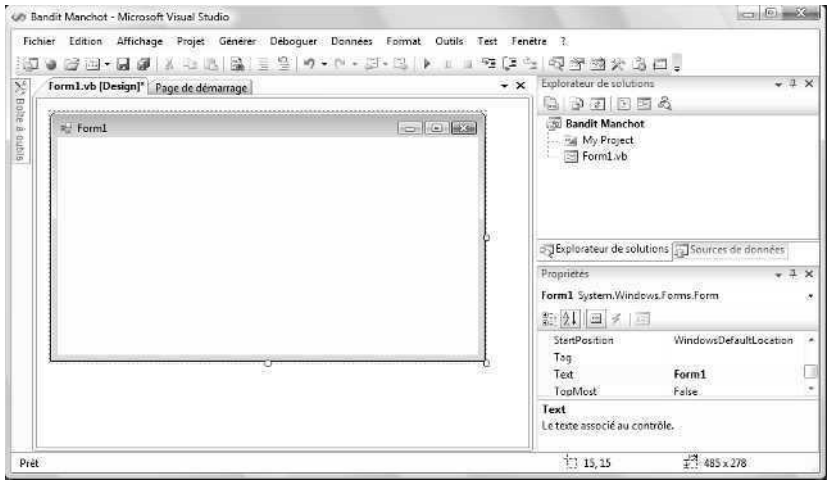
Vous allez maintenant agrandir le formulaire et créer les deux boutons de l'interface.

Créer l'interface utilisateur

1. Placez le pointeur de la souris au-dessus de l'angle inférieur droit du formulaire jusqu'à ce qu'il prenne la forme d'une poignée de dimensionnement, puis faites glisser pour agrandir le formulaire et libérer de l'espace pour les objets de votre programme.

   Quand vous agrandissez le formulaire, des barres de défilement peuvent apparaître dans le Concepteur pour vous permettre d'accéder à l'ensemble du formulaire que vous créez. Selon votre résolution d'écran et les outils de Visual Studio que vous avez ouverts, il se peut que vous ne puissiez pas voir l'ensemble du formulaire à l'écran. Ne vous en faites pas, que votre formulaire soit petit ou qu'il remplisse tout l'écran, les barres de défilement vous permettent d'accéder à l'intégralité du formulaire.

Dimensionnez votre formulaire comme sur l'illustration. Si vous souhaitez reproduire l'exemple à l'identique, utilisez les dimensions qui apparaissent dans l'angle inférieur droit de l'écran (485 pixels x 278 pixels).



Pour voir l'intégralité du formulaire, vous pouvez redimensionner ou fermer les autres outils de programmation, comme vous l'avez appris au chapitre 1. Reprenez-le si vous avez encore des questions sur le redimensionnement des fenêtres ou des outils.

Vous allez maintenant ajouter un objet bouton au formulaire.

2. Cliquez sur l'onglet Boîte à outils pour faire apparaître la fenêtre Boîte à outils dans l'EDI.

Celle-ci contient tous les contrôles nécessaires à l'élaboration des programmes Visual Basic de cet ouvrage. Vous avez sélectionné le type de projet Application Windows Forms : les contrôles nécessaires pour créer une application Windows apparaissent. Ils sont classés par type, et la catégorie Contrôles communs est affichée par défaut. Si la Boîte à outils n'apparaît pas, cliquez sur Boîte à outils dans le menu Affichage.

3. Dans la Boîte à outils, double-cliquez sur le contrôle *Button*, puis déplacez le pointeur de la souris à l'extérieur de la Boîte à outils.

   Visual Studio crée un objet bouton de taille standard sur le formulaire et masque la Boîte à outils, comme suit :

Le bouton est intitulé *Button1* car c'est le premier bouton du programme. Souvenez-vous de ce nom, il réapparaîtra lorsque vous écrirez le code. Le nouvel objet bouton est sélectionné et entouré de poignées de redimensionnement. Lorsque Visual Basic est en *mode conception* (c'est-à-dire lorsque l'EDI est actif), vous pouvez déplacer des objets sur le formulaire en les faisant glisser à l'aide de la souris, et les redimensionner à l'aide des poignées de redimensionnement. En revanche, il est impossible de déplacer des éléments de l'interface pendant l'exécution d'un programme, à moins que vous n'ayez modifié une propriété dans le programme pour permettre cette opération. Vous allez maintenant déplacer et redimensionner le bouton.

Déplacer et redimensionner un bouton

1. Placez le pointeur de la souris au-dessus du bouton pour qu'il se transforme en flèche à quatre pointes, puis faites glisser le bouton vers le bas et vers la droite.

   Le bouton se déplace au-dessus du formulaire. Si vous déplacez l'objet près du bord du formulaire ou près d'un autre objet (si d'autres objets sont affichés), il s'aligne automatiquement sur une grille masquée. Une petite « marque » bleue apparaît également pour vous aider à évaluer la distance entre cet objet et le bord du formulaire ou l'autre objet. Contrairement aux versions antérieures de Visual Studio, la grille n'est pas affichée par défaut sur le formulaire, mais vous pouvez utiliser la marque pour évaluer les distances.

Astuce Si vous souhaitez afficher la grille du mode conception comme dans Microsoft Visual Studio .NET 2003 et Visual Basic 6, dans le menu Outils, cliquez sur la commande Options, puis sur Concepteur Windows Form et Général. Réglez *ShowGrid* sur *SnapToGrid*. Pour que les modifications prennent effet, vous devrez fermer le projet et le réouvrir.

2. Placez le pointeur de la souris dans l'angle inférieur droit du bouton.

   Lorsque le pointeur de la souris s'attarde sur l'une des poignées de redimensionnement d'un objet sélectionné, il se transforme en pointeur de redimensionnement. Vous pouvez l'utiliser pour modifier la taille d'un objet.

3. Agrandissez le bouton en faisant glisser le pointeur vers le bas et vers la droite.

   Lorsque vous relâchez le bouton de la souris, le bouton change de taille et s'ajuste à la grille.

4. Utilisez la poignée de redimensionnement pour redonner au bouton sa taille d'origine.

Vous allez maintenant ajouter un second bouton au formulaire, en dessous du premier.

Ajouter un deuxième bouton

1. Cliquez sur l'onglet Boîte à outils pour afficher celle-ci.
2. Cliquez (une seule fois) sur le contrôle *Button* dans la Boîte à outils, puis déplacez le pointeur de la souris au-dessus du formulaire.

   Il se change en pointeur en croix et en icône bouton. Le pointeur en croix est conçu pour vous permettre de dessiner la forme rectangulaire du bouton, et vous pouvez utiliser cette méthode au lieu de double-cliquer pour créer un contrôle de taille standard.
3. Faites glisser le pointeur vers le bas et vers la droite. Relâchez le bouton de la souris pour terminer le bouton : vous le voyez s'ajuster sur le formulaire.
4. Redimensionnez l'objet bouton pour que sa taille soit la même que celle du premier bouton, puis placez-le en dessous du premier bouton. Utilisez la marque pour vous aider.

Astuce Vous pouvez à tout moment supprimer un objet et recommencer, en sélectionnant l'objet sur le formulaire puis en appuyant sur SUPPR. N'hésitez pas à créer et à supprimer des objets pour vous entraîner à créer votre interface utilisateur.

Vous allez maintenant ajouter les étiquettes de numéros du programme. Une étiquette (*label*) est un élément particulier de l'interface utilisateur, conçu pour afficher du texte, des nombres ou des symboles lors de l'exécution d'un programme. Lorsque l'utilisateur clique sur le bouton Lancer du programme Bandit Manchot, trois numéros aléatoires apparaissent dans les cases étiquette. Si l'un des trois numéros est un 7, l'utilisateur gagne.

Ajouter les étiquettes de numéro

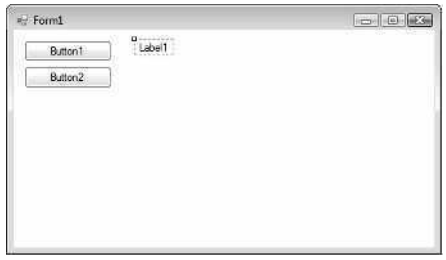
1. Double-cliquez sur le contrôle *Label* dans la Boîte à outils.

   Visual Studio crée un objet étiquette sur le formulaire. Si vous êtes habitué aux anciennes versions de Visual Studio ou Visual Basic, vous remarquerez que l'objet

étiquette par défaut est plus petit. Il est juste assez large pour contenir le texte de l'objet, mais il peut être redimensionné.

2. Faites glisser l'objet *Label1* à droite des deux objets bouton.

   Votre formulaire présente un résultat similaire à



3. Dans la Boîte à outils, double-cliquez sur le contrôle *Label* pour créer un deuxième objet étiquette. Cet objet sera intitulé *Label2* dans le programme.
4. Double-cliquez une nouvelle fois sur le contrôle *Label* pour créer un troisième objet étiquette.
5. Sur le formulaire, placez les deuxième et troisième objets étiquette à droite du premier.

   Laissez beaucoup d'espace entre les trois étiquettes, car vous allez les utiliser pour afficher de grands numéros lors de l'exécution du programme.

   Vous allez maintenant utiliser le contrôle *Label* pour ajouter une étiquette descriptive à votre formulaire. Ce sera la quatrième et dernière étiquette de votre formulaire.
6. Dans la Boîte à outils, double-cliquez sur le contrôle *Label*.
7. Faites glisser l'objet *Label4* sous les deux boutons de commande.

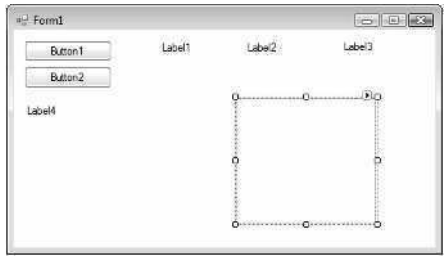
   Lorsque vous avez terminé, vos quatre étiquettes doivent ressembler à celles de l'illustration suivante. Vous pouvez déplacer vos objets étiquette s'ils ne sont pas au bon endroit.

Vous allez maintenant ajouter une zone d'image au formulaire, pour afficher le résultat obtenu lorsque vous tirez un 7 et touchez le jackpot. Une zone d'image est conçue pour afficher dans un programme des images en mode point, des icônes, des photos numériques et d'autres illustrations. L'une des meilleures utilisations d'une zone d'image consiste à afficher un fichier image JPEG.

Ajouter une image

1. Dans la Boîte à outils, cliquez sur le contrôle *PictureBox*.
2. À l'aide du pointeur de dessin, créez une grande zone rectangulaire sous les deuxième et troisième étiquettes.

   Laissez un peu d'espace sous les étiquettes, dont la taille va augmenter comme je l'ai mentionné précédemment. Lorsque vous avez terminé, votre objet zone d'image ressemble à :



   Cet objet sera intitulé *PictureBox1* dans votre programme. Vous utiliserez ce nom plus tard dans le code.

Vous êtes maintenant prêt à personnaliser votre interface en définissant quelques propriétés.

Définir les propriétés

Comme vous l'avez appris au chapitre 1, vous pouvez modifier les propriétés en sélectionnant des objets sur le formulaire et en modifiant leurs paramètres dans la fenêtre Propriétés. Vous allez commencer par modifier les paramètres de propriétés des deux boutons.

Définir les propriétés des boutons

1. Cliquez sur le premier bouton (*Button1*).

   Le bouton sélectionné est entouré de poignées de dimensionnement.

2. Cliquez sur la barre de titre de la fenêtre Propriétés.

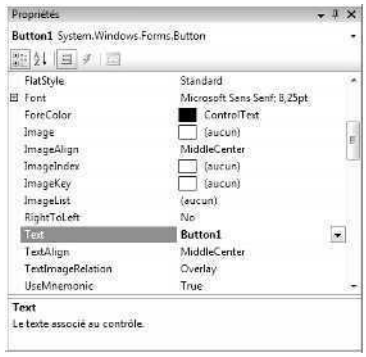
> **Astuce** Si la fenêtre Propriétés n'apparaît pas, cliquez sur la commande Fenêtre Propriétés du menu Affichage ou appuyez sur F4.

3. En haut de la fenêtre Propriétés, cliquez sur le bouton Catégories.

   Pour plus d'informations sur les propriétés catégorisées, reportez-vous à la section « Fenêtre Propriétés » du Chapitre 1.

4. Si nécessaire, redimensionnez la fenêtre Propriétés de manière à voir le nom des propriétés et leurs paramètres actuels.

   Quand vous serez habitué à définir des propriétés, vous utiliserez la fenêtre Propriétés sans l'agrandir, mais au début, cela peut être utile. Sur l'illustration suivante, la taille de la fenêtre Propriétés est convenable :



   La fenêtre Propriétés affiche la liste des paramètres du premier bouton. Il s'agit entre autres de la couleur du fond, du texte, de la taille de police et de la largeur du bouton. Vu le nombre important de propriétés, Visual Studio les classe en catégories et les affiche en mode plan. Pour afficher les propriétés d'une catégorie, cliquez sur le signe plus (+) adjacent à son titre.

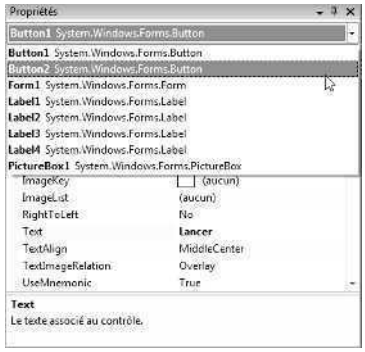
5. Déroulez la fenêtre Propriétés jusqu'à la propriété *Text*, située dans la catégorie Apparence.

6. Double-cliquez sur la propriété *Text* dans la colonne de gauche de la fenêtre Propriétés. Le paramètre actuel de *Text* ("Button1") apparaît en surbrillance.

7. Tapez **Lancer** et appuyez sur Entrée.

   La propriété *Text* devient « Lancer » dans la fenêtre Propriétés et sur le bouton. Vous allez maintenant modifier la propriété *Text* du second bouton en « Arrêter ». Cette fois, vous allez procéder différemment.

8. Ouvrez la liste d'objets au sommet de la fenêtre Propriétés. Une liste d'objets d'interface apparaît comme suit :



9. Cliquez sur Button2 System.Windows.Forms.Button (le deuxième bouton) dans la zone de liste.

   Les paramètres de propriétés du deuxième bouton apparaissent dans la fenêtre Propriétés et Visual Studio met *Button2* en surbrillance.

10. Double-cliquez sur la propriété *Text* actuelle (« Button2 »), tapez **Arrêter** et appuyez sur Entrée. Le texte du deuxième bouton se change en « Arrêter ».

> **Astuce** L'utilisation de la liste d'objets permet de passer facilement d'un objet à l'autre. Vous pouvez également le faire sur le formulaire, en cliquant sur chaque objet.

Vous allez maintenant définir les propriétés des étiquettes dans le programme. Les trois premières étiquettes comporteront les nombres aléatoires générés par le programme ; leurs paramètres de propriétés sont identiques. Vous définirez la plupart d'entre eux simultanément. Les paramètres de l'étiquette descriptive seront traités à part.

Définir les propriétés des étiquettes de numéro

1. Cliquez sur la première étiquette de numéro (*Label1*), appuyez sur la touche MAJ et maintenez-la enfoncée, cliquez sur les deuxième et troisième étiquettes de numéro, puis relâchez la touche MAJ. Si la fenêtre Propriétés fait obstacle, déplacez-la.

   Un rectangle de sélection et des poignées de dimensionnement apparaissent autour des étiquettes sélectionnées. Vous allez modifier les propriétés *TextAlign*, *BorderStyle* et *Font* de manière à ce que les numéros qui doivent apparaître dans les étiquettes soient centrés, encadrés, de police et de taille identiques. Toutes ces propriétés sont situées dans la catégorie Apparence de la fenêtre Propriétés. Attribuez la valeur *False* à la propriété *AutoSize* pour pouvoir modifier la taille des étiquettes selon des spécifications précises. La propriété *AutoSize* est située dans la catégorie Disposition.

> **Astuce** Lorsque plusieurs objets sont sélectionnés, seules les propriétés modifiables du groupe apparaissent dans la fenêtre Propriétés.

2. Dans la fenêtre Propriétés, cliquez sur la propriété *AutoSize*, puis sur la flèche qui apparaît à droite.
3. Pour pouvoir dimensionner manuellement les étiquettes, attribuez la valeur False à la propriété *AutoSize*.
4. Cliquez sur la propriété *TextAlign*, puis sur la flèche qui apparaît à droite.

   Une série d'options d'alignement illustrées apparaît dans la zone de liste. Ces paramètres vous permettent d'aligner du texte n'importe où dans les limites de l'objet étiquette.
5. Cliquez sur l'option de centrage (MiddleCenter).

   La propriété *TextAlign* de chacune des étiquettes sélectionnées devient Middle Center.
6. Cliquez sur la propriété *BorderStyle*, puis sur la flèche qui apparaît à droite.

   Les paramètres de propriété disponibles (None, FixedSingle, et Fixed3D) apparaissent dans la zone de liste.
7. Dans la zone de liste, cliquez sur FixedSingle pour ajouter une fine bordure à chaque étiquette.
8. Cliquez sur la propriété *Font*, puis sur le bouton équipé de points de suspension (situé à côté du paramètre de police actuel).

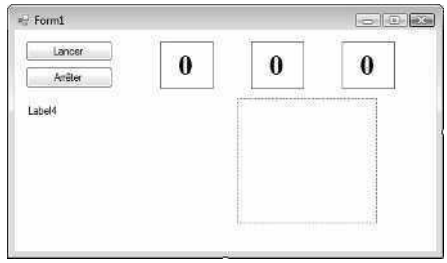
   La boîte de dialogue Police apparaît.
9. Sélectionnez une police Times New Roman, le style Gras et un corps de 24, puis cliquez sur OK.

   Le texte de l'étiquette apparaît avec les caractéristiques que vous lui avez données.

Vous allez maintenant donner la valeur 0 au texte des trois étiquettes (un bon « paramètre fictif » pour les nombres qui rempliront ces cases dans votre jeu). Le programme générant les vrais numéros, vous pourriez également effacer le texte, mais le fait d'utiliser un paramètre fictif vous donne une idée de la taille des étiquettes.

10. Cliquez sur une zone vierge du formulaire pour désélectionner les trois étiquettes, puis cliquez sur la première.
11. Double-cliquez sur la propriété *Text*, tapez **0** puis appuyez sur Entrée.

    Le texte de l'objet *Label1* devient 0. Plus loin dans ce chapitre, vous définirez cette propriété comme un numéro aléatoire de machine à sous à l'aide du code de programme.
12. Procédez de même pour le texte des deuxième et troisième étiquettes.
13. Déplacez et redimensionnez les étiquettes pour qu'elles soient suffisamment espacées. Votre formulaire présente un résultat similaire à



Vous allez maintenant modifier les propriétés *Text* et *Forecolor* de la quatrième étiquette.

Définir les propriétés de l'étiquette descriptive

1. Cliquez sur le quatrième objet étiquette (*Label4*) du formulaire.
2. Modifiez la propriété *Text* de la fenêtre Propriétés en **Bandit Manchot**.
3. Cliquez sur la propriété *Font*, puis sur les trois points de suspension.
4. Dans la boîte de dialogue, fixez la police à Arial, le style à Gras et le corps à 18. Ensuite, cliquez sur OK.

   La police de l'objet *Label4* est mise à jour et l'étiquette est redimensionnée automatiquement pour s'adapter à la police, car la propriété *AutoSize* est réglée sur True.

5. Cliquez sur la propriété *ForeColor* dans la fenêtre Propriétés, puis sur la flèche de la deuxième colonne.

   Visual Studio affiche une zone de liste comportant les onglets Personnaliser, Web et System pour le paramétrage des couleurs de premier plan (la couleur du texte) de l'objet étiquette. L'onglet Personnaliser propose un grand nombre de couleurs disponibles dans votre système. L'onglet Web présente les couleurs des pages web et permet de choisir les couleurs à l'aide de leur nom. L'onglet System affiche les couleurs utilisées actuellement pour les éléments de l'interface utilisateur dans votre système.

6. Cliquez sur la couleur pourpre dans l'onglet Personnaliser.

   Le texte de la zone d'étiquette devient pourpre.

Désormais, vous êtes prêt à définir les propriétés du dernier objet.

Lire des propriétés dans des tableaux

Dans ce chapitre, vous avez défini les propriétés du programme Bandit Manchot étape par étape. Dans les chapitres suivants, ces instructions seront présentées sous forme de tableau sauf si une propriété pose un problème particulier. Dans ce tableau se trouvent les propriétés du programme Bandit Manchot que vous avez définies jusqu'à présent. Les paramètres que vous devez saisir figurent entre guillemets. Vous ne devez pas taper les guillemets.

Objet	Propriété	Paramètre
Button1	*Text*	« Lancer »
Button2	*Text*	« Arrêter »
Label1, Label2, Label3	*AutoSize*	False
	BorderStyle	Fixed Single
	Font	Times New Roman, Gras, 24 points
	Text	« 0 »
	TextAlign	MiddleCenter
Label4	*Text*	« Bandit Manchot »
	Font	Arial, Gras, 18 points
	ForeColor	Pourpre
PictureBox1	*Image*	« c:\vb08epe\chap02\Jackpot.jpg »
	SizeMode	StretchImage
	Visible	False

Les propriétés de la zone d'image

Lorsque le joueur remporte le jackpot (c'est-à-dire quand au moins un 7 apparaît dans les étiquettes de numéro), l'objet zone d'image affiche la représentation d'une personne distribuant de l'argent. Cette image a été numérisée à partir d'un manuscrit allemand du quatorzième siècle non publié, et enregistrée au format JPEG. Vous devez définir la propriété *SizeMode* pour dimensionner correctement l'image et définir la propriété *Image* pour spécifier le nom du fichier JPEG que vous voulez afficher dans la zone d'image. Il vous faudra également définir la propriété *Visible*, qui précise l'état de l'image au début du programme.

Définir les propriétés de la zone d'image

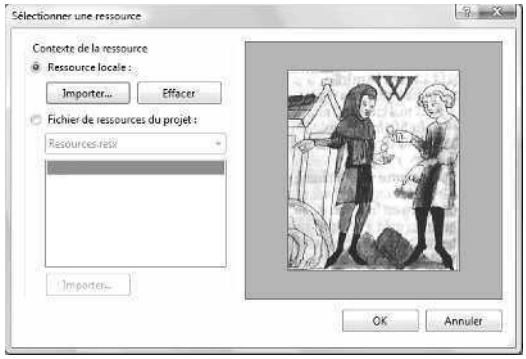
1. Cliquez sur l'objet zone d'image du formulaire.
2. Dans la fenêtre Propriétés, cliquez sur la propriété *SizeMode* (dans la catégorie Comportement), cliquez sur la flèche adjacente, puis sur *StretchImage*.

   Régler la propriété *SizeMode* sur *StretchImage* avant d'ouvrir une image oblige Visual Studio à redimensionner l'image aux dimensions de la zone d'image. En général, cette propriété est définie avant la propriété *Image*.
3. Cliquez sur la propriété *Image* dans la fenêtre Propriétés, puis sur les points de suspension dans la deuxième colonne.

   La boîte de dialogue Sélectionner une ressource apparaît.
4. Cliquez sur l'option Ressource locale, puis sur le bouton Importer.
5. Dans la boîte de dialogue Ouvrir, recherchez le dossier c:\vb08epe\chap02.

   Ce dossier contient la photo numérique Jackpot.jpg.
6. Sélectionnez le fichier Jackpot.jpg, puis cliquez sur Ouvrir.

   Une illustration médiévale représentant une personne distribuant de l'argent apparaît dans la boîte de dialogue Sélectionner une ressource. La lettre W signifie *winning* (gagnant).



7. Cliquez sur OK.

La photo est affichée dans la zone d'image. Celle-ci étant relativement petite (24 ko), elle s'ouvre rapidement.

8. Redimensionnez l'objet zone d'image pour éviter toute déformation de l'image.

Mon objet zone d'image mesure 148 pixels de large pour 138 pixels de haut. Vous pouvez lui conférer la même taille à l'aide des dimensions situées en bas à droite de l'EDI de Visual Studio. Les dimensions de l'objet sélectionné figurent en bas à droite, et la position de l'angle supérieur gauche de l'objet sur le formulaire figure à gauche des dimensions.

Cette image s'affiche de manière optimale lorsque l'objet zone d'image prend une forme presque carrée.

Remarque Examinez l'objet zone d'image : vous remarquerez une petite flèche de raccourci près de l'angle supérieur droit. Cette flèche est un bouton sur lequel vous pouvez cliquer pour modifier rapidement quelques paramètres standard de zone d'image et ouvrir la boîte de dialogue Sélectionner une ressource. Vous retrouverez cette fonction dans le chapitre 4, « Travailler avec les menus, les barres d'outils et les boîtes de dialogue », pour l'utilisation du contrôle *Toolstrip*.

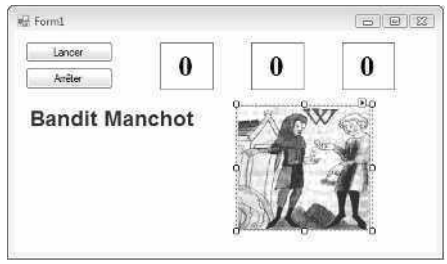
Vous allez désormais régler la propriété *Visible* sur False pour masquer l'image au démarrage du programme.

9. Cliquez sur la propriété *Visible* dans la catégorie Comportement de la fenêtre Propriétés, puis sur la flèche adjacente.

Les paramètres de la propriété *Visible* apparaissent sous forme de liste.

10. Cliquez sur False pour rendre l'image invisible au démarrage du programme.

Régler la propriété *Visible* sur False modifie la zone d'image pendant l'exécution du programme, mais pas en mode conception. Voici à quoi ressemble votre formulaire final :



Astuce Vous pouvez également double-cliquer sur les noms des propriétés qui possèdent des paramètres *True* et *False* (appelées propriétés *booléennes*) pour passer de *True* à *False* et inversement. Les propriétés booléennes par défaut apparaissent en texte normal, les paramètres modifiés en gras.

11. Vous avez achevé la définition des propriétés pour l'instant. Si votre fenêtre Propriétés est flottante, double-cliquez sur sa barre de titre pour qu'elle retrouve sa position ancrée.

Écrire le code

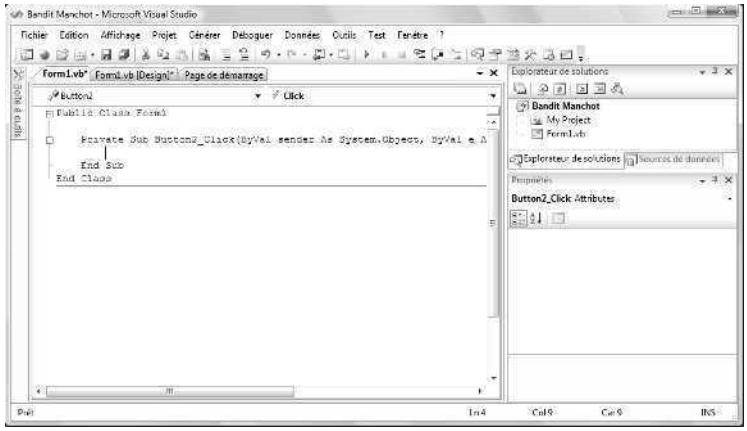
Vous voici prêt à écrire le code du programme Bandit Manchot. La plupart des objets que vous avez créés « connaissent » déjà leur fonction pendant l'exécution du programme, ils sont prêts à recevoir des informations de la part de l'utilisateur et à les traiter. La fonction inhérente des objets est l'une des grandes forces de Visual Studio et Visual Basic : une fois les objets placés sur un formulaire et leurs propriétés définies, ils sont prêts à fonctionner sans aucune autre programmation. Toutefois, il manque toujours au programme de jeu Bandit Manchot sa « matière » (le code, qui calcule les nombres aléatoires, les affiche dans les cases et détecte le jackpot). Cette logique informatique peut être introduite dans l'application uniquement à l'aide d'instructions (le code explique clairement ce que le programme doit faire à chaque étape). Les boutons Lancer et Arrêter dirigent le programme : vous allez leur associer le code du jeu. La saisie et la modification des instructions du programme Visual Basic a lieu dans l'Éditeur de code.

Au cours des étapes suivantes, vous allez saisir le code de programme pour Bandit Manchot dans l'Éditeur de code.

Utiliser l'Éditeur de code

1. Dans le formulaire, double-cliquez sur le bouton Arrêter.

   L'Éditeur de code apparaît sous forme de document avec onglet au milieu de l'EDI de Visual Studio, comme suit :



   Dans l'Éditeur de code, vous trouverez des instructions de code associées au formulaire courant. En général, les instructions utilisées conjointement pour effectuer une action sont regroupées dans un concept de programmation appelé *procédure*. Les sous-procédures, parfois appelées *sous-programmes*, sont un type de procédure. Les sous-procédures comprennent un mot clé *Sub* dans la première ligne et se terminent par *End Sub*. Les procédures sont exécutées automatiquement lorsque des événements se produisent, comme un clic sur un bouton. Lorsqu'une procédure est associée à un objet particulier et à un événement, elle est appelée *gestionnaire d'événements* ou *procédure événementielle*.

   Lorsque vous avez double-cliqué sur le bouton Arrêter (*Button2*), Visual Studio a automatiquement ajouté la première et la dernière ligne de la procédure d'événement *Button2_Click,* comme le montre le code suivant. La première ligne a été découpée pour tenir dans les marges de l'ouvrage. Vous pourriez remarquer la présence d'autres fragments de code, comme les mots Public et Class. Visual Studio les a ajouté pour définir des caractéristiques importantes du formulaire. Je les passerais sous silence pour le moment.

```
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
End Sub
```

   Le corps d'une procédure tient entre ces lignes et il s'exécute dès qu'un utilisateur active l'élément de l'interface associé à la procédure. Dans ce cas, l'événement est un clic de souris, mais comme vous le verrez plus loin dans cet ouvrage, il peut s'agir d'un autre type d'événement.

2. Tapez **End**, puis appuyez sur la touche ENTRÉE.

   Lors de la saisie de l'instruction, Visual Basic reconnaît *End* comme l'un des termes réservés, ou *mots clés*, du langage Visual Basic et l'affiche dans une zone de liste dotée d'onglets Commun et Tous. Cette zone de liste est nommée IntelliSense, car elle tente de vous aider intelligemment lors de l'écriture de code. Vous pouvez parcourir alphabétiquement différents objets et mots clés Visual Basic. Le langage est aussi partiellement explorable depuis l'EDI lui-même.

   Après avoir appuyé sur la touche ENTRÉE, le mot *End* s'affiche en bleu et s'indente, confirmant ainsi que Visual Basic l'a reconnu comme un parmi la centaine de mots clés du langage Visual Basic. Le mot clé *End* vous permet d'arrêter votre programme, lequel ne s'affiche plus à l'écran. Dans ce cas, *End* est également une *instruction* complète, une instruction autonome exécutée par le *compilateur Visual Basic*, la partie de Visual Studio qui traite, ou *analyse*, chaque ligne de *code source* Visual Basic, en combinant le résultat avec d'autres ressources pour créer un fichier exécutable. Les instructions ressemblent à des phrases complètes dans le langage humain. Leur longueur peut varier, mais elles doivent suivre les « règles » de syntaxe du compilateur. Dans Visual Studio, les instructions peuvent être composées de mots clés, de propriétés, de noms d'objets, de variables, de nombres, de symboles spéciaux, et d'autres valeurs. Vous en apprendrez davantage sur la construction des instructions dans le chapitre 5.

   Lorsque vous saisissez des instructions et effectuez d'autres modifications, l'Éditeur de code se charge de nombreux détails de mise en forme à votre place. Il ajuste le retrait et l'espacement, et ajoute les parenthèses nécessaires. L'orthographe, l'ordre et l'espacement exacts des éléments dans les instructions sont ce qui porte le nom de *syntaxe des instructions*.

   Lorsque vous avez appuyé sur la touche ENTRÉE, l'instruction *End* a été mise en retrait pour la séparer des instructions *Private Sub* et *End Sub*. Ce modèle de retrait est l'une des conventions de programmation que vous allez observer tout au long de cet ouvrage pour assurer la clarté et la lisibilité de vos programmes. L'ensemble des conventions concernant l'organisation du code dans un programme est ce que l'on appelle le *style de programmation*.

Maintenant que vous avez écrit le code du bouton Arrêter, vous allez écrire celui du bouton Lancer. Ces instructions un peu plus longues vous donneront l'occasion d'en apprendre davantage à propos de la syntaxe et du style de programmation. Vous étudierez de nombreuses instructions plus loin dans cet ouvrage. Inutile de tout savoir dès maintenant. Concentrez-vous sur la structure générale du code et veillez à reproduire fidèlement les instructions.

Écrire le code du bouton Lancer

1. Cliquez sur le bouton Concepteur de vues de la fenêtre Explorateur de solutions pour afficher de nouveau le formulaire.

Remarque Lorsque l'Éditeur de code est à l'écran, vous ne pouvez pas voir le formulaire sur lequel vous travaillez. Le bouton Concepteur de vues est un mécanisme qui permet de l'afficher à nouveau. Si plusieurs formulaires sont chargés dans l'Explorateur de solutions, cliquez sur celui que vous souhaitez afficher en premier. Vous pouvez également cliquer sur l'onglet Form1.vb [Design] au sommet de l'Éditeur de code. Si vous ne voyez pas les onglets en haut de l'Éditeur de code, activez l'affichage sous forme de documents avec onglet dans la boîte de dialogue Options, comme mentionné dans une astuce du chapitre 1.

2. Double-cliquez sur le bouton Lancer.

   L'Éditeur de code s'affiche, et une procédure d'événement associée au bouton *Button1* apparaît à côté de la procédure d'événement de *Button2*.

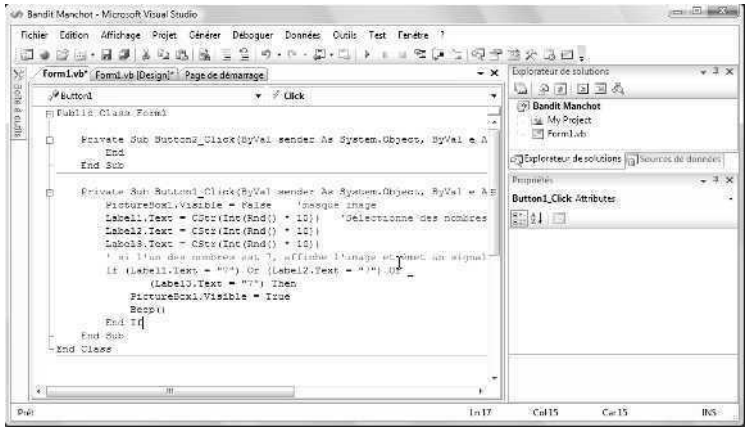
   Bien que vous ayez modifié le texte de ce bouton pour l'appeler « Lancer », son nom dans le programme reste *Button1*. Le nom et le texte d'un élément d'interface peuvent être différents, selon les besoins du programmeur. Chaque objet peut être associé à plusieurs procédures, une pour chaque événement qu'il reconnaît. Intéressez-vous désormais à l'événement clic : les utilisateurs vont cliquer sur les boutons Lancer et Arrêter lorsqu'ils exécuteront le programme.

3. Tapez les lignes de programme suivantes entre les instructions *Private Sub* et *End Sub*. Appuyez sur ENTRÉE à la fin de chaque ligne, appuyez sur TAB pour mettre en retrait, et veillez à reproduire fidèlement les instructions. L'Éditeur de code défilera vers la gauche quand vous saisirez les lignes plus longues. Si vous faites une erreur (habituellement signalée par un soulignement dentelé), supprimez les instructions erronées et réessayez.

Astuce Lorsque vous saisissez du code, Visual Basic le met en forme et affiche certaines parties du programme en couleur pour vous aider à identifier les différents éléments. Lorsque vous commencez à taper une propriété, Visual Basic affiche dans une zone de liste les propriétés disponibles pour l'objet que vous utilisez ; vous pouvez cliquer sur la propriété ou continuer à taper pour la saisir vous-même. Si Visual Basic affiche un message d'erreur, il se peut que vous ayez mal orthographié une instruction. Vérifiez que la ligne correspond au texte de cet ouvrage, faites les corrections nécessaires, et continuez la saisie. Vous pouvez également supprimer une ligne et la saisir à nouveau. En outre, Visual Basic peut ajouter du code automatiquement si nécessaire. Par exemple, lorsque vous tapez le code suivant, Visual Basic ajoute automatiquement la ligne *End If*. Les lecteurs des éditions précédentes ont été nombreux à trouver que ce premier exercice de saisie était la partie la plus difficile du chapitre : « Je suis sûr de l'avoir tapé *exactement* comme vous l'avez écrit ! » Accordez toute votre attention à ce code de programme. Je vous assure qu'il fonctionne !

```
PictureBox1.Visible = False    ' masque image
Label1.Text = CStr(Int(Rnd() * 10)) ' sélectionne des nombres aléatoires
Label2.Text = CStr(Int(Rnd() * 10))
Label3.Text = CStr(Int(Rnd() * 10))
' si l'un des nombres est 7, affiche l'image et émet un signal sonore
If (Label1.Text = "7") Or (Label2.Text = "7") _
Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
    Beep()
End If
```

Une fois que vous avez terminé, l'Éditeur de code se présente ainsi :



4. Cliquez sur la commande Enregistrer tout du menu Fichier pour enregistrer vos ajouts au programme.

La commande Enregistrer tout enregistre l'intégralité de votre projet : le fichier projet, le fichier formulaire, tous les modules de code et les autres composants relatifs à l'application. C'est la première fois que vous enregistrez le projet. La boîte de dialogue Enregistrer un projet vous demande le nom et l'emplacement du projet. Si vous avez configuré Visual Studio pour vous demander un emplacement au moment de la création du projet, vous ne verrez pas la boîte de dialogue Enregistrer un projet. Visual Studio enregistrera simplement vos modifications.

5. Cliquez sur le bouton Parcourir à droite de la zone de texte Emplacement et sélectionnez un emplacement pour vos fichiers.

Je vous recommande d'utiliser le dossier c:\vb08epe\chap02 (l'emplacement des fichiers d'exemples de l'ouvrage). Puisque vous avez utilisé le préfixe "Mon" quand vous avez ouvert votre projet pour la première fois, cette version n'écrasera pas le fichier d'exercice Bandit Manchot que j'ai élaboré pour vous et que vous avez téléchargé pour réaliser les exercices de ce livre.

6. Éliminez la coche de l'option Créer le répertoire pour la solution.

   Lorsque cette option est cochée, elle crée un second dossier pour les fichiers de solution du programme, ce qui est superflu pour les solutions qui ne renferment qu'un unique projet, comme la plupart des programmes de ce livre.

7. Cliquez sur Enregistrer pour enregistrer vos fichiers.

Remarque Si vous ne souhaitez enregistrer que l'élément sur lequel vous travaillez actuellement (le formulaire, le module de code, ou autre), utilisez la commande Enregistrer du menu Fichier. Si vous souhaitez enregistrer l'élément actuel sous un autre nom, utilisez la commande Enregistrer sous.

La procédure *Button1_Click*

La procédure *Button1_Click* s'exécute lorsque l'utilisateur clique sur le bouton Lancer. Celle-ci utilise des instructions assez complexes, que je n'ai pas présentées et qui peuvent sembler délicates. Toutefois, si vous y prêtez attention, certains éléments vous sembleront familiers. Jetez un œil au contenu de ces procédures : vous aurez un aperçu du type de code que vous allez écrire plus loin dans cet ouvrage. Si vous préférez passer cette prévisualisation, rendez-vous à la section suivante, « Exécuter des applications Visual Basic ».

La procédure *Button1_Click* effectue trois tâches :

- Elle masque la photo numérique.
- Elle crée trois nombres aléatoires pour les étiquettes de numéro.
- Elle affiche la photo lorsque le chiffre 7 apparaît.

Examinons tour à tour chacune de ces étapes.

La ligne suivante permet de masquer la photo :

```
PictureBox1.Visible = False ' masque image
```

Elle se compose de deux parties : une instruction et un commentaire.

L'instruction `PictureBox1.Visible = False` fixe la propriété *Visible* de l'objet zone d'image (*PictureBox1*) à False (l'un des deux paramètres possibles). Vous vous souvenez sans doute que vous avez déjà affecté la valeur False à cette propriété à l'aide de la fenêtre de Propriétés. Vous devez répéter l'opération dans le code car la première tâche est une mise à jour et vous devez effacer une photo qui a pu être affichée lors d'une partie précédente. La propriété va être modifiée pendant l'exécution et non pendant la conception, vous devez donc la définir à l'aide du code. C'est une fonction très pratique de Visual Basic, dont je parlerai plus en détail dans le chapitre 3, « Travailler avec les contrôles de la Boîte à outils ».

La deuxième partie de la première ligne (en vert sur votre écran) est appelée *commentaire*. Les commentaires sont des notes explicatives incluses dans le code, introduites par un guillemet simple ('). Les programmeurs utilisent les commentaires pour décrire le fonctionnement des instructions les plus importantes. Ces notes ne sont pas traitées par Visual Basic pendant l'exécution du programme ; elles expliquent seulement ce que fait le programme. Vous utiliserez souvent les commentaires lorsque vous écrirez des programmes Visual Basic afin de rendre votre travail compréhensible.

Les trois lignes suivantes contiennent les calculs des nombres aléatoires. Ce concept vous semble étrange ? Vous pouvez demander à Visual Basic de générer des nombres aléatoires dans un contexte particulier. En d'autres termes, vous pouvez générer des nombres aléatoires pour des loteries, des jeux de dés, ou d'autres modèles statistiques. La fonction *Rnd* de chaque ligne génère un nombre aléatoire situé entre 0 et 1 (un nombre suivi d'une virgule et de plusieurs décimales) et la fonction *Int* renvoie la partie entière du résultat en multipliant le nombre aléatoire par 10. Ce calcul génère des nombres aléatoires entre 0 et 9 (ce dont vous avez besoin pour cette application de machine à sous).

```
Label1.Text = CStr(Int(Rnd() * 10)) ' sélectionne nombres aléatoires
```

Maintenant, vous allez devoir « faire un petit saut » dans le code. Vous devez copier ces nombres aléatoires dans les trois zones d'étiquette du formulaire, mais les nombres doivent d'abord être convertis en texte par la fonction *CStr* (convertir en chaîne). Observez la façon dont *CStr, Int,* et *Rnd* sont reliées dans l'instruction : elles fonctionnent ensemble, et le résultat ressemble à une formule mathématique. Après le calcul et la conversion, les valeurs sont assignées aux propriétés *Text* des trois premières étiquettes du formulaire, et donc affichées dans une police Times New Roman gras 24 points dans les trois étiquettes de numéro.

L'illustration suivante montre comment Visual Basic évalue une ligne de code étape par étape pour générer le nombre aléatoire 7 et le copier dans un objet étiquette. Visual Basic évalue l'expression comme un mathématicien résoudrait une formule mathématique.

Graphic 2-15

Example:
Label1.Text = CStr(Int(Rnd() * 10))

Code	Result
Rnd()	0.7055475
Rnd() * 10	7.055475
Int(Rnd() * 10)	7
CStr(Int(Rnd() * 10))	"7"
Label1.Text = CStr(Int(Rnd() * 10))	7

Le dernier groupe d'instructions vérifie si l'un des nombres aléatoires est 7. Si c'est le cas, le programme affiche la représentation médiévale d'une personne distribuant de l'argent et un signal sonore annonce les gains.

```
' si l'un des nombres est 7, affiche image et émet un signal sonore
If (Label1.Text = "7") Or (Label2.Text = "7") _
Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
   Beep()
End If
```

La procédure *Button1_Click* est exécutée, ou *invoquée*, chaque fois que l'utilisateur clique sur le bouton Lancer, et les instructions de la procédure sont réexécutées.

Exécuter des applications Visual Basic

Félicitations ! Vous voici prêt à exécuter votre premier véritable programme. Pour exécuter un programme Visual Basic à partir de l'environnement de développement, vous pouvez :

- Cliquer sur Démarrer le débogage dans le menu Déboguer.
- Cliquer sur le bouton Démarrer le débogage dans la barre d'outils Standard.
- Appuyer sur F5.

Essayez maintenant d'exécuter votre programme Bandit Manchot. Si Visual Basic affiche un message d'erreur, il se peut que vous ayez commis des erreurs de codage. Essayez d'y remédier en comparant la version imprimée de cet ouvrage avec celle que vous avez tapée, ou chargez Bandit Manchot à partir de votre disque dur et exécutez-le.

Exécuter le programme Bandit Manchot

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme Bandit Manchot est compilé et exécuté dans l'EDI. Après quelques instants, l'interface utilisateur apparaît comme vous l'avez conçue.

2. Cliquez sur le bouton Lancer.

   Le programme choisit trois nombres aléatoires et les affiche dans les étiquettes du formulaire :

Un 7 apparaît dans la première zone d'étiquette. La photo numérique représentant une personne distribuant de l'argent apparaît, et l'ordinateur émet un son. Vous avez gagné ! Le son que vous entendez dépend des réglages de l'option Son et périphériques audio du Panneau de configuration Windows. Pour rendre ce son plus agréable, modifiez le Son par défaut et optez pour un signal sonore plus dynamique.

3. Cliquez sur le bouton Lancer une quinzaine de fois, et observez les résultats qui apparaissent dans les cases de numéro.

Vous remportez le jackpot environ une fois sur trois. La probabilité est grande. La véritable probabilité est de 2,8 sur 10. Vous avez la chance du débutant. Plus tard, vous pourrez compliquer le jeu en affichant la photo seulement lorsque deux ou trois 7 apparaissent, ou en créant un cumul des gains.

4. Après avoir testé le programme, cliquez sur le bouton Arrêter. Le programme s'arrête et vous revenez à l'environnement de développement.

Astuce Si vous relancez ce programme, vous remarquerez que Bandit Manchot affiche exactement la même série de nombres aléatoires. Rien d'extraordinaire à cela : la fonction *Rnd* de Visual Basic a été conçue pour afficher d'abord une série de nombres à répétition pour que vous puissiez tester votre code à l'aide d'un résultat reproductible. Pour générer de véritables nombres « aléatoires », utilisez la fonction *Randomize* dans votre code, comme le montre l'exercice à la fin de ce chapitre. Le .NET Framework, dont vous découvrirez l'utilisation plus tard, fournit également des fonctions de nombres aléatoires.

Exemples de projets du site compagnon

Si vous n'avez pas recréé le projet Bandit Manchot de A à Z (ou si vous l'avez fait, mais que vous voulez le comparer à celui que j'ai créé pour vous), prenez le temps d'ouvrir et d'exécuter le projet Bandit Manchot situé dans le dossier vb085epe\chap02\Bandit Manchot de votre disque dur (l'emplacement par défaut des fichiers d'exercices de ce chapitre). Si vous avez besoin d'un cours de rattrapage sur l'ouverture de projets, suivez les instructions détaillées du chapitre 1. Si on vous demande si vous souhaitez enregistrer les modifications apportées au projet Mon Bandit Manchot, cliquez sur Enregistrer.

Cet ouvrage vous guide pas à pas : rien de tel que de créer vos propres projets et les tester. Toutefois, une fois que vous avez terminé, c'est souvent une bonne idée de les comparer avec la « solution » du fichier d'exercices fournie, surtout si vous avez rencontré des difficultés. Pour faciliter les choses, dans la plupart des exercices pas à pas, je vous donne le nom des fichiers d'exercices avant que vous n'exécutiez le programme complet.

Après avoir comparé votre projet Mon Bandit Manchot au fichier d'exercice correspondant, ouvrez encore une fois Mon Bandit Manchot et préparez-vous à le compiler dans un fichier exécutable. Si vous n'avez pas créé Mon Bandit Manchot, utilisez mon fichier d'exercice pour réaliser les étapes de la prochaine section.

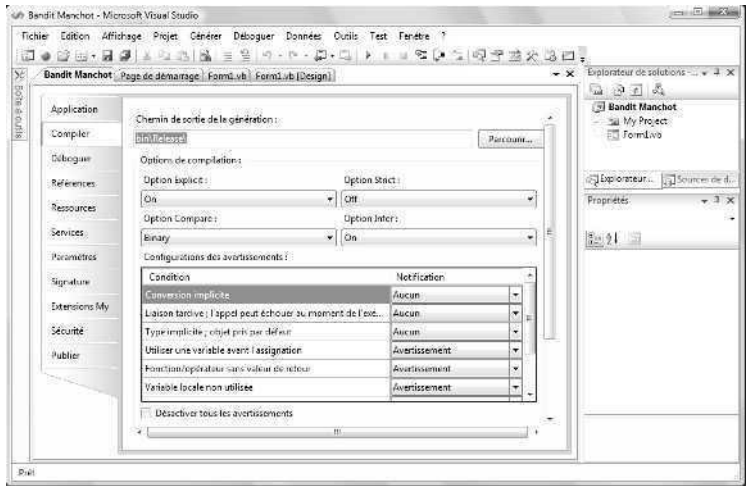
Créer un fichier exécutable

Vous terminerez ce chapitre en achevant le processus de développement, ce qui consiste à créer une application Windows, ou *fichier exécutable*. Les applications Windows créées à l'aide de Visual Studio prennent l'extension .exe et peuvent être exécutées dans tous les systèmes équipés de Windows et autres fichiers nécessaires. (Visual Basic les installe automatiquement, y compris les fichiers du .NET Framework). Si vous prévoyez de distribuer vos applications, référez-vous à la section « Déployer votre application » plus loin dans le chapitre.

À ce stade, vous devez savoir que Visual Studio peut créer deux types de fichiers exécutables : une version de débogage et une version définitive.

Visual Studio crée automatiquement les versions de débogage lorsque vous créez et testez un programme. Elles sont stockées dans le dossier bin\debug du dossier du projet. Les fichiers exécutables de débogage contiennent des données de débogage, qui ralentissent un peu l'exécution du programme.

Les versions définitives sont des fichiers exécutables optimisés ; elles sont stockées dans le dossier bin\release de votre projet. Pour personnaliser les paramètres de vos fichiers définitifs, cliquez sur la commande Propriétés de [*Nom du projet*] du menu Projet, puis sur l'onglet Compiler, où apparaît la liste d'options de compilation suivante :



Essayez maintenant de créer la version de débogage du programme Mon Bandit Manchot.exe.

Créer un fichier exécutable

1. Dans le menu Générer, cliquez sur la commande Générer Bandit Manchot.

   La commande Générer crée un dossier bin\release pour stocker votre projet (si le dossier n'existe pas déjà) et compile le code source de votre projet. Il en résulte un fichier exécutable dénommé Mon Bandit Manchot.exe. Pour ne pas perdre de temps, Visual Studio crée souvent des fichiers exécutables temporaires pendant que vous développez votre application. Il est toujours préférable de recompiler votre application manuellement à l'aide des commandes Générer ou Régénérer lorsque vous atteignez une étape importante.

   Essayez d'exécuter ce programme en dehors de l'EDI Visual Studio à partir du menu Démarrer de Windows.

2. Dans la barre des tâches Windows, cliquez sur Démarrer.

3. La commande suivante dépend de la version de Windows employée. Avec Windows Vista, saisissez **Exécuter** dans la zone de texte Rechercher et appuyez sur ENTRÉE pour ouvrir la boîte de dialogue Exécuter. Avec Windows XP ou antérieur, cliquez sur la commande Exécuter pour ouvrir la boîte de dialogue Exécuter.

4. Cliquez sur Parcourir, puis recherchez le dossier c:\vb08epe\chap02\Mon Bandit Manchot\bin\release.

5. Cliquez sur l'icône de l'application Mon Bandit Manchot.exe, cliquez sur Ouvrir puis sur OK.

   Le programme Bandit Manchot se charge et s'exécute sous Windows. Comme il s'agit d'une simple application de test dépourvue de certificat de publication officiel garantissant son innocuité ou son authenticité, vous pourriez obtenir le message suivant : « L'éditeur n'a pas pu être vérifié. Êtes-vous sûr de vouloir exécuter ce programme ? ». Si cela se produit, cliquez sur Oui pour exécuter malgré tout le programme. La création de tels certificats dépasse la portée de ce livre, mais ce programme est parfaitement inoffensif.

6. Cliquez plusieurs fois sur Lancer pour vérifier que le jeu fonctionne, puis cliquez sur Arrêter.

Astuce Vous pouvez également exécuter des applications Windows, y compris les programmes Visual Basic compilés, en ouvrant l'Explorateur Windows et en double-cliquant sur le fichier exécutable. Pour créer un raccourci vers Mon Bandit Manchot.exe sur le bureau Windows, cliquez droit sur le bureau Windows, pointez sur Nouveau, puis cliquez sur Raccourci. Lorsqu'on vous demande l'emplacement de votre fichier d'application, cliquez sur Parcourir, puis sélectionnez le fichier exécutable Mon Bandit Manchot.exe. Cliquez sur OK, Suivant puis Terminer. Windows place une icône sur le bureau, sur laquelle vous pouvez double-cliquer pour exécuter votre programme.

7. Dans le menu Fichier, cliquez sur Quitter pour fermer Visual Studio et le projet Mon Bandit Manchot. L'environnement de développement de Visual Studio se ferme.

Déployer une application

Visual Studio vous aide à distribuer des applications Visual Basic en fournissant plusieurs options de *déploiement*, ce qui consiste à installer l'application sur un ou plusieurs systèmes informatiques. Alors que Visual Basic 6 requiert un programme d'installation complexe qui copie les bibliothèques de liens dynamiques (DLL) et les fichiers de support et enregistre l'application auprès du système d'exploitation, les applications de Visual Studio 2008 sont compilées sous forme d'*assemblys*, des unités de déploiement qui comprennent un ou plusieurs fichiers nécessaires à l'exécution du programme. Les assemblys comportent quatre éléments : du code *Microsoft intermediate language* (MSIL), des métadonnées, un manifeste, des fichiers de support et des ressources.

Les assemblys sont tellement complets et auto-décrits que les applications Visual Studio ne doivent pas nécessairement être enregistrées dans le système d'exploitation pour être exécutées. Cela signifie qu'une application Visual Basic 2008 peut théoriquement être installée simplement en copiant l'assembly du programme sur un deuxième ordinateur qui possède une version adéquate du .NET Framework (un procédé de type *installation XCOPY*, du nom de la commande MS-DOS XCOPY qui copie une structure de répertoire (dossier) d'un emplacement à un autre). Dans la pratique, il n'est cependant pas pratique

de déployer des applications Visual Basic à l'aide d'une procédure telle que XCOPY (*via* l'Invite de commandes) ou de l'Explorateur Windows. Pour les applications commerciales, on préfère souvent un programme d'installation possédant une interface utilisateur, et il est souvent souhaitable d'enregistrer le programme avec le système d'exploitation, pour qu'il puisse être désinstallé par la suite à l'aide du Panneau de configuration.

Même si les options avancées relatives au déploiement et à la sécurité dépassent la portée de ce livre, vous devez connaître les différentes options de déploiement. Pour gérer le processus d'installation, Visual Studio 2008 propose deux techniques de déploiement, *ClickOnce* et *Installateur Windows*.

ClickOnce sert à créer un service d'installation pour des applications de bureau auxquelles les utilisateurs peuvent accéder avec une interaction minimale. *ClickOnce* permet de spécifier des exigences préalables comme le .NET Framework. Il est facile de publier des mises à jour au fur et à mesure des améliorations apportées à votre programme. Votre programme peut être publié vers un serveur web ou un serveur de fichiers. Vous pouvez débuter à tout moment avec *ClickOnce* à l'aide de la commande Publier du menu Générer. Vous pouvez également spécifier les paramètres de *ClickOnce* à l'aide de la commande Propriétés du menu Projet. Vous configurez les paramètres de *ClickOnce* sur l'onglet Publier du Concepteur de projet.

L'Installateur Windows est un processus d'installation plus classique. Vous ajoutez depuis Visual Studio un projet de configuration ou d'Installateur Windows, ce qui crée automatiquement un programme d'installation pour l'application. Ce projet d'installation peut être personnalisé pour autoriser différentes méthodes d'installation, comme depuis des CD-ROM ou des serveurs web. Vous pouvez à tout moment tester les Installateurs Windows à l'aide de la commande Nouveau projet du menu Fichier et créer un projet d'installateur Windows personnalisé. Sous Autres types de projets, sélectionnez l'option Setup And Deployment pour afficher une liste de modèles et d'assistants d'installation.

Aller plus loin : Ajouter un élément à un programme

Vous pouvez redémarrer Visual Studio à tout moment et travailler sur un projet de programmation que vous avez stocké sur votre disque dur. Redémarrez Visual Studio et ajoutez une instruction *Randomize* au programme Bandit Manchot.

Recharger Bandit Manchot

1. Cliquez sur Démarrer dans la barre des tâches, cliquez sur Tous les programmes, sur Microsoft Visual Studio 2008, puis cliquez sur l'icône du programme Microsoft Visual Studio 2008.

   Une liste de projets récents apparaît sur la page de démarrage de Visual Studio. Vous venez juste de travailler sur Bandit Manchot : le projet Mon Bandit Manchot est le premier de la liste.

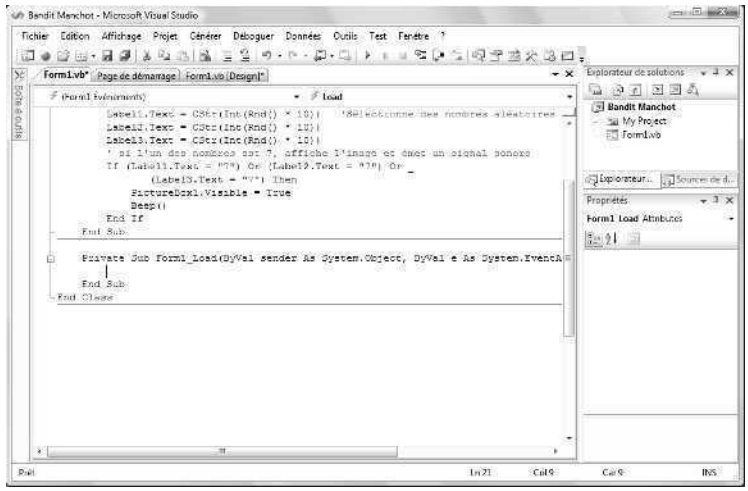
2. Cliquez sur le lien Mon Bandit Manchot pour ouvrir le projet Bandit Manchot.

   Le programme Bandit Manchot s'ouvre et le formulaire Mon Bandit Manchot apparaît. Si vous ne voyez pas le formulaire, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Vous allez maintenant ajouter l'instruction *Randomize* à la procédure *Form_Load*, une procédure spéciale associée au formulaire et exécutée à chaque démarrage du programme.

3. Double-cliquez sur le formulaire (pas sur l'un des objets) pour afficher la procédure *Form_Load*.

   La procédure *Form_Load* apparaît dans l'Éditeur de code, comme suit :



4. Tapez **Randomize**, puis appuyez sur la touche ENTRÉE.

   L'instruction *Randomize* est ajoutée au programme et sera exécutée à chaque démarrage du programme. *Randomize* utilise l'horloge système pour créer un point de départ aléatoire, ou *graine*, pour l'instruction *Rnd* employée dans la procédure *Button1_Click*. Comme je l'ai indiqué précédemment, sans l'instruction *Randomize*, le programme Bandit Manchot générerait la même chaîne de nombres aléatoires à chaque redémarrage du programme. Avec *Randomize*, le programme « tourne » de manière aléatoire chaque fois qu'il s'exécute et les nombres ne suivent pas un modèle reconnaissable.

5. Exécutez la nouvelle version de Bandit Manchot, puis enregistrez le projet. Si vous prévoyez d'utiliser souvent la nouvelle version, vous pouvez également créer un nouveau fichier .exe.

6. Lorsque vous avez terminé, cliquez sur le bouton Fermer le projet dans le menu Fichier.

   Les fichiers associés au programme Bandit Manchot sont fermés.

Rappel du chapitre 2

Pour	Faites ceci
Créer une interface utilisateur	Utilisez les contrôles de la Boîte à outils pour disposer les objets sur votre formulaire, puis définissez les paramètres nécessaires. Redimensionnez le formulaire et les objets.
Déplacer un objet	Placez le pointeur de la souris au-dessus de l'objet jusqu'à ce que la flèche à quatre pointes apparaisse, puis faites glisser l'objet.
Redimensionner un objet	Cliquez sur l'objet pour le sélectionner, puis faites glisser la poignée de dimensionnement correspondant à la partie de l'objet que vous souhaitez redimensionner.
Supprimer un objet	Cliquez sur l'objet, puis appuyez sur la touche SUPPR.
Ouvrir l'Éditeur de code	Double-cliquez sur un objet du formulaire (ou sur le formulaire lui-même). *ou* Sélectionnez un formulaire ou un module dans l'Explorateur de solutions, puis cliquez sur le bouton Afficher le code.
Écrire le code de programme	Dans l'Éditeur de code, tapez les instructions du programme Visual Basic associées aux objets.
Enregistrer un programme	Dans le menu Fichier, cliquez sur la commande Enregistrer tout. *ou* Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout.
Enregistrer un fichier de formulaire	Veillez à ce que le formulaire soit ouvert, puis cliquez sur la commande Enregistrer du menu Fichier. *ou* Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer.
Créer un fichier .exe	Dans le menu Générer, cliquez sur la commande Générer ou Régénérer.
Déployer une application à l'aide de la technologie ClickOnce	Cliquez sur la commande Publier du menu Générer, puis utilisez l'assistant Publication pour spécifier l'emplacement et les paramètres de l'application.
Recharger un projet	Dans le menu Fichier, cliquez sur la commande Ouvrir Projet. *ou* Dans le menu Fichier, pointez sur Projets récents, puis cliquez sur le projet souhaité. *ou* Cliquez sur le projet dans la liste de projets récents de la page de démarrage de Visual Studio.

Chapitre 3

Travailler avec les contrôles de la Boîte à outils

À la fin de ce chapitre, vous saurez :

- Utiliser les contrôles *TextBox* et *Button* pour créer le programme Bonjour
- Utiliser le contrôle *DateTimePicker* pour afficher votre date de naissance
- Utiliser les contrôles *CheckBox*, *RadioButton*, *ListBox* et *ComboBox* pour traiter les saisies utilisateur
- Utiliser le contrôle *LinkLabel* et la méthode *Process.Start* pour afficher une page web dans le navigateur par défaut de votre système

Comme vous l'avez appris dans les chapitres précédents, les contrôles de Microsoft Visual Studio 2008 sont les outils graphiques utilisés pour construire l'interface utilisateur d'un programme Microsoft Visual Basic. Les contrôles se trouvent dans la Boîte à outils de l'environnement de développement et ils permettent de créer des objets sur un formulaire en quelques cliquer-déplacer.

Les contrôles des formulaires Windows sont spécialement conçus pour créer des applications Windows. Vous les trouverez dans l'onglet Tous les Windows Forms de la Boîte à outils, bien que la plupart des contrôles soit également accessibles dans des onglets tels que Contrôles communs, Conteneurs et Impression. Vous avez utilisé quelques-uns de ces contrôles dans le chapitre précédent. Plus loin dans cet ouvrage, vous allez apprendre à utiliser d'autres contrôles, dont les outils de création d'applications de bases de données et de pages web.

Dans ce chapitre, vous allez apprendre à afficher des informations dans une zone de texte, à travailler avec la date et l'heure de votre système, à traiter les saisies utilisateur et à afficher une page web dans un programme Visual Basic. Les exercices de ce chapitre vous aideront à concevoir vos propres applications et vous en apprendront davantage à propos des objets, des propriétés et du code.

L'utilisation de base des contrôles : Le programme Bonjour

Le programme Bonjour, qui illustre la construction et l'exécution de l'utilitaire le plus simple dans un langage de programmation, est un grand classique des ouvrages d'initiation à la programmation. À l'époque de la programmation textuelle, Bonjour était habituellement un programme de deux ou trois lignes tapé dans un éditeur de programme et

assemblé à l'aide d'un compilateur autonome. Toutefois, avec l'avènement de systèmes d'exploitation et d'outils de programmation complexes, le classique Bonjour est devenu un programme plus sophistiqué contenant des dizaines de lignes, et sa création requiert plusieurs outils de programmation. Heureusement, avec Visual Studio et Visual Basic 2008, la création d'un programme Bonjour reste relativement simple. Pour construire une interface utilisateur complète, il suffit de créer deux objets, de définir deux propriétés et de saisir une ligne de code. Essayez.

Créer un programme Bonjour

1. Démarrez Visual Studio 2008 si nécessaire.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

   Visual Studio affiche la boîte de dialogue Nouveau projet, qui vous invite à indiquer le nom du projet et le modèle à employer.

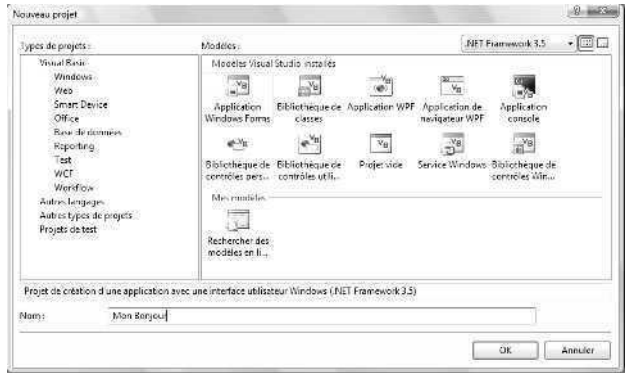
> **Remarque** Utilisez les instructions suivantes chaque fois que vous souhaitez créer un nouveau projet sur votre disque dur.

3. Veillez à sélectionner le type de projet Visual Basic et la catégorie Windows, puis cliquez sur le modèle Application Windows Forms.

   Ces sélections indiquent que vous allez construire une application Visual Basic autonome, qui s'exécutera sous Windows.
4. Supprimez le nom du projet par défaut (WindowsApplication1) de la zone de texte Nom, puis tapez **Mon Bonjour.**

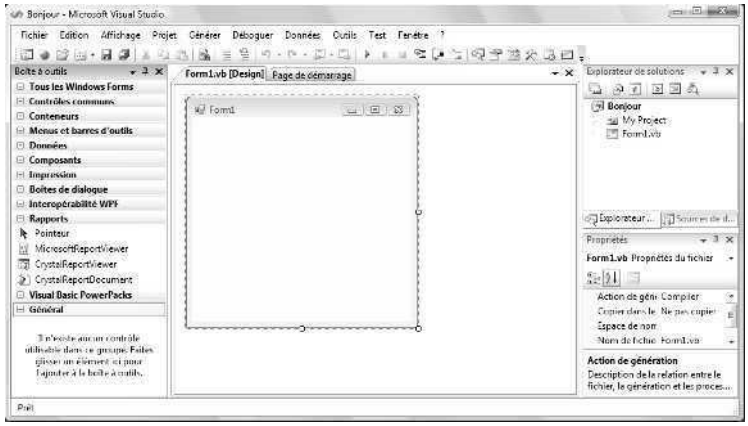
> **Remarque** Dans cet ouvrage, je vous invite à créer des projets test avec le préfixe Mon, pour différencier votre travail des fichiers d'exercices que vous avez téléchargés depuis le site *www.dunod.com*. Toutefois, les projets apparaissent généralement dans les copies d'écran sans ce préfixe, puisque je les ai créés ainsi.

La boîte de dialogue Nouveau Projet ressemble à ce qui suit :



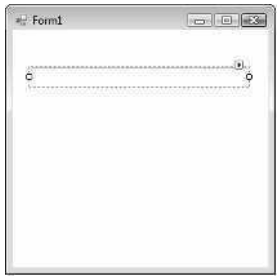
5. Cliquez sur OK pour créer votre nouveau projet.

Visual Basic crée le nouveau projet et un formulaire vierge apparaît dans le Concepteur, comme le montre l'illustration suivante. Les deux contrôles que vous allez utiliser dans cet exercice, *Button* et *TextBox*, se trouvent dans la Boîte à outils, qui apparaît sur l'illustration sous forme de fenêtre ancrée. Si la configuration de vos outils de programmation est différente, prenez le temps de les organiser conformément à l'illustration. Si vous avez besoin d'un cours de rattrapage, le chapitre 1 « Explorer l'environnement de développement intégré de Visual Studio », explique comment configurer l'EDI.



6. Dans la Boîte à outils, cliquez sur le contrôle *TextBox* de l'onglet Contrôles communs.

7. Dessinez une zone de texte comme celle-ci :



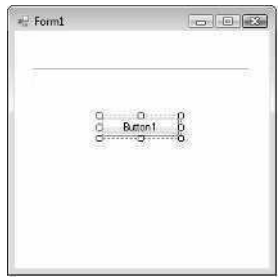
Les *zones de texte* sont utilisées pour afficher du texte sur un formulaire ou recueillir les saisies de l'utilisateur pendant l'exécution du programme. Le fonctionnement d'une zone de texte dépend de votre définition de ses paramètres et de la référence à la zone de texte dans le code. Dans ce programme, on utilise un objet zone de texte pour afficher le message « Bonjour le monde ! » lorsqu'on clique sur un objet bouton du formulaire.

Remarque Les lecteurs qui ont travaillé avec des versions antérieures de Visual Basic remarqueront que la valeur par défaut de la propriété *Text* n'est plus « TextBox1 ». La zone de texte est désormais vide par défaut.

Vous allez maintenant ajouter un bouton au formulaire.

8. Dans la Boîte à outils, cliquez sur le contrôle *Button*.
9. Sur le formulaire, dessinez un bouton en dessous de la zone de texte.

   Votre formulaire présente un résultat similaire à :



Comme vous l'avez appris au chapitre 2, « Écrire votre premier programme », les boutons sont utilisés pour recueillir les saisies utilisateur les plus basiques. Lorsqu'un utilisateur clique sur un bouton, il demande que le programme effectue immédiatement une action spécifique. Pour Visual Basic, l'utilisateur appuie sur le bouton pour créer un *événement* à traiter par le programme. Les boutons standard d'un programme sont le bouton OK, sur lequel l'utilisateur clique pour accepter une liste d'options et indiquer qu'il est prêt à poursuivre ; le bouton Annuler, sur lequel l'utilisateur clique pour supprimer une liste d'options ; et le bouton Quitter, sur lequel l'utilisateur clique pour quitter le programme. Dans tous les cas, vous devez utiliser ces boutons de manière à ce qu'ils fonctionnent comme prévu lorsque l'utilisateur clique dessus. Les caractéristiques d'un bouton (comme celles de tous les objets) peuvent être modifiées à l'aide des paramètres de propriétés et des références à l'objet dans le code.

10. À l'aide de la fenêtre Propriétés, définissez les propriétés suivantes pour l'objet bouton :

Objet	Propriété	Paramètre
Button1	Text	« OK »

Pour en savoir plus sur la définition de propriétés et leur lecture sous forme de tableau, voir la section du chapitre 1 « La fenêtre Propriétés ».

11. Dans l'Éditeur de code, double-cliquez sur le bouton OK et tapez l'instruction suivante entre les instructions *Private Sub Button1_Click* et *End Sub*.

```
TextBox1.Text = "Bonjour le monde !"
```

Remarque Tandis que vous saisissez des instructions, Visual Studio affiche une zone de liste qui renferme tous les éléments valides qui correspondent à votre texte. Après avoir saisi le nom de l'objet *TextBox1*, suivi d'un point, Visual Studio affiche, pour vous rafraîchir la mémoire, une zone de liste qui contient toutes les propriétés et méthodes valides pour les objets zone de texte. Cette zone de liste se nomme Microsoft IntelliSense et peut être très utile lors de l'écriture du code. Si vous cliquez sur un élément de la zone de liste, vous obtenez généralement une infobulle qui procure une brève description de l'élément. Vous pouvez ajouter à votre code une propriété depuis la liste en double-cliquant dessus ou en vous servant des touches fléchées pour la sélectionner puis en appuyant sur la touche TAB. Vous pouvez également poursuivre votre saisie. En général, je continue à taper, sauf si je passe en revue de nouvelles fonctions.

L'instruction que vous avez saisie transforme la propriété *Text* de la zone de texte en « Bonjour le monde ! » lorsque l'utilisateur clique sur le bouton pendant l'exécution. Le signe égal (=) assigne tout ce qui se trouve entre guillemets à la propriété *Text* de l'objet *TextBox1*. Dans cet exemple, une propriété est modifiée pendant l'exécution ; c'est l'une des utilisations les plus courantes du code dans un programme Visual Basic.

Vous êtes prêt à exécuter le programme Bonjour.

Exécuter le programme Bonjour

Astuce Le programme Bonjour complet est disponible dans le dossier c:\vb08epe\chap03\Bonjour.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme Bonjour est compilé, puis s'exécute dans l'EDI Visual Studio.

2. Cliquez sur OK.

   Le programme affiche le message d'accueil « Bonjour le monde ! » dans la zone de texte :

Lorsque vous avez cliqué sur le bouton OK, le code a transformé la propriété *Text* de la zone de liste vierge *TextBox1* en « Bonjour le monde ! » et a affiché ce texte dans la zone. Si vous n'obtenez pas ce résultat, répétez les étapes de la section précédente et recommencez de zéro. Il se peut que vous ayez mal défini une propriété ou fait une erreur de syntaxe. Dans l'Éditeur de code, celles-ci sont soulignées en dents de scie.

3. Cliquez sur le bouton Fermer dans l'angle supérieur droit de la fenêtre du programme Bonjour pour arrêter le programme.

Remarque Pour arrêter un programme en cours d'exécution dans Visual Studio, vous pouvez également cliquer sur le bouton Arrêter le débogage de la barre d'outils Standard.

4. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer votre nouveau projet sur le disque dur.

   Visual Studio vous demande un nom et un emplacement pour le projet.

5. Cliquez sur le bouton Parcourir.

   La boîte de dialogue Emplacement du projet s'ouvre. Utilisez-la pour spécifier l'emplacement du projet et, le cas échéant, créer de nouveaux dossiers pour le projet. Bien que vous puissiez choisir l'emplacement de vos projets (le dossier Mes Documents\Visual Studio 2008\Projects par exemple), dans cet ouvrage je vous conseille de les enregistrer dans le dossier c:\vb08epe, l'emplacement par défaut des fichiers d'exercices *Étape par étape*. Si par la suite vous souhaitez supprimer tous les fichiers associés à ce cours de programmation, vous saurez où les trouver, et vous pourrez les supprimer très simplement en effaçant tout le dossier.

6. Naviguez jusqu'au dossier c:\vb08epe\chap03.

7. Cliquez sur le bouton Ouvrir ou Sélectionner un dossier pour ouvrir le dossier spécifié.

8. Si elle est cochée, supprimez la coche de la case Créer le répertoire pour la solution.

   Cette solution ne contenant qu'un projet (comme la plupart des exercices de cet ouvrage), il est inutile de créer un dossier racine à part pour les fichiers solution du projet. Cependant, libre à vous de le créer si vous le souhaitez.

9. Cliquez sur Enregistrer pour enregistrer le projet et ses fichiers.

Félicitations ! Vous faites partie des développeurs qui ont écrit un programme Bonjour. Essayons maintenant un nouveau contrôle.

Utiliser le contrôle *DateTimePicker*

Certains contrôles Visual Basic affichent des informations, d'autres recueillent des informations utilisateur ou traitent des données à l'arrière-plan. Dans cet exercice, vous allez travailler avec le contrôle *DateTimePicker,* qui invite l'utilisateur à saisir une date ou une heure à l'aide d'un calendrier graphique muni de flèches de défilement. Bien que vos connaissances du contrôle soient encore rudimentaires, c'est en expérimentant *DateTimePicker* que vous vous ferez une idée de tout ce que peuvent faire automatiquement pour vous les contrôles Visual Basic, et de la manière de traiter les informations qui en émanent.

Le programme Anniversaire

Le programme Anniversaire utilise un contrôle *DateTimePicker* et un contrôle *Button* pour demander à l'utilisateur sa date d'anniversaire. Il affiche ensuite cette information dans une boîte de message. Essayez.

Créer le programme Anniversaire

1. Dans le menu Fichier, cliquez sur Fermer le projet pour fermer le projet Mon Bonjour.

   Les fichiers du programme se ferment.

2. Dans le menu Fichier, cliquez sur Nouveau Projet.

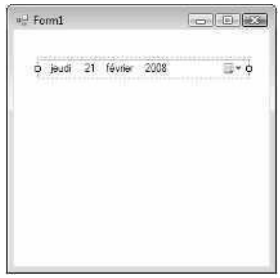
   La boîte de dialogue Nouveau projet s'affiche.

3. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Anniversaire**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans le Concepteur.

4. Dans la Boîte à outils, cliquez sur le contrôle *DateTimePicker*.

5. Dessinez un objet sélectionneur de date/heure au milieu du formulaire, comme suit :



Par défaut, cet objet affiche la date du jour, mais vous pouvez modifier la date affichée en changeant la propriété *Value* de l'objet. L'affichage de la date est un excellent guide de conception : il permet de dimensionner l'objet sélectionneur de date/heure pendant sa création.

6. Dans la Boîte à outils, cliquez sur le contrôle *Button*, puis ajoutez un objet bouton sous le sélectionneur de date/heure.

   Ce bouton vous permettra d'afficher votre date d'anniversaire et vérifier que le sélectionneur de date/heure fonctionne.

7. Dans la fenêtre Propriétés, transformez la propriété *Text* de l'objet bouton en **Afficher mon anniversaire**.

   Vous allez maintenant ajouter quelques lignes de code à une procédure associée à l'objet bouton. Il s'agit d'une procédure événementielle, car elle s'exécute quand un événement, comme un clic de souris, se produit ou est *initié* dans l'objet.

8. Sur le formulaire, double-cliquez sur l'objet bouton pour afficher sa procédure événementielle par défaut, puis tapez les instructions suivantes entre *Private Sub* et *End Sub* dans la procédure événementielle *Button1_Click* :

```
MsgBox("Vous êtes né le " & DateTimePicker1.Text)
MsgBox("Jour de l'année : " & _
    DateTimePicker1.Value.DayOfYear.ToString())
```

   Ces instructions font apparaître deux petites boîtes de dialogue contenant les informations de l'objet sélectionneur de date/heure. La première ligne utilise la propriété *Text* du sélectionneur pour afficher les informations sur la date d'anniversaire que vous sélectionnez en utilisant l'objet pendant l'exécution. La fonction *MsgBox* ajoute la chaîne « Vous êtes né le » à la valeur texte de la propriété *Text* du sélectionneur de date/heure. Ces deux informations sont jointes par l'opérateur de concaténation (&). Vous en apprendrez davantage sur la fonction *MsgBox* et l'opérateur de concaténation au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework ».

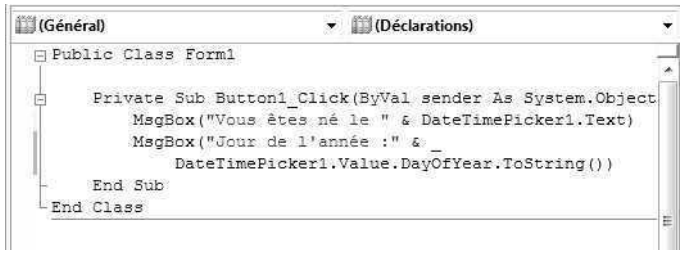
   Les deux lignes suivantes forment une seule instruction ; elles ont été séparées par le caractère de suite de ligne (_) car l'instruction était trop longue pour être imprimée dans cet ouvrage.

Remarque Dans l'Éditeur de code, les lignes de programme peuvent contenir 65 000 caractères, mais il est souvent plus simple de travailler avec des lignes de 80 caractères, voire moins. Vous pouvez diviser les instructions longues en plusieurs lignes en terminant chaque ligne de l'instruction, sauf la dernière, par un espace et un caractère de suite de ligne (_). Toutefois, il est impossible de procéder ainsi pour séparer une chaîne entre guillemets. Dans cet exercice, j'utilise le caractère de suite de ligne pour découper la deuxième ligne de code en deux.

L'instruction `DateTimePicker1.Value.DayOfYear.ToString()` utilise l'objet sélectionneur de date/heure pour calculer à quel jour de l'année correspond votre anniversaire, à partir du 1er janvier. La propriété *DayOfYear* et la méthode *ToString* le permettent : elles convertissent le résultat numérique du calcul de la date en valeur textuelle, plus simple à afficher par la fonction *MsgBox*.

Les *méthodes* sont des instructions spéciales qui effectuent une action ou un service pour un certain objet, comme la conversion d'un nombre en chaîne ou l'ajout d'éléments à une zone de liste. Les méthodes diffèrent des propriétés, qui contiennent une valeur, et des procédures événementielles, qui s'exécutent lorsque l'utilisateur manipule un objet. Les méthodes ne sont pas spécifiques à un objet. Ainsi, lorsque vous apprenez à utiliser une certaine méthode, vous pouvez souvent l'appliquer dans d'autres circonstances. Dans cet ouvrage, nous aborderons plusieurs méthodes importantes.

Après avoir saisi le code de la procédure événementielle *Button1_Click*, l'Éditeur de code présente un résultat similaire à :



9. Cliquez sur le bouton Enregistrer tout pour enregistrer vos modifications et choisissez le dossier de destination c:\vb08epe\chap03.

Vous êtes prêt à exécuter le programme Anniversaire.

Exécuter le programme Anniversaire

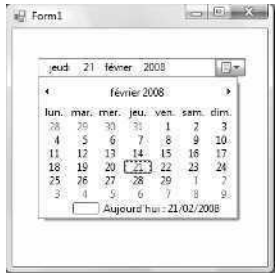
Astuce Le programme Anniversaire complet est disponible dans le dossier c:\vb08epe\chap03\Anniversaire.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme Anniversaire s'exécute dans l'EDI. La date du jour est affichée dans le sélectionneur de date/heure.

2. Cliquez sur la flèche dans le sélectionneur pour afficher l'objet en mode calendrier.

   Votre formulaire ressemble à l'illustration suivante, avec une date différente.



3. Cliquez sur la flèche de défilement de gauche pour voir les mois précédents du calendrier.

   Remarquez que la partie zone de texte de l'objet se transforme également lorsque vous faites défiler la date. En revanche, la valeur « Aujourd'hui » située en bas du calendrier reste la même.

   Bien qu'il soit possible de revenir en arrière jusqu'à votre date d'anniversaire, vous n'aurez peut-être pas la patience de faire défiler le calendrier mois par mois. Pour arriver plus rapidement à votre date d'anniversaire, sélectionnez l'année dans le sélectionneur de date/heure et saisissez une nouvelle valeur.

4. Sélectionnez l'année à quatre chiffres dans la zone de texte du sélectionneur de date/heure.

   Lorsque vous sélectionnez l'année, le sélectionneur de date/heure se ferme.

5. Tapez votre année d'anniversaire à la place de l'année sélectionnée, puis cliquez à nouveau sur la flèche.

   Le calendrier de votre année de naissance apparaît.

6. Cliquez à nouveau sur la flèche de défilement pour déterminer votre mois de naissance, puis cliquez sur votre jour de naissance.

   Si vous ne saviez pas quel jour de la semaine vous êtes né, c'est désormais chose faite !

   Lorsque vous sélectionnez la date finale, le sélectionneur de date/heure se ferme et votre date d'anniversaire apparaît dans la zone de texte. Vous pouvez cliquer sur l'objet bouton pour voir comment les autres objets de votre formulaire accèdent à ces informations.

7. Cliquez sur le bouton Afficher mon anniversaire.

   Visual Basic exécute le code et affiche une boîte de message avec le jour et la date de votre anniversaire. Les dates correspondent :

8. Cliquez sur OK dans la boîte de message.

   Une deuxième boîte de message apparaît ; elle indique quel jour de l'année vous êtes né. Tout semble fonctionner ! Ce contrôle est très ingénieux : non seulement il garde en mémoire la nouvelle date ou heure que vous avez saisie, mais il n'oublie pas non plus la date et l'heure du jour, et il peut l'afficher dans différents formats très pratiques.

> **Remarque** Pour configurer l'objet sélectionneur de date/heure afin qu'il affiche des heures et non des dates, réglez la propriété *Format* sur Time.

9. Cliquez OK pour fermer la boîte de message, puis sur le bouton Fermer du formulaire.

Vous en avez terminé avec le contrôle *DateTimePicker*.

Un mot de terminologie

Dans cet ouvrage, j'ai utilisé différents termes pour décrire les éléments d'un programme Visual Basic. Savez-vous ce qu'ils signifient ? Pour éviter toute confusion, voici quelques définitions.

Instruction.
Une instruction est une ligne de code d'un programme Visual Basic, un ordre autonome exécuté par le compilateur Visual Basic, qui accomplit quelque chose d'utile dans l'application. La longueur des instructions peut varier (certaines ne contiennent qu'un mot clé Visual Basic), mais toutes doivent respecter les règles de syntaxe définies et appliquées par le compilateur Visual Basic. Dans Visual Studio 2008, les instructions de programme peuvent être composées de mots clés, de propriétés, de noms d'objets, de variables, de nombres, de symboles spéciaux et d'autres valeurs (voir chapitres 2 et 5).

Mot clé.
Un mot clé est un mot réservé du langage Visual Basic, reconnu par le compilateur Visual Basic et chargé d'une tâche. Par exemple, le mot clé *End* interrompt l'exécution du programme. Les mots clés constituent l'un des blocs de construction des instructions. Avec les objets, les propriétés, les variables et autres valeurs, ils forment des lignes de code complètes et donc, des instructions pour le compilateur et le système d'exploitation. La plupart des mots clés sont représentés en bleu dans l'Éditeur de code (voir chapitre 2).

Variable.
Une variable est un conteneur spécial utilisé pour conserver temporairement des données dans un programme. Le programmeur crée des variables à l'aide de l'instruction *Dim,* puis les utilise pour stocker les résultats d'un calcul, des noms des fichiers, des saisies, etc. Les variables peuvent stocker des nombres, des noms et des valeurs de propriétés (voir chapitre 5).

Contrôle.
Un contrôle est un outil utilisé pour créer des objets dans un programme Visual Basic (généralement un formulaire). Vous sélectionnez les contrôles dans la Boîte à outils et les utilisez pour dessiner des objets sur un formulaire à l'aide la souris. La plupart des contrôles servent à créer des éléments de l'interface utilisateur, comme les boutons, les zones d'image et les zones de liste (voir en particulier les chapitres 2 à 4).

Objet.
Un objet est un élément de l'interface utilisateur créé dans un programme Visual Basic à l'aide d'un contrôle de la Boîte à outils. Un objet peut également être fourni par d'autres composants système et nombre d'entre eux contiennent des données. Dans Visual Basic, le formulaire lui-même est aussi un objet. Techniquement parlant, un objet est une instance d'une classe qui prend en charge des méthodes, des propriétés et des événements. Les objets possèdent également ce qui est nommé

une *fonctionnalité inhérente* : ils savent comment fonctionner et peuvent répondre par eux-mêmes à certaines situations. Une zone de liste « sait » comment défiler, par exemple. (voir les chapitres 1 à 4).

Classe.

Une classe est un modèle ou patron pour un ou plusieurs objets. Elle définit ce qu'un objet peut faire, mais pas l'objet lui-même. Vous pouvez utiliser les classes Visual Studio existantes (comme *System.Math* et *System.Windows.Forms.Form*) ou construire vos propres classes en *héritant* de propriétés, de méthodes et d'événements des précédentes. L'héritage permet à une classe d'acquérir l'interface préexistante et les caractéristiques de comportement d'une autre classe. Bien que les classes restent peut-être encore une notion ésotérique pour vous, elles constituent des caractéristiques essentielles de Visual Studio 2008. Vous allez les utiliser dans cet ouvrage pour construire rapidement des interfaces utilisateur et étendre votre travail à d'autres projets de programmation (voir les chapitres 5 et 16).

Espace de noms.

Un espace de noms est une bibliothèque de classes hiérarchisée organisée sous un nom unique, comme *System.Windows* ou *System.Diagnostics*. Pour accéder aux classes et aux objets sous-jacents d'un espace de noms, placez une instruction *Imports* au début du code. Tous les projets Visual Studio possèdent également un espace de noms racine, qui est défini à l'aide de la page Propriétés du projet. Les ouvrages et la documentation sur Visual Studio font référence aux espaces de noms en tant que bibliothèques d'objets ou bibliothèques de classes (voir chapitre 5).

Propriété.

Une propriété est une valeur ou une caractéristique d'un objet. Par exemple, un objet bouton possède une propriété *Text* pour spécifier le texte qui apparaît sur le bouton et une propriété *Image* pour spécifier le chemin vers un fichier image devant apparaître sur le bouton. Dans Visual Basic, on définit les propriétés au moment de la conception, à l'aide de la fenêtre Propriétés, ou en cours d'exécution, à l'aide d'instructions dans le code. Dans le code, le format de définition d'une propriété est

```
Objet.Propriété = Valeur
```

où *Objet* est le nom de l'objet que vous personnalisez, *Propriété* la caractéristique que vous souhaitez modifier et *Valeur* le nouveau paramètre de propriété. Par exemple,

```
Button1.Text = "Bonjour"
```

permet de régler la propriété *Text* de l'objet *Button1* sur « Bonjour » (voir les chapitres 1 à 3).

Procédure événementielle.

Une procédure événementielle est un bloc de code qui s'exécute lorsqu'un objet est manipulé dans un programme. Par exemple, lorsque l'on clique sur l'objet *Button1*,

la procédure événementielle *Button1_Click* s'exécute. En général, les procédures événementielles évaluent et définissent les propriétés et utilisent d'autres instructions de programme pour effectuer les tâches du programme (voir les chapitres 1 à 3).

Méthode.

Une méthode est une instruction spéciale qui effectue une action ou un service pour un certain objet du programme. Dans le code, l'utilisation d'une méthode est notée de la manière suivante :

```
Objet.Méthode(Valeur)
```

où *Objet* est le nom de l'objet avec lequel vous souhaitez travailler, *Méthode* représente l'action que vous souhaitez effectuer, et *Valeur* représente un argument facultatif pouvant être utilisé par la méthode. Par exemple, l'instruction

```
ListBox1.Items.Add("Vérifier")
```

utilise la méthode *Add* pour placer le mot *Vérifier* dans la zone de liste *ListBox1*. Les méthodes et les propriétés sont souvent identifiées par leur position dans une collection ou une bibliothèque d'objets. Ne soyez pas surpris de voir de longues références comme *System.Drawing.Image.FromFile*, qui signifie « la méthode *FromFile* qui fait partie de la classe *Image*, elle-même appartenant à la bibliothèque d'objets *System.Drawing*. » (voir les chapitres 1 à 5).

Contrôles de recueil de saisies

Visual Basic fournit plusieurs mécanismes pour recueillir des saisies dans un programme. Les *zones de texte* acceptent les saisies clavier, les *menus* présentent des commandes pouvant être cliquées ou sélectionnées au clavier et les *boîtes de dialogue* proposent différents éléments pouvant être sélectionnés individuellement ou dans un groupe. Dans cet exercice, vous allez apprendre à utiliser quatre contrôles importants qui permettent de recueillir des saisies dans plusieurs situations différentes. Vous allez découvrir les contrôles *RadioButton*, *CheckBox*, *ListBox* et *ComboBox*. Vous allez passer en revue chacun de ces objets à l'aide d'un programme Visual Basic appelé Saisie, interface utilisateur d'un système de commande simple et illustratif. En exécutant le programme, vous allez acquérir de l'expérience avec les objets de saisie. Dans le chapitre suivant, j'expliquerai comment ces objets peuvent être utilisés parallèlement aux menus dans un programme à part entière.

À titre d'expérience, essayez maintenant d'utiliser le contrôle *CheckBox* pour voir comment la saisie utilisateur est traitée sur un formulaire et dans le code.

Expérimenter le contrôle *CheckBox*

1. Dans le menu Fichier, cliquez sur Fermer le projet pour fermer le projet Anniversaire.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

La boîte de dialogue Nouveau projet s'affiche.

3. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Cocher Case**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans le Concepteur.

4. Dans la Boîte à outils, cliquez sur le contrôle *CheckBox*.

5. Sur le formulaire, dessinez deux objets case à cocher, l'un en dessous de l'autre.

   Les cases à cocher apparaissent sous forme d'objets dans le formulaire. Pour créer la deuxième case à cocher, cliquez une nouvelle fois sur *Checkbox* dans la Boîte à outils.

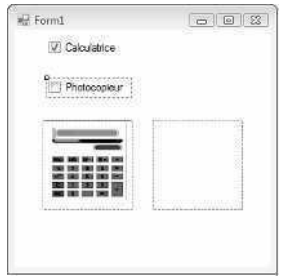
6. À l'aide du contrôle *PictureBox,* dessinez deux objets zone d'image carrés sous les deux cases à cocher.

7. Définissez les propriétés suivantes pour les objets case à cocher et zone d'image :

Objet	Propriété	Paramètre
CheckBox1	*Checked*	True
	Text	« Calculatrice »
CheckBox2	*Text*	« Photocopieur »
PictureBox1	*Image*	c:\vb08epe\chap03\calculette.bmp
	SizeMode	StretchImage
PictureBox2	*SizeMode*	StretchImage

Dans cette description, vous allez utiliser les cases à cocher pour afficher et masquer les images d'une calculatrice et d'une photocopieuse. La propriété *Text* de l'objet case à cocher détermine le contenu de l'étiquette case à cocher dans l'interface utilisateur. Avec la propriété *Checked*, vous pouvez donner une valeur par défaut à la case à cocher. Régler *Checked* sur True place une coche dans la case, et régler *Checked* sur False (paramètre par défaut) retire la coche. Les propriétés *SizeMode* dans les zones d'image permettent d'ajuster les images à la zone d'image.

Votre formulaire présente un résultat similaire à :



8. Double-cliquez sur le premier objet case à cocher pour ouvrir la procédure événementielle *CheckBox1_CheckedChanged* dans l'Éditeur de code, puis saisissez le code suivant :

```
If CheckBox1.CheckState = 1 Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
        ("c:\vb08epe\chap03\Calculette.bmp")
    PictureBox1.Visible = True
Else
    PictureBox1.Visible = False
End If
```

La procédure événementielle *CheckBox1_CheckedChanged* s'exécute seulement si l'utilisateur clique sur le premier objet case à cocher. Elle utilise une structure de décision *If... Then* (décrite au chapitre 6 « Utiliser les structures de décision ») pour confirmer le statut actuel, ou *état*, de la première case à cocher, et elle affiche l'image d'une calculette à partir du dossier c:\vb08epe\ chap03 si la case est cochée. La propriété *CheckState* prend la valeur 1 si la case est cochée et 0 si elle n'est pas cochée (vous pourriez également recourir à l'énumération *CheckState.Checked* qui apparaît dans IntelliSense lors de votre saisie, comme alternative à fixer la valeur à 1). J'utilise la propriété *Visible* pour afficher l'image si la case est cochée ou pour la masquer si la case n'est pas cochée. Remarquez que j'ai coupé la longue ligne qui charge l'image dans l'objet zone d'image à l'aide du caractère de suite (_).

9. Dans l'Explorateur de solutions, cliquez sur le bouton Concepteur de vues pour réafficher le formulaire, double-cliquez sur la deuxième case à cocher, puis ajoutez le code suivant à la procédure événementielle *CheckBox2_CheckedChanged* :

```
If CheckBox2.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
        ("c:\vb08epe\chap03\Photocop.bmp")
    PictureBox2.Visible = True
Else
    PictureBox2.Visible = False
End If
```

Cette procédure événementielle est presque identique à celle que vous venez de saisir. Seuls le nom de l'image (photocop.bmp), de l'objet case à cocher (*CheckBox2*) et de l'objet zone d'image (*PictureBox2*) changent.

10. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap03.

Exécuter le programme Cocher Case

Astuce Le programme Cocher Case complet est disponible dans le dossier c:\vb08epe\chap03\Cocher Case.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Visual Studio exécute le programme dans l'EDI. Sur le formulaire, l'image de la calculatrice apparaît dans une zone d'image et la première case est cochée.

2. Sélectionnez la case à cocher Photocopieuse.

   Visual Basic affiche l'image de la photocopieuse :

3. Essayez plusieurs combinaisons de cases à cocher, en sélectionnant ou en désélectionnant plusieurs fois les cases pour tester le programme. La logique de programmation que vous avez ajoutée avec quelques courtes lignes de code Visual Basic gère les cases à la perfection. Dans les chapitres suivants, vous découvrirez davantage de détails de programmation.
4. Cliquez sur le bouton Fermer sur le formulaire pour arrêter le programme.

La version de démonstration du programme Saisie

Maintenant que vous avez un peu d'expérience avec les cases à cocher, exécutez et examinez le programme de démonstration Saisie que j'ai créé pour simuler un environnement visuel de prise de commandes qui utilise davantage de cases à cocher, d'options, une zone de liste et une zone de liste déroulante. Si la saisie de commandes vous intéresse, vous souhaiterez peut-être transformer ce programme en un programme de saisie de commandes visuel et complet. Après avoir testé le programme Saisie, prenez le temps d'examiner le fonctionnement des quatre contrôles de saisie dans le programme. Ils ont été créés en quelques étapes à l'aide de Visual Basic et des techniques que vous venez d'apprendre.

Exécuter le programme Saisie

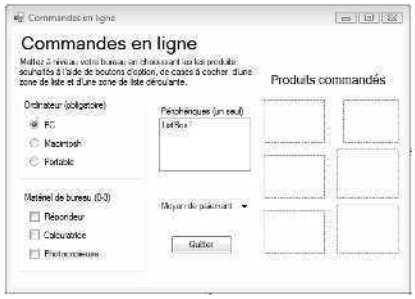
1. Dans le menu Fichier, cliquez sur Ouvrir un projet.

   La boîte de dialogue Ouvrir un projet s'affiche.
2. Ouvrez le dossier c:\vb08epe\chap03\Saisie, puis double-cliquez sur le fichier projet (Saisie.vbproj).

Comme je l'ai déjà indiqué, vous pouvez ouvrir soit le fichier projet (Saisie.vbproj) soit le fichier solution (Saisie.sln) pour ouvrir des solutions ne comportant qu'un projet. Dans tous les cas, le projet Saisie s'ouvre dans l'EDI.

3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.
4. Sur le formulaire, déplacez ou fermez les fenêtres gênantes de manière à voir la disposition des objets.

   Votre formulaire ressemble à :



Le formulaire du projet Saisie contient une option, une case à cocher, une zone de liste, une zone déroulante, un bouton et des objets étiquette. Ensemble, ces objets permettent de créer un simple programme de saisie de commandes qui illustre le fonctionnement des objets de saisie Visual Basic. Lors de son exécution, le programme Saisie charge des images à partir du dossier c:\vb08epe\chap03\Saisie et les affiche dans les six zones d'image du formulaire.

> **Remarque** Si vous avez installé les fichiers d'exercices dans un autre emplacement que le dossier par défaut c:\vb08epe, le chemin indiqué par les instructions du programme pour charger les images à partir du disque est incorrect. Comme vous allez le voir, chaque instruction commence par c:\vb08epe\chap03\Saisie. Si c'est votre cas, vous pouvez faire fonctionner le programme en renommant le dossier de vos fichiers d'exercices –vb08epe, ou en modifiant les chemins d'accès dans l'Éditeur de code à l'aide des touches d'édition ou de la commande Remplacement rapide du menu Édition.

5. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute dans l'EDI.
6. Dans la zone Ordinateur, cliquez sur la case Portable.

   L'image d'un ordinateur portable apparaît dans la zone Produits commandés située à droite du formulaire. L'utilisateur peut cliquer sur différentes options et son choix est illustré dans la zone de commande située à droite. Dans la zone Ordinateur, un groupe de cases d'options est utilisé pour recueillir la saisie de l'utilisateur.

Les *options* obligent l'utilisateur à choisir *un seul* élément dans une liste de choix. Sur un formulaire, lorsque des options sont placées dans un objet zone de groupe, elles sont considérées comme faisant partie d'un groupe et une seule option peut être sélectionnée. Pour créer une zone de groupe, cliquez sur le contrôle *GroupBox* de l'onglet Conteneurs dans la Boîte à outils, puis dessinez le contrôle sur votre formulaire (le contrôle *GroupBox* remplace le contrôle *Frame* de Visual Basic 6). Comme je l'ai fait, vous pouvez donner un titre au groupe de cases d'options en définissant la propriété *Text* de l'objet zone de groupe. Lorsque vous déplacez un objet zone de groupe sur le formulaire, les contrôles qui sont à l'intérieur se déplacent aussi.

7. Cochez les cases Répondeur, Calculatrice et Photocopieuse dans la zone Matériel de bureau.

Dans un programme, les *cases à cocher* sont utilisées pour permettre à l'utilisateur de sélectionner simultanément plusieurs options dans une liste. Cliquez pour supprimer la coche de la case calculatrice : l'image de la calculatrice disparaît de la zone de commande. Chaque élément de l'interface utilisateur répond à des événements clic : lorsqu'ils se produisent, les choix de commande sont immédiatement modifiés. Le code qui effectue ces tâches est presque identique au code que vous avez saisi précédemment dans le programme Ma Saisie.

8. Cliquez sur Antenne satellite dans la zone de liste Périphériques.

L'image d'une antenne satellite s'ajoute à la zone de commande.

Les *zones de liste* sont utilisées pour obtenir une réponse unique dans une liste de choix. Elles sont créées à l'aide du contrôle *ListBox*, et peuvent contenir plusieurs éléments au choix. Si la liste d'éléments est plus longue que la zone de liste, des barres de défilement apparaissent. Contrairement aux options, une zone de liste ne propose pas de sélection par défaut à l'utilisateur. Du point de vue de la programmation, les éléments d'une zone de liste peuvent être ajoutés, supprimés ou organisés pendant l'exécution du programme. Si vous souhaitez afficher des coches à côté des éléments de votre zone de liste, utilisez le contrôle *Checked ListBox* de la Boîte à outils à la place du contrôle *ListBox*.

9. Choisissez maintenant Dollars US (désolé, pas de crédit) dans la liste de paiements de la zone de liste déroulante Mode de paiement.

Les *zones de liste déroulantes* sont identiques aux zones de liste normales, mais occupent moins de place. Visual Basic gère automatiquement l'ouverture, la fermeture et le défilement de la zone de liste déroulante. Le développeur n'a qu'à créer la zone à l'aide du contrôle *ComboBox* de la Boîte à outils, définir la propriété *Text* pour donner des instructions ou une valeur par défaut, et écrire le code permettant d'ajouter des éléments à la zone et de traiter la sélection de l'utilisateur dans cette zone. Dans la section suivante, vous allez voir des exemples de chaque tâche dans le code de la version de démonstration du programme Saisie.

Après avoir passé vos commandes, votre écran présente un résultat similaire à l'illustration suivante :

10. Dans le programme, essayez d'apporter quelques modifications à la liste de commande (testez différents ordinateurs, périphériques et modes de paiement), puis cliquez sur le bouton Quitter pour quitter le programme.

Lorsque vous cliquez sur Quitter, le programme se ferme, et l'EDI apparaît.

Le code du programme Saisie

Bien que vous n'ayez pas encore beaucoup d'expérience de la programmation, il est important de passer en revue quelques procédures événementielles du programme Saisie pour voir comment le programme traite les informations provenant des éléments de l'interface utilisateur. Dans ces procédures, vous allez voir fonctionner les instructions *If…Then* et *Select Case*. Vous découvrirez ces structures de décision, et bien d'autres, dans le chapitre 6. Pour l'instant, concentrez-vous sur la propriété *CheckState*, qui change lorsqu'une case est cochée et sur la propriété *SelectedIndex*, qui change lorsqu'une zone de liste est sélectionnée.

Analyser le code d'une case à cocher et d'une zone de liste

1. Veillez à ce que le programme ne soit plus en cours d'exécution, puis double-cliquez sur la case à cocher Répondeur de la zone de groupe Matériel de bureau pour afficher la procédure événementielle *CheckBox1_CheckedChanged* dans l'Éditeur de code.

Le code suivant apparaît :

```
'Si la propriété CheckState d'une case à cocher prend la valeur 1, elle est cochée
If CheckBox1.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
        ("c:\vb08epe\chap03\Saisie\Repondeur.bmp")
    PictureBox2.Visible = True
Else
'S'il n'y a pas de coche, masque l'image
    PictureBox2.Visible = False
End If
```

Comme vous l'avez appris au chapitre 2, la première ligne de cette procédure événementielle est un commentaire. Les commentaires apparaissent en vert et sont de simples notes du programmeur pour décrire ce qui est important ou intéressant dans cette partie du code. Parfois, les commentaires sont générés par des outils de programmation automatisés qui compilent des programmes ou insèrent des fragments de code. J'ai écrit ce commentaire pour me rappeler que la propriété *CheckState* contient une valeur essentielle dans ce sous-programme : la valeur 1 si la première case a été cochée.

Le reste de la procédure événementielle est presque identique à celle que vous venez d'écrire dans le programme Saisie. En faisant défiler l'Éditeur de code vers le bas, vous découvrirez une procédure événementielle similaire pour les objets *CheckBox2* et *CheckBox3*.

2. Au sommet de l'Éditeur de code, cliquez sur l'onglet Form1.vb [Design] pour afficher de nouveau le formulaire, puis double-cliquez sur la zone de liste Périphériques.

La procédure événementielle *ListBox1_SelectedIndexChanged* apparaît dans l'Éditeur de code. Les instructions de programme suivantes apparaissent :

```
'L'article choisi(0-2) se trouve dans la propriété SelectedIndex
Select Case ListBox1.SelectedIndex
    Case 0
        PictureBox3.Image = System.Drawing.Image.FromFile _
            ("c:\vb08epe\chap03\Saisie\DisqueDur.bmp")
    Case 1
        PictureBox3.Image = System.Drawing.Image.FromFile _
            ("c:\vb08epe\chap03\Saisie\Imprimante.bmp")
    Case 2
        PictureBox3.Image = System.Drawing.Image.FromFile _
            ("c:\vb08epe\chap03\Saisie\Satellite.bmp")
End Select
```

Il s'agit du code qui s'exécute lorsque l'utilisateur clique sur un élément de la zone de liste Périphérique. Dans ce cas, le mot clé important est *ListBox1.SelectedIndex*, qui signifie « la propriété *SelectedIndex* de l'objet zone de liste nommé *ListBox1* ». Lorsque l'utilisateur clique sur un élément de la zone de liste, la propriété *SelectedIndex* renvoie un nombre qui correspond à l'emplacement d'un élément dans la zone de liste. Le premier élément est numéroté 0, le deuxième 1, et ainsi de suite.

Dans le code précédent, *SelectedIndex* est évaluée par la structure de décision *Select Case*, et une image différente est chargée en fonction de la valeur de la propriété *SelectedIndex*. Si la valeur est 0, l'image d'un disque dur est chargée ; si la valeur est 1, l'image d'une imprimante est chargée ; et si la valeur est 2, l'image d'une antenne satellite est chargée. Vous en apprendrez davantage sur le fonctionnement de la structure de décision *Select Case* au chapitre 6.

3. En haut de l'Éditeur de code, cliquez sur l'onglet Form1.vb [Design] pour réafficher le formulaire, puis double-cliquez sur le formulaire (et non pas sur l'un des objets) pour afficher le code qui lui est associé.

La procédure événementielle *Form1_Load* s'affiche dans l'Éditeur de code. C'est cette procédure qui est exécutée à chaque fois que le programme Saisie est chargé dans la mémoire. Les programmeurs placent des instructions de programme dans cette procédure particulière lorsqu'ils souhaitent qu'elles soient exécutées à chaque chargement d'un formulaire. Votre programme peut afficher plusieurs formulaires ou aucun, mais Visual Basic charge et exécute par défaut la procédure événementielle *Form1_Load* chaque fois que l'utilisateur exécute le programme). Souvent, comme dans le programme Saisie, ces instructions définissent un aspect de l'interface utilisateur qui ne pourrait pas être créé à l'aide des contrôles de la Boîte à outils ou de la fenêtre Propriétés.

Voici la procédure événementielle *Form1_Load* de ce programme :

```
'Ces instructions s'exécutent au chargement du formulaire
PictureBox1.Image = System.Drawing.Image.FromFile _
    ("c:\vb08epe\chap03\Saisie\PC.bmp")
'Ajoute éléments à une zone de liste comme suit :
ListBox1.Items.Add("Disque dur")
ListBox1.Items.Add("Imprimante")
ListBox1.Items.Add("Antenne satellite")
'Les zones de liste déroulantes sont également peuplées via la méthode Add :
ComboBox1.Items.Add("Dollars US")
ComboBox1.Items.Add("Chèque")
ComboBox1.Items.Add("Livres anglaises")
```

Trois des lignes de cette procédure événementielle sont des commentaires (en vert dans l'Éditeur de code). La deuxième ligne de la procédure événementielle charge l'image de l'ordinateur dans la première zone d'image. Cette ligne a été coupée à l'aide d'un espace et du caractère de suite, mais le compilateur la considère toujours comme une seule ligne. Le chargement d'une image établit le paramètre par défaut de la zone de groupe d'options Ordinateur. Remarque : le texte entre guillemets apparaît en rouge.

Les trois lignes suivantes ajoutent des éléments à la zone de liste Périphériques (*ListBox1*) dans le programme. Les termes entre guillemets simples apparaissent dans la zone de liste lorsqu'elle est à l'écran. Les éléments de la zone de liste déroulante Mode de paiement (*ComboBox1*) sont spécifiés en dessous des instructions de programme de la zone de liste. Le mot clé important de ces deux groupes est *Add*, une fonction spéciale, ou méthode, qui permet d'ajouter des éléments à des objets zone de liste et zone de groupe.

Vous en avez terminé avec le programme Saisie. Prenez le temps d'examiner toutes les autres parties du programme qui vous intéressent, puis passez à l'exercice suivant.

Astuce Comme signalé précédemment, la plupart des images de cet exemple simple sont chargées à l'aide d'un chemin d'accès absolu dans le code du programme. Cela fonctionne parfaitement tant que les images sont présentes à l'emplacement indiqué. Dans une application commerciale toutefois, vous ne pouvez toujours être certain que l'utilisateur ne va pas déplacer les fichiers de votre application, si bien qu'un programme comme celui-ci génère une erreur lorsque les fichiers dont ils se servent ne sont plus à l'emplacement prévu. Pour rendre vos applications plus fiables et plus robustes, mieux vaut généralement employer des chemins relatifs pour accéder à des images et autres ressources. Vous pouvez également incorporer ces images et ressources à votre application. Pour plus d'informations sur cette technique utile, soigneusement décrite dans les fichiers de documentation de votre Visual Studio, reportez-vous à « Comment : Créer des ressources incorporées » et « Accès aux ressources de l'application » de la documentation de Visual Studio 2008.

Aller plus loin : Utiliser le contrôle *LinkLabel*

La plupart des applications Windows fournissent désormais un accès au web. Avec Visual Studio, l'ajout de cette fonctionnalité est d'une simplicité enfantine. Vous pouvez créer un programme Visual Basic qui s'exécute à partir d'un serveur web en créant un projet Web Forms à l'aide des contrôles de la Boîte à outils optimisée pour le web. Vous pouvez également utiliser Visual Basic pour créer une application Windows qui ouvre un Navigateur Web dans l'application et fournit un accès au web tout en restant un programme Windows qui s'exécute sur un ordinateur client. Nous aborderons l'écriture des projets Web Forms plus tard dans cet ouvrage. Dans l'exercice suivant, vous allez apprendre à utiliser le contrôle *LinkLabel* de la Boîte à outils pour créer un lien dans un programme Windows permettant d'accéder à Internet *via* Internet Explorer ou le Navigateur Web par défaut de votre système.

Remarque Pour en apprendre davantage sur l'écriture d'applications Visual Basic 2008 pour le web, lisez le chapitre 20 « Créer des sites et des pages web avec Microsoft Visual Web Developper et ASP.NET ».

Créer le programme WebLink

1. Dans le menu Fichier, cliquez sur Fermer le projet pour fermer le projet Saisie.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.
3. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon WebLink**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans le Concepteur.

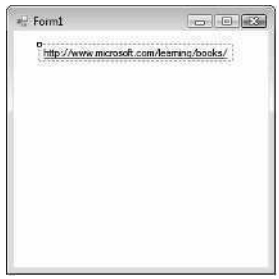
4. Dans la Boîte à outils, cliquez sur le contrôle *LinkLabel* et dessinez un objet étiquette de lien sur votre formulaire.

   Les objets étiquette de lien ressemblent aux objets étiquette, mais le texte de l'étiquette apparaît en bleu et en souligné sur le formulaire.

5. Réglez la propriété *Text* de l'objet étiquette de lien sur l'adresse URL de la page d'accueil de Microsoft Press :

   http://www.microsoft.com/learning/books/

   Votre formulaire présente un résultat similaire à ce qui suit :



6. Dans l'EDI, cliquez sur le formulaire pour le sélectionner. Cliquez sur le formulaire même, pas sur l'objet étiquette de lien.

   C'est la technique employée pour visualiser les propriétés du formulaire par défaut, Form1, dans la fenêtre Propriétés. Comme tous les objets de votre projet, le formulaire possède également des propriétés que vous pouvez définir.

7. Donnez la valeur **Test du lien** à la propriété *Text* de l'objet formulaire.

   La propriété *Text* d'un formulaire définit l'apparence de la barre de titre du formulaire lors de la conception et de l'exécution. Bien que cette personnalisation ne soit pas exclusive au web, il est préférable d'acquérir cette compétence avant de passer à d'autres projets. Nous personnaliserons la barre de titre dans la plupart des programmes construits.

8. Double-cliquez sur l'objet étiquette de lien, puis tapez le code suivant dans la procédure événementielle *LinkLabel1_LinkClicked* :

```
' Modifie la couleur du lien en définissant LinkVisited à True.
LinkLabel1.LinkVisited = True
' Utilise la méthode Process.Start pour ouvrir le navigateur par défaut
' avec l'URL Microsoft Press :
System.Diagnostics.Process.Start
    ("http://www.microsoft.com/learning/books/")
```

J'ai ajouté des commentaires dans le code pour vous entraîner à les saisir. Dès que vous avez saisi le caractère guillemet simple ('), Visual Studio modifie la couleur de la ligne en vert : il identifie la ligne comme étant un commentaire. Les commentaires ne sont là qu'à titre de documentation ; ils ne sont ni évalués, ni exécutés par le compilateur.

Les deux instructions de programme définissent le fonctionnement du lien. Fixer la propriété *LinkVisited* à *True* donne au lien sa couleur violette, qui indique dans de nombreux navigateurs que le document HTML associé au lien a déjà été visité. Bien que la définition de cette propriété ne soit pas nécessaire pour afficher une page web, c'est une bonne habitude de programmation que de fournir à l'utilisateur une information conforme aux usages.

La deuxième instruction (que j'ai scindée en deux lignes) exécute le navigateur par défaut (comme Internet Explorer) si ce n'est pas déjà le cas. Si le navigateur est ouvert, l'adresse URL se charge automatiquement. La méthode *Start* de la classe *Process* effectue le travail essentiel, en démarrant un processus ou une session de programme exécutable dans la mémoire pour le navigateur. La classe *Process*, qui gère de nombreux autres aspects de l'exécution du programme, est un membre de l'espace de noms *System.Diagnostics*. En insérant une adresse Internet ou URL parallèlement à la méthode *Start*, je fais savoir à Visual Basic que je souhaite visiter un site web. Visual Basic est assez intelligent pour savoir que le navigateur par défaut du système est l'outil nécessaire pour afficher cette URL, même si je n'ai pas identifié le navigateur par son nom.

L'une des caractéristiques intéressantes de la méthode *Process.Start* est qu'elle peut être utilisée pour exécuter d'autres applications Windows. Si je souhaite identifier un certain navigateur par son nom pour ouvrir une URL, je peux le faire à l'aide de la syntaxe suivante. Ici, je demande le navigateur Internet Explorer.

```
System.Diagnostics.Process.Start("IExplore.exe", _
    "http://www.microsoft.com/learning/books/")
```

Deux arguments séparés par une virgule sont utilisés avec la méthode *Start*. L'emplacement exact du programme IExplore.exe dans mon système n'est pas spécifié, mais Visual Basic va rechercher son chemin d'accès dans le système pendant l'exécution du programme.

Pour exécuter une autre application avec la méthode *Start* (par exemple, l'application Microsoft Word, et ouvrir le document c:\malettre.doc), je peux utiliser la syntaxe suivante :

```
System.Diagnostics.Process.Start("Winword.exe", _
    "c:\malettre.doc")
```

Comme vous le voyez, la méthode *Start* de la classe *Process* est très utile.

Maintenant que vous avez saisi le code, vous devez enregistrer votre projet. Si vous avez testé la syntaxe *Start* comme je vous l'ai montré, restaurez d'abord le code d'origine représenté au début de l'étape 8.

9. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos modifications et choisissez le dossier de destination c:\vb08epe\Chap03.

Vous pouvez à présent exécuter le programme.

Exécuter le programme WebLink

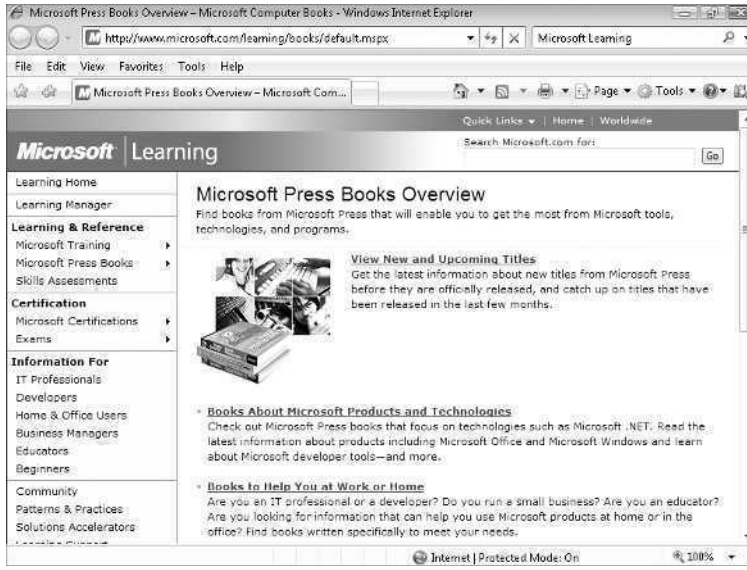
> **Astuce** Le programme WebLink complet est disponible dans le dossier c:\vb08epe\chap03\weblink.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer pour exécuter le programme WebLink.

   Le formulaire s'ouvre et s'exécute, représentant le lien et un texte de barre de titre.

2. Cliquez sur le lien pour ouvrir le site web *http://www.microsoft.com/learning/books/*.

   N'oubliez pas : ce n'est qu'une coïncidence si la propriété *Text* de l'étiquette de lien contient la même URL que le site que vous avez mentionné dans le code. Ces deux éléments ne correspondent pas obligatoirement. Vous pouvez saisir le texte de votre choix dans l'étiquette de lien. Vous pouvez également utiliser la propriété *Image* d'une étiquette de lien pour spécifier une image à afficher à l'arrière-plan de l'étiquette de lien. La figure suivante représente la page web Microsoft Press (en anglais) affichée lorsque le programme WebLink utilise Internet Explorer.



3. Affichez à nouveau le formulaire. S'il ne s'affiche pas, cliquez sur l'icône du formulaire Test du lien dans la barre des tâches Windows.

   Remarque : le lien est maintenant estompé. Comme un lien standard, votre étiquette de lien indique que le lien a été utilisé par sa couleur et son intensité (mais il reste actif).

4. Sur le formulaire, cliquez sur le bouton Fermer pour quitter l'utilitaire de test.

   Dans ce chapitre, vous avez appris à écrire du code et vous maîtrisez mieux certains contrôles de la Boîte à outils permettant de créer des applications Windows Forms. Continuons !

Rappel du chapitre 3

Pour	Faites ceci
Créer une zone de texte	Cliquez sur le contrôle *TextBox* et dessinez la zone.
Créer un bouton	Cliquez sur le contrôle *Button* et dessinez le bouton.
Modifier une propriété en cours d'exécution	Modifiez la valeur de la propriété à l'aide du code. Par exemple : `Label1.Text = "Bonjour !"`
Créer une zone d'options	Utilisez le contrôle *RadioButton*. Pour créer plusieurs zones d'options, placez plusieurs objets bouton dans une zone créée à l'aide du contrôle *GroupBox*.
Créer une case à cocher	Cliquez sur le contrôle *CheckBox* et dessinez une case à cocher.
Créer une zone de liste	Cliquez sur le contrôle *ListBox* et dessinez une zone de liste.
Créer une zone de liste déroulante	Cliquez sur le contrôle *ComboBox* et dessinez une zone de liste déroulante.
Ajouter des éléments à une zone de liste	Insérez des instructions comportant la méthode *Add* dans la procédure événementielle *Form1_Load* de votre programme. Par exemple : `ListBox1.Items.Add("Imprimante")`
Utiliser un commentaire dans le code	Tapez un guillemet simple (') dans l'Éditeur de code, puis saisissez un commentaire descriptif : il sera ignoré par le compilateur. Par exemple : `' Utilise la méthode Process.Start pour lancer IE`
Afficher une page web	Créez un lien vers la page web à l'aide du contrôle *LinkLabel*, puis ouvrez le lien dans un navigateur à l'aide de la méthode *Process.Start*.

Chapitre 4

Travailler avec les menus, les barres d'outils et les boîtes de dialogue

À la fin de ce chapitre, vous saurez :

- Ajouter des menus à vos programmes à l'aide du contrôle *MenuStrip*
- Traiter les sélections des menus et des barres d'outils à l'aide de procédures événementielles et de l'Éditeur de code
- Ajouter des barres d'outils et des boutons à l'aide du contrôle *ToolStrip*
- Utiliser les contrôles *OpenFileDialog* et *ColorDialog* pour créer des boîtes de dialogue standard
- Ajouter des touches d'accès rapide et de raccourci aux menus

Dans le chapitre 3, « Travailler avec les contrôles de la Boîte à outils », vous avez utilisé plusieurs contrôles Microsoft Visual Studio 2008 pour recueillir des informations de la part de l'utilisateur pendant l'exécution d'un programme. Dans ce chapitre, vous allez apprendre à proposer des choix à l'utilisateur en créant des menus, des barres d'outils et des boîtes de dialogue de qualité professionnelle.

Les menus sont placés dans la barre de menus, chaque menu contenant une liste de commandes. Une barre d'outils contient des boutons et d'autres outils utiles d'un programme. La plupart des commandes de menu et de barre d'outils s'exécutent dès que l'on clique dessus. Par exemple, lorsque l'utilisateur clique sur la commande Copier du menu Édition, l'information copiée est instantanément placée dans le Presse-papiers. En revanche, si une commande de menu est suivie de points de suspension (...), le fait de cliquer sur la commande affiche une boîte de dialogue demandant davantage d'informations avant d'effectuer la commande. De nombreux boutons de barres d'outils affichent également des boîtes de dialogue.

Dans ce chapitre, vous allez apprendre à utiliser les contrôles *MenuStrip* et *Toolstrip* pour donner un aspect professionnel à l'interface utilisateur de votre application. Vous allez également apprendre à traiter les commandes de menus, les boutons de barres d'outils et les options des boîtes de dialogue.

Ajouter des menus à l'aide du contrôle *MenuStrip*

Le contrôle *MenuStrip* est un outil qui permet d'ajouter des menus à votre programme. Vous pouvez le personnaliser *via* les paramètres de propriétés de la fenêtre Propriétés. *MenuStrip* permet d'ajouter de nouveaux menus, de modifier et réorganiser des menus existants et d'effacer d'anciens menus. Vous pouvez également créer automatiquement une configuration de menu standard et améliorer vos menus avec des effets spéciaux, comme les touches d'accès rapide, les coches d'activation/désactivation et les raccourcis clavier. Les menus ressemblent en tout point à ceux des applications professionnelles Microsoft Windows, mais *MenuStrip* ne crée que la partie *visible* de vos menus et commandes. Il ne vous dispense pas d'écrire les procédures événementielles qui traitent les sélections des menus et assurent le fonctionnement des commandes. Dans l'exercice suivant, vous allez commencer ce traitement à l'aide du contrôle *MenuStrip* et créer un menu Horloge comportant des commandes pour afficher la date et l'heure en cours.

Créer un menu

1. Démarrez Visual Studio.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.
3. Créez un nouveau projet Application Windows Forms appelé **Mon Menu**.
4. Dans l'onglet Menus et barres d'outils de la Boîte à outils, cliquez sur le contrôle *MenuStrip,* puis dessinez un contrôle de menu sur votre formulaire.

   Ne vous souciez pas de la position du contrôle. Visual Studio le déplace et le redimensionne automatiquement. Votre formulaire présente un résultat similaire à :

L'objet barre de menus n'apparaît pas sur votre formulaire, mais en dessous. Son fonctionnement diffère de celui de Visual Basic 6, qui affiche, d'une manière ou d'une autre, tous les objets dans le formulaire (y compris ceux qui n'ont pas de représentation visuelle lors de l'exécution du programme, comme le contrôle *Timer*). Toutefois, dans Visual Studio, les objets non visibles, comme les menus ou les horloges, figurent dans l'EDI dans une sous-fenêtre à part appelée *zone des*

composants et vous pouvez les sélectionner, définir leurs propriétés ou les effacer depuis cette sous-fenêtre.

Outre l'objet barre de menus dans la zone des composants, Visual Studio affiche une représentation visuelle du menu que vous avez créé en haut du formulaire. L'étiquette Tapez ici vous invite à cliquer sur l'étiquette et à saisir le titre de votre menu. Après avoir saisi le titre du premier menu, vous pouvez saisir les titres des sous-menus et les noms d'autres menus en appuyant sur les touches de direction et en tapant des noms supplémentaires. Mieux encore : vous pouvez revenir ultérieurement à ce Concepteur de menus en ligne et modifier votre travail ou ajouter d'autres éléments de menu (l'objet barre de menus est entièrement personnalisable et permet de créer une interface utilisateur digne des meilleures applications Windows).

5. Cliquez sur l'étiquette Tapez ici, tapez **Horloge** et appuyez sur ENTRÉE.

Le mot « Horloge » est le nom de votre premier menu et deux autres étiquettes Tapez ici apparaissent pour vous permettre de créer les commandes du menu Horloge ou d'autres titres de menu.

6. Tapez **Date** afin de créer une commande Date dans le menu Horloge, puis appuyez sur ENTRÉE.

Visual Studio ajoute la commande Date au menu et sélectionne l'élément suivant.

7. Tapez **Heure** afin de créer une commande Heure et appuyez sur ENTRÉE.

Votre menu Horloge possède désormais deux commandes, Date et Heure. Vous pourriez ajouter d'autres menus ou commandes, mais ce n'est pas nécessaire pour cet exemple de programme. Votre formulaire présente un résultat similaire à :

8. Cliquez sur le formulaire pour fermer le Concepteur de menus.

Le Concepteur de menus se ferme et votre formulaire s'ouvre dans l'EDI avec un nouveau menu Horloge.

Vous êtes prêt à personnaliser le menu.

Ajouter des touches d'accès rapide aux commandes de menu

La plupart des applications permettent d'accéder aux commandes de menu et de les exécuter à l'aide du clavier. Par exemple, dans Visual Studio, vous pouvez ouvrir le menu Fichier en appuyant sur la touche ALT puis sur la touche F. Une fois le menu Fichier ouvert, vous pouvez ouvrir un projet en appuyant sur la touche P. Cette combinaison de touches sert à exécuter la commande correspondante du menu ouvert. Elle porte le nom de *touche d'accès rapide*. Vous pouvez identifier la touche d'accès rapide d'une commande de menu à son soulignement.

Visual Studio permet de créer facilement ces touches d'accès rapide. Pour ce faire, activez le Concepteur de menus puis tapez une esperluette (&) avant la lettre choisie dans le nom de la commande. Lorsque vous cliquez sur la commande pendant l'exécution du programme, votre programme reconnaît automatiquement la touche d'accès rapide. Essayez maintenant d'ajouter des touches d'accès rapide au menu Horloge.

Conventions des menus

Par convention, la première lettre de chaque nom de menu et de commande d'une application Windows est une majuscule. Les menus Fichier et Édition sont souvent les deux premiers de la barre de menus, et Aide est habituellement le dernier. Affichage, Format et Fenêtre sont d'autres menus courants. Peu importent les menus que vous utilisez dans vos applications, veillez seulement à être clair et cohérent. Menus et commandes doivent être simples d'utilisation et ressembler autant que possible à ceux des autres applications Windows. Lorsque vous créez des éléments de menu, suivez ces conseils :

- Utilisez des légendes courtes, spécifiques, d'un ou deux mots maximum.
- Assignez une touche d'accès rapide à chaque élément de menu. Utilisez si possible la première lettre de l'élément ou la touche d'accès rapide habituelle (comme Q pour Quitter).
- Les éléments de menu de même niveau doivent avoir chacun une touche d'accès rapide.
- Si une commande fonctionne en mode activation/désactivation, placez une coche à gauche de l'élément lorsqu'il est actif. Pour ce faire, ajoutez une coche dans la fenêtre Propriétés en réglant la propriété Checked de la commande de menu sur True.
- Placez des points de suspension (...) après une commande de menu qui va nécessiter une saisie utilisateur pour être exécutée. Les points de suspension indiquent qu'une boîte de dialogue s'ouvrira lorsque l'utilisateur sélectionnera cet élément.

Remarque Par défaut, la plupart des versions de Windows n'affichent pas le soulignement des touches d'accès rapide dans un programme tant que vous n'avez pas appuyé sur la touche ALT. Avec Windows 2000, vous pouvez désactiver cette option à partir de l'icône Affichage du Panneau de configuration, dans l'onglet Effets. Avec Windows XP et Windows Server 2003, vous pouvez désactiver cette option depuis les Propriétés de Affichage du Panneau de configuration, dans l'onglet Apparence. Avec Windows Vista, vous le faites en cliquant dans le Panneau de configuration sur Apparence et personnalisation, puis en sélectionnant dans Options d'ergonomie l'option Souligner les raccourcis clavier et les touches d'accès rapide. Cochez ou décochez ensuite les options adéquates.

Essayons maintenant d'ajouter des touches d'accès rapide au menu Horloge.

Ajouter des touches d'accès rapide

1. Sur le formulaire, cliquez sur le nom de menu Horloge, attendez un moment, puis cliquez de nouveau dessus.

   Le nom du menu est mis en surbrillance et un pointeur en I clignotant (le curseur d'édition) apparaît à la fin de la sélection. À l'aide du pointeur en I, vous pouvez modifier le nom du menu ou ajouter l'esperluette (&) désignant la touche d'accès rapide. Si vous avez double-cliqué sur le nom du menu, l'Éditeur de code s'est peut-être ouvert. Si tel est le cas, fermez-le et réitérez l'étape 1.

2. Appuyez cinq fois sur la flèche de direction gauche pour positionner la barre en I juste devant le nom du menu Horloge.

   La barre en I clignote devant la lettre H de Horloge.

3. Tapez **&** pour faire de la lettre H la touche d'accès rapide du menu Horloge.

   Dans la zone de texte, une esperluette apparaît avant le mot Horloge.

4. Dans le menu, cliquez une première fois sur la commande Date, puis une seconde fois pour faire apparaître la barre en I.

5. Tapez **&** devant la lettre D.

   La lettre D est désormais la touche d'accès rapide de la commande Date.

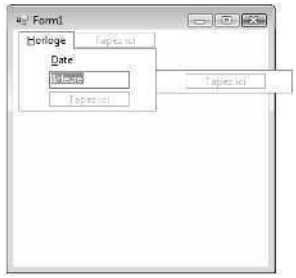
6. Dans le menu, cliquez une première fois sur la commande Heure, puis une seconde fois pour faire apparaître la barre en I.

7. Tapez **&** devant la lettre H.

   La lettre H est désormais la touche d'accès rapide de la commande Heure.

8. Appuyez sur Entrée.

   Vos modifications sont enregistrées. Voici à quoi ressemble votre formulaire :



Vous allez maintenant utiliser le Concepteur de menus pour modifier l'ordre des commandes Date et Heure du menu Horloge. Il est important de savoir modifier l'ordre des éléments de menu, car vous serez bientôt amené à mieux définir vos menus.

Modifier l'ordre des éléments de menu

1. Sur le formulaire, cliquez sur le menu Horloge pour afficher les éléments du menu.

   Pour modifier l'ordre d'un élément de menu, faites-le simplement glisser vers un nouvel emplacement dans le menu. Essayez.

2. Faites glisser la commande Heure au-dessus de la commande Date, puis relâchez le bouton de la souris.

   En faisant glisser un élément de menu au-dessus d'un autre, vous placez le premier élément avant le deuxième. En un rien de temps, Visual Studio a déplacé la commande Heure au-dessus de la commande Date.

Vous avez créé l'interface utilisateur du menu Horloge. Vous allez maintenant utiliser les procédures événementielles de menu pour traiter les sélections de l'utilisateur par programmation.

Remarque Pour effacer un élément d'un menu, cliquez sur l'élément indésirable, puis appuyez sur la touche SUPPR (souvenez-vous que Visual Studio possède également une commande Annuler, située à la fois dans le menu Édition et dans la barre d'outils Standard : vous pouvez tester cette fonction et annuler l'effacement).

Traiter les choix dans le menu

Une fois les menus et commandes configurés à l'aide de l'objet barre de menus, ils deviennent eux aussi des objets du programme. Pour que les objets menu fonctionnent, vous devez écrire des procédures événementielles les concernant. Les procédures événementielles de menu contiennent généralement des instructions de programme affichant ou traitant des informations sur le formulaire de l'interface utilisateur, et modifiant une ou plusieurs propriétés de menu. Si davantage d'informations sont requises de la part l'utilisateur pour traiter la commande sélectionnée, vous pouvez écrire votre procédure événementielle de manière à ce qu'elle affiche une boîte de dialogue ou l'un des contrôles de saisie utilisés au chapitre 3.

Dans l'exercice suivant, vous allez ajouter un objet étiquette à votre formulaire pour afficher le résultat des commandes Heure et Date du menu Horloge.

Ajouter un objet étiquette au formulaire

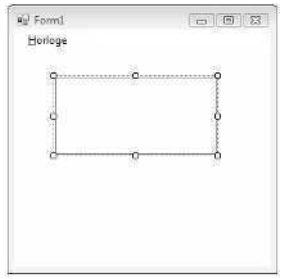
1. Dans la Boîte à outils, cliquez sur le contrôle *Label*.
2. Créez une étiquette au centre du formulaire.

   L'objet étiquette apparaît sur le formulaire et porte le nom de *Label1* dans le code.
3. Définissez les propriétés d'étiquette suivantes :

Objet	Propriété	Paramètre
Label1	*AutoSize*	False
	BorderStyle	FixedSingle
	Font	Microsoft Sans Serif, Gras, 14 points
	Text	(vide)
	TextAlign	MiddleCenter

4. Agrandissez l'objet étiquette (il devra contenir les valeurs d'heure et de date) et placez-le au centre du formulaire.

   Voici à quoi doit ressembler votre formulaire :



Vous allez maintenant ajouter des instructions aux procédures événementielles de Heure et Date pour traiter les commandes de menu.

Remarque Dans l'exercice suivant, vous allez saisir du code pour traiter les choix de menu. Si le fonctionnement du code est encore un peu flou, ne vous en faites pas, vous allez en apprendre davantage sur les instructions dans les chapitres 5 à 7.

Modifier les procédures événementielles de menu

1. Sur le formulaire, cliquez sur le menu Horloge pour en afficher les commandes.
2. Dans le menu, double-cliquez sur la commande Heure pour ouvrir une procédure événementielle dans l'Éditeur de code.

   La procédure événementielle *HeureToolStripMenuItem_Click* s'affiche dans l'Éditeur de code. Le nom *HeureToolStripMenuItem_Click* inclut le nom *Heure* que vous avez donné à cette commande de menu. Les mots *ToolStripMenuItem* indiquent que le contrôle *MenuStrip* est lié au contrôle *ToolStrip* dans la technologie d'arrière-plan. (Nous verrons plusieurs exemples plus loin dans ce chapitre.) La syntaxe *_Click* signifie que la procédure événementielle s'exécute lorsque l'utilisateur clique sur l'élément du menu.

   Nous allons conserver ce nom de menu pour le moment, mais si vous souhaitez donner vos propres noms aux objets du menu, sélectionnez l'objet, ouvrez la fenêtre Propriétés et modifiez sa propriété *Name*. Je ne vais pas vous ennuyer avec cette étape supplémentaire dans ce chapitre, mais plus loin dans cet ouvrage, vous serez amené à renommer des objets dans votre programme pour les rendre conforme aux usages de la programmation professionnelle.
3. Tapez l'instruction suivante :

```
Label1.Text = TimeString
```

   Cette instruction affiche l'heure courante (à partir de l'horloge système) dans la propriété *Text* de l'objet *Label1* et remplace (le cas échéant) l'ancien texte *Label1*. La propriété *TimeString* contient l'heure courante, formatée pour l'affichage ou l'impression. Vous pouvez utiliser *TimeString* à tout moment dans vos programmes pour afficher l'heure exacte à la seconde près (elle remplace l'ancienne instruction *TIME$* de Visual Basic).

Remarque La propriété *TimeString* de Visual Basic renvoie l'heure système en cours. Vous pouvez définir celle-ci à l'aide de l'icône Date/Heure du Panneau de configuration Windows. Vous pouvez en modifier le format à l'aide de l'icône Options régionales (ou Options régionales et linguistiques) du Panneau de configuration.

4. Appuyez sur la touche ENTRÉE.

   Visual Basic interprète la ligne et adapte si nécessaire l'emploi des majuscules et l'espacement.

   Il vérifie les erreurs de syntaxe de chaque ligne saisie.

Astuce Vous pouvez saisir une ligne en appuyant sur ENTRÉE ou sur ECHAP.

5. Dans l'Explorateur de solutions, cliquez sur le bouton du Concepteur de vues, puis double-cliquez sur la commande Date du menu Horloge.

   La procédure événementielle *DateToolStripMenuItem_Click* s'affiche dans l'Éditeur de code. Cette procédure événementielle s'exécute lorsque l'utilisateur clique sur la commande Date du menu Horloge.

6. Tapez l'instruction suivante :

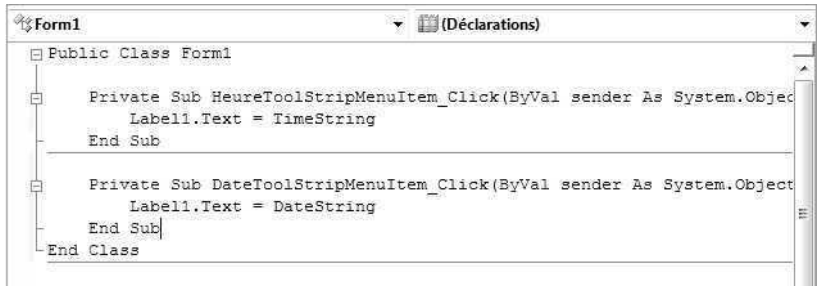
```
Label1.Text = DateString
```

   Cette instruction affiche la date courante (à partir de l'horloge système) dans la propriété *Text* de l'objet *Label1* et remplace l'ancien texte *Label1*. La propriété *DateString* a également un usage général dans vos programmes. Affectez *Date String* à la propriété *Text* d'un objet lorsque vous souhaitez afficher la date du jour sur un formulaire.

Remarque La propriété *DateString* de Visual Basic renvoie la date système en cours. Vous pouvez la définir à l'aide de la catégorie Horloge, Langue et Région du Panneau de configuration de Windows Vista.

7. Appuyez sur la touche ENTRÉE.

   Votre écran ressemble à l'illustration suivante :



Vous avez terminé la saisie du programme de démonstration des menus. Vous allez maintenant enregistrer les modifications apportées au projet et l'exécuter.

8. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout et choisissez le dossier de destination c:\vb08epe\chap04.

Exécuter le programme Menu

Astuce Le programme Menu complet est disponible dans le dossier c:\vb08epe\chap04\Menu.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme Menu s'exécute dans l'EDI.

2. Dans la barre de menus, cliquez sur le menu Horloge.

   Le menu Horloge apparaît.

3. Cliquez sur la commande Heure.

   L'heure système courante apparaît dans la zone d'étiquette, comme suit :

   Vous allez maintenant essayer d'afficher la date du jour à l'aide des touches d'accès rapide du menu.

4. Appuyez sur la touche ALT, puis relâchez-la et appuyez sur la touche H.

   Le menu Horloge s'ouvre et le premier élément est mis en surbrillance.

5. Appuyez sur D pour afficher la date du jour.

   La date du jour apparaît dans la zone d'étiquette.

6. Cliquez sur le bouton Fermer de la barre de titre pour arrêter le programme.

Félicitations ! Vous avez créé un programme qui utilise les menus et les touches d'accès rapide. Dans l'exercice suivant, vous allez apprendre à utiliser les barres d'outils.

Propriétés et fonctions de l'horloge système

Vous pouvez utiliser différentes propriétés et fonctions pour recueillir des valeurs chronologiques à partir de l'horloge système et créer des calendriers personnalisés, des horloges et des alarmes dans vos programmes. Le tableau suivant énumère les fonctions les plus utiles de l'horloge système. Pour davantage d'informations, consultez la documentation de Visual Studio.

Propriété ou fonction	Description
TimeString	Cette propriété définit l'heure courante ou la récupère à partir de l'horloge système.
DateString	Cette propriété définit la date du jour ou la récupère à partir de l'horloge système.
Now	Cette propriété renvoie une valeur codée représentant la date et l'heure courantes. Elle est très utile en tant qu'argument pour d'autres fonctions de l'heure système.
Hour (date)	Cette fonction extrait la partie heures de la valeur date/heure spécifiée (0 à 23).
Minute (date)	Cette fonction extrait la partie minutes de la valeur date/heure spécifiée (0 à 59).
Second (date)	Cette fonction extrait la partie secondes de la valeur date/heure spécifiée (0 à 59).
Month (date)	Cette fonction extrait un nombre entier représentant le mois (1 à 12).
Year (date)	Cette fonction extrait la partie années de la valeur date/heure spécifiée.
Weekday (date)	Cette fonction extrait un nombre entier représentant le jour de la semaine (1 pour dimanche, 2 pour lundi, etc.).

Ajouter des barres d'outils à l'aide du contrôle *ToolStrip*

Parallèlement au contrôle *MenuStrip*, le contrôle Visual Studio *ToolStrip* permet d'ajouter rapidement des barres d'outils à l'interface utilisateur de votre programme. Tout comme le contrôle *MenuStrip*, le contrôle *ToolStrip* est placé dans un formulaire Visual Basic, mais il est situé dans la zone des composants de l'EDI. Vous pouvez ajouter de nombreuses caractéristiques à vos barres d'outils, comme les étiquettes, les zones combinées, les zones de texte, et les boutons séparateurs. Les barres d'outils sont très bien conçues, mais n'oubliez pas que, comme pour les commandes de menu, vous devez écrire dans votre programme une procédure événementielle pour chaque bouton que vous souhaitez utiliser. Il n'en reste pas moins que ce que réalise pour vous l'EDI en matière de programmation et de configuration des barres d'outils est incomparable aux anciennes versions de Visual Basic. Essayez maintenant de créer une barre d'outils.

Créer une barre d'outils

1. Dans l'onglet Menus et barres d'outils de la Boîte à outils, cliquez sur le contrôle *ToolStrip,* puis dessinez un contrôle de barre d'outils sur votre formulaire.

   Ne vous souciez pas de la position de la barre d'outils : Visual Studio la crée automatiquement sur votre formulaire et l'étire sur la largeur de la fenêtre. L'objet barre d'outils lui-même apparaît en dessous du formulaire, dans la zone des composants. Sur le formulaire, la barre d'outils par défaut comporte un seul bouton. Vous allez maintenant utiliser un raccourci spécial pour remplir automatiquement la barre d'outils.

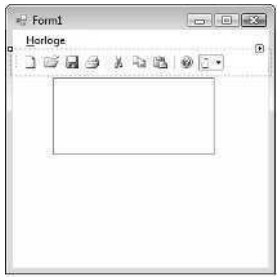
2. Cliquez sur la petite flèche de raccourci dans le coin supérieur droit de la nouvelle barre d'outils.

   La flèche de raccourci pointe vers la droite et ressemble à la flèche de raccourci que nous avons vue avec le contrôle *PictureBox*, dans le chapitre 2, « Écrire son premier programme ». Lorsque vous cliquez sur la flèche, une fenêtre comportant quelques-unes des tâches et des propriétés habituelles des barres d'outils s'ouvre.

   Ces commandes permettent de configurer rapidement la barre d'outils.

3. Cliquez sur Insérer des éléments standard.

   Visual Studio ajoute un ensemble de boutons de barre d'outils Standard, dont Nouveau, Ouvrir, Enregistrer, Imprimer, Couper, Copier, Coller et Aide. Voici à quoi ressemble votre formulaire :



Il n'est pas nécessaire de commencer avec une barre d'outils complète comme je l'ai fait ici. Il ne s'agit que d'une démonstration de l'une des fonctions « automatiques » de Visual Studio 2008. Rien ne vous empêche de créer les boutons de votre barre d'outils un par un, à l'aide des commandes d'édition *ToolStrip*, comme nous allons le voir. Cependant, pour de nombreuses applications, la fonction Insérer des éléments standards permet de gagner du temps. N'oubliez pas : bien que ces boutons aient un aspect professionnel, ils ne fonctionnent pas encore : il manque leurs procédures événementielles.

4. Cliquez sur le bouton Ajouter ToolStripButton à droite de la barre d'outils, puis cliquez sur l'élément Button.

   Cette commande ajoute des éléments à votre barre d'outils : boutons, étiquettes, boutons séparateurs, zones de texte, zones combinées et autres éléments de l'interface utilisateur. Vous avez créé un bouton de barre d'outils personnalisé. Par défaut, il contient l'image d'une montagne et d'un soleil.

5. Élargissez la fenêtre du formulaire pour vérifier que vous pouvez voir tous les éléments de barre d'outils.

6. Effectuez un clic droit sur le nouveau bouton, pointez sur DisplayStyle, puis cliquez sur ImageAndText.

   Sur la barre d'outils, votre nouveau bouton affiche désormais à la fois du texte et une image. Dans le programme, Visual Studio appelle votre nouveau bouton *ToolStripButton1*, et ce nom apparaît par défaut sur la barre d'outils. Élargissez si nécessaire la fenêtre du formulaire pour voir le nouveau bouton, car il contient la valeur texte par défaut ToolStripButton1.

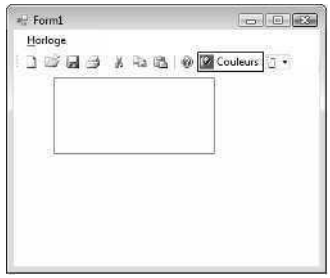
7. Sélectionnez l'objet *ToolStripButton1*.

8. Fixez sa propriété *Text* à Couleurs, qui est le nom de votre bouton sur le formulaire, puis appuyez sur ENTRÉE.

   Le bouton Couleurs apparaît dans la barre d'outils. Vous utiliserez ce bouton plus tard, pour modifier la couleur du texte du formulaire dans le programme. Insérez maintenant une image personnalisée pour votre bouton.

9. Effectuez un clic droit sur le bouton Couleurs, puis cliquez sur la commande Définir l'image.
10. Sélectionnez l'option Ressource locale (si elle ne l'est pas déjà), puis cliquez sur le bouton Importer.
11. Parcourez l'arborescence jusqu'au dossier c:\vb08epe\chap04, cliquez sur le fichier image ColorButton que j'ai créé pour vous, cliquez sur Ouvrir, puis sur OK.

    Visual Studio charge l'icône rose, bleue et jaune dans le bouton Couleurs, comme suit :



Votre nouveau bouton est terminé, et vous avez appris à ajouter vos propres boutons à la barre d'outils, en plus des éléments par défaut fournis par Visual Studio. Vous allez maintenant apprendre à supprimer et à réorganiser des boutons de barre d'outils.

Déplacer et supprimer des boutons de barre d'outils

1. Faites glisser le nouveau bouton Couleurs sur le côté gauche de la barre d'outils.

   Visual Studio vous permet de réorganiser vos boutons de barre d'outils à l'aide de simples mouvements de glisser-déposer.
2. Effectuez un clic droit sur le deuxième bouton de la barre d'outils (Nouveau), puis choisissez la commande Supprimer.

   Le bouton Nouveau est supprimé de la barre d'outils. À l'aide de la commande Supprimer, vous pouvez supprimer des boutons indésirables. Elle permet de personnaliser facilement les boutons de barre d'outils standards fournis par le contrôle *ToolStrip*.
3. Supprimez les boutons Enregistrer et Imprimer, mais conservez les boutons Couleurs et Ouvrir.

   Vous allez maintenant apprendre à utiliser les contrôles de la boîte de dialogue et à les lier à des boutons de barre d'outils.

Utiliser des contrôles de boîte de dialogue

Les onglets Boîtes de dialogue et Impression de la Boîte à outils de Visual Studio comportent huit contrôles standards de boîte de dialogue. Ces dernières sont prêtes à l'emploi, inutile de créer vos boîtes de dialogue personnalisées pour les tâches les plus courantes dans les applications Windows, comme l'ouverture, l'enregistrement et l'impression de fichiers. La plupart du temps, vous devrez écrire le code de procédure événementielle qui relie ces boîtes de dialogue à votre programme, mais les interfaces utilisateur ont été générées pour vous et appliquent les normes des applications Windows.

Le tableau suivant répertorie les huit contrôles de boîte de dialogue standards disponibles. À quelques exceptions près, ils sont similaires aux objets fournis par le contrôle *CommonDialog* dans Visual Basic 6. Le contrôle *PrintPreviewControl* n'est pas cité ici, mais vous en aurez l'utilité si vous utilisez le contrôle *PrintPreviewDialog*.

Nom du contrôle	Fonction
OpenFileDialog	Récupère le lecteur, le nom du dossier et du fichier pour un fichier existant
SaveFileDialog	Récupère le lecteur, le nom du dossier et du fichier pour un nouveau fichier
FontDialog	Permet à l'utilisateur de choisir une nouvelle police et un nouveau style
ColorDialog	Permet à l'utilisateur de choisir une couleur sur une palette
FolderBrowserDialog	Permet à l'utilisateur de parcourir l'arborescence des dossiers d'un ordinateur
PrintDialog	Permet à l'utilisateur de définir les options d'impression
PrintPreviewDialog	Affiche une boîte de dialogue d'aperçu avant impression comme le programme Microsoft Word
PageSetupDialog	Permet à l'utilisateur de contrôler les options de mise en page, comme les marges, la taille du papier et la disposition

Dans les exercices suivants, vous allez utiliser les contrôles *OpenFileDialog* et *ColorDialog*. Avec le premier, votre programme ouvre des fichiers image bitmap et avec le second, il modifie la couleur de l'horloge. Vous allez relier ces boîtes de dialogue à la barre d'outils que vous venez de créer, mais vous pourriez tout aussi facilement les relier aux commandes de menu.

Ajouter des contrôles *OpenFileDialog* et *ColorDialog*

1. Dans l'onglet Boîtes de dialogue de la Boîte à outils, cliquez sur le contrôle *OpenFileDialog*, puis sur le formulaire.

   Un objet Boîte de dialogue Ouvrir apparaît dans la zone des composants.

2. Dans l'onglet Boîtes de dialogue de la Boîte à outils, cliquez sur le contrôle *ColorDialog*, puis cliquez de nouveau sur le formulaire.

   La fenêtre de composants ressemble désormais à :

Tout comme les objets barre de menus et barre d'outils, les objets Boîte de dialogue Ouvrir et Boîte de dialogue Couleurs apparaissent dans la fenêtre de composants, et peuvent être personnalisés à l'aide des paramètres de propriétés.

Vous allez maintenant créer un objet zone d'image à l'aide du contrôle *PictureBox*. Comme vous l'avez vu, l'objet zone d'image affiche une image sur le formulaire. Cette fois-ci, vous allez utiliser l'objet Boîte de dialogue Ouvrir pour afficher une image dans la zone d'image.

Ajouter un objet zone d'image

1. Dans la Boîte à outils, cliquez sur le contrôle *PictureBox*.
2. Dessinez un objet zone d'image sur le formulaire, en dessous de l'étiquette.
3. Utilisez la flèche de raccourci dans l'objet zone d'image pour régler la propriété *SizeMode* de la zone d'image sur StretchImage.

Vous allez maintenant créer des procédures événementielles pour les boutons Couleurs et Ouvrir de la barre d'outils.

Procédures événementielles qui gèrent les boîtes de dialogue courantes

Après avoir créé un objet boîte de dialogue, voici comment l'afficher dans un programme :

- Saisissez le nom de la boîte de dialogue à l'aide de la méthode *ShowDialog* dans une procédure événementielle associée à un bouton de barre d'outils ou à une commande de menu.
- Si nécessaire, définissez une ou plusieurs propriétés de la boîte de dialogue dans le code avant d'ouvrir la boîte de dialogue.
- Utilisez le code pour répondre aux sélections de l'utilisateur dans la boîte de dialogue après la manipulation et la fermeture de celle-ci.

Dans l'exercice suivant, vous allez saisir le code de la procédure événementielle *OuvrirToolStripButton_Click*, le sous-programme qui s'exécute lors d'un clic sur la commande Ouvrir. Vous allez définir la propriété *Filter* de l'objet *OpenFileDialog1* pour déterminer le type de fichier de la boîte de dialogue Ouvrir habituelle (vous spécifierez images bitmap). Ensuite, vous utiliserez la méthode *ShowDialog* pour afficher la boîte de dialo-

gue Ouvrir. Une fois que l'utilisateur a sélectionné un fichier et refermé cette boîte de dialogue, vous afficherez le fichier sélectionné dans une zone d'image en positionnant la propriété *Image* de l'objet zone d'image sur le nom du fichier sélectionné par l'utilisateur.

Éditer la procédure événementielle du bouton Ouvrir

1. Dans la barre d'outils de votre formulaire, double-cliquez sur le bouton Ouvrir.

   La procédure événementielle *OuvrirToolStripButton_Click* s'affiche dans l'Éditeur de code.

2. Tapez le code suivant dans la procédure événementielle. Veillez à reproduire exactement chaque ligne et appuyez sur la touche ENTRÉE après chaque ligne.

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
        (OpenFileDialog1.FileName)
End If
```

   Les trois premières instructions de la procédure événementielle se rapportent aux trois différentes parties de l'objet Boîte de dialogue Ouvrir. La première instruction utilise la propriété *Filter* pour établir une liste de fichiers valables (dans le cas présent, la liste ne contient qu'un élément : *.bmp). Cette précision est importante pour les boîtes de dialogue Ouvrir, car un objet zone d'image peut présenter de nombreux types de fichier, dont :

   - Les images bitmap (fichiers .bmp)
   - Les métafichiers Windows (fichiers .emf et .wmf)
   - Les icônes (fichiers .ico)
   - Le format JPEG (fichiers .jpg et .jpeg)
   - Le format PNG (fichiers .png)
   - Le format GIF (fichiers .gif)

   Pour ajouter d'autres éléments à la liste *Filter*, saisissez une barre verticale (|) entre les éléments. Par exemple, cette instruction

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp|Metafiles (*.wmf)|*.wmf"
```

   permet de choisir à la fois les images bitmap et les métafichiers Windows dans la boîte de dialogue Ouvrir.

   La deuxième instruction de la procédure événementielle affiche la boîte de dialogue Ouvrir dans le programme. *ShowDialog* ressemble à la méthode *Show* dans Visual Basic 6, mais elle peut être utilisée avec tout type de formulaire Windows. La méthode *ShowDialog* renvoie un résultat appelé *DialogResult*, qui indique sur quel bouton de la boîte de dialogue l'utilisateur a cliqué. Pour déterminer s'il a cliqué sur le bouton Ouvrir, on utilise une structure de décision *If…Then* qui vérifie que le

résultat renvoyé est bien *DialogResult.OK*. Si tel est le cas, un chemin d'accès à un fichier .bmp valable doit être enregistré dans la propriété *FileName* de l'objet OpenFileDialog. Vous en apprendrez davantage sur la syntaxe des structures de décision *If…Then* dans le chapitre 6, « Utiliser des structures de décision ».)

La troisième instruction utilise le nom du fichier sélectionné par l'utilisateur dans la boîte de dialogue. Lorsque l'utilisateur sélectionne un lecteur, un dossier et un nom de fichier, puis clique sur Ouvrir, le chemin d'accès complet est transféré dans le programme *via* la propriété *OpenFileDialog1.FileName*. Ensuite, on utilise la méthode *System.Drawing.Image.FromFile*, qui charge une image électronique, pour copier l'image bitmap spécifiée dans l'objet zone d'image (vu sa longueur, j'ai coupé cette instruction avec un caractère de continuation de ligne (_)).

Vous allez maintenant écrire une procédure événementielle pour le bouton Couleurs que vous avez ajouté à la barre d'outils.

Écrire la procédure événementielle du bouton Couleurs

1. Affichez de nouveau le formulaire, puis dans la barre d'outils que vous avez ajoutée au formulaire, double-cliquez sur le bouton Couleurs.

   Une procédure événementielle appelée *ToolStripButton1_Click* apparaît dans l'Éditeur de code. Le nom de l'objet contient *Button1* parce qu'il a été le premier bouton ajouté à la barre d'outils standards (vous pouvez modifier le nom de cet objet et l'appeler par exemple *ToolStripButtonCouleurs*, en cliquant sur le bouton dans le formulaire et en modifiant la propriété *Name* dans la fenêtre Propriétés).

2. Tapez les instructions suivantes dans la procédure événementielle :

   ```
   ColorDialog1.ShowDialog()
   Label1.ForeColor = ColorDialog1.Color
   ```

   La première instruction fait appel à la méthode *ShowDialog* pour ouvrir la boîte de dialogue Couleurs. Comme vous l'avez appris précédemment dans ce chapitre, *ShowDialog* est la méthode utilisée pour ouvrir tout formulaire sous forme de boîte de dialogue, y compris les formulaires créés à l'aide de l'un des contrôles de boîte de dialogue standards fournis par Visual Studio. La deuxième instruction de la procédure événementielle assigne la couleur sélectionnée par l'utilisateur dans la boîte de dialogue à la propriété *ForeColor* de l'objet *Label1*. Souvenez-vous, *Label1* est la zone d'étiquette que vous avez utilisée pour afficher l'heure et la date courantes sur le formulaire. Vous allez utiliser la couleur retournée par la boîte de dialogue Couleurs pour définir la couleur du texte dans l'étiquette.

   La boîte de dialogue Couleurs peut servir à définir la couleur de tout élément de l'interface utilisateur qui comporte une couleur : formulaire, ombres sur le formulaire, premier plan et arrière-plan.

3. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

Contrôler le choix des couleurs en définissant les propriétés de la boîte de dialogue Couleurs

Pour personnaliser davantage la boîte de dialogue Couleurs, vous pouvez contrôler les choix de couleurs proposés à l'utilisateur à l'ouverture de la boîte de dialogue en ajustant les paramètres de couleur dans la fenêtre Propriétés ou en définissant les propriétés dans le code avant d'afficher la boîte de dialogue à l'aide de la méthode *ShowDialog*. Le tableau suivant décrit les propriétés les plus utiles du contrôle *Color Dialog*. Chaque propriété doit être positionnée sur *True* pour activer l'option ou sur *False* pour la désactiver.

Propriété	Signification
AllowFullOpen	Positionnée sur *True* pour activer le bouton Définir les couleurs personnalisées dans la boîte de dialogue.
AnyColor	Positionnée sur *True* si l'utilisateur peut sélectionner toutes les couleurs dans la boîte de dialogue.
FullOpen	Positionnée sur *True* pour afficher la zone Couleurs personnalisées la première fois que la boîte de dialogue s'ouvre.
ShowHelp	Positionnée sur *True* pour activer le bouton d'aide dans la boîte de dialogue.
SolidColorOnly	Positionnée sur *True* si vous souhaitez que l'utilisateur ne sélectionne que des couleurs unies (les couleurs dégradées, composées de pixels de différentes couleurs, sont désactivées).

Vous allez maintenant exécuter le programme Menu et tester les menus et les boîtes de dialogue que vous avez créés.

Exécuter le programme Menu

Astuce Le programme Menu complet est disponible dans le dossier c:\vb08epe\chap04\Menu.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute, le menu Horloge et la barre d'outils apparaissent dans la partie supérieure de l'écran.

2. Cliquez sur Ouvrir dans la barre d'outils du formulaire.

   La boîte de dialogue Ouvrir apparaît. Comment la trouvez-vous ? Remarquez l'entrée Bitmaps (*.bmp) dans la zone Fichiers de type. Vous avez défini cette entrée *via* l'instruction

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
```

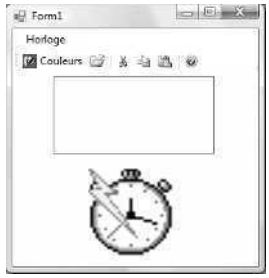
dans la procédure événementielle *OuvrirToolStripButton_Click*. La première partie du texte entre guillemets (Bitmaps (*.bmp)) détermine quels éléments sont énumérés dans la zone Fichiers de type. La deuxième partie (*.bmp) détermine l'extension des fichiers qui apparaîtront dans la boîte de dialogue.

3. Ouvrez sur votre système un dossier qui contient des images bipmap. Je me suis servi d'un dossier créé pour l'occasion, situé dans Images publiques et qui renferme des icônes au format bmp.

4. Sélectionnez l'un des fichiers bitmap, puis cliquez sur le bouton Ouvrir.

   La représentation de l'image bitmap apparaît dans la zone d'image (j'ai sélectionné le fichier Clock.bmp.)

   Voici à quoi ressemble votre formulaire :



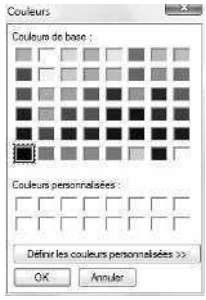
   Testons à présent le menu Horloge.

5. Dans le menu Horloge, cliquez sur la commande Heure.

   L'heure courante apparaît dans la zone d'étiquette.

6. Dans la barre d'outils, cliquez sur le bouton Couleurs.

La boîte de dialogue Couleurs s'affiche :



La boîte de dialogue Couleurs comporte des éléments permettant de modifier la couleur du texte Horloge dans votre programme. Le paramètre de couleur actuel, noir, est sélectionné.

7. Cliquez sur la case bleue, puis sur OK.

La boîte de dialogue Couleurs se ferme et le texte dans l'étiquette Horloge prend la couleur bleue. Ce n'est pas hélas visible sur cette copie d'écran, mais vous le constaterez sur votre écran.

8. Dans le menu Horloge, cliquez sur la commande Date.

La date du jour est affichée en bleu. La couleur du texte dans l'étiquette étant définie, elle reste bleue jusqu'à la prochaine modification ou jusqu'à la fermeture du programme.

9. Fermez le programme.

   L'application se termine et l'EDI de Visual Studio apparaît.

Ça y est ! Vous avez découvert plusieurs commandes et techniques de création de menus, de barres d'outils et de boîtes de dialogue dans vos programmes. Une fois que vous en saurez davantage sur le code, vous pourrez mettre ces connaissances en pratique dans vos propres programmes.

Ajouter d'autres boîtes de dialogue à des programmes

Comment ajouter une boîte de dialogue qui n'est pas fournie par l'un des huit contrôles de boîte de dialogue de Visual Studio à votre programme ? Aucun problème. Vous devrez seulement effectuer un travail de conception supplémentaire. Comme vous allez le voir dans les chapitres suivants, un programme Visual Basic peut exploiter plusieurs formulaires pour recevoir et diffuser des informations. Pour créer des boîtes de dialogue non standard, vous devez ajouter de nouveaux formulaires à votre programme, ajouter des objets d'entrée et de sortie et traiter les clics dans les boîtes de dialogue dans le code (ces techniques feront l'objet du chapitre 14, « Gérer les formulaires et les contrôles Windows à l'exécution »). Dans le chapitre 5, « Variables et formules Visual Basic et environnement .NET Framework », vous apprendrez à utiliser deux boîtes de dialogue conçues spécialement pour recevoir des entrées textuelles (*InputBox*) et diffuser des sorties textuelles (*MsgBox*). Ces boîtes de dialogue permettent de faire le lien entre les contrôles de boîte de dialogue et les boîtes de dialogue que vous créez par vous-même.

Aller plus loin : Assigner des touches de raccourci aux menus

Le contrôle *MenuStrip* permet d'assigner aux menus des touches de raccourci, des combinaisons de touches sur lesquelles l'utilisateur appuie pour activer une commande sans passer par la barre de menus. Par exemple, dans le menu Édition standard d'une application Windows comme Microsoft Word, vous copiez le texte sélectionné dans le Presse-papiers en appuyant sur CTRL+C. La propriété *ShortcutKeys* du contrôle *MenuStrip* permet de personnaliser ce paramètre. Essayons maintenant d'assigner deux touches de raccourci au menu Horloge du programme Menu.

Assigner des touches de raccourci au menu Horloge

1. Veillez à ce que le programme soit arrêté et en mode conception.

   Vous pouvez modifier un programme seulement s'il n'est pas en cours d'exécution. Pour découvrir une exception à cette règle, reportez-vous au chapitre 8, « Déboguer les programmes Visual Basic ».

2. Cliquez sur le menu Horloge, puis sur la commande Heure pour la sélectionner : avant de définir la touche de raccourci d'une commande de menu, vous devez la sélectionner. Assignez une touche de raccourci en définissant la propriété *ShortcutKeys* de la commande dans la fenêtre Propriétés (dans Visual Basic .NET 2002 et 2003, cette propriété était appelée *Shortcut*).

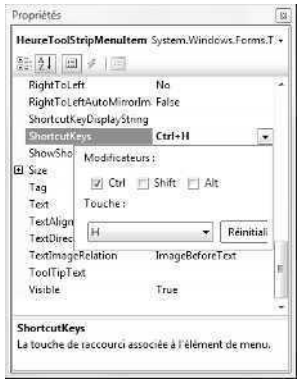
   L'objet barre de menus vous permet de le faire facilement.

3. Ouvrez la fenêtre Propriétés, cliquez sur la propriété *ShortcutKeys*, puis cliquez sur la flèche dans la deuxième colonne.

   Un menu contextuel apparaît pour vous aider à assigner la touche de raccourci.

4. Cochez la case Ctrl, cliquez sur la flèche de liste déroulante des touches et sélectionnez la lettre H dans la liste.

   Voici à quoi ressemble la fenêtre Propriétés :



Astuce Visual Basic affiche généralement la combinaison de touches de raccourci dans le menu lorsque l'on exécute le programme, pour indiquer aux utilisateurs sur quelles touches appuyer. Pour masquer les combinaisons de touches à l'utilisateur (si vous n'avez pas assez de place), positionnez la propriété *ShowShortcutKeys* sur *False*. Les touches de raccourci fonctionnent toujours, mais les utilisateurs n'ont pas de rappel visuel. Vous pouvez également déterminer l'affichage des touches de raccourci dans le programme en définissant la propriété *ShortcutKeyDisplayString*.

5. Cliquez sur la commande Date, puis réglez sa propriété *ShortcutKeys* sur CTRL+D.

   Vous allez maintenant exécuter le programme et tester les touches de raccourci.
6. Cliquez sur le formulaire pour fermer le menu Horloge.
7. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.
8. Appuyez sur CTRL+D pour exécuter la commande Date.

   La date du jour apparaît dans le programme.
9. Appuyez sur CTRL+H pour exécuter la commande Heure.

   L'heure courante apparaît dans le programme.
10. Cliquez sur le menu Horloge.

    Les touches de raccourci figurent à côté des commandes Heure et Date, comme le montre l'illustration suivante. Visual Basic ajoute ces combinaisons de touches lorsque vous définissez les raccourcis à l'aide de la propriété *ShortcutKeys*.

11. Fermez le programme.

    Le programme Menu se ferme et l'environnement de développement apparaît.

Vous êtes prêt à passer à l'écriture de programmes proprement dite, dans la partie de l'ouvrage que j'intitule « Les bases de la programmation ».

Rappel du chapitre 4

Pour	Faites ceci
Créer un élément de menu	Cliquez sur le contrôle *MenuStrip* et dessinez un menu sur votre formulaire. Sur le formulaire, cliquez sur l'étiquette Tapez ici et saisissez le nom des menus et des commandes que vous voulez créer.
Ajouter une touche d'accès rapide à un élément de menu	Cliquez deux fois sur l'élément de menu pour afficher le pointeur en I, puis tapez une esperluette (&) suivie de la lettre à utiliser comme touche d'accès rapide.
Ajouter une touche de raccourci à un élément de menu	Définissez la propriété *ShortcutKeys* de l'élément de menu dans la fenêtre Propriétés. Une liste de touches de raccourci standard est fournie.
Modifier l'ordre des éléments de menu	Faites glisser l'élément de menu que vous souhaitez déplacer.
Ajouter une barre d'outils à votre programme	Cliquez sur le contrôle *ToolStrip* et dessinez une barre d'outils sur votre formulaire. Cliquez droit sur les boutons pour les personnaliser. Double-cliquez sur les boutons et écrivez des procédures événementielles pour les configurer.
Utiliser une boîte de dialogue standard dans votre programme.	Ajoutez l'un des huit contrôles de boîtes de dialogue standard à votre formulaire, puis personnalisez-le à l'aide des paramètres de propriété et du code. Les contrôles de boîtes de dialogue sont situés dans les onglets Boîtes de dialogue et Impression de la Boîte à outils.
Afficher une boîte de dialogue Ouvrir	Ajoutez le contrôle *OpenFileDialog* à votre formulaire. Affichez la boîte de dialogue à l'aide de la méthode *ShowDialog*. La propriété *FileName* contient le nom du fichier sélectionné.
Afficher une boîte de dialogue Couleurs	Ajoutez le contrôle *ColorDialog* à votre formulaire. Affichez la boîte de dialogue à l'aide de la méthode *ShowDialog*. La propriété *Color* contient la couleur sélectionnée par l'utilisateur.

Partie II

Les bases de la programmation

Dans cette partie :

Dans la première partie, « Démarrer avec Visual Basic 2008 », vous avez appris à créer l'interface utilisateur d'un programme Microsoft Visual Basic 2008 et à construire et exécuter un programme dans l'environnement de développement de Microsoft Visual Studio 2008. Dans les neuf chapitres de la deuxième partie, « Les bases de la programmation », vous allez en apprendre davantage sur le code Visual Basic (les instructions et les mots clés, noyau d'un programme Visual Basic). Vous allez également apprendre à gérer l'information dans les programmes et à contrôler l'exécution de votre code. Vous découvrirez comment exploiter des structures de décision, des boucles, des horloges, des tableaux, des collections et des fichiers texte. Vous verrez comment déboguer vos programmes et gérer les éventuelles erreurs d'exécution. À la fin de la deuxième partie, vous serez prêt à aborder des sujets plus avancés, comme la personnalisation de l'interface utilisateur, la programmation de bases de données et la programmation web.

Chapitre 5

Variables et formules Visual Basic et l'environnement .NET Framework

À la fin de ce chapitre, vous saurez :

- Utiliser des variables pour stocker des données dans vos programmes
- Utiliser la fonction *InputBox* pour recueillir des entrées
- Utiliser la fonction *MsgBox* pour afficher des messages
- Travailler avec différents types de données
- Utiliser des variables et des opérateurs pour manipuler des données
- Utiliser des méthodes du .NET Framework
- Utiliser des opérateurs mathématiques et des fonctions dans des formules

Dans ce chapitre, vous allez apprendre à utiliser des variables et des constantes pour stocker temporairement des données dans un programme ainsi que les fonctions *Input Box* et *MsgBox* pour recueillir et présenter des informations à l'aide de boîtes de dialogue. Nous verrons comment exploiter les fonctions et les formules pour effectuer des calculs et employer les opérateurs arithmétiques pour accomplir des tâches comme la multiplication et la concaténation. Vous découvrirez enfin comment vous servir des classes et des méthodes du .NET Framework 3.5 de Microsoft pour effectuer des calculs mathématiques et d'autres tâches utiles.

Structure d'une instruction Visual Basic

Comme vous l'avez appris dans le chapitre 2, « Écrire son premier programme », dans un programme Visual Basic une ligne de code est appelée *instruction*. Une instruction est une combinaison de mots clés, de propriétés, de noms d'objets, de variables, de nombres, de symboles spéciaux et d'autres valeurs Visual Basic qui, ensemble, forment une instruction valable reconnue par le compilateur Visual Basic. Un simple mot clé peut être une instruction, comme

```
End
```

qui interrompt l'exécution d'un programme Visual Basic. Il peut s'agir d'une combinaison d'éléments, comme l'instruction suivante, qui utilise la propriété *TimeString* pour assigner l'heure système courante à la propriété *Text* de l'objet *Label1* :

```
Label1.Text = TimeString
```

L'ensemble des règles de construction des instructions est nommé syntaxe. De nombreuses règles de syntaxe de Visual Basic sont communes aux anciennes versions du langage de programmation BASIC et à d'autres compilateurs de langage. Pour écrire des instructions correctes, il faut apprendre la syntaxe des éléments de langage les plus exploités, puis se servir convenablement ces éléments pour traiter les données dans le programme. Heureusement, Visual Basic effectue une grande partie du travail pour vous. Le temps que vous passez à écrire le code est relativement court et les résultats peuvent être réemployés dans d'autres programmes. L'EDI Visual Studio signale en outre de potentielles erreurs de syntaxe et suggère des corrections, de façon analogue au dispositif de correction automatique de Word.

Dans ce chapitre et les suivants, vous allez découvrir les principaux mots clés et instructions de programmation Visual Basic, ainsi que de nombreux objets, propriétés et méthodes fournis par les contrôles Visual Studio et le .NET Framework. Ces mots clés et ces objets complètent les compétences de programmation que vous avez déjà acquises et vous aideront à écrire vos programmes à l'avenir. Les premiers sujets abordés, variables et types de données, sont des caractéristiques essentielles de la plupart des programmes.

Utiliser des variables pour stocker des informations

Une *variable* représente un emplacement provisoire de stockage de données dans le programme. Dans le code, vous utilisez une ou plusieurs variables qui peuvent contenir des mots, des nombres, des dates, des propriétés ou d'autres valeurs grâce auxquelles vous donnez un nom court et facile à mémoriser à chaque donnée avec laquelle vous souhaitez travailler. Les variables peuvent contenir les informations saisies par l'utilisateur pendant l'exécution, le résultat d'un calcul particulier ou une donnée à afficher sur votre formulaire. Pour résumer, les variables sont des conteneurs que vous utilisez pour stocker et conserver tout type d'information.

L'utilisation des variables dans un programme Visual Basic demande une certaine planification. Avant de pouvoir utiliser une variable, il vous faut réserver de l'espace pour son utilisation. Ce processus ressemble à la réservation d'une place au théâtre ou à un match de football. Le processus de réservation d'une variable, ou *déclaration*, fera l'objet de la prochaine section.

Réserver de l'espace pour les variables : L'instruction *Dim*

Depuis Microsoft Visual Basic .NET 2003, vous devez déclarer explicitement vos variables avant de les utiliser. Ce n'était pas le cas dans Visual Basic 6 et les versions plus anciennes de Visual Basic, où vous pouviez (dans certaines circonstances) déclarer implicitement des variables (en d'autres termes, les utiliser sans l'instruction *Dim*). Cette pratique est flexible mais assez risquée : elle peut donner lieu à une confusion entre variables et à une mauvaise orthographe des noms de variables, et donc à des bogues dans le code qui peuvent ne pas être découverts ultérieurement.

Visual Basic 2008 effectue un petit retour vers le passé en matière de déclaration de variable. Il est à nouveau possible de déclarer une variable de façon implicite. Comme je déconseille toutefois cette pratique, je n'aborderais pas ce nouveau dispositif tant que vous n'aurez pas acquis les méthodes de programmation recommandées par les plus éminents des programmateurs expérimentés.

Pour déclarer une variable dans Visual Basic 2008, tapez le nom de la variable après l'instruction *Dim* (Dim est l'abrégé de *Dimension*). Cette déclaration réserve de l'espace en mémoire pour la variable lors de l'exécution du programme et informe Visual Basic du type de données auquel il doit s'attendre. Bien que cette déclaration puisse avoir lieu à tout endroit du code (tant que la déclaration a lieu avant l'utilisation de la variable), la plupart des programmeurs déclarent les variables au début de leurs procédures événementielles ou de leurs modules de code.

Par exemple, l'instruction suivante libère de l'espace pour la variable *Nom*, qui contient une valeur textuelle, ou *chaîne* (String) :

```
Dim Nom As String
```

En plus d'identifier la variable par son nom, le mot clé *As* attribue un certain type à la variable identifiée *via* le mot clé *String* (vous découvrirez d'autres types de données dans ce chapitre). Une variable chaîne comporte des informations textuelles : des mots, des lettres, des symboles et même des nombres. J'utilise beaucoup les variables chaîne pour les noms, les lieux, les lignes de poème, les contenus de fichiers et bien d'autres données textuelles.

Pourquoi déclarer les variables ? Visual Basic demande d'identifier le nom et le type des variables à l'avance de sorte que le compilateur puisse réserver la mémoire nécessaire au programme pour stocker et traiter les informations contenues dans les variables. La gestion de la mémoire ne vous semble peut-être pas très importante (après tout, les ordinateurs modernes possèdent une mémoire RAM importante et des gigaoctets d'espace disque libre), mais certains programmes consomment rapidement de la mémoire et l'allocation de cette dernière doit être prise très au sérieux, même au début. Comme vous allez le constater, le besoin d'espace et les limites de taille varient selon les types de variables.

Remarque Certaines versions antérieures de Visual Basic ne nécessitent pas de type de variables particulier (comme *String* ou *Integer*) ; l'information est conservée à l'aide d'un type de données générique (et très gourmand en mémoire) appelé *Variant*, qui peut contenir des données de toutes tailles et de tous formats. Visual Basic 2008 ne prend pas en charge les types *Variant*. Bien qu'ils soient très pratiques pour les programmeurs débutants, leur conception les rend lents et inefficaces. De surcroît, ils permettent trop facilement la conversion de variables d'un type à un autre, ce qui provoque souvent des résultats inattendus. Comme vous l'apprendrez toutefois par la suite, vous pouvez encore stocker des informations dans des conteneurs génériques nommées *Object*. Ils proposent des fonctionnalités presque universelles mais restent plutôt inefficaces en termes de taille.

Après avoir déclaré une variable, vous pouvez lui affecter des informations dans le code *via* l'opérateur d'affectation (=). Par exemple, l'instruction suivante affecte le nom de famille « Jefferson » à la variable *Nom :*

```
Nom = "Jefferson"
```

J'ai affecté une valeur textuelle à la variable *Nom* car le type de données est *String*. Il est également possible d'affecter à la variable des valeurs avec des espaces, des symboles ou des nombres, comme

```
Nom = "1313 rue de l'oiseau moqueur"
```

mais la variable est toujours considérée comme une valeur de type chaîne. La partie numérale ne peut être utilisée dans une formule mathématique qu'après conversion préalable en nombre entier ou en valeur à virgule flottante à l'aide de l'une des fonctions de conversion que nous aborderons plus tard.

Une fois une valeur affectée à la variable *Nom*, elle remplace le nom « Jefferson » dans le code. Par exemple, l'instruction d'affectation

```
Label1.Text = Nom
```

affiche « Jefferson » dans l'étiquette *Label1* de votre formulaire.

Déclaration implicite de variable

Pour déclarer des variables « à l'ancienne » dans Visual Basic 2008 (c'est-à-dire sans les déclarer explicitement à l'aide de l'instruction *Dim*), placez l'instruction *Option Explicit Off* au début de votre formulaire ou du code du module, avant toute procédure événementielle. Les variables ne devront alors plus être impérativement déclarées avant leur utilisation, comme le demande par défaut Visual Basic. Je vous déconseille d'ajouter systématiquement cette instruction à votre code, mais elle peut s'avérer utile pour convertir d'anciens programmes Visual Basic en Visual Studio 2008.

Une autre possibilité consiste à employer la nouvelle instruction *Option Infer* de Basic 2008. Si *Option Infer* est fixé à *On*, Visual Basic déduit le type d'une variable en examinant sa première affectation. Cela permet de déclarer des variables sans identifier spécifiquement le type employé : Visual Basic effectue la détermination.

Par exemple, l'expression

```
Dim attendance = 100
```

va déclarer la variable nommée *attendance* comme *Integer*, puisque 100 est une expression entière. En d'autres termes, lorsque *Option Infer* est fixé à *On*, cela est équivalent à saisir

```
Dim attendance As Integer = 100
```

De même, l'expression

```
Dim address = "1012 Daisy Lane"
```

déclare la variable *address* comme de type *String*, puisque son affectation initiale est de type *String*. Si toutefois vous fixez *Option Infer* à *Off*, Visual Basic va déclarer la variable comme de type *Object* : un conteneur générique volumineux et quelque peu inefficace pour tout type de données. Si vous envisagez d'employer *Option Infer* pour autoriser ce type de déclaration de variable déduite (une approche souple mais susceptible d'aboutir potentiellement à des résultats inattendus), placez les deux instructions suivantes en haut de votre module de code, au-dessus de l'instruction *Class Form* :

```
Option Explicit Off
Option Infer On
```

Option Explicit Off autorise la déclaration de variables lors de leur utilisation tandis que *Option Infer On* permet à Visual Basic d'en déterminer automatiquement le type. Vous pouvez également configurer ces options à l'aide de la commande Options du menu Outils, comme examiné au chapitre 1, « Explorer l'environnement de développement intégré de Visual Studio ».

Utiliser des variables dans un programme

Les variables peuvent garder la même valeur au cours du programme ou changer de valeur plusieurs fois, selon les besoins. L'exercice suivant illustre comment la variable *Nom* peut contenir différentes valeurs textuelles et comment elle peut être affectée à des propriétés d'objet.

Modifier la valeur d'une variable

1. Démarrez Visual Studio.
2. Dans le menu Fichier, cliquez sur Ouvrir un projet.

   La boîte de dialogue Ouvrir un projet s'affiche.
3. Ouvrez le projet Test de variable dans le dossier c:\vb08epe\chap05\Test de variable.
4. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Le formulaire Test de variable apparaît dans le Concepteur. Test de variable est un *programme squelette*. Il contient un formulaire avec des étiquettes et des boutons pour afficher les résultats, mais peu de code. Je crée ces programmes squelette pour vous faire gagner du temps, bien que vous puissiez créer le projet de A à Z. Dans cet exercice, vous allez ajouter du code.

Voici à quoi ressemble le formulaire Test de variable :

Le formulaire comporte deux étiquettes et deux boutons. Vous allez utiliser les variables pour afficher des informations dans chacune des étiquettes.

Remarque Les objets étiquette ressemblent à des cases car j'ai positionné leur propriété *BorderStyle* à *Fixed3D*.

5. Double-cliquez sur le bouton Montrer.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

6. Tapez les instructions de programme suivantes pour déclarer et utiliser la variable *Nom* :

```
Dim Nom As String
Nom = "Luther"
Label1.Text = Nom
Nom = "Bodenstein von Karlstadt"
Label2.Text = Nom
```

Les instructions sont organisées en trois groupes. La première instruction se sert de l'instruction *Dim* et du type *String* pour déclarer la variable *Nom*. Une fois que vous avez saisi cette ligne, Visual Studio souligne la variable *Nom* d'une ligne dentelée, car elle a été déclarée mais pas utilisée dans le programme. Rien de grave : Visual Studio vous rappelle seulement que la nouvelle variable créée attend d'être utilisée.

Astuce Si vous avez terminé d'écrire le programme et que le nom de la variable est toujours souligné, vous l'avez peut-être mal orthographié dans le code.

Les deuxième et troisième lignes affectent le nom « Luther » à la variable *Nom* puis affichent ce nom dans la première étiquette du formulaire. Cet exemple illustre l'un des usages courants des variables dans un programme : le transfert d'informations vers une propriété. Comme vous l'avez vu précédemment, toutes les valeurs chaîne affectées à des variables sont affichées en rouge.

La quatrième ligne affecte le nom « Bodenstein von Karlstadt » à la variable *Nom* (en d'autres termes, elle modifie le contenu de la variable). Notez que la deuxième chaîne est plus longue que la première et comporte des espaces. Lorsque vous affectez des chaînes de texte à des variables ou que vous les utilisez à d'autres endroits, vous devez insérer le texte entre guillemets (c'est inutile pour les nombres).

Enfin, souvenez-vous d'une autre caractéristique importante des variables déclarées dans cette procédure événementielle : elles conservent leur *portée*, ou gardent leur valeur, seulement dans la procédure événementielle dans laquelle vous les utilisez. Vous apprendrez plus tard à déclarer des variables exploitables dans toutes les procédures événementielles du formulaire.

7. Cliquez sur l'onglet Form1.vb [Design] pour afficher à nouveau le formulaire.

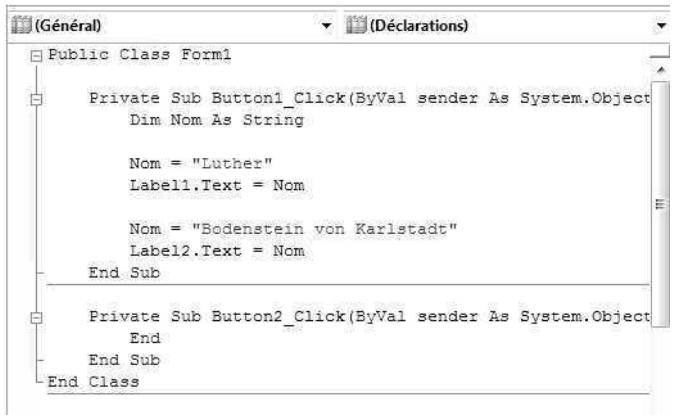
8. Double-cliquez sur le bouton Quitter.

La procédure événementielle *Button2_Click* s'affiche dans l'Éditeur de code.

9. Tapez l'instruction suivante pour interrompre le programme :

```
End
```

Votre écran présente un résultat similaire à :



10. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

11. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le programme s'exécute dans l'EDI.

12. Cliquez sur le bouton Montrer.

Le programme déclare la variable, lui affecte deux valeurs et copie chaque valeur dans l'étiquette appropriée du formulaire. Le résultat du programme est le suivant :

13. Cliquez sur le bouton Quitter pour interrompre le programme.

Le programme s'arrête et vous revenez à l'environnement de développement.

Conventions de nommage des variables

Le nommage des variables peut s'avérer difficile, car vous devez utiliser des noms courts mais intuitifs et faciles à retenir. Pour éviter toute confusion, respectez les conventions suivantes lorsque vous nommez des variables :

- Commencez chaque variable par une lettre ou un caractère de soulignement. Les noms des variables ne peuvent comporter que des lettres, des caractères de soulignement ou des nombres.
- Bien que les noms des variables n'aient pas de longueur maximum, essayez de les limiter à 33 caractères pour faciliter leur lecture (dans Visual Basic 6, les noms des variables étaient limités à 255 caractères, mais ce n'est plus le cas).
- Servez-vous de noms de variables descriptifs, en combinant plusieurs mots si nécessaire. Par exemple, le nom de variable *TVAVentes* est bien plus clair que *TVA* ou *Taxes*.
- Utilisez une combinaison de majuscules, de minuscules et de nombres. Par convention, la première lettre de chaque mot d'une variable est une majuscule ; par exemple, *DateDeNaissance*. Toutefois, certains programmeurs préfèrent utiliser la casse « en chameau » (qui consiste à mettre la première lettre d'une variable en minuscule) pour distinguer les noms de variables des fonctions et des noms de modules, qui commencent généralement par une majuscule, par exemple : *dateDeNaissance*, *nomEmployé* et *compteur*.

- N'utilisez pas les mots clés, les objets ou les propriétés de Visual Basic comme noms de variables. Une erreur se produirait à l'exécution du programme.
- Vous pouvez éventuellement commencer chaque nom de variable par une abréviation de deux ou trois caractères correspondant au type de données stockées dans la variable. Par exemple, utilisez *strNom* pour montrer que la variable *Nom* contient des données de type chaîne. Vous n'avez pas à vous inquiéter de ce détail pour l'instant, mais prenez note de cette convention : vous la retrouverez dans divers emplacements de la documentation de Visual Studio et dans de nombreux ouvrages de programmation Visual Basic avancée (ce modèle de convention et d'abréviation a été créé par l'informaticien de Microsoft Charles Simonyi et porte le nom de notation hongroise).

Utiliser une variable pour stocker des entrées

Une variable sert souvent à stocker des informations entrées par l'utilisateur. S'il est possible de faire appel à un objet tel qu'une zone de liste ou qu'une zone de texte pour recueillir ces informations, il est parfois préférable de traiter directement avec l'utilisateur et d'enregistrer les entrées dans une variable plutôt que dans une propriété. Pour recueillir des entrées, servez-vous de la fonction *InputBox*, qui affiche une boîte de dialogue à l'écran, puis d'une variable pour stocker le texte saisi par l'utilisateur. Vous testerez cette technique dans l'exemple suivant.

Recueillir des entrées à l'aide de la fonction *InputBox*

1. Dans le menu Fichier, pointez sur Ouvrir Projet.

   La boîte de dialogue Ouvrir un projet s'affiche.

2. Ouvrez le projet Zone de saisie qui se trouve dans le dossier c:\vb08epe\chap05\Zone de saisie.

   Le projet Zone de saisie s'ouvre dans l'EDI. Il s'agit d'un programme squelette.

3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Le formulaire comporte une étiquette et deux boutons. Vous allez utiliser la fonction *InputBox* pour recueillir une entrée de l'utilisateur, puis afficherez celle-ci dans l'étiquette du formulaire.

4. Double-cliquez sur le bouton Zone de saisie.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

5. Tapez les instructions de programme suivantes pour déclarer deux variables et appeler la fonction *InputBox* :

```
Dim Prompt, NomComplet As String
Prompt = "Veuillez saisir votre nom."
NomComplet = InputBox(Prompt)
Label1.Text = NomComplet
```

Cette fois-ci, l'instruction *Dim* déclare deux variables du type *String* : *Prompt* et *NomComplet*. Vous pouvez déclarer autant de variables que vous voulez sur la même ligne, du moment qu'elles sont du même type. Dans Visual Basic 6, cette syntaxe aurait donné des résultats différents. *Dim* aurait créé la variable *Prompt* de type *Variant* (aucun type n'étant spécifié) et la variable *NomComplet* de type *String*. Cependant, cette contradiction a été supprimée dans les versions 2002 et ultérieures de Visual Basic.

La deuxième ligne de la procédure événementielle affecte une chaîne de texte à la variable *Prompt*. Ce message est utilisé comme argument textuel pour la fonction *InputBox*. Un *argument* est une valeur ou une expression transmise à une procédure ou à une fonction. La ligne suivante appelle la fonction *InputBox* et affecte le résultat (la chaîne de texte saisie par l'utilisateur) à la variable *NomComplet*. *Input Box* est une fonction Visual Basic particulière qui affiche une boîte de dialogue à l'écran et demande une saisie de l'utilisateur. Outre les chaînes d'invite, la fonction *InputBox* accepte d'autres arguments. Pour en savoir plus, consultez la documentation Visual Basic.

Une fois que *InputBox* a renvoyé une chaîne de texte au programme, la quatrième instruction de la procédure place le nom de l'utilisateur dans la propriété *Text* de l'objet *Label1*, qui l'affiche sur le formulaire.

Remarque Dans les anciennes versions de BASIC, la fonction *InputBox* insérait un caractère $ à la fin pour rappeler aux programmeurs que la fonction avait renvoyé l'information sous forme de chaîne ($). Parfois, les variables de type chaîne étaient également identifiées par le symbole $. Aujourd'hui nous n'utilisons plus d'abréviations pour les types de données. *String* ($), *Integer* (%) et les autres abréviations n'existent plus.

6. Enregistrez vos modifications.

Vous rappelez-vous sur quel bouton de la barre d'outils cliquer pour enregistrer votre projet ? Si vous avez oublié, rendez-vous à l'étape 10 de l'exercice précédent.

7. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le programme s'exécute dans l'EDI.

8. Cliquez sur le bouton Zone de saisie.

Visual Basic exécute la procédure événementielle *Button1_Click* et la boîte de dialogue Zone de saisies apparaît à l'écran, comme suit :

9. Tapez votre nom, puis cliquez sur OK.

La fonction *InputBox* renvoie votre nom au programme et le place dans la variable *NomComplet*. Le programme utilise ensuite la variable pour afficher votre nom sur le formulaire, comme suit :

Utilisez la fonction *InputBox* dans vos programmes à chaque fois que vous souhaitez demander des informations à l'utilisateur. Vous pouvez employer cette fonction parallèlement aux autres contrôles de saisie pour réguler le flux de données qui entre dans le programme et qui en sort. Dans l'exercice suivant, vous allez apprendre à exploiter une fonction similaire pour afficher du texte dans une boîte de dialogue.

10. Sur le formulaire, cliquez sur le bouton Quitter pour arrêter le programme.

Le programme s'arrête et vous revenez à l'environnement de développement.

Qu'est-ce qu'une fonction ?

InputBox est un mot clé spécial de Visual Basic connu sous le nom de *fonction*. Une fonction est une instruction qui effectue certaines tâches (comme inviter l'utilisateur à saisir des informations ou calculer une équation) puis renvoie un résultat au programme. La valeur renvoyée par une fonction peut être affectée à une variable, comme dans le programme Input Box, ou à une propriété, une autre instruction ou fonction. Les fonctions Visual Basic utilisent souvent un ou plusieurs arguments pour définir leurs actions. Par exemple, la fonction *InputBox* que vous venez d'exécuter utilise la variable *Prompt* pour présenter à l'utilisateur les instructions de la boîte de dialogue. Lorsqu'une fonction utilise plusieurs arguments, ceux-ci sont séparés par des virgules et le groupe d'arguments est mis entre parenthèses. L'instruction suivante représente l'appel d'une fonction à deux arguments :

```
NomComplet = InputBox(Prompt, Titre)
```

Remarque : dans cette description de syntaxe, j'utilise l'italique pour indiquer les paramètres fictifs pour les informations que vous spécifiez. Vous trouverez ce style tout au long de cet ouvrage et dans la documentation de Visual Studio.

Utiliser une variable en tant que sortie

Vous pouvez afficher le contenu d'une variable en affectant la variable à une propriété (comme la propriété *Text* d'un objet étiquette) ou en la transférant sous forme d'argument vers une fonction de boîte de dialogue, comme la fonction *MsgBox*. Lorsque vous appelez la fonction *MsgBox*, elle affiche une boîte de dialogue, parfois dénommée *boîte de message*, dont vous pouvez spécifier les options. Comme *InputBox*, elle accepte un ou plusieurs arguments en tant qu'entrées et les résultats de l'appel de fonction peuvent être affectés à une variable. La syntaxe de la fonction *MsgBox* est

```
BoutonCliqué = MsgBox(Prompt, Buttons, Titre)
```

où *Prompt* est le texte qui sera affiché dans la boîte de message, *Buttons* correspond à un nombre qui définit les boutons, les icônes et les autres options de la boîte de message à afficher et *Titre* représente le texte qui apparaît dans la barre de titre de la boîte de message. La variable *BoutonCliqué* est affectée au résultat renvoyé par la fonction. Elle indique sur quel bouton l'utilisateur a cliqué dans la boîte de dialogue.

Si vous utilisez la fonction *MsgBox* seulement pour afficher un message, la variable *BoutonCliqué*, l'opération d'affectation (=), l'argument *Buttons* et l'argument *Titre* sont facultatifs. Dans l'exercice suivant, vous allez utiliser l'argument *Titre*, mais pas les autres. Pour en savoir plus (notamment sur les différents boutons que vous pouvez insérer dans *MsgBox* et d'autres options), recherchez la fonction *MsgBox* dans la documentation de Visual Studio.

Remarque Pour afficher du texte dans une boîte de message, Visual Basic propose la fonction *MsgBox* et la classe *MessageBox*. La classe *MessageBox* est membre de l'espace de noms *System.Windows.Forms*, accepte des arguments de type *MsgBox* et est affichée par la méthode *Show*. Dans cet ouvrage, j'utilise *MsgBox* et *MessageBox*.

Vous allez maintenant ajouter une fonction *MsgBox* au programme Zone de saisie pour afficher le nom saisi par l'utilisateur dans la boîte de dialogue.

Afficher un message à l'aide de la fonction *MsgBox*

1. Si l'Éditeur de code n'apparaît pas, double-cliquez sur le bouton Zone de saisie du formulaire Zone de saisie.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code (il s'agit du code que vous avez saisi dans l'exercice précédent).

2. Sélectionnez l'instruction suivante dans la procédure événementielle (la dernière ligne) :

   ```
   Label1.Text = NomComplet
   ```

   Il s'agit de l'instruction qui affiche le contenu de la variable *NomComplet* dans l'étiquette.

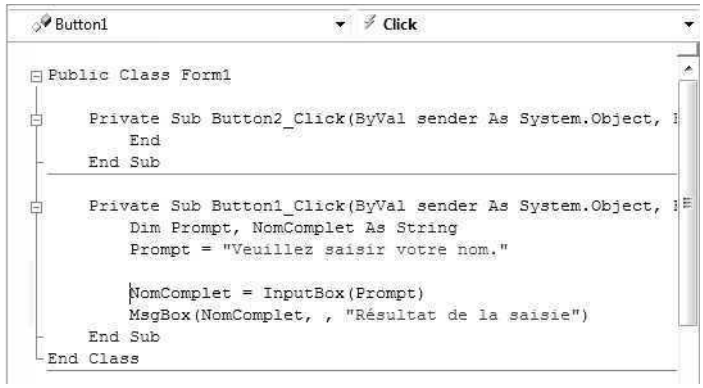
3. Appuyez sur la touche SUPPR pour supprimer la ligne.

   L'instruction est éliminée de l'Éditeur de code.

4. À la place, tapez la ligne suivante dans la procédure événementielle :

   ```
   MsgBox(NomComplet, , "Résultat de la saisie")
   ```

   Cette nouvelle instruction va appeler la fonction *MsgBox*, afficher le contenu de la variable *NomComplet* dans la boîte de dialogue et placer les mots *Résultat de la saisie* dans la barre de titre. L'argument *Buttons* et la variable *BoutonCliqué*, facultatifs, ne s'appliquent pas ici. Ils ont été omis. Votre procédure événementielle présente un résultat similaire à :



5. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

6. Cliquez sur le bouton Zone de saisie, saisissez votre nom dans la zone de saisie, puis cliquez sur OK.

   Visual Basic stocke dans la variable *NomComplet* les entrées dans le programme, puis les affiche dans une boîte de message. Votre écran ressemble à :

7. Cliquez sur OK pour fermer la boîte de message. Cliquez sur Quitter pour fermer le programme.

   Le programme se ferme et vous revenez à l'environnement de développement.

Travailler avec les types de données particuliers

Le type de données *String* permet de gérer du texte dans vos programmes, mais qu'en est-il des nombres, des dates et autres types d'information ? Afin de permettre une gestion efficace de la mémoire pour tous les types de données, Visual Basic propose plusieurs autres types de données destinés aux variables. Il s'agit pour la plupart de types courants dans les versions antérieures de BASIC ou Visual Basic. Certains sont apparus avec Visual Studio 2005 pour permettre un traitement efficace des données dans les nouveaux ordinateurs 64 bits.

Le tableau suivant présente les types de données fondamentaux de Visual Basic. Quatre nouveaux types de données ont été ajoutés dans Visual Basic 2005 : *SByte*, *UShort*, *UInteger* et *ULong*. *SByte* tient compte des valeurs « signées », c'est-à-dire des nombres positifs et négatifs. *UShort*, *Uinteger* et *ULong* sont des types de données « non signés », c'est-à-dire qu'ils ne peuvent pas contenir des nombres négatifs. En revanche, comme le montre le tableau suivant, la plage des types de données non signés est deux fois plus étendue que la plage positive de leurs homologues signés. Si vos programmes effectuent de nombreux calculs, choisir le type de données correct pour vos variables (la taille ne doit être ni trop petite, ni trop grande) améliorera leurs performances. Dans l'exercice suivant, vous allez découvrir le fonctionnement de ces types de données.

Remarque La taille de stockage des variables se mesure en bits. La quantité d'espace requis pour stocker un caractère de clavier standard (ASCII) en mémoire est de 8 bits, soit un octet.

Type de données	Taille	Plage	Exemple d'utilisation
Short	16 bits	–32 768 à 32 767	`Dim Oiseaux As Short` `Oiseaux = 12500`
Ushort	16 bits	0 à 65 535	`Dim Jours As Ushort` `Jours = 55000`
Integer	32 bits	–2 147 483 648 à 2 147 483 647	`Dim Insectes As Integer` `Insectes = 37500000`
Uinteger	32 bits	0 à 4 294 967 295	`Dim Joies As Uinteger` `Joies = 3000000000`
Long	64 bits	–9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	`Dim PopMondiale As Long` `PopMondiale = 4800000004`

Ulong	64 bits	0 à 18 446 744 073 709 551 615	`Dim Etoiles As Ulong`
			`Etoiles = _`
			`1800000000000000000`

Single	32 bits à virgule flottante	–3,4028235E38 à 3,4028235E38	`Dim Prix As Single` `Prix = 899.99`

Type de données	Taille	Plage	Exemple d'utilisation
Double	64 bits à virgule flottante	–1,79769313486231E308 à 1,79769313486231E308	`Dim Pi As Double` `Pi = 3.1415926535`
Decimal	128 bits	0 à +/– 79228 162 514 264 337 593 543 950 335 (+/-7.9...E+28) sans décimale ; 0 à +/- 7,9228162514264337593543950335 avec 28 décimales. Ajoutez « D » si vous voulez imposer à Visual Basic d'initialiser un *Decimal*.	`Dim Dette As Decimal` `Dette = 7600300.5D`
Byte	8 bits	0 à 255 (pas de nombres négatifs)	`Dim Clé As Byte` `lé = 13`
Sbyte	8 bits	–128 à 127	`Dim ValNeg As Sbyte` `ValNeg = -20`
Char	16 bits	Tout symbole Unicode compris entre 0 et 65 535. Ajoutez « c » lors de l'initialisation d'un *Char*.	`Dim CarUnicode As Char` `CarUnicode = " "c`
String	Généralement 16 bits par caractère	0 à environ 2 milliards de caractères Unicode 16 bits	`Dim Chien As String` `Chien = "pointer"`
Boolean	16 bits	*True* ou *False* (pendant les conversions, 0 est converti en *False*, les autres valeurs en *True*)	`Dim Drapeau as Boolean` `Drapeau = True`
Date	64 bits	Du 1er janvier 0001 au 31 décembre 9999	`Dim Anniversaire as Date` `Anniversaire = #3/1/ 1963#`
Object	32 bits	Tout type peut être stocké dans une variable de type *Objet*	`Dim MonApp As Object` `MonApp = CreateObject _("Word.Application")`

Utiliser les types de données spéciaux dans le code

1. Dans le menu Fichier, cliquez sur Ouvrir un projet.

   La boîte de dialogue Ouvrir un projet s'affiche.

2. Ouvrez le projet Types de données qui se trouve dans le dossier c:\vb08epe\chap05\Types de données.

3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Types de données est un programme Visual Basic complet que j'ai créé pour illustrer le fonctionnement des types de données essentiels. Vous allez exécuter le programme pour voir à quoi ressemblent les types de données, puis observer comment les variables sont déclarées et utilisées dans le code. Vous apprendrez également à placer les déclarations de variables pour que toutes les procédures événementielles du programme puissent y accéder.

4. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   La fenêtre d'application suivante apparaît :

   Le programme Types de données permet de tester 11 types de données, dont les types entier, simple précision à virgule flottante et date. Le programme affiche un exemple de chaque type lorsque vous cliquez sur son nom dans la zone de liste.

5. Cliquez sur le type *Integer* dans la zone de liste.

   Le nombre 37500000 apparaît dans la case Exemple de données, comme le montre l'illustration suivante. Il est impossible d'insérer ou d'afficher des virgules avec les types de données *Short*, *Integer* et *Long*. Pour afficher des virgules, vous devez utiliser la fonction *Format*.

6. Cliquez sur le type Date dans la zone de liste.

   La date 01/03/1963 apparaît dans la case Exemple de données.

7. Dans la zone de liste, cliquez sur chaque type de données pour voir comment Visual Basic l'affiche dans la case Exemple de données.
8. Cliquez sur le bouton Quitter pour interrompre le programme.

   Vous allez maintenant observer comment les types de données de base sont déclarés au début du formulaire et comment ils sont utilisés dans la procédure événementielle *ListBox1_SelectedIndexChanged*.
9. Double-cliquez sur le formulaire (et non pas sur l'un des objets), puis agrandissez l'Éditeur de code pour voir davantage de code.

   Voici à quoi ressemble l'Éditeur de code :

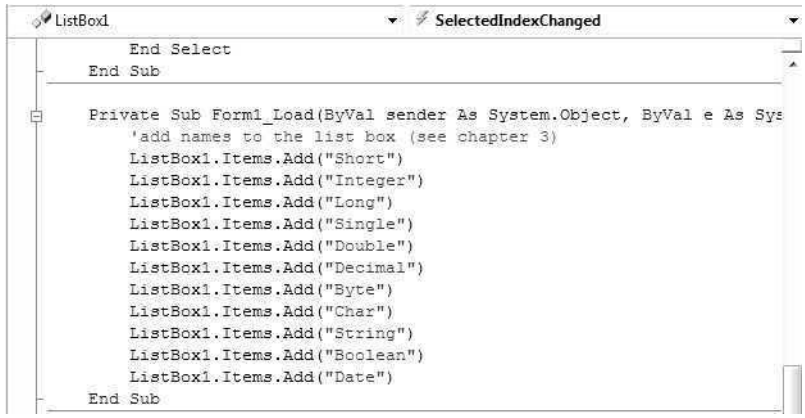
Faites défiler jusqu'au début de l'Éditeur de code pour voir la dizaine d'instructions de programme ajoutées pour déclarer 11 variables dans le programme (une pour chaque type de données dans Visual Basic). Je n'ai pas créé d'exemple pour les types *SByte*, *UShort*, *UInteger* et *ULong* car ils ressemblent à leurs homologues signés ou non signés. En plaçant toutes les instructions *Dim* au début de la zone d'initialisation du code du formulaire, on s'assure que les variables seront valides, ou auront une *portée*, pour toutes les procédures événementielles du formulaire. De cette manière, il est possible de définir la valeur d'une variable dans une procédure événementielle et la lire dans une autre. En règle générale, les variables ne sont valides que dans la procédure événementielle dans laquelle elles ont été déclarées. Pour les rendre valides sur l'ensemble du formulaire, vous devez déclarer les variables au début du code de votre formulaire.

Remarque Toutes les variables portent le même nom que le type de données qui figure dans le tableau précédent, pour que vous puissiez voir les exemples dans le code.

10. Faites défiler l'Éditeur de code vers le bas et observez la procédure événementielle *Form1_Load*.

Vous allez voir les instructions suivantes, qui ajoutent des éléments à l'objet zone de liste du programme.

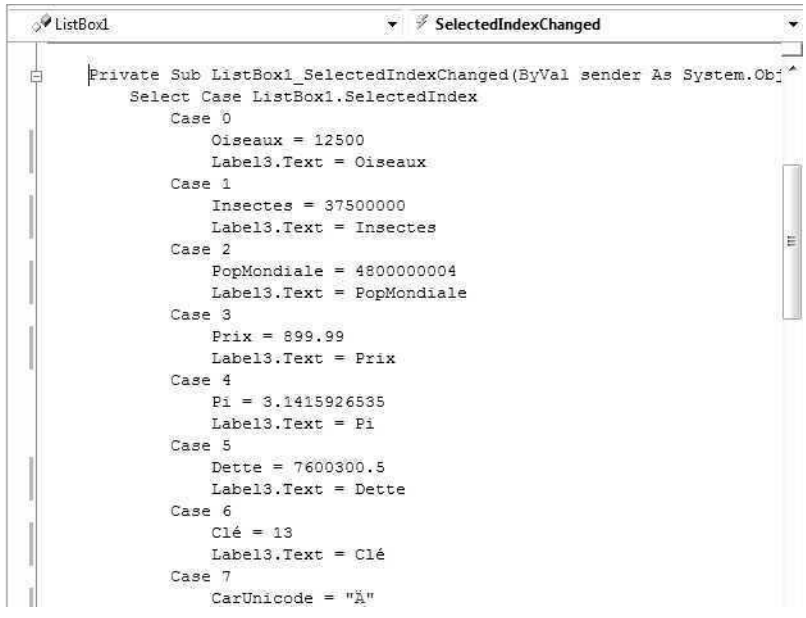
Vous vous souvenez peut-être de cette syntaxe, vue dans le chapitre 3, « Travailler avec les contrôles de la Boîte à outils ». J'ai réutilisé quelques-unes des instructions.




```
        End Select
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As Sys
        'add names to the list box (see chapter 3)
        ListBox1.Items.Add("Short")
        ListBox1.Items.Add("Integer")
        ListBox1.Items.Add("Long")
        ListBox1.Items.Add("Single")
        ListBox1.Items.Add("Double")
        ListBox1.Items.Add("Decimal")
        ListBox1.Items.Add("Byte")
        ListBox1.Items.Add("Char")
        ListBox1.Items.Add("String")
        ListBox1.Items.Add("Boolean")
        ListBox1.Items.Add("Date")
    End Sub
```

11. Faites défiler vers le bas et observez la procédure événementielle *ListBox1_SelectedIndexChanged*.

Elle traite les sélections que vous faites dans la zone de liste et ressemble à :




```
    Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Obj
        Select Case ListBox1.SelectedIndex
            Case 0
                Oiseaux = 12500
                Label3.Text = Oiseaux
            Case 1
                Insectes = 37500000
                Label3.Text = Insectes
            Case 2
                PopMondiale = 4800000004
                Label3.Text = PopMondiale
            Case 3
                Prix = 899.99
                Label3.Text = Prix
            Case 4
                Pi = 3.1415926535
                Label3.Text = Pi
            Case 5
                Dette = 7600300.5
                Label3.Text = Dette
            Case 6
                Clé = 13
                Label3.Text = Clé
            Case 7
                CarUnicode = "Ä"
```

La structure de décision *Select Case* constitue le noyau de la procédure événementielle. Dans le prochain chapitre, nous verrons comment ce groupe d'instructions de programme sélectionne un choix parmi plusieurs. Pour l'instant, remarquez comment chaque section du bloc *Select Case* affecte un exemple de valeur à l'un des types de données de base, puis affecte la variable à la propriété *Text* de l'objet *Label4* sur le formulaire. J'ai utilisé le même code dans le chapitre 3 pour traiter les choix de zones de liste. Vous pouvez utiliser ces techniques pour travailler avec les zones de liste et les types de données dans vos propres programmes.

Remarque Si votre projet comporte plusieurs formulaires, vous devez déclarer les variables d'une manière un peu différente (et les placer à un autre endroit) pour qu'elles portent sur l'ensemble du programme (c'est-à-dire pour tous les formulaires du projet). Le type de variable que vous déclarez est une variable publique ou globale. Elle est déclarée dans un *module*, un fichier spécial contenant des déclarations et des procédures qui ne sont pas associées à un formulaire particulier. Pour en savoir plus sur la création de variables publiques dans des modules, reportez-vous au chapitre 10, « Créer des modules et des procédures ».

12. Parcourez la procédure événementielle *ListBox1_SelectedIndexChanged* et examinez chaque affectation de variable.

Essayez de modifier les données de quelques instructions d'affectation de variable. Exécutez de nouveau le programme pour voir à quoi elles ressemblent. Essayez d'affecter à des variables des valeurs qui sortent de leur plage (elle figure dans le tableau des types de données précédent). Si vous commettez une telle erreur, Visual Basic souligne la valeur incorrecte d'une ligne dentelée dans l'Éditeur de code et le programme ne s'exécutera pas tant que vous ne l'aurez pas modifiée. Pour en savoir plus sur votre erreur, maintenez le pointeur de la souris au-dessus de la valeur soulignée et lisez le message d'erreur de l'infobulle concernant le problème.

Astuce Par défaut, une ligne dentelée verte indique un avertissement, une ligne dentelée rouge une erreur de syntaxe, une ligne dentelée bleue une erreur de compilation et une ligne dentelée pourpre indique un autre type d'erreur.

13. Si vous souhaitez enregistrer vos modifications, cliquez sur le bouton Enregistrer tout de la barre d'outils Standard.

Types de données personnalisés

Visual Basic permet également de créer vos propres types de données. Cette fonction est utile lorsque les données du groupe avec lequel vous travaillez s'assemblent naturellement, mais appartiennent à différentes catégories. Créez un type personnalisé à l'aide de l'instruction *Structure* et déclarez les variables associées au nouveau type à l'aide de l'instruction *Dim*. Attention : l'instruction *Structure* ne peut se situer dans une procédure événementielle : elle doit figurer au début du formulaire, avec d'autres déclarations de variables, ou dans un module de code.

Par exemple, la déclaration suivante crée un type de données personnalisé appelé *Employé*, qui peut stocker le nom, la date de naissance et la date d'embauche d'un salarié :

```
Structure Employé
    Dim Nom As String
    Dim DateDeNaissance As Date
    Dim DateEmbauche As Date
End Structure
```

Après avoir créé un type de données, vous pouvez l'utiliser dans le code pour les procédures événementielles du formulaire ou du module. Les instructions suivantes utilisent le nouveau type *Employé*. La première instruction crée une variable appelée *DirecteurProduit*, de type *Employé* et la deuxième instruction affecte le nom « Greg Baker » au composant *Nom* de la variable :

```
Dim DirecteurProduit As Employé
DirecteurProduit.Nom = "Greg Baker"
```

Cette procédure ressemble à la définition d'une propriété, n'est-ce pas ? Visual Basic utilise la même notation pour la relation entre les objets et les propriétés et pour la relation entre des types de données définis par l'utilisateur et les variables de composants.

Les constantes : des variables qui ne changent pas

Si l'une des variables de votre programme comporte une valeur immuable (comme π, une entité mathématique fixe), vous pouvez la stocker en tant que constante et non en tant que variable. Une *constante* est un nom qui remplace un nombre ou une chaîne de texte immuable. L'intérêt des constantes est qu'elles améliorent la lisibilité du code. Elles réduisent les erreurs de programmation et facilitent les modifications globales à accomplir ultérieurement. Les constantes agissent comme beaucoup de variables, mais vous ne pouvez pas modifier leur valeur pendant l'exécution. Elles sont déclarées avec le mot clé *Const*, comme dans l'exemple suivant :

```
Const Pi As Double = 3.14159265
```

Cette instruction crée une constante appelée *Pi* qui peut remplacer la valeur ¼ dans le code. Pour permettre à tous les objets et procédures événementielles du formulaire d'accéder à une constante, placez l'instruction au début du formulaire avec les autres déclarations de variables et de structure qui porteront sur l'ensemble des procédures événementielles du formulaire. Pour que tous les formulaires et modules d'un programme puissent accéder à la constante (et pas seulement *Form1*), créez la constante dans un module de code en la faisant précéder du mot clé *Public*. Par exemple :

```
Public Const Pi As Double = 3.14159265
```

L'exercice suivant illustre l'utilisation d'une constante dans une procédure événementielle.

Utiliser une constante dans une procédure événementielle

1. Dans le menu Fichier, cliquez sur Ouvrir un projet.

   La boîte de dialogue Ouvrir un projet s'affiche.

2. Ouvrez le projet Testeur de constante dans le dossier c:\vb08epe\chap05\Testeur de constante.

3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Le formulaire Testeur de constante apparaît dans le Concepteur. Il s'agit d'un programme squelette.

   L'interface utilisateur est terminée, mais vous devez saisir le code.

4. Dans le formulaire, double-cliquez sur le bouton Afficher constante.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

5. Saisissez les instructions suivantes dans la procédure événementielle *Button1_Click* :

   ```
   Const Pi As Double = 3.14159265
   Label1.Text = Pi
   ```

> **Astuce** L'emplacement de vos déclarations dépend de la façon dont vous prévoyez d'utiliser les constantes ou les variables. En général, les programmeurs limitent au maximum la portée des déclarations, mais les rendent disponibles pour le code qui doit les utiliser. Par exemple, si une constante est uniquement requise par une procédure événementielle, placez la déclaration de la constante dans cette procédure. Vous pouvez également placer la déclaration au début du code du formulaire, ce qui la rendra accessible pour toutes les procédures événementielles du formulaire.

6. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

7. Cliquez sur le bouton Afficher constante.

   La constante *Pi* apparaît dans la zone d'étiquette, comme suit :

8. Cliquez sur le bouton Quitter pour interrompre le Programme.

   Les constantes sont utiles dans le code, particulièrement lorsqu'elles sont incluses dans des formules mathématiques comme Surface = π^2. La section suivante décrit l'utilisation des opérateurs et des variables pour écrire de telles formules.

Travailler avec les opérateurs Visual Basic

Une *formule* est une instruction qui combine des nombres, des variables, des opérateurs et des mots clés pour créer une nouvelle valeur. Plusieurs éléments du langage Visual Basic sont conçus pour être utilisés dans des formules. Dans cette section, vous allez travailler avec des *opérateurs* arithmétiques (ou mathématiques), les symboles utilisés pour relier les différentes parties d'une formule. Sauf exceptions, les symboles arithmétiques que vous allez utiliser sont ceux que vous employez au quotidien et leurs opérations sont très intuitives. Chaque opérateur est illustré dans les prochains exercices.

Visual Basic propose les opérateurs arithmétiques suivants :

Opérateur	Description
+	Addition
–	Soustraction
*	Multiplication
/	Division
\	Division entière
Mod	Division modulaire
^	Élévation à une puissance
&	Concaténation de chaînes (combinaison)

Mathématiques de base : les opérateurs +, –, * et /

Les opérateurs d'addition, de soustraction, de multiplication et de division sont très simples et peuvent être utilisés dans toute formule comportant des nombres ou des variables numériques. L'exercice suivant illustre leur utilisation dans un programme.

Travailler avec les opérateurs de base

1. Dans le menu Fichier, cliquez sur Ouvrir un projet.
2. Ouvrez le projet Maths de base dans le dossier c:\vb08epe\chap05\Maths de base.
3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Le formulaire Tests de Maths de base apparaît dans le Concepteur. Le programme illustre le fonctionnement des opérateurs d'addition, de soustraction, de multiplication et de division avec les nombres que vous saisissez. Il montre également comment utiliser des objets zone de texte, case à cocher et bouton pour traiter les saisies utilisateur dans un programme.
4. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute dans l'EDI. Le programme affiche deux zones de texte pour saisir des valeurs numériques, un groupe de cases opérateur à cocher, une zone de résultats et deux objets bouton (Calculer et Quitter).
5. Tapez **100** dans la zone de texte Variable 1, puis appuyez sur la touche TAB.

   Le point d'insertion, ou *cible de saisie (focus)*, se déplace vers la deuxième zone de texte.
6. Tapez **17** dans la zone de texte Variable 2.

   Vous pouvez maintenant appliquer tous les opérateurs mathématiques aux valeurs des zones de texte.
7. Cliquez sur la case Addition, puis sur le bouton Calculer.

   L'opérateur est appliqué aux deux valeurs et le nombre 117 apparaît dans la zone Résultats, comme le montre la figure suivante :

8. Essayez les opérateurs de soustraction, de multiplication et de division avec les deux nombres des zones Variable (cliquez sur Calculer pour exécuter chaque formule).

   Les résultats apparaissent dans la zone Résultats. N'hésitez pas à tester différents nombres dans les zones de texte Variable (essayez quelques nombres décimaux si vous le souhaitez). J'ai employé le type *Double* pour déclarer les variables, vous pouvez donc utiliser de très grands nombres.

   Faites maintenant le test suivant :

9. Tapez **100** dans la zone de texte Variable 1. Tapez **0** dans la zone de texte Variable 2. Cochez la case Division, puis cliquez sur Calculer.

   La division par zéro n'est pas possible dans les calculs mathématiques, car le résultat produit est infini. Cependant, Visual Basic peut effectuer ce calcul et affiche la valeur Infini dans la zone de texte Résultats. Visual Basic 2008 gère automatiquement la division par zéro.

10. Lorsque vous avez terminé vos tests, cliquez sur le bouton Quitter.

    Le programme s'arrête et vous revenez à l'environnement de développement.

Observez maintenant le code pour voir comment les résultats ont été calculés. Maths de base utilise quelques-uns des contrôles de saisie standard que vous avez découverts au chapitre 3 et une procédure événementielle qui emploie des variables et des opérateurs pour traiter de simples formules mathématiques. Le programme déclare ses variables au début du formulaire : elles peuvent être utilisées par toutes les procédures événementielles de Form1.

Examiner le code de Math de base

1. Dans le formulaire, double-cliquez sur le bouton Calculer.

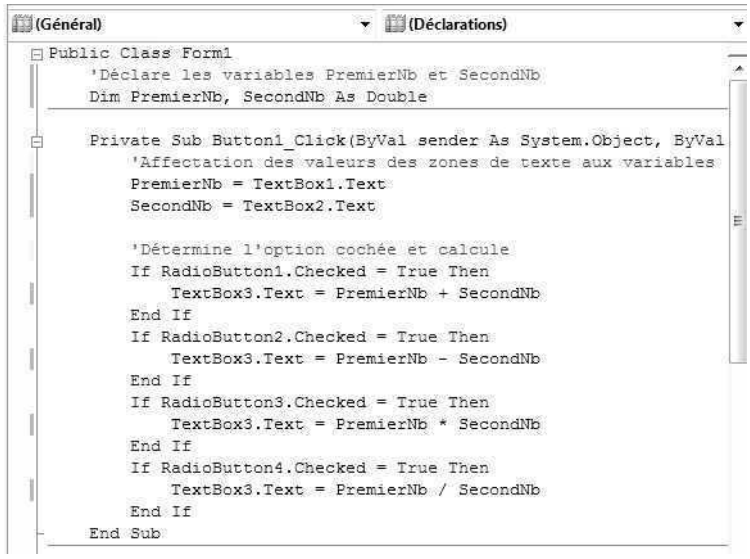
   L'Éditeur de code affiche la procédure événementielle *Button1_Click*. Au début du code du formulaire, vous trouvez l'instruction suivante, qui déclare deux variables de type *Double* :

   ```
   'Déclare les variables PremierNb et SecondNb
   Dim PremierNb, SecondNb As Double
   ```

   On utilise le type *Double* pour créer une variable polyvalente pouvant gérer différents types de nombres : entiers, décimaux, très grands nombres, petits nombres, etc. Les variables sont déclarées sur la même ligne, à l'aide de leurs notations abrégées. *PremierNb* et *SecondNb* sont de type *Double* et elles contiennent respectivement les entrées de valeurs de la première et de la deuxième zone de texte.

2. Faites défiler l'Éditeur de code pour voir le contenu de la procédure événementielle *Button1_Click*.

   Votre écran ressemble à :



Les deux premières instructions de la procédure événementielle transmettent les données saisies dans les objets zone de texte aux variables *PremierNb* et *SecondNb*.

```
'Affectation des valeurs des zones de texte aux variables
PremierNb = TextBox1.Text
SecondNb = TextBox2.Text
```

Le contrôle *TextBox* gère la transmission à la propriété *Text* (qui accepte du texte saisi par l'utilisateur et permet au programme de l'utiliser). J'utiliserai souvent le contrôle *TextBox* dans cet ouvrage. Lorsqu'il est réglé sur multiligne et redimensionné, il peut afficher plusieurs lignes de texte (voire un fichier entier !).

Une fois que les valeurs des zones de texte ont été affectées aux variables, la procédure événementielle détermine quelle option a été sélectionnée, calcule la formule mathématique et affiche le résultat dans une troisième zone de texte. Le test de la première case à cocher ressemble à

```
'Détermine l'option cochée et calcule
If RadioButton1.Checked = True Then
     TextBox3.Text = PremierNb + SecondNb
End If
```

Dans le chapitre 3, nous avons vu que dans un objet zone de groupe, un seul objet case à cocher peut être sélectionné à la fois. La propriété *Checked* vous permet de savoir si une case a été cochée ou non. Si elle est positionnée à *True*, la case a été cochée. Si elle prend la valeur *False*, la case n'a pas été sélectionnée. Après ce simple test, vous êtes prêt à calculer le résultat et à l'afficher dans le troisième objet zone de texte. C'est tout ce qu'il faut savoir à propos des opérateurs arithmétiques de base (vous en apprendrez davantage sur la syntaxe des tests *If…Then* dans le chapitre 6, « Utiliser les structures de décision »).

Vous en avez terminé avec le programme Maths de base.

Nouveaux opérateurs abrégés

Visual Basic propose des opérateurs abrégés pour les opérations mathématiques et les opérations sur les chaînes qui concernent la modification de la valeur d'une variable existante. Par exemple, en combinant le symbole + avec le symbole =, vous pouvez compléter une variable sans répéter son nom dans la formule. Ainsi, vous pouvez écrire la formule X = X + 6 avec la syntaxe X += 6. Le tableau suivant présente des exemples de ces opérateurs condensés.

Opération	Syntaxe longue	Syntaxe abrégée
Addition (+)	X = X + 6	X += 6
Soustraction (–)	X = X – 6	X –= 6
Multiplication (*)	X = X * 6	X *= 6
Division (/)	X = X / 6	X /= 6
Division entière (\)	X = X \ 6	X \= 6
Élévation à une puissance (^)	X = X ^ 6	X ^= 6
Concaténation (&)	X = X & "ABC"	X &= "ABC"

Utiliser des opérateurs avancés : \, *Mod*, ^ et &

Outre les quatre opérateurs arithmétiques de base, Visual Basic comporte quatre opérateurs avancés, responsables de la division entière (\), de la division modulaire (*Mod*), de l'élévation à la puissance (^) et de la concaténation (*&*). Ces opérateurs sont intéressants dans les formules mathématiques à usage spécial et les applications de traitement de texte. L'utilitaire suivant (une variante du programme Maths de base) illustre l'utilisation de chacun de ces opérateurs dans un programme.

Travailler avec les opérateurs avancés

1. Dans le menu Fichier, cliquez sur Ouvrir un projet.

   La boîte de dialogue Ouvrir un projet s'affiche.

2. Ouvrez le projet Math Plus dans le dossier c:\vb08epe\chap05\Math Plus.

3. Si vous ne voyez pas le formulaire du projet, cliquez sur Form1.vb dans l'Explorateur de solutions, puis sur le bouton du Concepteur de vues.

   Le formulaire Testeur de Math Plus apparaît dans le Concepteur. Ce programme est identique au programme Maths de base, à l'exception des opérateurs représentés dans les cases à cocher et dans le programme.

4. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme affiche deux zones de texte pour saisir des valeurs numériques, un groupe d'options opérateur à sélectionner, une zone de résultat et deux boutons.

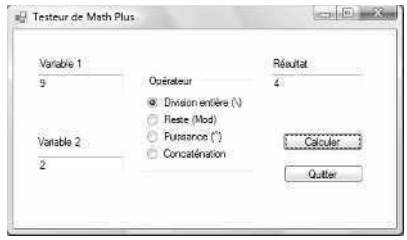
5. Tapez **9** dans la zone de texte Variable 1, puis appuyez sur la touche TAB.

6. Tapez **2** dans la zone de texte Variable 2.

   Vous pouvez maintenant appliquer tous les opérateurs mathématiques avancés aux valeurs des zones de texte.

7. Cliquez sur Division entière, puis sur le bouton Calculer.

   L'opérateur est appliqué aux deux valeurs et le nombre 4 apparaît dans la zone Résultat, comme suit :



   La division entière ne donne que le quotient entier de l'opération de division. Bien que 9 divisé par 2 égale 4,5, la division entière ne donne que la première partie du quotient, un nombre entier (4). Ce résultat peut être utile lorsque vous travaillez avec des quantités indivisibles, comme le nombre d'adultes pouvant prendre place dans une voiture.

8. Cliquez sur l'option Reste, puis sur le bouton Calculer.

   Le nombre 1 apparaît dans la zone Résultat. La division modulaire donne le reste de la division de deux nombres. 9 divisé par 2 égale 4, avec un reste de 1 (2 * 4 + 1 = 9). Le résultat fourni par l'opérateur *Mod* est donc 1. L'opérateur *Mod* peut vous aider à repérer les restes de vos calculs, comme la quantité d'argent restant après une transaction financière.

9. Cliquez sur l'option Puissance, puis sur le bouton Calculer.

   Le nombre 81 apparaît dans la zone Résultats. Le caret (^) élève un nombre à une puissance donnée. Par exemple, 9 ^ 2 correspond à 9^2, soit 81. Dans une formule Visual Basic, 9^2 s'écrit 9 ^ 2.

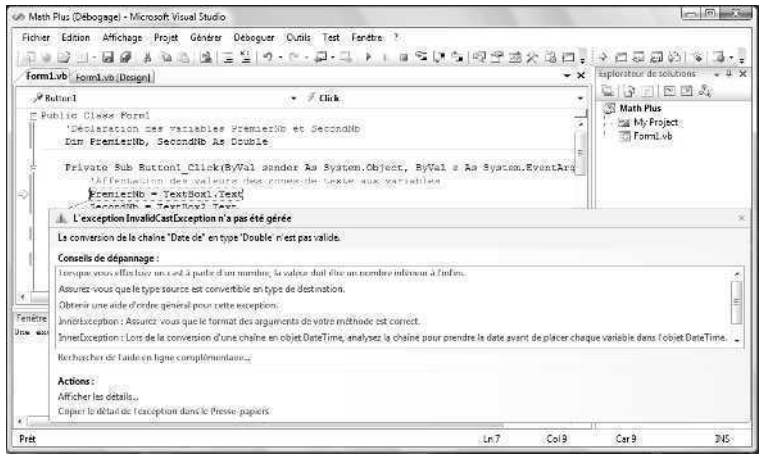
10. Cliquez sur l'option Concaténation, puis sur le bouton Calculer.

Le nombre 92 apparaît dans la zone Résultat. L'opérateur de concaténation (*&*) combine deux chaînes dans une formule, mais ne les additionne pas. Le résultat est une combinaison du caractère « 9 » et du caractère « 2 ». La concaténation peut être réalisée sur des variables numériques, mais elle est plus fréquente sur des chaînes de valeurs ou des variables.

Les variables *PremierNb* et *SecondNb* sont de type *Double* : vous ne pouvez pas combiner des mots ou des lettres à l'aide de ce code. À titre d'exemple, faites le test suivant, qui provoque une erreur et interrompt le programme.

11. Tapez **date de** dans la zone de texte Variable 1. Tapez **naissance** (en faisant précéder ce mot d'un espace) dans la zone de texte Variable 2. Cliquez sur Concaténation, puis sur Calculer.

Visual Basic ne peut traiter les valeurs textuelles que vous avez saisies. Le programme s'arrête, et un message d'erreur apparaît à l'écran :



Ce type d'erreur est appelé *erreur d'exécution*. Elle n'apparaît ni pendant la conception et ni pendant la compilation du programme, mais plus tard, lorsque le programme est en cours d'exécution et rencontre une condition qu'il ne sait pas comment traiter. Si cela vous semble étrange, imaginez-vous que Visual Basic vous propose une interprétation du vieil adage : « On ne mélange pas les serviettes et les torchons ». Le message « La conversion de la chaîne "Date" en type "Double" n'est pas valide » signifie que Visual Basic n'a pas pu convertir, ou *fondre*, les mots que vous avez saisis dans les zones de texte (« date » et « naissance ») en variables de type *Double*. Les types *Double* ne peuvent contenir que des nombres.

Comme nous le détaillerons plus tard, Visual Studio ne vous laisse pas dans l'expectative. Il affiche une boîte de dialogue avec différents types d'informations pour vous aider à résoudre l'erreur d'exécution. Vous avez appris une autre leçon importante sur les types de données et vous savez quand il ne faut pas les mélanger.

12. Dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage pour interrompre le programme.

Votre programme s'arrête et vous revenez à l'environnement de développement.

> **Remarque** Dans le chapitre 8, « Déboguer les programmes Visual Basic », vous allez découvrir le mode débogage, qui permet de repérer les erreurs, ou *bogues*, de votre code.

Observez maintenant la déclaration des variables et l'utilisation des opérateurs avancés dans le code.

13. Rendez-vous dans la partie supérieure de l'Éditeur de code.

Le commentaire et l'instruction suivante apparaissent :

```
'Déclare les variables PremierNb et SecondNb
Dim PremierNb, SecondNb As Double
```

Comme nous l'avons vu dans l'exercice précédent, *PremierNb* et *SecondNb* sont les variables qui contiennent les nombres provenant des objets *TextBox1* et *TextBox2*.

14. Passez du type de données *Double* à *String*, pour tester correctement le fonctionnement de l'opérateur de concaténation (*&*).

15. Faites défiler l'Éditeur de code vers le bas pour voir l'utilisation des opérateurs avancés dans le code.

Le code suivant apparaît :

```
'Affectation des valeurs des zones de texte aux variables
PremierNb = TextBox1.Text
SecondNb = TextBox2.Text
'Détermine l'option cochée et calcule
If RadioButton1.Checked = True Then
     TextBox3.Text = PremierNb \ SecondNb
End If
If RadioButton2.Checked = True Then
TextBox3.Text = PremierNb Mod SecondNb
End If
If RadioButton3.Checked = True Then
     TextBox3.Text = PremierNb ^ SecondNb
End If
If RadioButton4.Checked = True Then
     TextBox3.Text = PremierNb & SecondNb
End If
```

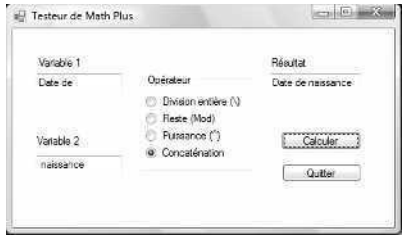
Comme le programme Maths de base, ce programme charge des données à partir des zones de texte et les place dans les variables *PremierNb* et *SecondNb*. Ensuite, le programme vérifie quelle case l'utilisateur a cochée et effectue le calcul. Cette procédure événementielle se sert des opérateurs de division entière (\), de division modulaire (*Mod*), d'élévation à une puissance (^) et de concaténation (*&*). Maintenant que vous avez positionné le type de données des variables sur *String*, exécutez

de nouveau le programme pour voir le fonctionnement de l'opérateur & sur du texte.

16. Cliquez sur le bouton Démarrer le débogage.

17. Tapez **Date de** dans la zone de texte Variable 1. Tapez « **naissance »** (sans les guillemets, mais en faisant précéder le mot d'un espace) dans la zone de texte Variable 2. Cliquez sur la case Concaténation, puis cliquez sur Calculer.

    Le programme concatène les chaînes de valeurs et ne provoque aucune erreur d'exécution :



18. Cliquez sur le bouton Quitter pour fermer le programme.

    Vous en avez terminé avec le programme Math Plus.

Astuce Il est difficile d'éviter toutes les erreurs d'exécution. Même les programmes les plus sophistiqués, comme Microsoft Word ou Excel, rencontrent parfois des conditions d'erreur qu'ils ne peuvent pas gérer. En découlent des erreurs d'exécution ou *pannes*. Concevez vos programmes de manière à gérer différents types de données et de conditions d'exécution : vos applications seront *fiables*. Dans le chapitre 9, « Gérer les erreurs avec la gestion structurée des exceptions », vous allez découvrir un autre outil de prévention des erreurs d'exécution : le gestionnaire d'erreurs.

Travailler avec des méthodes dans le .NET Framework de Microsoft

Vous devrez parfois faire d'autres manipulations de nombres dans vos programmes : arrondir un nombre, calculer une expression mathématique complexe ou introduire un caractère aléatoire dans vos programmes. Les méthodes mathématiques représentées dans le prochain tableau peuvent vous aider à manipuler des nombres dans les formules. Ces méthodes sont fournies par le .NET Framework de Microsoft, une bibliothèque de classes qui permet de profiter de la puissance du système d'exploitation Windows et d'accomplir de nombreuses tâches de programmation nécessaires à la création de vos projets. Le .NET Framework constitue une fonction essentielle de Visual Studio où elle est partagée par Visual Basic, Visual C++, Visual C# ainsi que d'autres outils de Visual Studio. Il s'agit d'une interface sous-jacente qui devient membre du système d'exploitation Windows lui-même et qui est installée sur chaque ordinateur exécutant des programmes Visual Studio.

Le .NET Framework est organisé en classes, que vous pouvez employer dans vos projets de programmation. Le processus est assez simple. Vous allez maintenant tester son fonctionnement à l'aide d'une méthode mathématique de la classe *System.Math* du .NET Framework.

Quoi de neuf dans la version 3.5 du .NET Framework de Microsoft ?

Visual Studio 2008 comporte une nouvelle version du .NET Framework : Microsoft .NET Framework 3.5. C'est une mise à jour du logiciel .NET Framework 3.0 qui prend en charge le système d'exploitation Windows Vista ainsi que du logiciel .NET Framework 2.0 qui accompagnait Visual Studio 2005 et offrait la prise en charge des processeurs 64 bits. La version 3.5 ajoute de nouvelles classes qui procurent des fonctionnalités complémentaires pour la distribution d'applications mobiles, la communication interprocessus, les opérations de fuseaux horaires, ASP.NET, Visual Web Developer et bien d'autres choses encore. Le .NET Framework 3.5 prend également en charge de nouvelles technologies avancées, comme LINQ (*Language-Integrated Query*) pour effectuer des requêtes vers différents types de données, WPF (*Windows Presentation Foundation*) pour la création d'applications graphiques complexes, WCF (*Windows Communication Foundation*) pour la création d'applications qui fonctionnent avec des services Web et WF (*Windows Workflow Foundation*) pour la création d'applications de type flux de travail. Vous découvrirez de nombreuses améliorations du .NET Framework lors de votre travail avec Visual Basic 2008. Certaines vous seront très utiles quand vous aborderez les techniques de programmation avancées.

Le tableau suivant fournit une liste partielle des méthodes mathématiques de la classe *System.Math*. L'argument *n* du tableau représente le nombre, la variable ou l'expression que vous voulez faire évaluer par la méthode. Si vous utilisez l'une de ces méthodes, veillez à placer l'instruction

```
Imports System.Math
```

au début du code de votre formulaire dans l'Éditeur de code.

Méthode	Fonction
Abs(n)	Renvoie la valeur absolue de *n*.
Atan(n)	Renvoie l'arc tangente de *n* en radians.
Cos(n)	Renvoie le cosinus de l'angle *n*. L'angle *n* est exprimé en radians.
Exp(n)	Renvoie la constante *e* élevée à la puissance *n*.
Sign(n)	Renvoie –1 si *n* est inférieur à 0, 0 si *n* égal 0 et +1 si *n* est supérieur à 0.
Sin(n)	Renvoie le sinus de l'angle *n*. L'angle *n* est exprimé en radians.
Sqrt(n)	Renvoie la racine carrée de *n*.
Tan(n)	Renvoie la tangente de l'angle *n*. L'angle *n* est exprimé en radians.

Utiliser la classe *System.Math* pour calculer des racines carrées

1. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.

2. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Math et Framework**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans le Concepteur.

3. Cliquez sur le contrôle *Button* de l'onglet Windows Form de la Boîte à outils et dessinez un objet bouton en haut de votre formulaire.

4. Dans la Boîte à outils, cliquez sur le contrôle *TextBox*. Dessinez une zone de texte en dessous de l'objet bouton.

5. Attribuez la valeur **Racine carrée** à la propriété *Text* du bouton.

6. Double-cliquez sur l'objet bouton pour afficher l'Éditeur de code.

7. Dans la partie supérieure de l'Éditeur de code, tapez l'instruction suivante au-dessus de l'instruction `Public Class Form1` :

   ```
   Imports System.Math
   ```

   La classe *System.Math* est une collection de méthodes proposées par le .NET Framework pour les opérations arithmétiques. Le .NET Framework est organisée de façon hiérarchique et peut être très complexe. L'instruction *Imports* facilite le référencement de classes, propriétés et méthodes dans votre projet. Par exemple, si vous omettez l'instruction *Imports*, pour appeler la méthode *Sqrt*, vous devez saisir *System.Math.Sqrt* plutôt que *Sqrt*.

   Cette instruction *Imports* doit être l'une des premières de votre programme. Elle doit être placée avant les variables que vous déclarez pour le formulaire et avant l'instruction `Public Class Form1` fournie automatiquement par Visual Basic. La bibliothèque particulière que vous avez choisie est la classe *System.Math*, un ensemble d'objets, de propriétés et de méthodes fournies par le .NET Framework pour les opérations mathématiques.

8. Faites défiler l'Éditeur de code vers le bas et ajoutez le code suivant entre les instructions *Private Sub* et *End Sub* de la procédure événementielle *Button1_Click :*

   ```
   Dim Résultat As Double
   Résultat = Sqrt(625)
   TextBox1.Text = Résultat
   ```

   Ces trois instructions déclarent la variable de type Double appelée *Résultat*, utilisent la méthode *Sqrt* pour calculer la racine carrée de 625 et affectent la variable *Résultat* à la propriété *Text* de l'objet zone de texte pour afficher la réponse.

9. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap05 comme emplacement.

10. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le programme Math et Framework s'exécute dans l'EDI.

11. Cliquez sur le bouton Racine carrée.

Visual Basic calcule la racine carrée de 625 et affiche le résultat (25) dans la zone de texte. La méthode *Sqrt* fonctionne !

12. Sur le formulaire, cliquez sur le bouton Fermer pour terminer le programme.

Pour faciliter le référencement de classes, propriétés et méthodes du .NET Framework, insérez l'instruction *Imports* et spécifiez l'espace de noms ou la classe adéquat. Vous pouvez utiliser cette technique pour toutes les classes du .NET Framework. Vous trouverez davantage d'exemples dans la suite de cet ouvrage.

Aller plus loin : Établir un ordre de priorité

Dans les exercices précédents, vous avez testé quelques opérateurs arithmétiques et un opérateur de concaténation. Visual Basic vous permet de mélanger autant d'opérateurs arithmétiques que vous le souhaitez dans une formule, tant que les variables numériques et les expressions sont séparées par un opérateur.

Par exemple, cette formule Visual Basic est acceptable :

```
Total = 10 + 15 * 2 / 4 ^ 2
```

La formule traite plusieurs valeurs et affecte le résultat à une variable appelée *Total*. Comment Visual Basic évalue-t-il une telle expression ? En d'autres termes, quelle séquence suit-il lorsqu'il résout la formule ? L'ordre d'évaluation joue un grand rôle dans cet exemple.

Visual Basic résout ce dilemme en établissant un *ordre de priorité* pour les opérations mathématiques. Cette liste de règles indique à Visual Basic quel opérateur utiliser en premier, en deuxième, etc., lorsqu'il évalue une expression qui comporte plusieurs opérateurs.

Le tableau suivant énumère les opérateurs dans l'ordre dans lequel ils sont évalués. Les opérateurs qui se situent au même niveau dans une expression sont évalués de la gauche vers la droite.

Opérateurs	Ordre de priorité
( )	Les valeurs entre parenthèses sont toujours évaluées en premier.
^	L'élévation à une puissance est évaluée en deuxième.
–	La négation est évaluée en troisième.
** /*	La multiplication et la division sont évaluées en quatrième.
\	La division entière est évaluée en cinquième.
Mod	La division modulaire est évaluée en sixième.
+ –	L'addition et la soustraction sont évaluées en dernier.

Vu l'ordre de priorité de ce tableau, l'expression

```
Total = 10 + 15 * 2 / 4 ^ 2
```

est évaluée par Visual Basic selon les étapes suivantes :

```
Total = 10 + 15 * 2 / 4 ^ 2
Total = 10 + 15 * 2 / 16
Total = 10 + 30 / 16
Total = 10 + 1.875
Total = 11.875
```

Utiliser des parenthèses dans une formule

Vous pouvez utiliser une ou plusieurs paires de parenthèses dans une formule pour clarifier l'ordre de priorité. Par exemple, Visual Basic calcule la formule

```
Nombre = (8 - 5 * 3) ^ 2
```

en déterminant la valeur entre parenthèses (–7) avant d'élever à la puissance (bien que l'ordre de priorité place l'élévation à une puissance avant la soustraction et la multiplication). Vous pouvez encore affiner le calcul en plaçant des parenthèses imbriquées dans la formule. Par exemple,

```
Nombre = ((8 - 5) * 3) ^ 2
```

Visual Basic calcule d'abord la différence dans la paire intérieure de parenthèses, effectue ensuite l'opération dans la paire extérieure de parenthèses, puis élève le résultat à la puissance 2. Le résultat des deux formules est différent : la première formule donne 49 et la deuxième 81.

Rappel du chapitre 5

Pour	Faites ceci
Déclarer une variable	Dans le code, tapez *Dim*, le nom de la variable, le mot clé *As* et le type de variable. Pour que la variable soit valable dans toutes les procédures événementielles d'un formulaire, placez cette instruction au début du code du formulaire, avant les procédures événementielles. Par exemple : `Dim Country As String`
Modifier la valeur d'une variable	Affectez une nouvelle valeur à l'aide de l'opérateur d'affectation (=). Par exemple : `Pays = "Japon"`
Recueillir des entrées à l'aide d'une boîte de dialogue	Utilisez la fonction *InputBox* et affectez le résultat à une variable. Par exemple : `NomUtilisateur = InputBox("Quel est votre nom ?")`

Afficher un résultat dans une boîte de dialogue	Utilisez la fonction *MsgBox* (la chaîne qui doit être affichée dans la boîte de dialogue peut être stockée dans une variable). Par exemple :
	`Prévision = "Pluie, principalement en plaine."`
	`MsgBox(Prévision, , "Rapport météorologique pour l'Espagne")`

Créer une constante	Tapez le mot clé *Const*, le nom de la constante, l'opérateur d'affectation (=), le type de données de la constante et la valeur fixe. Par exemple : `Const AgeJackBennys As Short = 39`
Créer une formule	Liez des variables numériques ou des valeurs à l'aide de l'un des sept opérateurs arithmétiques, puis affectez le résultat à une variable ou à une propriété. Par exemple : `Résultat = 1 ^ 2 * 3 \ 4 ' Ceci est égal à 0`
Combiner des chaînes textuelles	Utilisez l'opérateur de concaténation (*&*). Par exemple : `Msg = "Hello" & "," & " world!"`
Faciliter le référencement d'une bibliothèque de classes à partir du .NET Framework	Placez une instruction *Imports* au tout début du code du formulaire pour identifier la bibliothèque de classes. Par exemple : `Imports System.Math`
Appeler une méthode à partir d'une bibliothèque de classes insérée	Utilisez le nom de la méthode et insérez tous les arguments nécessaires pour qu'elle soit utilisée dans une formule ou une instruction. Par exemple, pour appeler la méthode *Sqrt* dans la bibliothèque de classes *System.Math :* `Hypoténuse = Sqrt(x ^ 2 + y ^ 2)`

Contrôler l'ordre d'évaluation dans une formule	Utilisez les parenthèses dans la formule. Par exemple :
	`Résultat = 1 + 2 ^ 3 \ 4 ' Ceci est égal à 3`
	`Résultat = (1 + 2) ^ ( 3 \ 4) ' Ceci est égal à 1`

Chapitre 6

Exploiter des structures de décision

À la fin de ce chapitre, vous saurez :

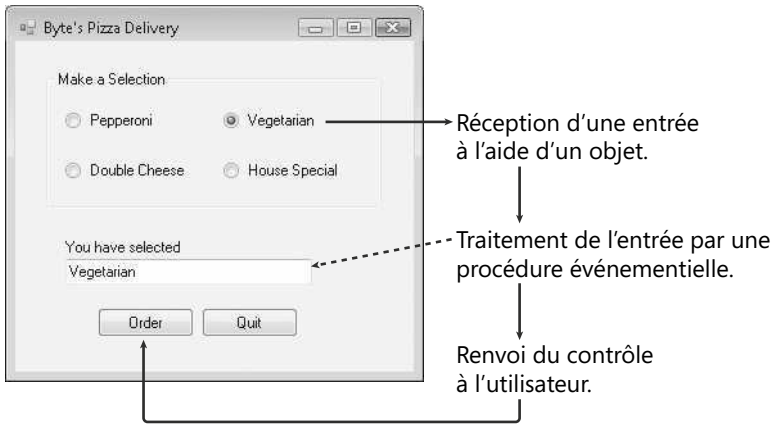
- Rédiger des expressions conditionnelles
- Exploiter une instruction *If…Then* pour se brancher sur un ensemble d'instructions selon sur une condition variable
- Exploiter le contrôle *MaskedTextBox* pour recevoir une entrée utilisateur dans un format spécifique
- Court-circuiter une instruction *If…Then*
- Exploiter une instruction *Select Case* pour effectuer un choix parmi plusieurs options dans le code
- Exploiter la propriété *Name* pour renommer des objets dans un programme
- Gérer des événements de la souris et rédiger un gestionnaire d'événements *MouseHover*

Dans les précédents chapitres, vous avez utilisé plusieurs fonctionnalités de Visual Basic 2008 pour gérer les entrées utilisateur. Vous avez exploité des menus, des barres d'outils, des boîtes de dialogue et d'autres contrôles de la Boîte à outils pour afficher les choix de l'utilisateur. Vous avez traité des entrées grâce à des paramètres de propriétés, des variables, des opérateurs, des formules et le Microsoft .Net Framework.

Dans ce chapitre, vous allez apprendre comment vous connecter de manière conditionnelle à une zone spécifique de votre programme en fonction de l'entrée utilisateur. Vous allez également découvrir comment évaluer une ou plusieurs propriétés ou variables grâce aux expressions conditionnelles, puis comment exécuter une ou plusieurs instructions en fonction des résultats obtenus. Vous allez en définitive accroître votre vocabulaire de programmation en créant en interne des blocs de code appelés *structures de décision*, également appelés *flots*, qui contrôlent la manière dont votre programme s'exécute.

Programmation pilotée par les événements

Les programmes que vous avez écrits jusqu'à présent dans ce livre affichaient à l'écran des contrôles de la Boîte à outils, des menus, des barres d'outils et des boîtes de dialogue ; grâce à ces programmes, les utilisateurs manipulent à l'envi les éléments de l'écran. L'utilisateur devient acteur ; le programme attend patiemment une réponse puis traite l'entrée comme prévu. Dans le milieu de la programmation, cette méthodologie est connue comme *programmation pilotée par les événements*. Vous construisez un programme en créant un groupe d'objets « intelligents » qui savent comment répondre lorsqu'un utilisateur interagit avec eux. Le programme traite ensuite l'entrée en utilisant des procédures événementielles associées aux objets. Le diagramme suivant illustre comment un programme piloté par les événements fonctionne dans Visual Basic.



Les entrées d'un programme peuvent également provenir du système lui-même. Votre programme peut, par exemple, être informé de l'arrivée d'une partie d'un courriel ou de l'écoulement d'une durée spécifique sur l'horloge système. C'est l'ordinateur, et non l'utilisateur, qui déclenche ces événements. Indépendamment de la manière dont un événement est déclenché, Visual Basic réagit en appelant la procédure événementielle associée à l'objet qui a reconnu l'événement. Jusqu'à présent, vous avez essentiellement expérimenté les événements *Click*, *CheckedChanged* et *SelectedIndexChanged*. Toutefois, les objets Visual Basic peuvent également répondre à d'autres types d'événements.

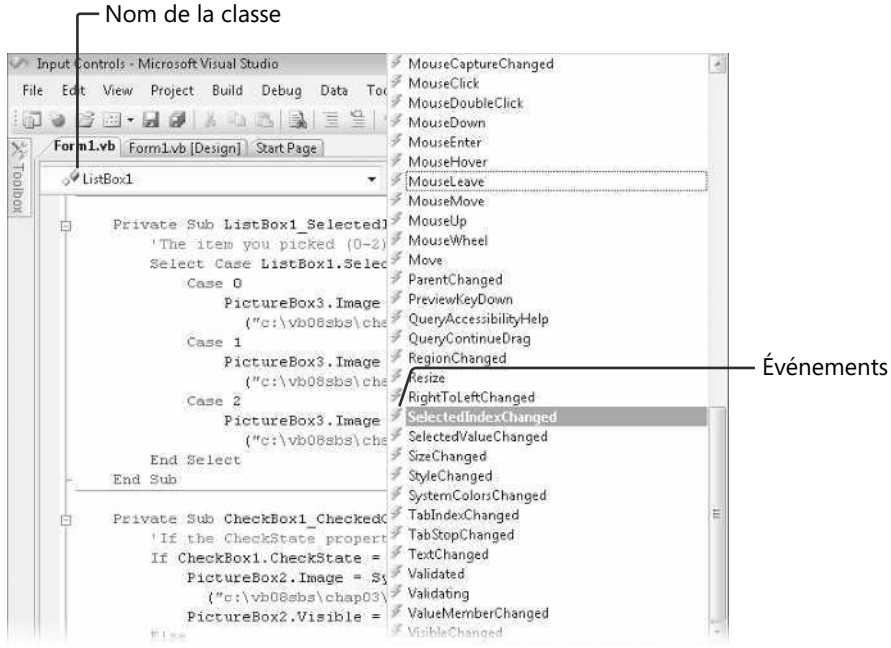
Le fait que Visual Basic soit piloté par les événements signifie que la plupart des calculs réalisés dans vos programmes s'accomplissent *via* des procédures événementielles. Ces blocs de code propres à des événements traitent les entrées, calculent de nouvelles valeurs, affichent les sorties et gèrent d'autres tâches.

Dans ce chapitre, vous allez apprendre à utiliser des structures de décision pour comparer des variables, des propriétés et des valeurs et à exécuter une ou plusieurs instructions en fonction des résultats. Au chapitre 7 « Utiliser les boucles et les minuteurs », vous allez utiliser des boucles pour exécuter à plusieurs reprises un groupe d'instructions jusqu'à ce

qu'une condition soit remplie ou tant qu'une condition spécifique est vraie. Ensemble, ces puissantes structures de contrôle de flot vous seront utiles pour construire vos procédures événementielles de sorte à répondre à la plupart des situations.

Événements pris en charge par les objets Visual Basic

Dans Visual Basic, chaque objet possède un jeu prédéfini d'événements auxquels il peut répondre. Ces événements sont listés lorsque vous sélectionnez un nom d'objet dans la zone de liste Nom de la classe en haut de l'Éditeur de code, puis que vous cliquez sur la flèche Nom de la méthode. (Dans Visual Studio, les événements sont identifiés visuellement par l'icône d'un éclair). On peut rédiger une procédure événementielle pour chacun de ces événements et si un événement se produit dans le programme, Visual Basic exécutera la procédure événementielle qui lui est associée. Par exemple, un objet zone de liste prend en charge plus de 60 événements, dont *Click*, *DoubleClick*, *DragDrop*, *DragOver*, *GotFocus*, *KeyDown*, *KeyPress*, *KeyUp*, *LostFocus*, *MouseDown*, *MouseMove*, *MouseUp*, *MouseHover*, *TextChanged* et *Validated*. Vous n'aurez sans doute pas besoin de rédiger le code pour plus de trois ou quatre de ces événements dans vos applications, mais il est bon de connaître l'éventail des choix disponibles lorsque vous créez les éléments de votre interface. L'illustration qui suit représente un listing partiel des événements destinés à un objet zone de liste dans l'Éditeur de code.



Expressions conditionnelles

Pour traiter les informations d'une procédure événementielle, l'expression conditionnelle est l'un des outils les plus utiles. Une *expression conditionnelle* est une partie d'une instruction de programme complète qui pose une question de type True ou False à propos d'une propriété, d'une variable ou de toute autre information contenue dans le code du programme. Par exemple, l'expression conditionnelle

```
Prix < 100
```

retourne True si la variable *Prix* contient une valeur inférieure à 100 et False si le *Prix* contient une valeur supérieure ou égale à 100.

Voici les opérateurs de comparaison qui peuvent être employés dans une expression conditionnelle.

Opérateur de comparaison	Signification
=	Égal à
<>	Différent de
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à

Le tableau qui suit montre quelques expressions conditionnelles et leurs résultats. Vous travaillerez à plusieurs reprises avec des expressions conditionnelles au cours de ce chapitre.

Expression conditionnelle	Résultat
`10 <> 20`	True (10 n'est pas égal à 20)
`Résultat < 20`	True si *Résultat* est inférieur à 20 ; sinon, False
`Résultat = Label1.Text`	True si la propriété *Text* de l'objet *Label1* contient la même valeur que la variable *Résultat* ; sinon, False
`TextBox1.Text = "Bill"`	True si le mot « Bill » se trouve dans la zone de texte *TextBox1* ; sinon, False

Structures de décision If…Then

Lorsqu'une expression conditionnelle est utilisée dans un bloc spécial d'instructions appelé *structure de décision*, elle vérifie si d'autres instructions de votre programme s'exécutent et dans quel ordre. Vous pouvez recourir à une structure de décision *If…Then* pour évaluer une condition dans le programme et entreprendre une action en fonction du résultat. Dans sa forme la plus simple, une structure de décision *If…Then* s'écrit sur une seule ligne :

```
If condition Then instruction
```

où *condition* est une expression conditionnelle et *instruction* est une instruction valide d'un programme Visual Basic. Par exemple,

```
If Résultat >= 20 Then Label1.Text = "Vous avez gagné !"
```

est une structure de décision *If…Then* qui utilise l'expression conditionnelle

```
Résultat >= 20
```

pour déterminer si le programme doit définir la propriété *Text* de l'objet objet *Label1* à « Vous avez gagné ! ». Si la variable *Résultat* contient une valeur supérieure ou égale à 20, Visual Basic définit la propriété *Text* ; sinon, il saute l'instruction d'assignation et exécute la ligne suivante dans la procédure événementielle. Ce type de comparaison se traduit toujours par une valeur True ou False. Une expression conditionnelle ne peut jamais donner un « peut-être ».

Tester plusieurs conditions dans une structure de décision If…Then

Visual Basic prend également en charge une structure de décision *If…Then* que l'on peut exploiter pour inclure plusieurs expressions conditionnelles. Ce bloc d'instructions peut contenir plusieurs lignes ainsi que des mots clés importants comme *ElseIf*, *Else* et *End If*.

```
If condition1 Then
    instructions exécutées si condition1 est True
ElseIf condition2 Then
    instructions exécutées si condition2 est True
[On peut placer ici d'autres conditions ElseIf et instructions]
Else
    instructions exécutées si aucune condition n'est True
End If
```

Dans cette structure, *condition1* est d'abord évaluée. Si cette expression conditionnelle est True, le bloc d'instructions situé en dessous s'exécute, instruction par instruction. (Vous pouvez inclure une ou plusieurs instructions de programme.) Si la première condition n'est pas True, on évalue alors la seconde expression conditionnelle (*condition2*). Si la seconde condition est True, on exécute le deuxième bloc d'instructions. (On peut ajouter des conditions *ElseIf* et des instructions supplémentaires s'il existe d'autres conditions à

évaluer.) Si aucune expression conditionnelle n'est True, les instructions en dessous du mot clé *Else* s'exécutent. Enfin, la structure globale est clôturée par les mots clés *End If*.

Le code qui suit montre comment une structure *If…Then* avec plusieurs lignes peut servir à déterminer le montant de l'impôt à payer dans une déclaration d'impôt sur le revenu tout à fait fictive qui fonctionnerait de manière progressive.

```
Dim RevenuImposable, ImpôtDû As Double
RevenuImposable = 50000

If RevenuImposable <= 7825 Then          'tranche d'imposition à 10 %
ImpôtDû = RevenuImposable * 0.1
ElseIf RevenuImposable <= 31850 Then     'tranche d'imposition à 15 %
ImpôtDû = 782.5 + ((RevenuImposable - 7825) * 0.15)
ElseIf RevenuImposable <= 77100 Then     'tranche d'imposition à 25 %
ImpôtDû = 4386.25 + ((RevenuImposable - 31850) * 0.25)
ElseIf RevenuImposable <= 160850 Then    'tranche d'imposition à 28 %
ImpôtDû = 15698.75 + ((RevenuImposable - 77100) * 0.28)
ElseIf RevenuImposable <= 349700 Then    'tranche d'imposition à 33 %
ImpôtDû = 39148.75 + ((RevenuImposable - 160850) * 0.33)
Else                                     'tranche d'imposition à 35 %
ImpôtDû = 101469.25 + ((RevenuImposable - 349700) * 0.35)
End If
```

Important Dans vos instructions *If…Then* et *ElseIf*, l'ordre des expressions conditionnelles est primordial. Dans l'exemple sur le calcul d'impôt, que se passerait-t-il si vous inversiez l'ordre des expressions conditionnelles et que dans la structure, les taux d'imposition apparaissent du plus élevé au plus bas ? Les contribuables situés dans les tranches d'imposition des 10, 15, 25, 28 et 33 pour cent seraient tous placés dans la tranche d'imposition des 35 pour cent car ils possèdent tous un revenu inférieur ou égal à 349 700. (Visual Basic s'arrête à la première expression conditionnelle qui est True, même si d'autres le sont également.) Comme toutes les expressions conditionnelles de cet exemple testent la même variable, elles doivent intervenir par ordre croissant afin que les contribuables soient placés au bon endroit. En définitive : lorsque vous utilisez plusieurs expressions conditionnelles, soyez attentif à leur ordre.

Cette structure de décision est fort intéressante pour tester la variable à double précision *RevenuImposable* au premier niveau de revenu et aux niveaux de revenus suivants jusqu'à ce qu'une des expressions conditionnelles retourne True, puis détermine en conséquence l'impôt sur le revenu du contribuable. Grâce à quelques modifications simples, on peut l'exploiter pour calculer l'impôt dû pour tout contribuable dans un système d'imposition progressif, comme celui qui s'applique aux États-Unis. Tant que les taux d'imposition sont complets et à jour et que la valeur de la variable *RevenuImposable* est correcte, ce programme retournera l'impôt correct dû par les contribuables célibataires nord-américains pour 2007. Si les taux d'imposition changent, il est très simple d'actualiser les expressions conditionnelles. Grâce à une structure de décision supplémentaire qui détermine la catégorie de contribuable, il est très simple d'étendre le programme pour les y inclure tous.

Astuce Les expressions qui peuvent être évaluées comme True ou False sont connues sous le nom d'*expressions booléennes*. Le résultat True ou False peut être affecté à une variable ou une propriété booléenne. Des valeurs booléennes peuvent être affectées à certaines propriétés d'objet ou à des variables booléennes créées grâce à l'instruction *Dim* et aux mots clés *As Boolean*.

Au prochain exercice, vous allez utiliser une structure de décision *If…Then* qui reconnaît l'utilisateur au moment où il ouvre le programme – ce qui représente une manière simple de commencer à rédiger vos propres structures de décision. Vous apprendrez également à exploiter le contrôle *MaskedTextBox* pour recevoir des entrées utilisateur dans un format spécifique.

Validation des utilisateurs avec If…Then

1. Démarrez Visual Studio et créez un nouveau projet Application Windows Forms appelé **Ma Validation Utilisateur**.

   Le nouveau projet est créé et un formulaire vide s'affiche dans le Concepteur.

2. Cliquez sur le formulaire et définissez la propriété *Text* du formulaire à « Validation Utilisateur ».

3. Utilisez le contrôle *Label* pour créer l'étiquette de votre formulaire et utilisez la Fenêtre de propriétés pour définir la propriété *Text* à « Tapez votre numéro de sécurité sociale ».

4. Utilisez le contrôle *Button* pour créer un bouton sur votre formulaire et définissez la propriété *Text* du bouton à « S'inscrire ».

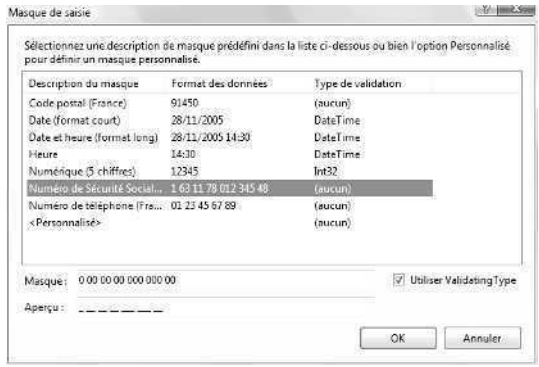
5. Dans la Boîte à outils, cliquez sur le contrôle *MaskedTextBox* de l'onglet Contrôles communs, puis créez un objet zone de texte masqué sur votre formulaire, en dessous de l'étiquette.

   Le contrôle *MaskedTextBox* ressemble au contrôle *TextBox* que vous avez déjà utilisé. Toutefois, *MaskedTextBox* permet de contrôler le format des informations saisies par l'utilisateur dans votre programme. Pour ce faire, il faut définir la propriété *Mask* ; il est possible d'exploiter un format prédéfini fourni par le contrôle ou de choisir votre propre format. Dans ce programme, vous utiliserez le contrôle *MaskedTextBox* pour demander aux utilisateurs de saisir un numéro de sécurité sociale au format standard à quinze chiffres.

6. Après avoir sélectionné l'objet *MaskedTextBox1,* cliquez sur la propriété *Mask* dans la Fenêtre de propriétés, puis cliquez sur le bouton adjacent (les trois points).

   La boîte de dialogue Masque de saisie s'affiche, présentant une liste de vos modèles prédéfinis de formats, ou *masques*.

7. Cliquez sur Numéro de sécurité sociale dans la liste. Voici la boîte de dialogue Masque de saisie :



Même si vous n'allez pas l'utiliser immédiatement, prenez un instant pour observer l'option <Personnalisé>, que vous utiliserez ultérieurement pour créer vos propres masques de saisie en utilisant des nombres et des caractères de séparation, comme par exemple des traits d'union (-).

8. Cliquez sur OK pour accepter Numéro de sécurité sociale comme masque de saisie.

   Visual Studio affiche votre masque de saisie dans l'objet *MaskedTextBox1*, comme le montre la figure qui suit.

9. Double-cliquez sur le bouton S'inscrire.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

10. Tapez les instructions suivantes dans la procédure événementielle :

```
If MaskedTextBox1.Text = "1 85 08 22 274 123 04" Then
     MsgBox("Bienvenue dans le système !")
Else
     MsgBox("Numéro inconnu")
End If
```

Cette simple structure de décision *If…Then* vérifie la valeur de la propriété *Text* de l'objet *MaskedTextBox1*. Si elle est égale à « 1 85 08 22 274 123 04 », la structure affiche le message « Bienvenue dans le système ». Si le numéro saisi par l'utilisateur correspond à une autre valeur, la structure affiche le message « Numéro inconnu ». Toutefois, la beauté de ce programme réside dans la manière dont l'objet *MaskedTextBox1* filtre automatiquement la saisie pour garantir que son format est correct.

11. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap06 comme emplacement.

12. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le programme s'exécute dans l'environnement de développement. Le formulaire invite l'utilisateur à saisir un numéro de sécurité sociale au format approprié et affiche des traits de soulignement et des tirets pour présenter à l'utilisateur l'agencement souhaité.

13. Tapez **abcd** pour tester le masque de saisie.

Visual Basic n'autorise pas l'affichage des lettres car celles-ci ne répondent pas au format demandé. Il faut un numéro de sécurité sociale à quinze chiffres.

14. Tapez **1234567890123456** pour tester le masque de saisie.

Visual Basic affiche le numéro 1 23 45 67 890 123 45 dans la zone de texte masquée en ignorant le seizième chiffre tapé. De nouveau, Visual Basic a forcé la saisie de l'utilisateur au format adéquat. Votre formulaire présente un résultat similaire à

15. Cliquez sur le bouton S'inscrire.

Visual Basic affiche le message « Numéro inconnu » car le numéro de sécurité sociale ne correspond pas à celui recherché par la structure de décision *If…Then*.

16. Cliquez sur OK, supprimez le numéro de sécurité sociale de la zone de texte masqué, saisissez le numéro **1 85 08 22 274 123 04** puis cliquez de nouveau sur le bouton S'inscrire.

    Cette fois, la structure de décision reconnaît le numéro et affiche le message de bienvenue. Le message suivant apparaît :

    Votre code a empêché un utilisateur non autorisé d'utiliser le programme et vous avez exigé une saisie spécifique afin de contrôler la saisie utilisateur.

17. Quittez le programme.

Opérateurs logiques dans les expressions conditionnelles

Vous pouvez tester plusieurs expressions conditionnelles dans des clauses *If…Then* et *ElseIf* pour inclure plusieurs critères de sélection dans votre structure de décision. Les conditions supplémentaires sont liées grâce à un ou plusieurs opérateurs logiques apparaissant dans le tableau suivant.

Opérateur logique	Signification
And	Si les deux expressions conditionnelles sont True, alors le résultat est True.
Or	Si une des deux expressions conditionnelles est True, alors le résultat est True.
Not	Si l'expression conditionnelle est False, alors le résultat est True. Si l'expression conditionnelle est True, alors le résultat est False.
Xor	Si une et seulement une des expressions conditionnelles est True, alors le résultat est True. Si les deux sont True ou False, alors le résultat est False. (Xor signifie Ou exclusif.)

Astuce Lorsque votre programme évalue une expression complexe contenant plusieurs types d'opérateurs différents, il évalue d'abord les opérateurs mathématiques, puis les opérateurs de comparaison et enfin les opérateurs logiques.

Le tableau qui suit répertorie des exemples d'opérateurs logiques à l'œuvre. Dans les expressions, on suppose que la variable de chaîne *Véhicule* contient la valeur « Vélo » et que la variable entière *Prix* contient la valeur 200.

Expression logique	Résultat
`Vehicule = "Vélo" And Prix < 300`	True (les deux conditions sont True)
`Véhicule = "Voiture" Or Prix < 500`	True (une condition est True)
`Not Prix < 100`	True (la condition est False)
`Vehicule = "Vélo" Xor Prix < 300`	False (les deux conditions sont True)

Dans l'exercice suivant, vous allez modifier le programme Validation Utilisateur pour demander à l'utilisateur de saisir un numéro d'identification personnel (PIN, *Personal Identification Number*) au cours du processus de validation. Pour ce faire, vous allez ajouter une deuxième zone de texte destinée au PIN de l'utilisateur, puis vous allez modifier la clause *If…Then* dans la structure de décision de sorte qu'elle utilise l'opérateur *And* pour vérifier le PIN.

Ajouter une protection par mot de passe grâce à l'opérateur *And*

1. Affichez le formulaire Validation Utilisateur et utilisez le contrôle *Label* pour ajouter une deuxième étiquette descriptive en dessous de la première zone de texte masqué.
2. Définissez la propriété *Text* de la nouvelle étiquette à « PIN ».
3. Ajoutez un deuxième contrôle *MaskedTextBox* sous la première zone de texte masqué et la nouvelle étiquette.
4. Cliquez sur la flèche de raccourci de l'objet *MaskedTextBox2* pour ouvrir la liste Tâches MaskedBoxText, puis sur la commande Définir le masque pour afficher la boîte de dialogue Masque de saisie.
5. Cliquez sur le masque de saisie Code postal (5 chiffres) puis cliquez sur OK.

   À l'instar de nombreux codes PIN que l'on trouve en ligne, ce code PIN contient 5 chiffres. Encore une fois, si l'utilisateur tape un mot de passe de longueur ou de format différents, celui-ci est rejeté.
6. Double-cliquez sur le bouton S'inscrire pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.
7. Modifiez la procédure événementielle afin qu'elle contienne le code suivant :

```
If MaskedTextBox1.Text = "1 85 08 22 274 123 04" _
And MaskedTextBox2.Text = "54321" Then
     MsgBox("Bienvenue dans le système !")
Else
     MsgBox("Numéro inconnu")
End If
```

L'instruction contient désormais l'opérateur logique *And* qui requiert que le code PIN de l'utilisateur corresponde à son numéro de sécurité sociale avant que l'utilisateur soit admis dans le système. (Dans ce cas, le code PIN valide est le 54321 ; dans un programme réel, cette valeur ainsi que le numéro de sécurité sociale doivent provenir d'une base de données sécurisée). J'ai modifié le programme précédent en ajoutant un caractère de suite à la fin de la première ligne, ainsi qu'une deuxième ligne commençant par *And*.

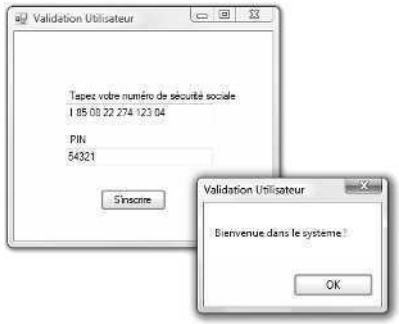
8. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute dans l'environnement de développement.

9. Tapez **1 85 08 22 274 123 04** dans la zone de texte masqué Numéro de sécurité sociale.

10. Tapez **54321** dans la zone de texte masquée PIN.

11. Cliquez sur le bouton S'inscrire. L'utilisateur est le bienvenu dans le programme, comme le montre la figure suivante.



12. Cliquez sur OK pour fermer la boîte de message.

13. Essayez d'autres valeurs pour le numéro de sécurité sociale et le code PIN.

    Testez attentivement le programme pour vous assurer que le message de bienvenue ne s'affiche pas avec d'autres codes PIN ou numéros de sécurité sociale.

14. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire. Le programme se ferme et vous revenez à l'environnement de développement.

Astuce Il est possible de personnaliser plus avant ce programme en utilisant la propriété *PasswordChar* dans les objets zone de texte masqué pour afficher une marque de réservation telle qu'un astérisque (*) au moment de la saisie. (Vous spécifiez ce caractère grâce à la fenêtre de propriétés.) Cela procure aux utilisateurs une confidentialité accrue au moment de la saisie de leur mot de passe protégé. Il s'agit d'une fonctionnalité standard pour ce type d'opérations.

Établir un court-circuit avec les opérateurs *AndAlso* et *OrElse*

Visual Basic propose deux opérateurs logiques à utiliser dans vos instructions conditionnelles, *AndAlso* et *OrElse*. Ces opérateurs fonctionnent de la même manière que *And* et *Or* respectivement, mais avec plus de subtilité dans leur mode d'évaluation, fait nouveau pour les programmeurs habitués à Visual Basic 6.

Imaginons une instruction *If* avec deux conditions connectées par un opérateur *AndAlso*. Pour que les deux instructions de la structure *If* s'exécutent, il faut que les deux conditions soient évaluées en True. Si la première condition s'évalue en False, Visual Basic passe à la ligne suivante ou immédiatement à l'instruction *Else*, sans tester la deuxième condition. Cette évaluation partielle, ou *court-circuit*, d'une instruction *If*, est logique. En effet, pourquoi Visual Basic continuerait-il à évaluer l'instruction *If* si les deux conditions ne peuvent pas être vraies ?

L'opérateur *OrElse* fonctionne de façon similaire. Considérons une instruction *If* avec deux conditions connectées par un opérateur *OrElse*. Pour que les instructions de la structure *If* s'exécutent, il faut qu'au moins une condition soit évaluée à True. Si la première condition s'évalue à True, Visual Basic commence immédiatement à exécuter les instructions de la structure *If*, sans tester la deuxième condition.

Voici un exemple de court-circuit dans Visual Basic. Il s'agit d'une routine qui exploite une instruction *If* et un opérateur *AndAlso* pour tester deux conditions et qui affiche le message « Dans If » si les deux conditions sont True.

```
Dim Nombre As Integer = 0
If Nombre = 1 AndAlso MsgBox("Test de la deuxième condition") Then
    MsgBox("Dans If")
Else
    MsgBox("Dans Else")
End If
```

La fonction *MsgBox* sert elle-même de deuxième test conditionnel, ce qui est en quelque sorte inhabituel, même si cette étrange syntaxe est tout à fait valide et nous offre une parfaite occasion d'observer de près le fonctionnement d'un court-circuit. Le texte « Test de la deuxième condition » s'affiche dans une boîte de message uniquement si la variable *Nombre* est définie à 1 ; sinon, l'opérateur *AndAlso* court-circuite l'instruction *If* et la deuxième condition n'est pas évaluée. Si vous testez réellement ce code, gardez à l'esprit qu'il ne s'agit que d'un exemple ; n'utilisez pas *MgsBox* avec cette syntaxe comme test car en réalité, elle ne teste rien. Cependant, en remplaçant dans la variable *Nombre* 0 par 1 et inversement, vous pouvez mieux comprendre la façon dont l'instruction *AndAlso* et les courts-circuits fonctionnent.

Voici un deuxième exemple de fonctions de court-circuitage dans Visual Basic lors de l'évaluation de deux conditions avec l'opérateur *AndAlso*. Vous employez cette fois un

test conditionnel plus complexe (`7 / AgeHumain <= 1`) après l'opérateur *AndAlso* pour déterminer ce que certains appellent « l'âge canin » d'une personne.

```
Dim AgeHumain As Integer
AgeHumain = 7
'Un an pour un chien correspond à sept ans pour un homme
    If AgeHumain <> 0 AndAlso 7 / AgeHumain <= 1 Then
MsgBox("Votre âge canin est d'au moins un an")
Else
    MsgBox("Vous avez moins d'un an canin")
End If
```

Dans un programme plus complet qui déterminerait l'âge canin supposé d'une personne en divisant son âge réel par 7, cette petite routine tente de définir si la valeur de la variable entière *AgeHumain* est d'au moins 7. Si vous n'avez encore jamais entendu parler du concept « d'âge canin », un peu d'indulgence – suivant cette logique, une personne de 28 ans aurait quatre ans en âge canin. Il s'agit là d'une manière intéressante d'établir un rapport entre les humains et les chiens car leur durée de vie correspond approximativement à un septième de celle des hommes. Ce code exploite deux conditions de l'instruction *If* et peut être utilisé dans une multitude de contextes différents – je l'ai utilisé dans la procédure événementielle *Click* d'un objet bouton. La première condition vérifie que l'on a placé un nombre différent de zéro dans la variable *AgeHumain* – je suis parti provisoirement du principe que l'utilisateur possède suffisamment de bon sens pour définir un âge positif dans *AgeHumain* car un nombre négatif générerait des résultats incorrects. La deuxième condition teste si la personne a au moins sept ans. Si les deux conditions évaluent True, le message « Vous avez au moins un an canin » s'affiche dans la boîte de message. Si la personne a moins de sept ans, le message « Vous avez moins d'un an canin » s'affiche.

Supposons maintenant que l'on ait remplacé la valeur 7 de la variable *AgeHumain* par 0. Que se passe-t-il ? Le compilateur Visual Basic évalue la première condition de l'instruction *If* comme False. Cette évaluation empêche que la deuxième condition soit évaluée, ce qui interrompt, ou court-circuite, l'instruction *If* et nous met à l'abri d'une fâcheuse erreur de « division par zéro » à laquelle nous pouvons être confrontés si nous divisons 7 par 0 (la nouvelle valeur de la variable *AgeHumain*). Dans Visual Basic 6, nous n'aurions pas été aussi chanceux. La définition de la variable *AgeHumain* à 0 aurait provoqué une erreur d'exécution et un arrêt du programme, car l'ensemble de l'instruction *If* aurait été évalué et la division par zéro n'est pas autorisée dans Visual Basic 6. Dans Visual Studio, nous pouvons tirer profit des avantages du court-circuitage.

En résumé, dans Visual Basic, les opérateurs *AndAlso* et *OrElse* ouvrent la porte à de nouvelles possibilités pour les programmeurs, dont celle de prévenir les erreurs d'exécution et autres résultats inattendus. Il est également possible d'améliorer la performance en plaçant des conditions longues à calculer à la fin de l'instruction conditionnelle car Visual Basic n'effectue pas ces calculs de condition gourmands tant que cela n'est pas nécessaire. Vous devez toutefois réfléchir attentivement à toutes les conditions possibles que vos instructions *If* sont susceptibles de rencontrer en fonction du changement d'état des variables pendant l'exécution du programme.

Structures de décision Select Case

Avec Visual Basic, il est également possible de contrôler l'exécution des instructions dans vos programmes grâce aux structures de décision *Select Case*. Aux chapitres 3 et 5 de ce livre, vous avez utilisé les structures *Select Case* pour rédiger des procédures événementielles destinées à traiter des sélections de zone de liste et de liste déroulante. Une structure *Select Case* ressemble à une structure *If…Then…ElseIf*, mais elle est plus efficace lorsque le branchement dépend d'une variable clé, ou *cas de test*. Vous pouvez également utiliser des structures *Select Case* pour améliorer la lisibilité de votre code.

Voici la syntaxe d'une structure *Select Case* :

```
Select Case variable
    Case valeur1
        instructions exécutées si valeur1 correspond à variable
    Case valeur2
        instructions exécutées si valeur2 correspond à variable
    Case valeur3
        instructions exécutées si valeur3 correspond à variable
…
    Case Else
        instructions exécutées si aucune correspondance
End Select
```

Une structure *Select Case* commence par les mots clés *Select Case* et finit par les mots clés *End Select*. Vous remplacez *variable* par la variable, la propriété ou une autre expression censée être la valeur principale, ou cas de test, de la structure. Vous remplacez *valeur1*, *valeur2* et *valeur3* par les nombres, les chaînes et les autres valeurs liées au cas de test en question. Si une des valeurs correspond à la variable, les instructions situées après la condition *Case* sont exécutées, puis Visual Basic passe à la ligne qui suit l'instruction *End Select* et reprend l'exécution à cet endroit. Une structure *Select Case* peut contenir autant de conditions *Case* que voulu et une condition *Case* peut contenir plusieurs valeurs. Si vous prévoyez plusieurs valeurs pour un cas, séparez-les par des virgules.

L'exemple qui suit montre comment utiliser une structure *Select Case* pour afficher le message adéquat sur l'âge et les étapes culturelles de la vie d'une personne dans un programme. Comme la variable *Age* contient la valeur 18, la chaîne « Vous pouvez voter » est assignée à la propriété *Text* de l'objet étiquette.

```
Dim Age As Integer
Age = 18

Select Case Age
    Case 16
        Label1.Text = "Vous pouvez conduire !"
    Case 18
        Label1.Text = "Vous pouvez voter !"
    Case 21
        Label1.Text = "Vous pouvez vous faire élire."
    Case 65
        Label1.Text = "Il est temps de prendre votre retraite et de vous amuser !"
End Select
```

Une structure *Select Case* prend également en charge une condition *Case Else* que l'on peut utiliser pour afficher un message si aucun des cas précédents ne correspond à la variable *Age*. Voici comment *Case Else* fonctionnerait dans l'exemple suivant – notez que j'ai remplacé la valeur *Age* par 25 afin de déclencher la condition *Case Else* :

```
Dim Age As Integer
Age = 25

Select Case Age
    Case 16
        Label1.Text = "Vous pouvez conduire !"
    Case 18
        Label1.Text = "Vous pouvez voter !"
    Case 21
        Label1.Text = "Vous pouvez vous faire élire."
    Case 65
        Label1.Text = "Il est temps de prendre votre retraite et de vous amuser !"
    Case Else
        Label1.Text = "Vous avez le bon âge ! Profitez-en !"
End Select
```

Opérateurs de comparaison avec une structure Select Case

Vous pouvez exploiter des opérateurs de comparaison pour insérer une plage de valeurs test dans une structure *Select Case*. Les opérateurs de comparaison de Visual Basic sont =, <>, >, <, >= et <=. Pour utiliser ces opérateurs de comparaison, vous devez insérer le mot clé *Is* ou *To* dans l'expression pour identifier la comparaison que vous effectuez. Le mot clé *Is* indique au compilateur de comparer la variable de test à l'expression apparaissant après le mot clé *Is*. Le mot clé *To* identifie une plage de valeurs. La structure suivante utilise *Is*, *To* et plusieurs opérateurs de comparaison pour tester la variable *Age* et afficher un des cinq messages suivants :

```
Select Case Age
    Case Is < 13
        Label1.Text = "Profitez de votre jeunesse !"
    Case 13 To 19
        Label1.Text = "Profitez de votre adolescence !"
    Case 21
        Label1.Text = "Vous pouvez vous faire élire."
    Case Is > 100
        Label1.Text = "Vous allez l'air en forme !"
    Case Else
        Label1.Text = "Quel bel âge !"
End Select
```

Si la valeur de la variable *Age* est inférieure à 13, le message « Profitez de votre jeunesse ! » s'affiche. De 13 à 19 ans, le message « Profitez de votre adolescence ! » s'affiche, et ainsi de suite.

Une structure de décision *Select Case* est habituellement bien plus claire qu'une structure *If…Then* et plus efficace si vous effectuez plus de deux décisions de branchement d'après une variable ou une propriété. Toutefois, si vous réalisez deux comparaisons ou moins, ou si vous travaillez avec plusieurs valeurs différentes, il vaut mieux utiliser une structure de décision *If…Then*.

Dans l'exercice suivant, vous allez découvrir comment utiliser une structure *Select Case* pour traiter les entrées d'une zone de liste. Vous allez exploiter les propriétés *ListBox1.Text* et *ListBox1.SelectedIndexChanged* pour collecter les entrées, puis utiliser une structure *Select Case* pour afficher un message de bienvenue dans une des quatre langues.

Utiliser une structure Select Case pour traiter les entrées d'une zone de liste

1. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.
2. Créez un nouveau projet Application Windows Forms nommé Mon **Select Case**.

   Un formulaire vierge s'affiche dans la fenêtre conception.
3. Cliquez sur le contrôle *Label* de la Boîte à outils, puis faites glisser une nouvelle étiquette en haut du formulaire pour afficher le titre du programme.
4. Utilisez le contrôle *Label* pour créer un deuxième objet étiquette sous le premier.

   Vous allez utiliser cette étiquette comme intitulé de la zone de liste.
5. Dans la Boîte à outils, cliquez sur le contrôle *ListBox* puis créez une zone de liste en dessous de la deuxième étiquette.
6. Utilisez le contrôle *Label* pour dessiner deux étiquettes supplémentaires en dessous de la zone de liste afin d'afficher la sortie du programme.
7. Utilisez le contrôle *Button* pour créer un petit bouton en bas du formulaire.
8. Ouvrez la fenêtre Propriétés, puis positionnez les propriétés présentées dans le tableau suivant ; elles correspondent aux objets que vous venez de créer.

   En raison de la quantité élevée d'objets, vous allez également assigner des propriétés *Name* afin de pouvoir aisément identifier les différents contrôles sur le formulaire et dans votre code. Lorsque les propriétés sont triées alphabétiquement dans la fenêtre Propriété, la propriété *Name* apparaît entre parenthèses vers le haut de la fenêtre Propriétés. Je vous recommande d'utiliser la propriété *Name* dès que votre programme contient plus de quatre ou cinq objets. Dans cet exemple, j'ai attribué

comme nom d'objet des préfixes à trois lettres pour identifier le type d'objet, comme btn (pour button, bouton), lbl (pour label, étiquette ou libellé) et lst (pour list, zone de liste).

Objet	Propriété	Paramètre
Form1	*Text*	« Page d'accueil Case »
Label1	*Font*	Times New Roman, Bold, 12 points
	Name	lblTitle
	Text	« Programme international de bienvenue »
Label2	*Name*	lblTextBoxLabel
	Text	« Choisir un pays »
Label3	*Font*	10 points
	Name	lblPays
	Text	(vide)
Label4	*AutoSize*	False
	BorderStyle	Fixed3D
	ForeColor	Red
	Name	lblGreeting
	Text	(vide)
ListBox1	*Name*	lstCountryBox
Button1	*Name*	btnQuit
	Text	« Quitter »

Une fois que vous avez terminé, voici à quoi ressemble votre formulaire :

Vous allez maintenant entrer dans le code du programme pour initialiser la zone de liste.

9. Double-cliquez sur le formulaire.

La procédure événementielle *Form1_Load* s'affiche dans l'Éditeur de code.

10. Tapez le code suivant pour initialiser la zone de liste :

```
lstCountryBox.Items.Add("France")
lstCountryBox.Items.Add("Allemagne")
lstCountryBox.Items.Add("Mexique")
lstCountryBox.Items.Add("Italie")
```

Dans ces lignes, on exploite la méthode *Add* de l'objet zone de liste pour ajouter des entrées à la zone de liste du formulaire.

11. Cliquez sur l'onglet Form1.vb [Design] en haut de l'Éditeur de code pour revenir au Concepteur, puis double-cliquez sur l'objet zone de liste de votre formulaire pour modifier sa procédure événementielle.

La procédure événementielle *lstZonePays_SelectedIndexChanged* s'affiche dans l'Éditeur de code.

12. Tapez les lignes suivantes pour gérer la sélection de zone de liste effectuée par l'utilisateur :

```
lblPays.Text = lstCountryBox.Text
Select Case lstCountryBox.SelectedIndex
     Case 0
         lblGreeting.Text = "Bonjour, programmeur"
     Case 1
         lblGreeting.Text = "Hallo, programmierer"
     Case 2
         lblGreeting.Text = "Hola, programador"
     Case 3
         lblGreeting.Text = "Ciao, programmatore"
End Select
```

La première ligne copie le nom de l'élément de zone de liste sélectionné dans la propriété *Text* de la troisième étiquette du formulaire (que vous avez renommé *lblPays*). Dans cette instruction, la propriété la plus importante est *lstCountryBox.Text*. Elle contient le texte exact de l'élément sélectionné dans la zone de liste. Les autres instructions font partie de la structure de décision Select Case. Cette structure utilise la propriété *lstCountryBox.SelectedIndex* comme variable de cas de test et la compare à plusieurs valeurs. La propriété *SelectedIndex* contient toujours le numéro de l'élément sélectionné dans la zone de liste ; l'élément situé en haut est 0 (zéro), le deuxième est 1, le troisième est 2, et ainsi de suite. Grâce à *SelectedIndex*, la structure *Select Case* peut rapidement identifier le choix de l'utilisateur et afficher le message de bienvenue approprié sur le formulaire.

13. Affichez de nouveau le formulaire et double-cliquez sur le bouton Quitter (*btnQuit*). La procédure événementielle *btnQuit_Click* s'affiche dans l'Éditeur de code.

14. Tapez **End** dans la procédure événementielle.

15. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap06 comme emplacement.

Exécutez à présent le programme et observez comment fonctionne l'instruction *Select Case*.

> **Astuce** Le projet Select Case complet se trouve dans le dossier c:\vb08epe\chap06\Select Case.

16. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

17. Cliquez sur chacun des noms de pays dans la zone de liste Choisir un pays.

Le programme affiche un message de bienvenue pour chacun des pays listés. La figure qui suit illustre le message de bienvenue pour l'Italie :

18. Cliquez sur le bouton Quitter pour arrêter le programme.

Le programme s'arrête et vous revenez à l'environnement de développement.

Dans ce chapitre, votre travail sur les structures de décision *If…Then* et *Select Case* est terminé. Toutefois, vous aurez d'autres occasions de vous en servir tout au long de ce livre. *If…Then* et *Select Case* sont deux mécanismes de décision essentiels du langage de programmation Visual Basic et vous découvrirez que vous allez les employer dans presque chaque programme que vous rédigerez.

Aller plus loin : Détection des événements de la souris

Ce chapitre commence par aborder quelques événements auxquels les programmes Visual Basic sont susceptibles de répondre. Vous avez progressivement appris à gérer plusieurs types d'événements grâce aux structures de décision *If…Then* et *Select Case*. Dans cette section, vous allez ajouter un gestionnaire d'événements au programme Select Case destiné à détecter le moment où le pointeur survole quelques instants la zone de liste Pays. Vous allez rédiger une routine spéciale, ou *gestionnaire d'événements*, en construisant une procédure événementielle de zone de liste pour l'événement *MouseHover*, qui est une des nombreuses activités liées à la souris prises en charge et contrôlées par Visual Basic. Cette procédure événementielle affichera un message « Cliquez sur le nom du pays » si l'utilisateur maintient quelques instants la souris au-dessus de la zone de liste des pays sans faire de choix, soit parce qu'il ne sait pas comment s'y prendre, soit parce qu'il est absorbé par une autre tâche.

Ajouter un gestionnaire d'événements de souris

1. Ouvrez l'Éditeur de code si ce n'est déjà fait.
2. En haut de l'Éditeur de code, cliquez sur la flèche Nom de la classe, puis cliquez sur l'objet *lstCountryBox*.

   Utilisez les Info-bulles pour identifier les éléments tels que la zone de liste Nom de la classe dans Visual Studio, qui est un autre exemple d'événement *MouseHover* dans l'environnement de développement.
3. Cliquez sur la flèche Nom de la méthode, puis cliquez sur l'événement *MouseHover*.

   Visual Basic ouvre la procédure événementielle *lstCountryBox_MouseHover* dans l'Éditeur de code, comme ci-après :

Chaque objet du formulaire possède une procédure événementielle qui s'ouvre automatiquement si vous double-cliquez sur l'objet. Vous devez ouvrir les autres procédures événementielles en utilisant la zone de liste Nom de la méthode.

4. Tapez les instructions suivantes dans la procédure événementielle *lstCountryBox_MouseHover* :

```
If lstCountryBox.SelectedIndex < 0 Or _
     lstCountryBox.SelectedIndex > 4 Then
     lblGreeting.Text = "Cliquez sur un nom de pays"
End If
```

   Cette instruction *If* évalue la propriété *SelectedIndex* de l'objet zone de liste en utilisant deux instructions conditionnelles et l'opérateur *Or*. Le gestionnaire d'événements suppose que s'il existe une valeur située entre 1 et 4 dans la propriété *SelectedIndex*, l'utilisateur n'a pas besoin d'aide pour choisir un nom de pays (il a déjà sélectionné un pays). En revanche, si la propriété *SelectedIndex* se situe en dehors de cette plage, le gestionnaire d'événements affiche le message « Cliquez sur un nom de pays » dans l'étiquette de bienvenue au bas du formulaire. Ce message s'affiche si l'utilisateur maintient le pointeur sur la zone de liste et disparaît si un nom de pays a été choisi.

5. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

6. Maintenez le pointeur sur la zone de liste des pays et patientez quelques instants.

   Le message « Cliquez sur un nom de pays » apparaît dans l'étiquette, comme suit :

7. Dans la zone de liste, cliquez sur un nom de pays.

   Le message de bienvenue traduit s'affiche dans l'étiquette et le message d'aide disparaît.

8. Cliquez sur le bouton Quitter pour arrêter le programme.

Vous venez de découvrir comment traiter des événements de souris dans un programme et avez appris que la rédaction de gestionnaires d'événement est relativement simple. Essayez au fur et à mesure d'en rédiger par vous-même – vous en saurez ainsi plus sur les événements disponibles pour les objets Visual Studio et vous vous entraînerez à utiliser les structures de décision *If…Then* et *Select Case*.

Rappel du chapitre 6

Pour	Faites ceci
Écrire une expression conditionnelle	Utilisez un des opérateurs de comparaison suivants entre deux valeurs : =, <>, >, <, >=, ou <=.

Utiliser une structure de décision *If...Then*	Utilisez la syntaxe suivante :
	`If condition1 Then`
	`    instructions exécutées si condition1 True`
	`ElseIf condition2 Then`
	`    instructions exécutées si condition2 True`
	`Else`
	`    instructions exécutées si aucune sont True`
	`End If`

Recevoir des entrées utilisateur dans un format spécifique	Ajoutez un contrôle *MaskedTextBox* à votre formulaire et spécifiez le format de la saisie en configurant la propriété *Mask*.

Utiliser une structure de décision *Select Case*	Utilisez la syntaxe suivante :
	`Select Case variable`
	`Case valeur1`
	`    instructions exécutées si valeur1 correspond`
	`Case valeur2`
	`    instructions exécutées si valeur2 correspond`
	`Case Else`
	`    instructions exécutées si aucune correspondance`
	`End Select`

Renommer un objet dans un programme	Sélectionnez l'objet que vous souhaitez renommer, puis modifiez sa propriété *Name* en utilisant la fenêtre Propriétés. Si vous attribuez à l'objet un préfixe à trois caractères qui identifie le type de l'objet (btn, lbl, lst, etc.), il est plus aisé de le localiser dans le code.
Effectuer deux comparaisons dans une expression conditionnelle	Utiliser un opérateur logique entre des comparaisons (*And*, *Or*, *Not* ou *Xor*).
Court-circuiter une instruction *If...Then*	On peut court-circuiter les instructions *If...Then* lorsque l'on utilise les opérateurs *AndAlso* et *OrElse* et que deux ou plusieurs expressions conditionnelles sont données. En fonction du résultat de la première condition, Visual Basic peut ne pas évaluer les autres conditions, et l'instruction est court-circuitée.
Écrire un gestionnaire d'événement	Dans l'Éditeur de code, cliquez sur le nom d'un objet dans la zone de liste Nom de la classe, puis cliquez sur un nom d'événement dans la zone de liste Nom de la méthode. Ajoutez des instructions à la procédure événementielle (appelée *gestionnaire d'événement*) qui répondent à l'événement que vous personnalisez.

Chapitre 7

Utiliser les boucles et les minuteurs

À la fin de ce chapitre, vous saurez :

- Utiliser une boucle *For…Next* pour exécuter des instructions un nombre défini de fois
- Afficher la sortie dans une boîte de texte multiligne en utilisant la concaténation
- Utiliser une boucle *Do* pour exécuter des instructions jusqu'à ce qu'une condition spécifique soit remplie
- Utiliser le contrôle *Timer* pour exécuter un code à des moments spécifiques
- Créer une horloge numérique et un utilitaire de mot de passe chronométré
- Utiliser la nouvelle commande Insérer un extrait pour insérer des modèles ou des extraits de code prêts à l'emploi dans l'Éditeur de code

Dans le chapitre 6, « Utiliser les structures de décision », vous avez appris à utiliser les structures de décision *If…Then* et *Select…Case* pour choisir quelles instructions exécuter dans un programme. Vous avez également appris à traiter l'entrée de l'utilisateur, à évaluer différentes conditions dans un programme et à déterminer quel bloc d'instructions exécuter en fonction de conditions changeantes. Dans ce chapitre, vous allez poursuivre votre étude de l'exécution du programme et du *contrôle du flux* en vous servant de boucles pour exécuter un bloc d'instructions maintes et maintes fois. Vous créerez également une horloge numérique ainsi que d'autres utilitaires intéressants qui réalisent des actions à des moments définis ou en relation avec les intervalles de l'horloge système de votre ordinateur.

Dans ce chapitre, vous allez employer une boucle *For…Next* pour exécuter des instructions un nombre défini de fois et une boucle *Do* pour exécuter des instructions jusqu'à ce qu'une expression conditionnelle soit remplie. Vous apprendrez à afficher plusieurs lignes de texte dans un objet zone de texte en utilisant l'opérateur de concaténation (&) et à employer le contrôle *Timer* de Visual Studio pour exécuter du code à des intervalles spécifiques dans votre programme. Pour finir, vous étudierez comment exploiter la commande Insérer un extrait pour insérer des modèles de code dans vos programmes : une fonctionnalité pratique de l'EDI de Microsoft Visual Studio.

Développer des boucles For…Next

Avec une boucle *For…Next*, vous exécutez un groupe spécifique d'instructions un nombre défini de fois au cours d'une procédure événementielle ou dans un module de code. Cette approche est intéressante si l'on réalise plusieurs calculs liés, exploite des éléments à l'écran ou traite diverses parties de l'entrée utilisateur. La boucle *For…Next* représente surtout une méthode abrégée pour écrire une longue liste d'instructions : chaque groupe d'instructions d'une telle liste effectue la même tâche. En conséquence, il suffit de définir un groupe d'instructions et de demander qu'il s'exécute autant de fois que nécessaire.

Voici la syntaxe d'une boucle *For…Next* :

```
For variable = début To fin
    Instructions à répéter
Next [variable]
```

Dans cette instruction syntaxique, *For*, *To* et *Next* sont des mots obligatoires, à l'instar de l'opérateur égal à (=). Remplacez *variable* par le nom d'une variable numérique qui compte le nombre actuel de boucles (la variable après *Next* est optionnelle) puis *début* et *fin* par des valeurs numériques représentant les points de départ et d'arrêt de la boucle (vous devez déclarer *variable* avant de l'utiliser dans l'instruction *For…Next*). Les lignes qui se trouvent entre les instructions *For* et *Next* sont répétées à chaque exécution de la boucle.

Par exemple, la boucle *For…Next* suivante émet une succession rapide de quatre signaux sonores par l'intermédiaire du haut-parleur de l'ordinateur (quoique le résultat puisse être difficile à entendre) :

```
Dim i As Integer
For i = 1 To 4
Beep()
Next i
```

Du point de vue fonctionnel, cette boucle revient à écrire quatre fois l'instruction *Beep* dans une procédure. Le compilateur la traite de la même manière que

```
Beep()
Beep()
Beep()
Beep()
```

La variable employée dans la boucle est *i*, une lettre qui, par convention, représente le premier compteur d'entier d'une boucle *For…Next* et que l'on déclare en tant que type *Integer*. Chaque fois que la boucle s'exécute, la variable compteur est incrémentée de un (au premier passage dans la boucle, la variable contient une valeur de 1, la valeur de *départ* ; au dernier passage, elle contient la valeur 4, la valeur de *fin*). Comme nous le verrons dans les prochains exemples, cette variable compteur présente d'énormes avantages dans les boucles.

Afficher une variable compteur dans un contrôle zone de texte

La variable compteur est similaire à toute autre variable d'une procédure événementielle. On peut lui affecter des propriétés, l'employer dans des calculs ou l'afficher dans un programme. L'affichage de la sortie dans un contrôle *TextBox* constitue l'une de ses utilisations pratiques. Vous avez utilisé le contrôle *TextBox* précédemment dans ce livre pour afficher une ligne de sortie. Dans ce chapitre, ce contrôle va servir à afficher plusieurs lignes de texte. Pour ce faire, il suffit de positionner la propriété *Multiline* du contrôle *TextBox* sur *True* et la propriété *ScrollBars* sur *Vertical*. Ces simples paramètres transforment un objet zone de texte en un objet zone de texte multiligne équipé de barres de défilement pour en simplifier l'accès.

Afficher des informations en utilisant un boucle For...Next

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms intitulé **Ma Boucle For**.

   Un formulaire vierge s'affiche dans la fenêtre Concepteur. Votre première étape de programmation consiste à ajouter un contrôle *Button* au formulaire, mais cette fois, vous allez employer une nouvelle méthode.

2. Double-cliquez sur le contrôle *Button* dans la Boîte à outils.

   Visual Studio place un objet bouton dans l'angle supérieur gauche du formulaire. Avec le contrôle *Button*, ainsi que bien d'autres contrôles, le double-clic permet de créer rapidement un objet de taille standard sur le formulaire. Vous pouvez à présent faire glisser l'objet bouton où bon vous semble et le personnaliser en vous servant des paramètres des propriétés.

3. Faites glisser l'objet bouton vers la droite et centrez-le à proximité du bord supérieure du formulaire.

4. Ouvrez la fenêtre Propriétés et attribuez la valeur **Boucle** à la propriété *Text* du bouton.

5. Double-cliquez sur le contrôle *TextBox* dans la Boîte à outils.

   Visual Studio crée un petit objet zone de texte sur le formulaire.

6. Positionnez la propriété *Multiline* de l'objet zone de texte sur *True* et sa propriété *ScrollBars* sur *Vertical*.

   Ces paramètres préparent la zone de texte pour l'affichage de plusieurs lignes de texte.

7. Déplacez la zone de texte sous le bouton et agrandissez-la de sorte qu'elle occupe les deux tiers du formulaire.

8. Dans le formulaire, double-cliquez sur le bouton *Boucle*.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

9. Tapez les instructions suivantes dans la procédure :

```
Dim i As Integer
Dim rChariot As String
rChariot= Chr(13) & Chr(10)
For i = 1 To 10
     TextBox1.Text = TextBox1.Text & "Ligne " & i & rChariot
Next i
```

Cette procédure événementielle déclare deux variables, l'une du type *Integer* (*i*) et l'autre du type *String* (*rChariot*). Elle affecte ensuite une valeur de chaîne représentant le caractère retour chariot à la deuxième variable.

> **Astuce** En termes de programmeur, le caractère retour chariot revient à appuyer sur la touche ENTRÉE. Dans le code du programme, j'ai créé une variable spéciale pour ce caractère que j'ai constitué d'éléments retour et saut de ligne pour codifier un retour chariot plus simple. L'élément retour, Chr(13) déplace le pointeur en I au début de la ligne. L'élément saut de ligne, Chr(10), réminiscence d'une ancienne machine à écrire, déplace le pointeur en I à la ligne suivante.

Après la déclaration et l'affectation de la variable, j'ai employé une boucle *For…Next* pour afficher la ligne *X* 10 fois dans l'objet zone de texte, où *X* représente la valeur actuelle de la variable compteur (autrement dit, Ligne 1 à Ligne 10). Les caractères de concaténation (&) assemblent les parties du composant dans chaque ligne de la zone de texte. Pour commencer, on ajoute la valeur entière qui se trouve dans la zone de texte, stockée dans la propriété *Text*, à l'objet de sorte que les lignes précédentes ne soient pas éliminées chaque fois qu'une nouvelle ligne s'ajoute. Ensuite, on combine la chaîne « Ligne », le numéro de la ligne actuelle et le caractère de retour chariot (*rChariot*) pour afficher une nouvelle ligne et déplacer le pointeur en I vers la marge de gauche et vers le bas d'une ligne. L'instruction *Next* termine la boucle.

Remarquez que Visual Studio ajoute automatiquement l'instruction *Next* à la fin de la boucle lorsque vous tapez *For* au début de la boucle. Dans ce cas, j'ai modifié l'instruction *Next* pour y inclure le nom de la variable *i*, une clarification optionnelle de la syntaxe (le nom de la variable identifie la variable mise à jour, en particulier dans les boucles *For…Next* imbriquées).

10. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap07.

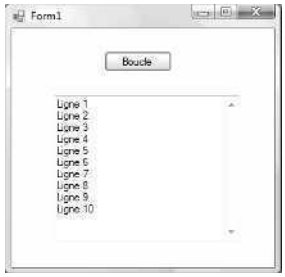
Vous êtes prêt à exécuter le programme.

Astuce Le programme Boucle For complet se trouve dans le dossier c:\vb08epe\chap07.

11. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

12. Cliquez sur le bouton Boucle.

La boucle *For…Next* affiche 10 lignes dans la zone de texte, comme l'illustre la figure suivante :



13. Cliquez à nouveau sur le bouton Boucle.

La boucle *For…Next* affiche 10 lignes supplémentaires dans le formulaire (pour visualiser les lignes non visibles, servez-vous de la barre de défilement verticale). Chaque fois que la boucle se répète, elle ajoute 10 lignes dans l'objet zone de texte.

Astuce Peur de manquer de place dans l'objet zone de texte ? Pas d'inquiétude : si vous affichez uniquement des lignes de texte simples, vous avez de la marge. Un objet zone de texte multiligne dispose d'une limite pratique de 64 Ko de texte ! Pour bénéficier d'un espace plus important ainsi que d'options de mise en forme, servez-vous du contrôle *RichTextBox* qui se trouve dans la Boîte à outils. Il s'agit d'un outil similaire, mais proposant un contrôle accru en matière d'affichage et de manipulation du texte.

14. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Comme vous pouvez le noter, la boucle *For…Next* simplifie considérablement le code et réduit le nombre total d'instructions à saisir. Dans l'exemple précédent, une boucle de trois lignes traite l'équivalent de 10 instructions suite à chaque clic sur le bouton Boucle.

Créer des boucles For…Next complexes

La variable compteur d'une boucle *For…Next* constitue un outil puissant au sein de vos programmes. Avec un peu d'imagination, vous pouvez l'employer pour créer des séquences de nombres intéressantes dans vos boucles. Pour créer une boucle avec un modèle de compteur autre que 1, 2, 3, 4, etc., il suffit de spécifier une valeur de départ différente puis d'utilise le mot clé *Step* pour incrémenter le compteur selon des intervalles différents. Par exemple, le code

```
Dim i As Integer
Dim rChariot As String
rChariot = Chr(13) & Chr(10)

For i = 5 To 25 Step 5
    TextBox1.Text = TextBox1.Text & "Ligne " & i & rChariot
Next i
```

affiche la séquence suivante de numéros de lignes dans une zone de texte :

```
Ligne 5
Ligne 10
Ligne 15
Ligne 20
Ligne 25
```

Vous pouvez également indiquer des valeurs décimales dans une boucle, si vous déclarez le type simple précision ou double précision pour la variable *i*. Par exemple, la boucle *For…Next*

```
Dim i As Single
Dim rChariot As String
rChariot = Chr(13) & Chr(10)

For i = 1 To 2.5 Step 0.5
    TextBox1.Text = TextBox1.Text & "Ligne " & i & rChariot
Next i
```

affiche les numéros de lignes dans une zone de texte :

```
Ligne 1
Ligne 1,5
Ligne 2
Ligne 2,5
```

Outre l'affichage de la variable compteur, vous pouvez exploiter le compteur pour définir des propriétés, calculer des valeurs ou traiter des fichiers. Le prochain exercice montre comment employer le compteur pour ouvrir des icônes Visual Basic stockées sur le disque dur dans des fichiers dont le nom contient des chiffres. Vous obtiendrez de nombreux fichiers d'icônes, de bitmaps et d'animations dans le fichier C:\Programmes \Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\1036\VS2008ImageLibrary.zip (avec Windows Vista). Il s'agit d'un fichier compressé .zip d'où vous devrez extraire les éléments. Sachez également que Microsoft modifie parfois l'emplacement de stockage de ces fichiers.

Ouvrir des fichiers en utilisant une boucle For...Next

1. Dans le menu Fichier, cliquez sur la commande Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.

2. Créez un nouveau projet Application Windows Forms intitulé **Ma Boucle For Icônes**.

   Le nouveau projet démarre et un formulaire vierge s'affiche dans la fenêtre Concepteur.

Remarque Si vous ouvrez le projet à partir des fichiers d'exercices d'accompagnement, vous verrez un code légèrement différent de celui montré à l'étape 7 ci-dessous, car le prochain exercice va modifier le projet Boucle For Icônes.

3. Dans la Boîte à outils, cliquez sur le contrôle *PictureBox* et tracez un objet zone d'image de taille moyenne, centré dans la moitié supérieure du formulaire.
4. Cliquez sur le contrôle *Button* et dessinez un large bouton sous la zone d'image (le nom que vous allez donner au bouton sera plus long que d'habitude).
5. Définissez les propriétés suivantes pour les deux objets :

Objet	Propriété	Paramètre
PictureBox1	*BorderStyle*	Fixed3D
	SizeMode	StretchImage
Button1	*Text*	« Afficher quatre visages »

6. Double-cliquez sur le bouton Afficher quatre visages sur le formulaire pour afficher la procédure événementielle de l'objet bouton.

   La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

7. Saisissez la boucle *For...Next* suivante :

```
Dim i As Integer
For i = 1 To 4
     PictureBox1.Image = System.Drawing.Image.FromFile _
         ("c:\vb08epe\chap07\face0" & i & ".ico")
     MsgBox ("Cliquez ici pour afficher le prochain visage")
Next
```

Astuce La méthode *FromFile* de cette procédure événementielle est trop longue pour être placée sur une même ligne de ce livre, je l'ai donc divisée en deux lignes en me servant d'un espace et du caractère de continuation de ligne (_). Vous pouvez utiliser ce caractère où bon vous semble dans le code du programme, excepté dans une expression de chaîne.

La boucle emploie la méthode *FromFile* pour charger les quatre fichiers d'icônes à partir du dossier c:\vb08epe\chap07 sur votre disque dur. Le nom du fichier est créé en faisant appel à la variable compteur et à l'opérateur de concaténation utilisé précédemment dans ce chapitre. Le code

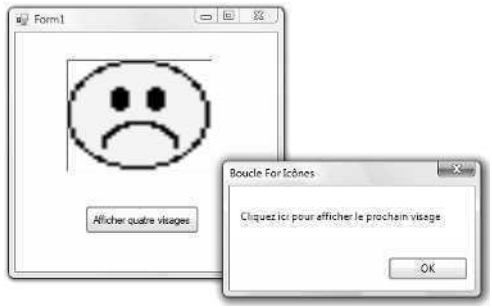
```
PictureBox1.Image = System.Drawing.Image.FromFile _
  ("c:\vb08epe\chap07\face0" & i & ".ico")
```

combine un chemin d'accès, un nom de fichier et l'extension .ico pour créer quatre noms de fichiers valides d'icônes se trouvant sur le disque dur. Dans cet exemple, vous chargez face01.ico, face02.ico, face03.ico et face04.ico dans la zone d'image. Cette instruction fonctionne parce que plusieurs fichiers du dossier c:\vb08epe\chap07 possèdent pour modèle de nom face*xx*.ico. En identifiant le modèle, vous pouvez construire une boucle *For…Next* basée sur les noms des fichiers.

Remarque La fonction boîte de message (*MsgBox*) sert principalement à ralentir l'action de sorte que vous puissiez observer ce qui se produit dans la boucle *For…Next*. Dans une application classique, vous n'utiliserez probablement pas une telle fonction (quoique rien ne vous en empêche).

8. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap07 comme emplacement.
9. Cliquez sur le bouton Démarrer le débogage pour exécuter le programme et cliquez sur le bouton Afficher quatre visages.

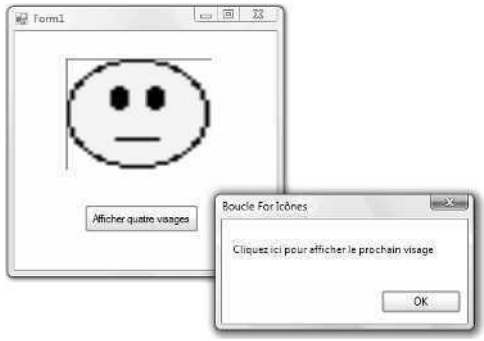
   La boucle *For…Next* charge le premier visage dans la zone d'image puis affiche cette boîte de message :



Remarque Si Visual Basic affiche un message d'erreur, assurez-vous que le code du programme ne comporte pas de faute d'orthographe et vérifiez que les fichiers d'icônes se trouvent à l'emplacement désigné par le chemin d'accès dans le programme. Si vous avez installé les fichiers d'exercices Étape par étape dans un autre dossier que celui par défaut ou si vous avez déplacé les fichiers d'icônes après l'installation, le chemin d'accès indiqué dans la procédure événementielle sera incorrect.

10. Cliquez sur OK pour afficher le visage suivant.

Votre écran présente un résultat similaire à



11. Cliquez trois autres fois sur OK pour afficher la collection complète de visages.

Vous pouvez répéter la séquence si vous le souhaitez.

12. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire.

Le programme s'arrête et vous revenez à l'environnement de développement.

Ouvrir des fichiers avec un compteur possédant une portée supérieure

Existe-t-il des situations dans lesquelles l'utilisation d'une boucle *For…Next* n'est pas efficace ou élégante ? Sans aucun doute. En fait, l'exemple précédent, bien qu'intéressant comme démonstration, est quelque peu alourdi par le comportement intrusif de la boîte de message, qui s'ouvre quatre fois dans la boucle *For…Next* et détourne l'utilisateur du formulaire sur lequel son attention doit se concentrer. Existe-t-il un moyen de se débarrasser de cette inopportune boîte de message ?

Une solution serait de supprimer la fonction *MsgBox* et la boucle *For…Next* et de les remplacer par une variable compteur dont la portée est plus importante au sein du formulaire. Comme vous l'avez appris au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework », il est possible de déclarer une variable ayant pour portée (ou conservant sa valeur) sur l'ensemble du formulaire en plaçant une instruction *Dim* pour la variable dans la partie supérieure du formulaire dans l'Éditeur de code, à un emplacement spécial, situé au-dessus des procédures événementielles. Dans le prochain exercice, vous allez utiliser une variable *Integer* intitulée *Compteur* qui conserve sa valeur entre les appels à la procédure événementielle *Button1_Click* et vous servir de cette variable pour ouvrir les mêmes fichiers d'icônes sans faire appel à la fonction *MsgBox* pour suspendre l'action.

Utiliser un compteur global

1. Ouvrez l'Éditeur de code du projet Ma Boucle For Icônes.
2. Placez le point d'insertion au-dessus de la procédure événementielle *Button1_Click* et directement sous l'instruction *Public Class Form1* puis déclarez une variable *Integer* intitulée *Compteur* en utilisant la syntaxe suivante :

   ```
   Dim Compteur As Integer = 1
   ```

   Remarquez que Visual Studio sépare la déclaration que vous venez de saisir de la procédure événementielle par une ligne pleine et affiche le mot « Déclarations » dans la zone de liste Nom de la méthode. Vous venez de faire quelque chose d'inhabituel : en dehors de la déclaration de la variable *Compteur*, vous lui avez également affecté la valeur 1. Déclarer et affecter simultanément n'est pas autorisé dans Visual Basic 6, mais constitue un dispositif bien pratique de Visual Basic depuis la version 2002. J'ai employé cette syntaxe au chapitre 5 pour déclarer une constante, mais c'est la première que j'y ai recours pour une déclaration de variable.
3. Au sein de la procédure événementielle *Button1_Click*, remplacez le code de sorte qu'il corresponde précisément au groupe d'instructions suivant (supprimez les instructions absentes).

   ```
   PictureBox1.Image = System.Drawing.Image.FromFile _
     ("c:\vb08epe\chap07\face0" & Compteur & ".ico")
   Compteur += 1
   If Compteur = 5 Then Compteur = 1
   ```

   J'ai supprimé la déclaration de l'entier *i*, les instructions *For* et *Next* ainsi que la fonction *MsgBox* et j'ai changé la manière dont fonctionne la méthode *FromFile* en remplaçant la variable *i* par la variable *Compteur*. J'ai également ajouté deux nouvelles instructions qui utilisent la variable *Compteur*. La première ajoute 1 à *Compteur* (`Compteur += 1`) et la deuxième réinitialise la variable *Compteur* si la valeur atteint 5 (réinitialiser ainsi la variable crée un cycle infini au sein des fichiers d'icônes). La syntaxe `Compteur += 1` est un raccourci de Visual Basic 2005 et 2008 qui représente l'équivalent fonctionnel de l'instruction

   ```
   Compteur = Compteur + 1
   ```

   Vous allez à présent exécuter le programme.

> **Astuce** Le programme modifié Boucle For Icônes est disponible dans le dossier c:\vb08epe\chap07\Boucle For Icônes.

4. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. Le programme s'exécute dans l'environnement de développement.
5. Cliquez plusieurs fois sur le bouton Afficher quatre visages (notez le changement d'humeur des visages).

6. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Comme vous l'aurez remarqué, cette solution est un peu plus élégante que celle de l'exemple précédent puisque l'utilisateur ne clique que sur un bouton et non pas sur un bouton de formulaire et un bouton de boîte de message. L'inconvénient de l'interface du premier programme n'est cependant pas lié à la boucle *For…Next*, mais à la limite définie pour la procédure événementielle *Button1_Click* qui peut uniquement employer des variables locales (autrement dit, des variables déclarées au sein de la procédure événementielle). Entre les clics sur le bouton, ces variables locales perdent leur valeur et la seule manière d'incrémenter le compteur a été de créer une boucle. En faisant appel à une variable *Integer* avec une portée plus étendue, la valeur de la variable *Compteur* est conservée entre les clics et cette information numérique peut être employée pour afficher des fichiers dans la procédure événementielle *Button1_Click*.

L'instruction *Exit For*

La plupart des boucles *For…Next* s'exécutent sans incident jusqu'à leur achèvement. De temps à autre, il peut toutefois être intéressant d'arrêter le calcul d'un boucle *For…Next* si une « condition d'arrêt » se présente. Visual Basic l'autorise en proposant l'instruction *Exit For*, que vous pouvez utiliser pour arrêter prématurément l'exécution d'une boucle *For…Next* et placer l'exécution au niveau de la première instruction qui suit la boucle.

Par exemple, la boucle *For…Next* suivante demande 10 noms à l'utilisateur et les affiche un par un dans une zone de texte jusqu'à ce que l'utilisateur saisisse le mot « Fini ».

```
Dim i As Integer
Dim Nom As String
For i = 1 To 10
    Nom = InputBox ("Saisissez un nom ou tapez Fini pour quitter")
    If Nom = "Fini" Then Exit For
    TextBox1.Text = Nom
Next i
```

Si l'utilisateur ne saisit pas « Fini », l'instruction *Exit For* termine la boucle et l'exécution se poursuit par l'instruction qui suit *Next*.

Développer des boucles Do

En alternative à la boucle *For…Next*, vous pouvez écrire une boucle *Do* qui exécute un groupe d'instructions jusqu'à ce qu'une condition donnée soit *True*. Les boucles *Do* sont particulièrement intéressantes lorsque l'on ne sait pas combien de fois la boucle se répète. Prenons l'exemple suivant : l'utilisateur saisit des noms destinés à une base de données dans une zone de saisie jusqu'à ce qu'il tape « Fini ». Dans ce cas, la boucle *Do* permet d'itérer indéfiniment jusqu'à ce que l'utilisateur saisisse la chaîne de texte « Fini ».

La boucle *Do* prend différentes formes, selon la manière et le moment où on évalue sa condition. Voici sa syntaxe classique

```
Do While condition
    Bloc d'instruction à exécuter
Loop
```

Par exemple, la boucle *Do* suivante demande une saisie à l'utilisateur et l'affiche dans une zone de texte jusqu'à ce que l'utilisateur tape le mot « Fini » dans la zone de texte :

```
Dim Nom As String
Do While Nom <> "Fini"
    Nom = InputBox ("Saisissez un nom ou tapez Fini pour quitter")
    If Nom <> "Fini" Then TextBox1.Text = Nom
Loop
```

L'instruction conditionnelle de la boucle est `Nom <> "Fini"` que le compilateur Visual Basic traduit par « boucler tant que la variable *Nom* ne contient pas exactement le mot 'Fini' ». Voilà qui met en évidence un point intéressant concernant les boucles *Do* : si la condition définie dans la partie supérieure de la boucle n'est pas *True* à la première évaluation de l'instruction *Do*, la boucle *Do* ne s'exécute jamais. Dans notre exemple, si la variable de chaîne *Nom* contient la valeur « Fini » avant le démarrage de la boucle (provenant éventuellement d'une assignation antérieure dans la procédure événementielle), Visual Basic ignore entièrement la boucle et continue par la ligne qui suit le mot clé *Loop*.

Pour toujours exécuter la boucle au moins une fois dans un programme, placez le test conditionnel dans la partie inférieure de la boucle. Par exemple, la boucle

```
Dim Nom As String
Do
    Nom = InputBox ("Saisissez un nom ou tapez Fini pour quitter")
    If Nom <> "Fini" Then TextBox1.Text = Nom
Loop While Nom <> "Fini"
```

est similaire à la boucle *Do* précédente, excepté que dans ce cas, la condition de la boucle est testée après réception du nom saisi par la fonction *InputBox*. Cette solution présente l'avantage d'actualiser la variable *Nom* avant le test conditionnel de la boucle de sorte que cette dernière ne soit pas ignorée en présence d'une valeur « Fini » préexistante. En testant la condition à la fin, la boucle s'exécute au moins une fois. Cette option vous oblige néanmoins souvent à ajouter quelques instructions supplémentaires pour traiter les données.

Remarque Les exemples de code précédents exigent de l'utilisateur qu'il saisisse le mot « Fini » pour quitter. Il est à noter que le test du texte saisi est sensible à la casse, ce qui signifie que si l'utilisateur saisit « fini » ou « FINI », le programme ne s'arrête pas. Pour rendre le test insensible à la casse, servez-vous de la fonction *StrComp*, que nous étudierons au chapitre 13, « Explorer le traitement des fichiers texte et des chaînes ».

Éviter une boucle sans fin

Compte tenu de la nature implacable des boucles *Do*, il est impératif de créer des conditions de test fournissant à chaque boucle un vrai point de sortie. Si l'évaluation d'un test de boucle ne produit jamais un résultat *False*, la boucle s'exécute indéfiniment et le programme ne répond plus à la saisie. Prenons l'exemple suivant :

```
Dim Nombre As Double
Do
    Nombre = InputBox ("Saisissez un nombre à élever au carré. Tapez -1 pour quitter.")
    Nombre = Nombre * Nombre
    TextBox1.Text = Nombre
Loop While Nombre >=0
```

Dans cette boucle, l'utilisateur saisit nombre après nombre et le programme élève au carré chaque nombre puis l'affiche dans la zone de texte. Malheureusement, lorsque l'utilisateur veut s'arrêter, il ne peut pas quitter puisque la condition de sortie annoncée ne fonctionne pas. Lorsque l'utilisateur saisit –1, le programme l'élève au carré et la variable *Nombre* prend la valeur 1 (le problème peut être résolu en définissant une condition de sortie différente). Il est essentiel de prêter attention aux boucles sans fin lorsque l'on écrit des boucles *Do*. Heureusement, elles sont assez simples à détecter si on teste attentivement les programmes.

Important Veillez à ce que chaque boucle possède une condition de sortie légitime.

Le prochain exercice montre comment se servir de la boucle *Do* pour convertir des températures en Fahrenheit en températures en Celsius. Ce programme simple utilise la fonction *InputBox* pour inviter l'utilisateur à saisir une valeur, convertit la température et affiche le résultat dans une boîte de message.

Convertir les températures avec une boucle Do

1. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.

2. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Ma Conversion Celsius**.

Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans la fenêtre Concepteur. Cette fois, vous allez placer tout le code du programme dans la procédure événementielle *Form1_Load* de sorte que Visual Basic vous demande immédiatement la température en Fahrenheit lorsque vous démarrez l'application. Vous allez vous servir d'une fonction *InputBox* pour demander les données Fahrenheit et d'une fonction *MsgBox* pour afficher la valeur convertie.

3. Double-cliquez sur le formulaire.

   La procédure événementielle *Form1_Load* s'affiche dans l'Éditeur de code.

4. Tapez le programme suivant dans la procédure événementielle *Form1_Load* :

```
Dim Fahrenheit, Celsius As Single
Dim strFahrenheit As String
Dim Invite As String = "Saisissez une température en Fahrenheit"
Do
     strFahrenheit = InputBox(Prompt, "Fahrenheit à Celsius")
     If strFahrenheit <> "" Then
         Fahrenheit = CSng(strFahrenheit)
         Celsius = Int((Fahrenheit + 40) * 5 / 9 - 40)
         MsgBox(Celsius, , "Température en Celsius")
     End If
Loop While strFahrenheit <> ""
End
```

Astuce N'oubliez pas l'instruction *End* à la fin de la procédure événementielle *Form1_Load*.

Ce code gère les calculs du projet. La première ligne déclare deux variables à simple précision, *Fahrenheit* et *Celsius*, pour conserver leurs températures relatives. La deuxième ligne déclare une variable de chaîne intitulée *strFahrenheit* qui contient une version chaîne de la température en Fahrenheit. La troisième ligne déclare une variable de chaîne nommée *Invite*, utilisée dans la fonction *InputBox*, et lui assigne une valeur initiale. La boucle *Do* invite répétitivement l'utilisateur à saisir une température en Fahrenheit, convertit le nombre en Celsius puis l'affiche à l'écran *via* la fonction *MsgBox*.

La valeur que l'utilisateur saisit dans la zone de saisie est stockée dans la variable *strFahrenheit*. La fonction *InputBox* retourne toujours une valeur de type chaîne, même si l'utilisateur saisit des nombres. Dans la mesure où un calcul doit être effectué sur la valeur saisie, la variable *strFahrenheit* doit être convertie en nombre. La fonction *CSng* convertit une chaîne en type de donnée *Single*. *CSng* représente l'une des nombreuses fonctions de conversion permettant de convertir une chaîne en différents types de données. Le programme stocke ensuite la valeur *Single* convertie dans la variable *strFahrenheit*.

La boucle s'exécute jusqu'à ce que l'utilisateur clique sur le bouton Annuler, qu'il appuie sur la touche ENTRÉE ou clique sur le bouton OK sans saisir de valeur. S'il clique sur le bouton Annuler ou ne saisit pas de valeur, le programme retourne une chaîne vide (""). La boucle vérifie que la chaîne est vide en se servant du test conditionnel *While* qui se trouve à la fin de la boucle. L'instruction

```
Celsius = Int((Fahrenheit + 40) * 5 / 9 - 40)
```

gère la conversion de Fahrenheit en Celsius dans le programme. Cette instruction emploie une formule de conversion standard et fait appel à la fonction *Int* pour retourner une valeur qui ne contient pas de décimales à la variable *Celsius* (tout ce qui se trouve à droite du séparateur décimal est éliminé). Cette coupe diminue la précision, mais évite les nombres longs et laids comme 21,11111, soit la valeur en Celsius de 70 degrés Fahrenheit.

5. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap07 comme emplacement.

   Vous allez à présent exécuter le programme.

6. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme démarre et la fonction *InputBox* vous invite à saisir une température en Fahrenheit.

7. Tapez **212**.

   Votre écran présente un résultat similaire à

8. Cliquez sur OK.

   Le programme convertit la température de 212 degrés Fahrenheit en 100 degrés Celsius, comme le montre cette boîte de message.

9. Cliquez sur OK. Tapez ensuite **72** dans la zone de saisie et cliquez à nouveau sur OK.

   Le programme convertit les 72 degrés Fahrenheit en 22 degrés Celsius.

10. Cliquez sur OK puis sur le bouton Annuler dans la boîte de saisie.

    Le programme se ferme et vous revenez à l'environnement de développement.

Utiliser le mot clé *Until* dans les boucles Do

Les boucles *Do* avec lesquelles vous avez travaillé pour l'instant utilisent le mot clé *While* pour exécuter un groupe d'instructions tant que la condition de bouclage demeure *True*. Avec Visual Basic, vous pouvez également faire appel au mot clé *Until* dans les boucles *Do* pour itérer jusqu'à ce qu'une certaine condition soit *True*. À l'instar du mot clé *While*, vous êtes libre de placer le mot clé *Until* au début ou à la fin de la boucle *Do* pour tester une condition. Par exemple, la boucle *Do* suivante utilise le mot clé *Until* pour itérer répétitivement jusqu'à ce que l'utilisateur saisisse le mot « Fini » dans la zone de saisie :

```
Dim Nom As String
Do
    Nom = InputBox ("Saisissez un nom ou tapez Fini pour quitter")
    If Nom <> "Fini" Then TextBox1.Text = Nom
Loop Until Nom = "Fini"
```

Comme vous pouvez le noter, la boucle qui emploie le mot clé *Until* est similaire à celle qui utilise le mot clé *While*, excepté que la condition de test contient habituellement l'opérateur opposé : l'opérateur = (égal à) au lieu de l'opérateur <> (différent de), dans ce cas. Si vous préférez le mot clé *Until*, n'hésitez pas à l'employer dans les conditions de test de vos boucles Do.

Le contrôle *Timer*

Dans le cadre des outils et des techniques de contrôle du flux que nous étudions au cours de ce chapitre, n'oublions pas le contrôle *Timer* de Visual Studio, qui permet d'exécuter un groupe d'instructions pendant une période donnée ou à des intervalles spécifiques. Le contrôle *Timer* n'est autre qu'un chronomètre invisible qui donne accès à l'horloge système dans les programmes. On peut l'exploiter comme sablier pour décompter à partir d'une heure donnée, provoquer un différé dans un programme ou répéter une action à intervalles définis.

Bien que les objets horloge ne soient pas visibles, chacun est associé à une procédure événementielle qui s'exécute chaque fois que l'intervalle prédéfini de l'horloge s'est écoulé. Pour définir cet intervalle, on utilise la propriété *Interval* et on active l'horloge en positionnant sa propriété *Enabled* sur *True*. Une fois l'horloge activée, elle s'exécute en continu, exécutant sa procédure événementielle à l'intervalle prévu, jusqu'à ce que l'utilisateur arrête le programme ou que l'objet horloge soit désactivé.

Créer une horloge numérique avec le contrôle *Timer*

Parmi les utilisations classiques du contrôle *Timer*, citons la création d'une horloge numérique personnalisée. Dans le prochain exercice, vous allez créer une horloge numérique simple qui récupère l'heure en cours à la seconde. Dans cet exemple, vous définirez la propriété *Interval* de l'horloge sur 1000, indiquant à Visual Studio d'actualiser l'heure toutes les 1000 millisecondes, soit une fois par seconde. Dans la mesure où le système d'exploitation Microsoft Windows est un environnement multitâche et que d'autres programmes réclament l'heure, Visual Studio n'actualise pas l'horloge exactement chaque seconde, mais il se rattrape en cas de retard. Pour vérifier l'heure avec un intervalle différent, comme une fois tous les dixièmes de seconde, il suffit d'ajuster le nombre de la propriété *Interval*.

Créer le programme Horloge numérique

1. Dans le menu Fichier, cliquez sur la commande Nouveau Projet et créez un nouveau projet Application Windows Forms intitulé **Mon Horloge numérique**.

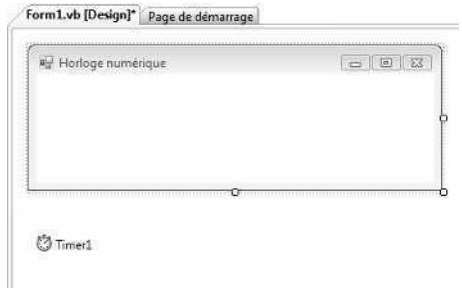
   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans la fenêtre Concepteur.

2. Redimensionnez le formulaire pour créer une petite fenêtre rectangulaire (plus large que haute).

   L'horloge ne doit pas occuper trop d'espace.

3. Sur l'onglet Composant de la Boîte à outils, double-cliquez sur le contrôle *Timer*.

   C'est la première fois que vous utilisez l'onglet Composant et le contrôle *Timer* dans ce livre (l'onglet Composants propose un certain nombre de contrôles intéressants qui travaillent « à l'arrière-plan » des programmes). Visual Studio crée un petit objet horloge dans la zone des composants, sous le formulaire, comme l'illustre la figure suivante :



Rappelez-vous : dans le chapitre 4, « Travailler avec les menus, les barres d'outils et les boîtes de dialogue », nous avons vu que certains contrôles Visual Studio ne possèdent pas de représentation visuelle sur le formulaire. Lors de la création d'objets pour ces contrôles, ils s'affichent dans la zone des composants, sous le formulaire (c'était le cas des contrôles *MenuStrip* et *ToolStrip* étudiés au chapitre 4). Pour définir leurs propriétés, il suffit de les sélectionner dans cette zone, comme nous le ferons pour l'objet horloge dans cet exercice.

4. Dans la Boîte à outils, cliquez sur le contrôle *Label* et tracez un très grand objet étiquette sur le formulaire, de la taille approximative du formulaire.

   Cette étiquette servira à afficher l'heure dans l'horloge et elle doit être de taille suffisante pour contenir des caractères 24 points.

Remarque Lorsque vous créez l'objet étiquette, il se dimensionne automatiquement pour contenir le texte « Label1 » à sa taille par défaut. Toutefois, quand vous positionnerez la propriété *AutoSize* sur *False* à la prochaine étape, l'objet intitulé reprend la taille initiale.

5. Ouvrez la fenêtre Propriétés et définissez les propriétés suivantes pour le formulaire et les deux objets du programme.

Objet	Propriété	Paramètre
Label1	*AutoSize*	False
	Font	Times New Roman, Gras, 24 points
	Text	(vide)
	TextAlign	MiddleCenter
Timer1	*Enabled*	True
	Interval	1000
Form1	*Text*	« Horloge numérique »

Astuce Pour ajouter une image à l'arrière-plan de l'horloge, fixez la propriété *BackgroundImage* de l'objet *Form1* au chemin d'accès du fichier graphique.

Vous allez à présent écrire le code de l'horloge.

6. Double-cliquez sur l'objet horloge dans la zone des composants.

   La procédure événementielle *Timer1_Tick* s'affiche dans l'Éditeur de code. Les programmeurs Visual Basic 6 expérimentés noteront que cette procédure événementielle a été renommée *Timer1_Tick* à la place de *Timer1_Timer*, ce qui clarifie ce qu'elle réalise dans le programme (autrement dit, la procédure événementielle s'exécute à chaque tic d'horloge).

7. Tapez l'instruction suivante :

```
Label1.Text = TimeString
```

Cette instruction récupère l'heure actuelle à partir de l'horloge système et l'assigne à la propriété *Text* de l'objet *Label1* (pour afficher la date dans l'horloge en même temps que l'heure, servez-vous de la propriété *System.DateTime.Now* à la place de la propriété *TimeString*). Ce programme n'exige qu'une instruction puisque vous avez défini la propriété *Interval* de l'horloge en vous servant de la fenêtre Propriétés. L'objet horloge gère le reste.

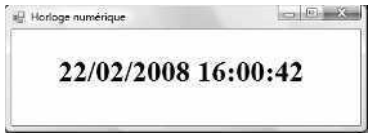
8. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap07 comme emplacement.

Astuce Le programme Horloge numérique complet est disponible dans le dossier c:\vb08epe\chap07\Horloge numérique.

9. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage pour exécuter l'horloge.

L'horloge s'affiche, comme le montre l'illustration suivante (votre heure sera sans doute différente).

En remplaçant la propriété *TimeString* par la propriété *System.DateTime.Now*, le résultat obtenu est le suivant :



J'ai du agrandir l'objet étiquette et le formulaire pour placer la date et l'heure sur une même ligne. Si les informations de votre horloge système passent également à la ligne, fermez le programme et redimensionnez l'étiquette et le formulaire.

10. Observez l'horloge pendant un instant.

Visual Basic actualise l'heure chaque seconde.

11. Cliquez sur le bouton Fermer dans la barre de titre du formulaire pour arrêter l'horloge.

Le programme Horloge numérique est si pratique que vous serez tenté de le compiler dans un fichier exécutable et de l'utiliser de temps en temps sur votre ordinateur. N'hésitez pas à le personnaliser en utilisant vos propres images, textes et couleurs.

Utiliser un objet *Timer* pour définir une limite de temps

L'objet horloge présente un autre intérêt : définir une période d'attente avant de permettre ou d'interdire une action. Vous pouvez également employer la technique de l'horloge pour afficher un message de bienvenue ou un message de copyright à l'écran ou pour répéter un événement à un intervalle donné, comme enregistrer un fichier toutes les dix minutes. Ces actions reviennent à insérer un sablier dans le programme. Vous définissez la propriété *Interval* avec le délai de votre choix puis démarrez l'horloge en attribuant la valeur *True* ou à propriété *Enabled*.

Le prochain exercice montre comment exploiter cette approche pour définir une limite de temps pour la saisie d'un mot de passe (le mot de passe de ce programme est « secret »). Le programme utilise une horloge pour se fermer si un mot de passe valide n'est pas saisi dans un délai de 15 secondes. Un tel programme est généralement intégré à une application plus importante.

Définir une limite de temps pour un mot de passe

1. Dans le menu Fichier, cliquez sur la commande Nouveau Projet et créez un nouveau projet Application Windows Forms intitulé **Mon Mot de passe chronométré**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans la fenêtre Concepteur.
2. Redimensionnez le formulaire pour créer une petite fenêtre rectangulaire de la taille d'une zone de saisie.
3. Dans la Boîte à outils, cliquez sur le contrôle *TextBox* et tracez un objet zone de texte pour le mot de passe au centre du formulaire.
4. Dans la Boîte à outils, cliquez sur le contrôle *Label* puis tracez une longue étiquette au-dessus de la zone de texte.
5. Dans la Boîte à outils, cliquez sur le contrôle *Button* puis tracez un bouton en dessous de la zone de texte.
6. Sur l'onglet Composant de la Boîte à outils, double-cliquez sur le contrôle *Timer*. Visual Studio ajoute un objet horloge dans la zone des composants sous le formulaire.
7. Définissez les propriétés du tableau suivant :

Objet	Propriété	Paramètre
Label1	*Text*	« Saisissez votre mot de passe dans les quinze prochaines secondes »
TextBox1	*PasswordChar*	« * »
Button1	*Text*	« Essayer le mot de passe »
Label1	*Test*	« Saisissez un mot de passe dans les 15 secondes »
TextBox1	*PasswordChar*	« * »
Button1	*Text*	« Essayer le mot de passe »

Objet	Propriété	Paramètre
Timer1	*Enabled*	True
	Interval	15000
Form1	*Text*	« Mot de passe »

Le paramètre *PasswordChar* affiche des caractères astérisque (*) dans la zone de texte au fur et à mesure de la saisie par l'utilisateur d'un mot de passe. La valeur 15000 de la propriété *Interval* laisse 15 secondes à l'utilisateur pour saisir un mot de passe et cliquer sur le bouton Essayer le mot de passe. La valeur *True* de la propriété *Enabled* lance l'exécution de l'horloge au démarrage du programme (pour lancer l'horloge plus tard dans le déroulement du programme, désactivez cette propriété et activez-la dans une procédure événementielle).

Votre formulaire présente un résultat similaire à

8. Double-cliquez sur l'objet horloge dans la zone des composants et tapez les instructions suivantes dans la procédure événementielle *Timer1_Tick* :

```
MsgBox("Désolé, le délai est écoulé.")
End
```

La première instruction affiche un message indiquant que le délai a expiré et la deuxième arrête le programme. Visual Basic exécute cette procédure événementielle si l'intervalle de l'horloge atteint 15 secondes et qu'aucun mot de passe valide n'a été saisi.

9. Affichez le formulaire, double-cliquez sur l'objet bouton et tapez les instructions suivantes dans la procédure événementielle *Button1_Click* :

```
If TextBox1.Text = "secret" Then
     Timer1.Enabled = False
     MsgBox("Bienvenue sur le système !")
     End
Else
     MsgBox("Désolé l'ami, je ne vous connais pas.")
End If
```

Ce code teste si le mot de passe saisi dans la zone de texte est bien « secret ». Si tel est le cas, il désactive l'horloge, affiche un message de bienvenue et se termine (un programme plus pratique continuerait à travailler au lieu de s'arrêter là). Si le mot de passe saisi ne correspond pas, l'utilisateur en est informé dans une boîte de message et peut retenter sa chance. Mais il n'a que quinze secondes pour y parvenir !

10. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap07 comme emplacement.

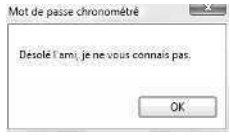
Tester le programme Mot de passe chronométré

Astuce Le programme Mot de passe chronométré complet est disponible dans le dossier c:\vb08epe\chap07\Mot de passe chronométré.

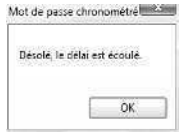
1. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme. Le programme démarrer et l'horloge de 15 secondes démarre le décompte.
2. Tapez **ouvrir** dans la zone de texte. Les astérisques masquent votre saisie, comme le montre l'illustration suivante :

3. Cliquez sur le bouton Essayer le mot de passe.

   La boîte de message suivante s'affiche à l'écran, notifiant une réponse incorrecte :

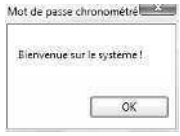


4. Cliquez sur OK et attendez patiemment que la période définie expire. Le programme affiche le message de dépassement suivant dans sa boîte de message :



5. Cliquez sur OK pour terminer le programme.
6. Exécutez à nouveau le programme, tapez **secret** (le mot de passe correct) dans la zone de texte et cliquez sur Essayer le mot de passe.

Le programme affiche le message suivant :



7. Cliquez sur OK pour terminer le programme.

   L'environnement de développement de Visual Basic s'affiche.

Comme vous pouvez l'imaginer, il existe de nombreuses utilisations pratiques pour les objets horloge. À l'instar des boucles *For…Next* et des boucles *Do*, vous pouvez vous servir des objets horloge pour répéter des commandes et des procédures autant de fois que nécessaire dans un programme. Avec ce que vous avez appris concernant les structures de décision *If…Then* et *Select Case* au chapitre 6, vous disposez à présent de plusieurs instructions, contrôles et techniques pour organiser vos programmes, les faire répondre à l'entrée de l'utilisateur et traiter les données de manière innovante. Apprendre à sélectionner le meilleur outil en fonction de la situation de contrôle du flux exige un peu de pratique, bien entendu, mais les prochains chapitres vous offriront de nombreuses opportunités d'essayer ces outils et techniques et de construire d'intéressantes applications. En fait, vous pourriez profiter immédiatement de vos nouvelles connaissances pour créer un ou deux projet(s) simple(s) à partir de rien avant d'attaquer le prochain chapitre, qui traite du débogage. Que pensez-vous de créer une horloge numérique qui affiche une image différente dans un objet zone d'image toutes les 30 secondes ?

Aller plus loin : Insérer des extraits de code

Si vous avez apprécié l'utilisation de l'horloge système et d'autres ressources Windows dans ce chapitre, ce nouvel exemple devrait vous plaire. Il utilise l'objet *Computer.Info* pour afficher des informations relatives au système d'exploitation que vous utilisez. Cet exemple présente également une nouvelle fonctionnalité intéressante de Visual Studio : la commande Insérer un extrait, qui permet d'insérer des modèles ou des extraits de code tous prêts dans l'Éditeur de code à partir d'une liste de tâches de programmation classiques. Visual Studio est automatiquement configuré avec une bibliothèque d'extraits de code, mais rien ne vous empêche d'ajouter des extraits issus de vos programmes ou de ressources en ligne comme MSDN. L'exercice suivant montre comment utiliser cette intéressante fonctionnalité.

Insérer l'extrait Version de Windows active

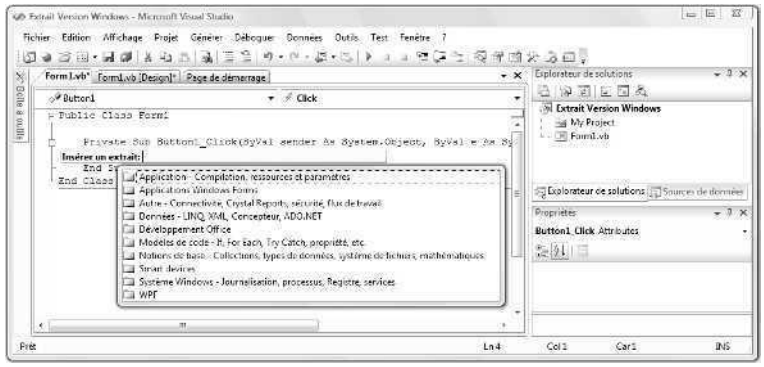
1. Dans le menu Fichier, cliquez sur la commande Nouveau Projet et créez un nouveau projet Application Windows Forms intitulé **Mon Extrait Version Windows**.

   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans la fenêtre Concepteur.

2. Créez un nouvel objet bouton au centre du formulaire et attribuez à la propriété *Text* du bouton la valeur « Afficher la version de Windows ».
3. Double-cliquez sur l'objet bouton pour afficher la procédure événementielle *Button1_Click*.

   Vous allez maintenant utiliser la commande Insérer un extrait pour insérer un modèle de code qui retourne automatiquement des informations relatives à la version de Windows installées sur l'ordinateur. Notez que cet extrait particulier n'est qu'un exemple issu d'une liste de plusieurs dizaines de modèles de code.
4. Dans le menu Edition, pointez vers le sous-menu IntelliSense et choisissez la commande Insérer un extrait.

   La zone de liste Insérer un extrait s'affiche dans l'Éditeur de code, comme le montre l'illustration suivante. Selon les composants de Visual Studio que vous avez installés, la liste des extraits peut varier.



> **Astuce** Vous pouvez également ouvrir la liste des extraits en effectuant un clic droit dans le Concepteur et en sélectionnant Insérer un extrait.

La zone de liste Insérer un extrait est un outil de navigation qui permet d'explorer la bibliothèque d'extraits et d'insérer des extraits dans le programme au point d'insertion. Pour ouvrir un dossier de la zone de liste, double-cliquez sur le nom du dossier. Pour revenir au dossier précédent dans la hiérarchie des dossiers, appuyez sur la touche RETOUR ARRIÈRE.

5. Faites défiler la liste jusqu'en bas et double-cliquez sur le dossier Système Windows – Journalisation, processus, Registre, Services.

   Dans ce dossier se trouvent des extraits relatifs à l'interrogation et à la configuration des paramètres du système d'exploitation.

6. Double-cliquez sur le dossier Windows – Informations système.

   Une liste d'extraits sur les informations système s'affiche. Vous allez à présent sélectionner l'extrait qui retourne les informations relatives à la version actuelle de Windows.

7. Double-cliquez sur l'extrait intitulé Déterminer la version de Windows active.

   Visual Studio insère les deux lignes de code suivantes dans la procédure événementielle *Button1_Click* au point d'insertion :

   ```
   Dim osVersion As String
   osVersion = My.Computer.Info.OSVersion
   ```

   Ces instructions déclarent la variable de chaîne *osVersion* pour contenir les informations de version relatives au système d'exploitation puis utilisent l'objet *Computer.Info* pour remplir la variable avec les informations actuelles. L'extrait utilise également l'espace de noms *My* pour collecter des informations sur l'ordinateur. Cet espace de noms est en nouvelle fonctionnalité « appel rapide » de Visual Studio conçue pour réduire le temps que le code utilise pour les tâches classiques. Nous l'étudierons de manière plus détaillée au chapitre 13, « Explorer le traitement des fichiers texte et des chaînes ».

   Cet extrait de code est appelé *modèle* : il fournit la majorité du code nécessaire pour insérer pour une tâche spécifique, mais ce code n'est pas encore pleinement intégré au projet. Nous devons ajouter une deuxième variable pour contenir le nom du système d'exploitation (puisqu'il existe différentes versions de Windows) et la fonction *MsgBox* pour afficher les résultats (dans d'autres circonstances, vous pourriez ajouter des contrôles au formulaire, créer de nouvelles variables ou des structures de données, voire écrire d'autres instructions qui exploitent l'extrait).

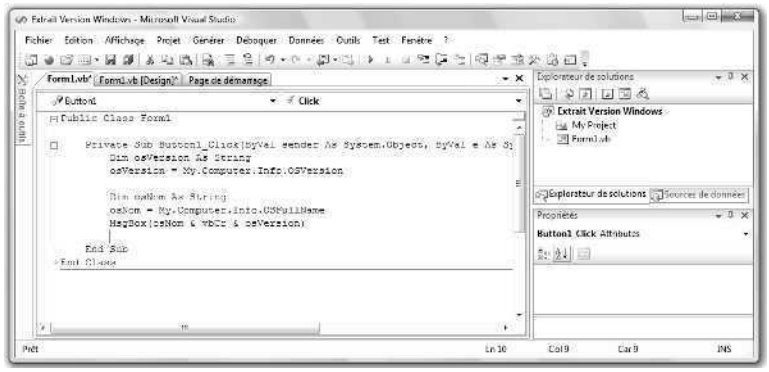
8. Appuyez deux fois sur la touche ENTRÉE pour ajouter une ligne vide sous l'extrait.

9. Tapez les instructions suivantes :

   ```
   Dim osNom As String
   osNom = My.Computer.Info.OSFullName
   MsgBox(osNom & vbCr & osVersion)
   ```

   Ces instructions déclarent une deuxième variable intitulée *osNom* qui contiendra la version de Windows récupérée de la propriété *OSFullName* de l'objet *Computer.Info*. La fonction *MsgBox* affiche les deux valeurs renvoyées : le nom du système d'exploitation (*osNom*) et le numéro de version du système d'exploitation (*osVersion*). Comme vous le savez probablement, le numéro de version du système d'exploitation est relativement détaillé dans Microsoft Windows, puisque Windows peut être automatiquement mis à jour sur le web chaque fois qu'une nouvelle mise à jour de sécurité ou amélioration est mise sur le marché.

Il est donc intéressant de connaître le numéro de version pour vérifier que votre système est à jour et sécurisé. Remarquez également l'emploi de *vbCr*. Cette constante représente un retour chariot. C'est une alternative à l'instruction Chr(13) employée précédemment dans ce chapitre. Plusieurs de ces constantes peuvent être utiles. Vous en découvrirez la liste en saisissant « vb » dans l'Éditeur de code. Votre écran présente un résultat similaire à

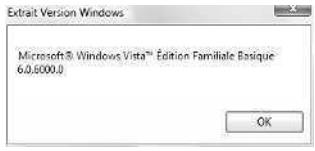


10. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap07.

11. Cliquez sur Démarrer le débogage pour démarrer le programme. Visual Studio exécute le programme dans l'environnement de développement.

12. Cliquez sur le bouton Afficher la version de Windows pour afficher les informations de version retournées par l'extrait.

Votre boîte de dialogue est similaire à la suivante :



13. Cliquez OK pour fermer la boîte de dialogue puis cliquez sur le bouton Fermer pour arrêter le programme.

Vous avez appris une technique pratique qui vous permettra d'insérer une variété de modèles de code dans vos programmes.

Astuce Pour insérer de nouveaux extraits ou réorganiser les extraits dont vous disposez, cliquez sur la commande Gestionnaire des extraits de code dans le menu Outils. La boîte de dialogue Gestionnaire des extraits de code permet de contrôler le contenu de la zone de liste Insérer un extrait. Elle contient également un mécanisme permettant de collecter des extraits de code en ligne.

Rappel du chapitre 7

Pour	Faites ceci

Exécuter un groupe d'instructions un nombre défini de fois	Insérez les instructions entre des instructions *For* et *Next* dans une boucle. Par exemple :
	`Dim i As Integer`
	`For i = 1 To 10`
	`    MsgBox("Appuyez déjà sur OK !")`
	`Next i`

Utiliser une séquence spécifique de nombres avec des instructions	Insérez les instructions dans une boucle *For…Next* et utilisez les mots clé *To* et *Step* pour définir les séquences de nombres. Par exemple :
	`Dim i As Integer`
	`For i = 2 To 8 Step 2`
	`    TextBox1.Text = TextBox1.Text & i`
	`Next i`

Éviter une boucle *Do* sans fin	Vérifiez que la boucle possède une condition de test qui peut prendre la valeur *False*.
Déclarer une variable et lui affecter simultanément une valeur	Servez-vous de *Dim* pour déclarer la variable puis affectez-lui une valeur à l'aide de l'opérateur égal (=). Par exemple : `Dim Compteur As Integer = 1`

Quitter prématurément une boucle *For…Next*	Utilisez l'instruction *Exit For*. Par exemple :
	`Dim Nom As String`
	`Dim i As Integer`
	`For i = 1 To 10`
	`    Nom = InputBox("Nom ?")`
	`    If Nom = "Milou" Then Exit For`
	`    TextBox1.Text = Nom`
	`Next i`

Exécuter un groupe d'instructions jusqu'à ce qu'une condition donnée soit satisfaite	Insérez les instructions entre les instructions *Do* et *Loop*. Par exemple :
	`Dim Question As String = ""`
	`Do While Question <> "Oui"`
	`    Question = InputBox("Milou ?")`
	`    If Question = "Oui" Then MsgBox ("Salut")`
	`Loop`

Boucler jusqu'à ce qu'une condition spécifique soit *True*	Utilisez une boucle *Do* avec le mot clé *Until*. Par exemple :
	`Dim Abandonne As String`
	`Do`
	`    Abandonne = InputBox("Dit 'Oncle'")`
	`Loop Until Abandonne = "Oncle"`

Pour	Faites ceci
Boucler pendant une période spécifique dans le programme	Utilisez un contrôle *Timer*.
Insérer un extrait de code dans le programme	Dans l'Éditeur de code, positionnez le point d'insertion (pointeur en I) à l'emplacement où insérer l'extrait. Dans le menu Edition, pointez sur IntelliSense et choisissez Insérer un extrait. Localisez l'extrait à utiliser et double-cliquez sur son nom.
Ajouter ou réorganiser des extraits dans la zone de liste Insérer un extrait	Cliquez sur la commande Gestionnaire des extraits de code dans le menu Outils.

Chapitre 8

Déboguer les programmes Visual Basic

À la fin de ce chapitre, vous saurez :

- Identifier plusieurs types d'erreurs dans vos programmes
- Utiliser les outils de débogage Microsoft Visual Studio pour placer des points d'arrêt et corriger des erreurs
- Utiliser les fenêtres Automatique et Espion pour examiner des variables pendant l'exécution du programme
- Utiliser un visualiseur pour examiner des types de données chaîne et complexes dans l'environnement de développement
- Utiliser les fenêtres Exécution et Commande pour modifier la valeur des variables et exécuter des commandes dans Visual Studio
- Supprimer des points d'arrêt

Au cours des précédents chapitres, vous avez eu maintes occasions de commettre des erreurs de programmation. Les fautes de grammaire et de prononciation occasionnelles n'entravent généralement pas le déroulement d'une conversation humaine. En revanche, la communication entre le développeur humain et le compilateur Microsoft Visual Basic ne fonctionne que si l'on respecte des règles précises ainsi que les normes du langage de programmation Visual Basic.

Dans ce chapitre, vous allez avancer dans votre découverte des défauts logiciels, ou *bogues*, qui interrompent l'exécution des programmes Visual Basic. Vous découvrirez plusieurs types d'erreurs différents ainsi que la manière d'utiliser les outils de débogage Visual Studio pour détecter et corriger ces défauts. Ces connaissances vous seront utiles pour les programmes développés dans ce livre ainsi que pour les programmes plus longs que vous rédigerez à l'avenir.

Pourquoi se pencher maintenant sur le débogage ? Certains livres de programmation omettent totalement ce sujet ou le renvoient en fin d'ouvrage (après que vous avez appris toutes les caractéristiques de langage d'un produit particulier). Ce choix répond à une certaine logique mais pour ma part, je pense qu'il est plus sensé de maîtriser les techniques de débogage tout en apprenant à programmer afin que la détection et la correction des erreurs fassent partie intégrante de votre approche de la programmation et de la résolution des problèmes. À ce stade de ce livre, vous en savez tout juste assez sur les objets, les structures de décision et la syntaxe des instructions pour développer des programmes intéressants, mais aussi pour vous exposer à un certain nombre de problèmes. Toutefois, comme nous le verrons bientôt, Microsoft Visual Studio 2008 vous permet de détecter facilement les erreurs pour revenir dans le droit chemin.

Localiser et corriger des erreurs

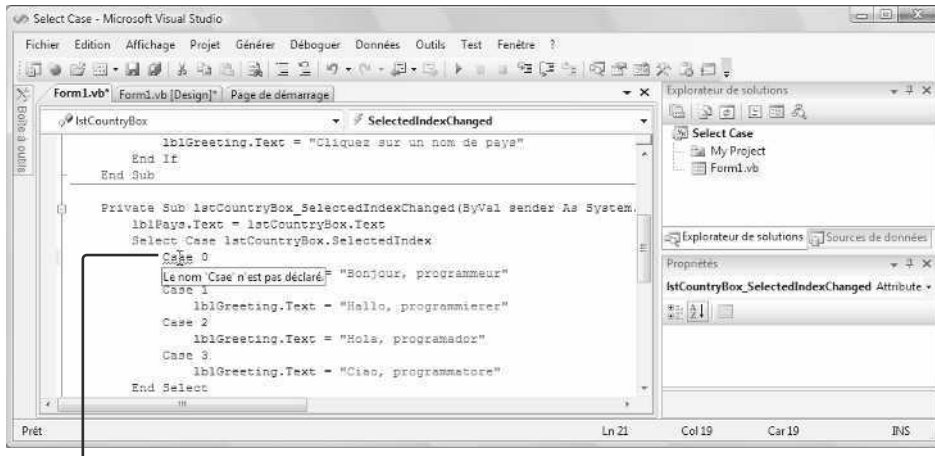
Les défauts que vous avez rencontrés jusqu'à présent dans vos programmes se résumaient probablement à des fautes de frappe ou à des erreurs de syntaxe. Mais qu'en est-il si vous découvrez un problème plus fâcheux dans votre programme – un problème que vous n'arrivez pas à détecter et à corriger en révisant simplement les objets, les propriétés et les instructions que vous avez utilisés ? L'environnement de développement de Visual Studio contient plusieurs outils pour vous aider à dépister et à corriger les erreurs dans vos programmes. Ces outils ne vous empêcheront pas de commettre des erreurs mais vous simplifieront la tâche lorsque vous en rencontrerez une.

Trois types d'erreurs

Un programme Visual Basic peut contenir trois types d'erreurs : les erreurs de syntaxe, les erreurs d'exécution et les erreurs de logique.

- Une *erreur de syntaxe* (ou *erreur de compilation*) est une erreur qui viole les règles de programmation de Visual Basic, comme une propriété ou un mot clé mal orthographiés. Visual Basic met en évidence plusieurs types d'erreurs de syntaxe dans vos programmes au fur et à mesure que vous tapez vos instructions et ne vous permet pas d'exécuter un programme tant que vous n'avez pas corrigé toutes les erreurs de syntaxe.
- Une *erreur d'exécution* est une erreur à l'origine de l'interruption inattendue d'un programme pendant l'exécution. Les erreurs d'exécution se produisent lorsqu'un événement extérieur ou une erreur de syntaxe cachée entraîne l'interruption d'un programme pendant son exécution. Par exemple, si vous avez mal orthographié un nom de fichier dans la méthode *System.Drawing.Image.FromFile* ou si vous tentez de lire une disquette alors que le lecteur est vide, votre code génère une erreur d'exécution.
- Une *erreur de logique* est une erreur humaine – une erreur de programmation qui fait que le code génère des résultats erronés. La plupart des efforts de débogage se concentrent sur la détection des erreurs de logique commises par le programmeur.

Si vous rencontrez une erreur de syntaxe, la documentation de Visual Basic peut vous aider à résoudre ce problème en vous proposant des informations supplémentaires sur le message d'erreur. Il est également possible de corriger l'erreur en examinant attentivement la syntaxe exacte des fonctions, des objets, des méthodes et des propriétés utilisés. Dans l'Éditeur de code, les instructions incorrectes sont soulignées d'une ligne dentelée. En maintenant le pointeur de la souris sur l'instruction, vous obtenez des informations supplémentaires sur l'erreur. La figure qui suit montre le message d'erreur qui s'affiche dans Visual Studio si vous tapez incorrectement le mot clé Case « Csae » et que vous maintenez le pointeur de la souris sur l'erreur. Ce message d'erreur s'affiche sous forme d'infobulle.



Erreur de syntaxe identifiée par le compilateur Visual Basic

Astuce Par défaut, une ligne dentelée verte signale un avertissement, une ligne dentelée rouge une erreur de syntaxe, une ligne dentelée bleue une erreur de compilation et une ligne dentelée pourpre un autre type d'erreur.

En cas d'erreur d'exécution, vous pouvez résoudre le problème en corrigeant la saisie. Par exemple, si une image se charge incorrectement dans un objet zone d'image, il peut s'agir simplement d'une erreur de chemin d'accès. Toutefois, bon nombre d'erreurs d'exécution nécessitent une solution plus approfondie. Pour ce faire, vous pouvez ajouter à vos programmes un *gestionnaire d'erreur structuré* – un bloc de code spécial qui reconnaît une erreur d'exécution lorsqu'elle se produit, supprime tous les messages d'erreur et modifie les conditions du programme afin de résoudre le problème. La nouvelle syntaxe des gestionnaires d'erreur structurés est abordée au chapitre 9, « Gérer les erreurs avec la gestion structurée des exceptions ».

Identifier les erreurs de logique

Dans vos programmes, les erreurs de logique sont souvent les plus difficiles à corriger. Elles résultent d'un raisonnement et d'une planification erronés, et non d'une mauvaise compréhension de la syntaxe Visual Basic. Observez la structure de décision *If…Then* suivante qui évalue deux expressions conditionnelles, puis affiche un message parmi deux en fonction du résultat.

```
If Age > 13 And Age < 20 Then
    TextBox2.Text = "Vous êtes un adolescent"
Else
    TextBox2.Text = "Vous n'êtes pas un adolescent"
End If
```

Pouvez-vous identifier le problème contenu dans cette structure de décision ? Un adolescent est une personne dont l'âge se situe entre 13 et 19 ans inclus, mais la structure est incapable d'identifier la personne qui a exactement 13 ans (pour cet âge, elle affiche à tort le message « Vous n'êtes pas un adolescent »). Il ne s'agit pas d'une erreur de syntaxe (car les instructions suivent les règles Visual Basic) ; il s'agit d'une erreur mentale, ou erreur de logique. La structure de décision appropriée doit contenir un opérateur supérieur ou égal à (>=) dans la première comparaison après l'instruction *If…Then*, comme suit :

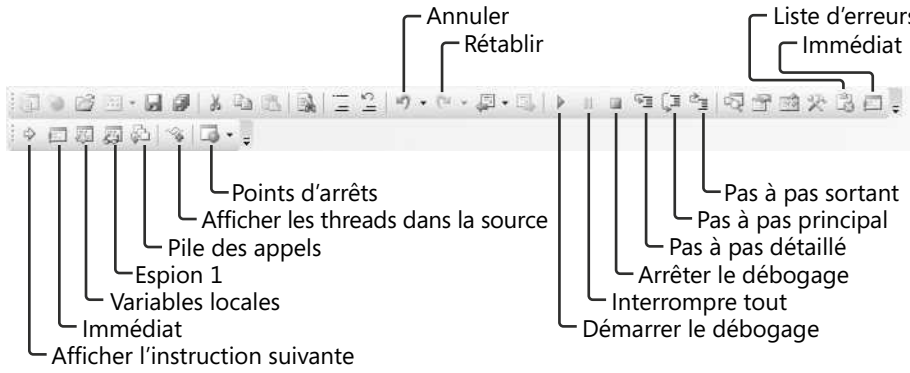
```
If Age >= 13 And Age < 20 Then
```

Croyez-moi ou non, il s'agit du type d'erreur le plus fréquent. Le code qui génère des résultats inattendus est le plus souvent, mais pas toujours, aussi le plus difficile à tester et à corriger.

Débogage 101 : Utilisation du mode Débogage

Pour identifier une erreur de logique, vous pouvez exécuter votre code ligne par ligne et examiner le contenu d'une ou plusieurs variables ou propriétés à mesure qu'elles changent. Pour ce faire, passez en *mode débogage* (en anglais, *break mode*) pendant l'exécution de votre programme, puis observez votre code dans l'Éditeur de code. Le mode Débogage offre un gros plan de votre programme pendant que le compilateur l'exécute. C'est comme si vous étiez placé derrière le pilote et le copilote d'un avion et que vous les regardiez piloter. En revanche, dans votre cas, vous avez accès aux contrôles.

Pendant le débogage de votre application, vous allez utiliser les boutons de la barre d'outils Standard et de la barre d'outils Déboguer, ainsi que les commandes du menu Déboguer et les boutons et fenêtres spéciaux de l'environnement de développement. La figure qui suit montre les boutons de débogage des barres d'outils Standard et Déboguer que l'on ouvre en pointant sur la commande Barres d'outils dans le menu Affichage et en cliquant sur Standard ou Déboguer (certains boutons ont été ajoutés *via* la commande Personnaliser, qui se trouve dans la partie inférieure de la liste des barres d'outils).



Dans l'exercice suivant, vous allez placer un point d'arrêt – un emplacement dans le programme où l'exécution s'interrompt. Vous allez ensuite utiliser le mode débogage pour localiser et corriger l'erreur de logique découverte précédemment dans la structure *If…Then* (cette erreur fait partie d'un programme réel). Pour isoler le problème, utilisez le bouton Pas à pas détaillé de la barre d'outils Standard pour exécuter une à une les instructions du programme. Utilisez ensuite la fenêtre Automatique pour examiner la valeur des principales variables et propriétés du programme. Observez attentivement cette stratégie de débogage. Elle vous servira à corriger de nombreux types de problèmes techniques dans vos propres programmes.

Déboguer le programme Test de débogage

1. Démarrez Visual Studio.
2. Dans le menu Fichier, cliquez sur Nouveau Projet.

   La boîte de dialogue Nouveau projet s'affiche.
3. Ouvrez le projet Test de débogage dans le dossier c:\vb08epe\chap08\Test de débogage.

   Le programme s'ouvre dans l'environnement de développement.
4. Affichez le formulaire si ce n'est déjà fait.

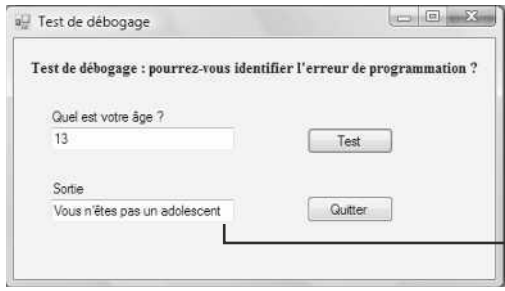
   Le programme Test de débogage demande à l'utilisateur son âge. Lorsque l'utilisateur clique sur le bouton Test, le programme indique à l'utilisateur s'il est un adolescent ou non. Toutefois, le programme présente toujours le problème lié aux personnes âgées de 13 ans, tel qu'identifié précédemment dans ce chapitre. Ouvrons maintenant la barre d'outils Déboguer et plaçons un point d'arrêt pour localiser le problème.
5. Si la barre d'outils Déboguer n'est pas visible, dans le menu Affichage, pointez sur Barres d'outils, puis cliquez sur Déboguer.

   La barre d'outils Déboguer s'affiche en dessous ou à droite de la barre d'outils Standard.
6. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute et affiche le formulaire Test de débogage.
7. Dans la zone de texte Age, supprimez le 0, tapez **14**, puis cliquez sur le bouton Test.

   Le programme affiche le message « Vous êtes un adolescent ». À ce stade, le programme affiche le résultat adéquat.
8. Tapez **13** dans la zone de texte, puis cliquez sur le bouton Test.

   Le programme affiche le message « Vous n'êtes pas un adolescent », comme ci-après.

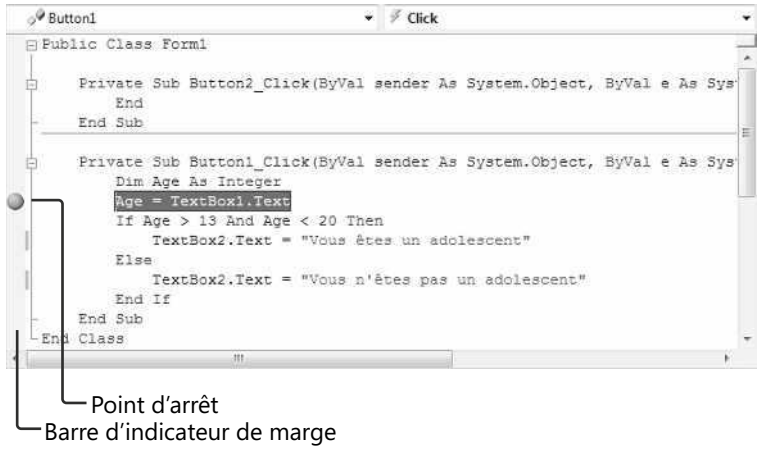


Cette réponse est incorrecte. Il faut examiner le code pour corriger le problème.

9. Cliquez sur le bouton Quitter du formulaire, puis ouvrez l'Éditeur de code.

10. Placez le pointeur de la souris sur la barre Indicateur de marge (la barre grise juste à côté de la marge de gauche dans la fenêtre de l'Éditeur de code), en regard de l'instruction `Age = TextBox1.Text` dans la procédure événementielle *Button1_Click*, puis cliquez sur la barre pour placer un point d'arrêt.

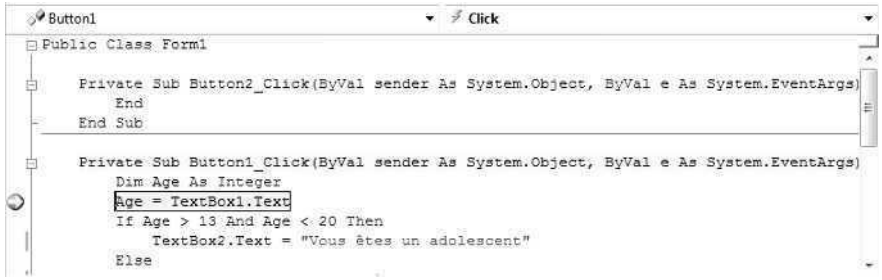
Le point d'arrêt s'affiche immédiatement en rouge. La figure qui suit montre l'emplacement et l'aspect du point d'arrêt.



11. Cliquez sur le bouton Démarrer le débogage pour exécuter de nouveau le programme. Le formulaire s'affiche exactement comme avant et vous pouvez poursuivre vos tests.

12. Tapez **13** dans la zone de texte, puis cliquez sur le bouton Test.

Visual Basic ouvre de nouveau l'Éditeur de code et affiche la procédure événementielle *Button1_Click* – le code qu'exécute actuellement le compilateur. L'instruction que vous avez sélectionnée comme point d'arrêt est surlignée en jaune et une flèche s'affiche dans la barre Indicateur de marge, comme dans la figure ci-après :



Vous savez que Visual Studio se trouve désormais en mode débogage car le mot « En cours de débogage » s'affiche dans sa barre de titre. En mode débogage, vous avez la possibilité de voir comment s'évalue la logique du programme.

Remarque Dans un programme Visual Basic, il est également possible de passer en mode débogage en introduisant l'instruction *Stop* dans votre code à l'emplacement où vous souhaitez interrompre l'exécution. Il s'agit d'une vieille méthode qui fonctionne toujours.

13. Placez le pointeur sur la variable *Age* dans l'Éditeur de code.

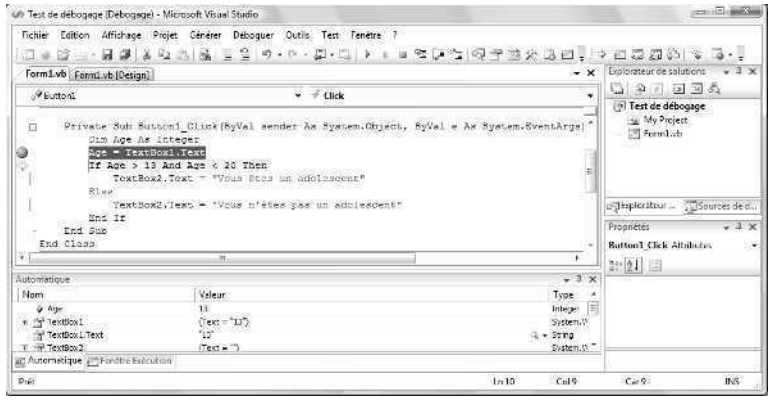
Visual Studio affiche le message « Age | 0 ». En mode débogage, vous pouvez afficher la valeur des variables et des propriétés en maintenant simplement le pointeur de la souris sur la valeur dans le code. La variable *Age* contient actuellement une valeur de 0 car elle n'a pas encore été remplie par la zone de texte *TextBox1* – il s'agit de la prochaine instruction que le compilateur va évaluer.

14. Dans la barre d'outils Déboguer, cliquez sur le bouton Pas à pas détaillé pour exécuter l'instruction suivante.

Ce bouton permet d'exécuter la prochaine instruction dans la procédure événementielle (la ligne actuellement en surbrillance). En cliquant sur ce bouton, vous pouvez observer le changement du programme chaque fois qu'une instruction est évaluée. Si vous maintenez le pointeur sur la variable *Age*, vous verrez qu'elle contient la valeur 13.

15. Dans le menu Déboguer, pointez sur Fenêtres, puis cliquez sur Automatique.

Le sous-menu Fenêtres offre un accès au jeu complet des fenêtres de débogage de Visual Studio. La fenêtre Automatique montre l'état des variables et des propriétés en cours d'utilisation (pas uniquement celles que vous définissez actuellement, toutes les autres également). Comme vous pouvez l'observer dans la figure suivante, la variable *Age* contient la valeur 13, la propriété *TextBox1.Text* contient la chaîne « 13 » et la propriété *TextBox2.Text* contient une chaîne vide ("").



16. Cliquez à deux reprises sur le bouton Pas à pas détaillé.

L'instruction *If* évalue l'expression conditionnelle à *False* et le compilateur passe à l'instruction *Else* dans la structure de décision. Voici notre bogue : la logique de la structure de décision est incorrecte car une personne âgée de 13 ans *est bien* un adolescent.

17. Sélectionnez le test conditionnel `Age > 13` puis maintenez le pointeur sur le texte sélectionné. Visual Studio évalue la condition et affiche le message « Age > 13 | False ».

18. Sélectionnez le test conditionnel `Age < 20` puis maintenez le pointeur sur le texte sélectionné.

Visual Studio affiche le message « Age < 20 | True ». Le pointeur nous a fourni une information supplémentaire – seul le premier test conditionnel génère un résultat incorrect. Comme une personne âgée de 13 ans est un adolescent, Visual Basic devrait évaluer le test à *True*, mais la condition `Age > 13` retourne une valeur False. Cela force l'exécution de la clause *Else* dans la structure de décision. Reconnaissez-vous ce problème ? La première comparaison requiert que l'opérateur supérieur ou égal (>=) analyse spécifiquement ce cas frontière de 13. Vous allez arrêter le débogage afin de corriger cette erreur de logique.

19. Dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage.
20. Dans l'Éditeur de code, ajoutez l'opérateur égal (=) à la première condition dans l'instruction *If*, comme suit :

    ```
    If Age >= 13 And Age < 20 Then
    ```

21. Revenez de nouveau au programme et testez votre solution en vous attardant tout particulièrement sur les nombres 12, 13, 19 et 20 : les cas frontière susceptibles de soulever des problèmes.

    Un point d'arrêt est encore défini : vous entrez en mode débogage lors de l'exécution du programme. Utilisez le bouton Pas à pas détaillé pour examiner le flot du programme autour de l'instruction *If* décisive et exploitez la fenêtre Automatique pour suivre la valeur de vos variables tout au long des tests. Lorsque le formulaire s'affiche, saisissez une nouvelle valeur et effectuez de nouveau ce test. Vous découvrirez en outre qu'en sélectionnant certaines expressions, comme les tests conditionnels, et en maintenant au-dessus le pointeur de la souris, vous comprendrez mieux comment ils sont évalués. Vous apprendrez plus loin dans ce chapitre comment supprimer le point d'arrêt.
22. Après avoir testé le mode débogage, dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage pour fermer le programme.

Félicitations ! Vous avez utilisé avec succès le mode débogage pour localiser et corriger une erreur de logique dans un programme.

Suivre des variables grâce à la fenêtre Espion

La fenêtre Automatique permet d'examiner l'état de certaines variables et propriétés au fur et à mesure que le compilateur les évalue, mais les éléments de cette fenêtre *persistent*, ou maintiennent leurs valeurs, uniquement dans l'instruction en cours (l'instruction en surbrillance dans le débogueur) et dans l'instruction précédente (celle qui vient de s'exécuter). Lorsque votre programme exécute du code qui n'exploite pas ces variables, elles disparaissent de la fenêtre Automatique.

Pour afficher le contenu des variables et des propriétés *pendant* l'exécution d'un programme, utilisez une fenêtre Espion. Il s'agit d'un outil Visual Studio spécial qui suit les valeurs importantes à votre place tant que vous travaillez en mode débogage. Dans Visual Basic 6, il est possible d'ouvrir une fenêtre Espion pour observer le changement des variables. Dans Visual Studio, vous pouvez ouvrir jusqu'à quatre fenêtres Espion, numérotées Espion 1, Espion 2, Espion 3 et Espion 4. Pour ce faire, en mode débogage, pointez sur la commande Fenêtres du menu Déboguer, pointez sur Espion, puis cliquez sur la fenêtre voulue dans le sous-menu Espion. Vous pouvez également ajouter dans une fenêtre Espion des expressions comme `Age >= 13`.

Ouvrir la fenêtre Espion

> **Astuce** Le projet Test de débogage se trouve dans le dossier c:\vb08epe\chap08\Test de débogage.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage pour exécuter de nouveau le programme Test de débogage.

   Je suis parti du principe que le point d'arrêt placé à la ligne `Age = TextBox1.Text` dans le précédent exercice est toujours présent. Dans le cas contraire, arrêtez le programme et définissez-le en cliquant dans la barre indicateur de marge en regard de l'instruction, comme nous l'avons vu à l'étape 10 du précédent exercice, puis redémarrez le programme.

2. Tapez **20** dans la zone de texte Age, puis cliquez sur le bouton Test.

   Le programme s'arrête au point d'arrêt et Visual Studio entre en mode débogage ; il s'agit du mode approprié pour ajouter des variables, des propriétés ou des expressions à une fenêtre Espion. Pour ajouter un élément, on sélectionne sa valeur dans l'Éditeur de code, on clique droit sur la sélection, puis sur la commande Ajouter un espion.

3. Sélectionnez la variable *Age*, effectuez dessus un clic droit, puis cliquez sur la commande Ajouter un espion.

   Visual Studio ouvre la fenêtre Espion 1 et y ajoute la variable *Age*. La valeur de cette variable est actuellement de 0 et la colonne Type de la fenêtre identifie la variable *Age* comme étant de type *Integer*.

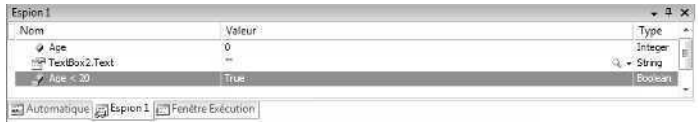
   Un autre moyen d'ajouter un élément consiste à effectuer un glisser-déposer de l'Éditeur de code dans la fenêtre Espion.

4. Sélectionnez la propriété *TextBox2.Text* et faites-la glisser dans la rangée vide de la fenêtre Espion.

   Lorsque vous relâchez le bouton de la souris, Visual Studio ajoute la propriété et affiche sa valeur (actuellement, cette propriété est une chaîne vide).

5. Sélectionnez l'expression `Age < 20` et ajoutez-la dans la fenêtre Espion.

   `Age < 20` est une expression conditionnelle. Utilisez la fenêtre Espion pour afficher sa valeur logique, ou booléenne, comme vous l'avez fait précédemment dans ce chapitre en maintenant le pointeur sur une condition. Votre fenêtre Espion présente un résultat similaire à



   Passez maintenant au code pour voir comment les valeurs de la fenêtre Espion changent.

6. Dans la barre d'outils Déboguer, cliquez sur le bouton Pas à pas détaillé.

> **Astuce** Au lieu de cliquer sur le bouton Pas à pas détaillé dans la barre d'outils Déboguer, vous pouvez appuyer sur la touche F8 du clavier.

La variable *Age* est définie à 20 et la condition `Age < 20` s'évalue à *False*. Ces valeurs s'affichent en rouge dans la fenêtre Espion car elles viennent tout juste d'être actualisées.

7. Cliquez à trois reprises sur le bouton Pas à pas détaillé.

   La clause *Else* s'exécute dans la structure de décision et la valeur de la propriété *TextBox2.Text* de la fenêtre Espion devient « Vous n'êtes pas un adolescent ». Ce test conditionnel fonctionne correctement. Comme cette condition vous satisfait, vous pouvez supprimer ce test de la fenêtre Espion.

8. Dans la fenêtre Espion, cliquez sur la rangée `Age < 20` puis appuyez sur Effacer tout.

   Visual Studio supprime la valeur de la fenêtre Espion. Comme vous pouvez le constater, l'ajout et la suppression de valeurs de la fenêtre Espion est un processus rapide.

Laissez pour le moment Visual Studio s'exécuter en mode débogage. Dans la prochaine section, nous allons continuer à utiliser la fenêtre Espion.

Visualiseurs : Les nouveaux outils de débogage qui affichent des données

Bien qu'il soit possible d'exploiter les fenêtres Espion, Automatique et Variables locales pour examiner des types de données simples tels que *Integer* et *String* dans l'environnement de développement, vous serez sans aucun doute amenés à rencontrer des données plus complexes dans vos programmes. Par exemple, vous pouvez examiner une variable ou une propriété contenant des informations structurées provenant d'une base de données (un dataset) ou une chaîne contenant des informations de formatage HTML ou XML issues d'une page web. Pour pouvoir examiner de plus près ce type d'élément au cours d'une session de débogage, l'environnement de développement de Visual Studio propose un jeu d'outils appelés visualiseurs. L'icône d'un visualiseur est une petite loupe.

L'environnement de développement de Visual Studio 2008 propose quatre visualiseurs standards : les visualiseurs de texte, HTML et XML (qui fonctionnent sur des objets chaîne) et le visualiseur dataset (qui fonctionne avec des objets *DataSet*, *DataView* et *DataTable*). Microsoft a laissé entendre qu'il proposerait ultérieurement en téléchargement d'autres visualiseurs. Visual Studio a été conçu de telle sorte que les développeurs tiers puissent rédiger leurs propres visualiseurs et les installer dans le débogueur de Visual Studio. Dans l'exercice suivant, vous allez observer le fonctionnement du visualiseur texte. Dans cet exercice, je suis parti du principe que vous vous trouvez toujours en mode débogage et que la fenêtre Espion est ouverte sur quelques expressions appartenant au programme Test de débogage.

Ouvrir un visualiseur de texte dans le débogueur

1. Localisez dans la partie droite de la fenêtre Espion l'icône d'une petite loupe.

   L'icône d'une loupe indique qu'un visualiseur est disponible pour la variable ou la propriété examinée dans une fenêtre Espion, Automatique ou Variables locales. Si vous avez achevé l'exercice précédent, la propriété *TextBox2.Text* présente désormais un visualiseur.

2. Cliquez sur la flèche du visualiseur.

   Si la propriété examinée est une propriété texte (*string*), Visual Studio propose trois visualiseurs : un simple visualiseur de texte qui affiche l'expression de chaîne sélectionnée en texte clair, un visualiseur HTML qui convertit le code HTML en page web et un visualiseur XML qui convertit le code XML en document affichable. La fenêtre Espion ressemble à la figure suivante.

3. Cliquez sur l'option Visualiseur de texte.

   Visual Studio ouvre une boîte de dialogue et affiche le contenu de la propriété *TextBox2.Text*. Votre écran présente un résultat similaire à

   Bien que ce résultat soit un peu plus riche en informations que ce que la fenêtre Espion vous a proposé, les avantages de l'outil visualiseur deviennent manifestes lorsque la propriété *Text* d'un objet zone de texte multiligne s'affiche ou lors de l'examen des variables ou des propriétés contenant des informations de base de données ou des documents web. Vous utiliserez ces types de données plus sophistiqués plus loin dans ce livre.

4. Cliquez sur Fermer pour fermer la boîte de dialogue Visualiseur de texte.

   Laissez Visual Studio s'exécuter en mode débogage. Dans la prochaine section, vous allez également utiliser la fenêtre Espion.

Astuce En mode débogage, les visualiseurs apparaissent également dans des fenêtres dans l'Éditeur de code. Lorsque vous pointez sur une variable ou une propriété dans l'Éditeur de code pendant une session de débogage, une Info-bulle apparaît. Cliquez sur l'icône de la loupe pour obtenir des informations supplémentaires, comme dans l'exercice précédent.

Fenêtres Exécution et Commande

Jusqu'à présent, vous avez utilisé les outils de débogage de Visual Studio qui vous permettent d'entrer en mode débogage, d'exécuter le code instruction par instruction et d'examiner la valeur des variables, des propriétés et des expressions importantes dans votre programme. Vous allez maintenant apprendre à modifier la valeur d'une variable grâce à la fenêtre Exécution et découvrir comment exécuter des commandes comme Enregistrer tout ou Imprimer dans l'environnement de développement Visual Studio grâce à la fenêtre Commande. Ces deux fenêtres possèdent des barres de défilement, si bien que vous pouvez exécuter plusieurs commandes et examiner les résultats en utilisant les flèches de direction.

Les exercices suivants montrent le fonctionnement des fenêtres Exécution et Commande. J'ai choisi de les traiter conjointement car les commandes spéciales suivantes vous permettent de basculer de l'une à l'autre.

- Dans la fenêtre Exécution, la commande >*cmd* permet de basculer vers la fenêtre Commande.
- Dans la fenêtre Commande, la commande *immed* permet de basculer vers la fenêtre Exécution.

Cet exercice part du principe que vous déboguez le programme Test de débogage en mode débogage.

Utiliser la fenêtre Exécution pour modifier une variable

1. Dans le menu Déboguer, pointez sur Fenêtres, puis cliquez sur Immédiat.

   Lorsque vous sélectionnez la commande, Visual Studio ouvre la fenêtre Exécution et prépare le compilateur à recevoir vos commandes *pendant l'exécution du programme Test de débogage*. Cette fonctionnalité est très pratique car elle vous permet de tester les conditions du programme à la volée, sans interrompre le programme ni insérer d'instructions dans l'Éditeur de code.

2. Dans la fenêtre Exécution, tapez **Age = 17** puis appuyez sur ENTRÉE.

   Vous venez d'utiliser la fenêtre Exécution pour modifier la valeur d'une variable. La valeur de la variable *Age* est immédiatement remplacée par 17 dans la fenêtre Espion. La prochaine fois que vous exécuterez l'instruction *If*, la valeur de la propriété *TextBox2.Text* sera remplacée par « Vous êtes un adolescent ». Votre fenêtre Exécution présente un résultat similaire à

3. Tapez l'instruction suivante dans la fenêtre Exécution, puis appuyez sur ENTRÉE.

```
TextBox2.Text = "Vous avez le bon âge !"
```

   La propriété *Text* de l'objet *TextBox2* est immédiatement remplacée par « Vous avez le bon âge ! » Dans la fenêtre Exécution, vous pouvez modifier la valeur des propriétés ainsi que celles des variables.

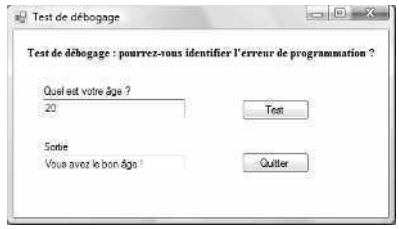
4. Affichez la fenêtre Espion 1 si elle n'est pas visible (cliquez sur l'onglet Espion 1 dans l'environnement de développement de Visual Studio.

   Votre fenêtre Espion présente un résultat similaire à

   Comme vous pouvez le constater, les deux éléments contiennent désormais de nouvelles valeurs, ce qui vous permet de tester plus avant le programme.

5. Cliquez à deux reprises sur le bouton Pas à pas détaillé pour afficher de nouveau le formulaire Test de débogage.

   Notez que la propriété *Text* de l'objet *TextBox2* a été modifiée à votre initiative, mais que la propriété *Text* de l'objet *TextBox1* contient toujours la valeur 20 (et non 17). En effet, vous avez modifié la variable *Age* dans le programme, mais pas la propriété qui attribue une valeur à *Age*. Votre écran présente un résultat similaire à



La fenêtre Exécution présente de nombreux usages : elle complète à merveille la fenêtre Espion et permet de tester des cas spécifiques qu'il serait difficile d'insérer autrement dans votre programme.

Basculer vers la fenêtre Commande

La fenêtre textuelle Commande vient en complément de la fenêtre Visual Studio Exécution. Rappelant les invites de commandes MS-DOS, elle sert à exécuter des commandes d'interface dans l'environnement de développement Visual Studio. Par exemple, le fait de taper la commande *Fichier.Enregistrertout* dans la fenêtre Commande enregistre tous les fichiers dans le projet en cours (cette commande équivaut à la commande Enregistrer tout du menu Fichier). Si vous avez déjà ouvert la fenêtre Exécution, vous pouvez basculer de la fenêtre Exécution à la fenêtre Commande en tapant les commandes *>cmd* et *immed*, respectivement. Vous pouvez également ouvrir la fenêtre Commande en cliquant sur le menu Affichage, en pointant vers Autres fenêtres puis en cliquant sur Fenêtre Commande. Dans le prochain exercice, vous allez vous entraîner à utiliser cette fenêtre.

Exécuter la commande Fichier.Enregistrertout

1. Dans la fenêtre Exécution, tapez **>cmd**, puis appuyez sur ENTRÉE pour basculer vers la fenêtre Commande.

   La fenêtre Commande s'ouvre et la fenêtre Exécution ou Espion peut maintenant être partiellement (ou totalement) masquée. Pour revenir à la fenêtre Exécution, cliquez sur son onglet ou tapez **immed** dans la fenêtre Commande. L'invite > apparaît. C'est un repère visuel qui vous indique que vous travaillez dans la fenêtre Commande.

2. Tapez **Fichier.Enregistrertout** dans la fenêtre Commande, puis appuyez sur ENTRÉE.

   En tapant Fichier, toutes les commandes Visual Studio associées au menu Fichier et aux opérations relatives aux fichiers apparaissent dans une zone de liste déroulante. Ce dispositif IntelliSense d'« achèvement de commande » est un moyen utile d'obtenir des informations sur les nombreuses commandes que l'on peut exécuter dans la fenêtre Commande. Après avoir tapé *Fichier.Enregistrertout* et appuyé sur

ENTRÉE, Visual Studio enregistre le projet en cours et l'invite de commandes revient, comme le montre la figure suivante :

3. Testez d'autres commandes si vous le souhaitez. Faites précéder vos commandes de noms de menu pour découvrir les différentes commandes disponibles. Lorsque vous avez terminé, cliquez sur le bouton Fermer dans les fenêtres Commande et Exécution. Votre travail sur ces fenêtres est terminé pour le moment.

Aller plus loin : Supprimer des points d'arrêt

Si vous avez suivi attentivement les instructions de ce chapitre, le programme Test de débogage est toujours en cours d'exécution et contient un point d'arrêt. Pour supprimer ce point d'arrêt et terminer le programme, suivez ces étapes. Vous aurez terminé de déboguer le programme Test de débogage.

Supprimer un point d'arrêt

1. Dans l'Éditeur de code, cliquez sur le cercle rouge associé au point d'arrêt dans la barre Indicateur de marge.

   Le point d'arrêt disparaît. Le tour est joué ! Si votre programme contient plusieurs points d'arrêt, vous pouvez tous les supprimer en cliquant sur la commande Supprimer tous les points d'arrêt dans le menu Déboguer. Visual Studio enregistre les points d'arrêt avec votre projet ; il est donc important de savoir comment les supprimer. Dans le cas contraire, ils demeureront dans votre programme, même si vous fermez Visual Studio et que vous redémarrez !

2. Dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage.

   Le programme Test de débogage se termine.

3. Dans le menu Affichage, pointez sur Barres d'outils, puis cliquez sur Déboguer.

   La barre d'outils Déboguer se ferme.

Vous venez d'apprendre les principales techniques de débogage des programmes Visual Basic avec Visual Studio. Placez un signet dans ce chapitre afin de pouvoir y revenir en cas de problème plus loin dans ce livre. Au prochain chapitre, vous allez apprendre à traiter les erreurs d'exécution grâce à des techniques de gestion des erreurs structurées.

Rappel du chapitre 8

Pour	Faites ceci
Afficher la barre d'outils Déboguer	Dans le menu Affichage, pointez sur Barres d'outils, puis cliquez sur Déboguer.
Placer un point d'arrêt	Dans l'Éditeur de code, cliquez sur la barre Indicateur de marge en regard de l'instruction où vous souhaitez interrompre l'exécution du programme. Lorsque le compilateur atteint le point d'arrêt, il entre en mode débogage. *ou* Placez une instruction *Stop* dans le code du programme à l'endroit où vous souhaitez entrer en mode débogage.
Exécuter une ligne de code dans l'Éditeur de code	Dans la barre d'outils Standard, cliquez sur le bouton Pas à pas détaillé.
Examiner une variable, une propriété ou une expression dans l'Éditeur de code	En mode débogage, sélectionnez la valeur dans l'Éditeur de code, puis maintenez le pointeur dessus.
Utiliser la fenêtre Automatique pour examiner une variable sur la ligne en cours ou la ligne précédente	En mode débogage, cliquez sur le menu Déboguer, pointez sur Fenêtres, puis cliquez sur Automatique.
Ajouter une variable, une propriété ou une expression à une fenêtre Espion	En mode débogage, cliquez sur le menu Déboguer, pointez sur Fenêtres, puis cliquez sur Ajouter un espion.
Afficher une fenêtre Espion	En mode débogage, cliquez sur le menu Déboguer, pointez sur Fenêtres, pointez sur Espion, puis cliquez sur la fenêtre.
Afficher des informations HTML, XML ou dataset pendant une session de débogage	Cliquez sur l'icône du visualiseur dans une fenêtre Espion, Variables locales ou l'infobulle pendant une session de débogage.
Ouvrir une fenêtre Exécution	Dans le menu Déboguer, pointez sur Fenêtres, puis cliquez sur Immédiat.
Exécuter une commande dans l'environnement de développement Visual Studio depuis la fenêtre Commande	À l'invite >, tapez le nom de la commande, puis appuyez sur Entrée. Par exemple, pour enregistrer le projet en cours, tapez **Fichier.Enregistrertout** puis appuyez sur ENTRÉE.
Basculer vers la fenêtre Commande depuis la fenêtre Exécution	Tapez **>cmd** puis appuyez sur ENTRÉE. Pour revenir vers la fenêtre Exécution, tapez **immed** puis appuyez sur ENTRÉE.
Supprimer un ou plusieurs points d'arrêt	Cliquez sur le point d'arrêt dans la barre Indicateur de marge de l'Éditeur de code. *ou* Dans le menu Déboguer, cliquez sur la commande Supprimer tous les points d'arrêt.
Arrêter le débogage	Dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage.

Chapitre 9

Gérer les erreurs avec la gestion structurée des exceptions

À la fin de ce chapitre, vous saurez :

- Gérer des erreurs d'exécution en utilisant le gestionnaire d'erreur *Try…Catch*
- Créer un gestionnaire d'erreur de disque qui teste des conditions d'erreurs spécifiques grâce à l'instruction *Catch When*
- Rédiger des gestionnaires d'erreur complexes qui utilisent l'objet *Err* et les propriétés *Err.Number* et *Err.Description* pour identifier des exceptions
- Construire des instructions *Try…Catch* imbriquées
- Associer des gestionnaires d'erreur à des techniques de programmation défensives
- Quitter prématurément des gestionnaires d'erreur grâce à l'instruction *Exit Try*

Au chapitre 8, « Déboguer les programmes Visual Basic », vous avez appris à reconnaître les erreurs d'exécution dans un programme Microsoft Visual Basic et à localiser les erreurs de logique ainsi que d'autres défauts du code grâce aux outils de débogage de Visual Studio 2008. Dans ce chapitre, vous allez apprendre à construire des blocs de code pour gérer des erreurs d'exécution, également appelées *exceptions*, qui surviennent suite à des conditions de fonctionnement normales – il peut s'agir, par exemple, d'un CD ou d'un DVD absent du lecteur, d'une connexion Internet interrompue ou d'une imprimante déconnectée. Ces routines s'appellent des *gestionnaires d'erreur structurés* (ou *gestionnaires d'exception structurés*). Vous pouvez les utiliser pour repérer des erreurs d'exécution, supprimer des messages d'erreur non désirés et adapter les conditions du programme afin que votre application retrouve le contrôle et puisse fonctionner de nouveau.

Heureusement, Visual Basic propose le puissant bloc de code *Try…Catch* pour gérer les erreurs. Dans ce chapitre, vous allez apprendre à détecter les erreurs d'exécution grâce à des blocs de code *Try…Catch* et à utiliser les propriétés *Err.Number* and *Err.Description* pour identifier des erreurs d'exécution spécifiques. Vous allez également apprendre à exploiter plusieurs instructions *Catch* pour rédiger des gestionnaires d'erreur plus souples, à construire des blocs de code *Try…Catch* imbriqués et à utiliser l'instruction *Exit Try* pour quitter prématurément un bloc de code *Try…Catch*. Les techniques de programmation que vous allez apprendre constituent des améliorations notables par rapport à ce que permettait Visual Basic 6. Elles ressemblent aux gestionnaires d'erreur structurés fournis par les langages de programmation les plus avancés comme Java et C++. Les programmes Visual Basic les plus fiables, ou *robustes*, exploitent plusieurs gestionnaires d'erreur pour gérer des événements inattendus et offrir aux utilisateurs des expériences cohérentes et fluides.

Gérer les erreurs grâce au bloc Try...Catch

Dans un programme Visual Basic, la panne d'un programme est un problème inattendu duquel le programme ne peut pas récupérer. Vous avez sans doute fait l'expérience de votre première panne lorsque Visual Basic ne pouvait pas charger une image à partir d'un fichier ou, dans le précédent chapitre, lorsque vous avez volontairement introduit des erreurs dans le code pendant le débogage. Ce n'est pas que Visual Basic ne soit pas suffisamment intelligent pour gérer l'incident. Simplement, il n'avait pas été « dit » au programme ce qu'il devait faire en cas de panne.

Heureusement, rien ne vous oblige à vivre avec des erreurs occasionnelles entraînant l'interruption de vos programmes. En effet, vous avez la possibilité de rédiger des routines Visual Basic spéciales, appelées *gestionnaires d'erreur structurés*, pour gérer et répondre aux erreurs d'exécution avant qu'elles ne contraignent le compilateur Visual Basic à mettre fin au programme. Un gestionnaire d'erreur gère une erreur d'exécution en indiquant au programme comment poursuivre si une de ses instructions ne fonctionne pas. On peut placer des gestionnaires d'erreur dans chaque procédure événementielle présentant un risque potentiel, ou dans des fonctions génériques ou des sous-programmes qui reçoivent le contrôle une fois que l'erreur s'est produite et qui gèrent systématiquement le problème. Vous en saurez plus sur la rédaction des fonctions et des sous-programmes au chapitre 10, « Créer des modules et des procédures ».

Les gestionnaires d'erreur gèrent, ou *piègent*, un problème grâce au bloc de code *Try...Catch* et un objet spécial de gestion des erreurs nommé *Err*. Ce dernier possède une propriété *Number* qui identifie le numéro de l'erreur et une propriété *Description* qui sert à afficher une description de l'erreur. Par exemple, si l'erreur d'exécution est associée au chargement d'un fichier depuis un lecteur de CD ou de DVD, votre gestionnaire d'erreur peut afficher un message d'erreur personnalisé qui identifie le problème et invite l'utilisateur à insérer un CD ou un DVD, plutôt que de permettre à l'opération qui a échoué d'interrompre le programme.

Quand utiliser les gestionnaires d'erreur ?

Vous pouvez utiliser des gestionnaires d'erreur dans toutes les situations où une action (attendue ou inattendue) est susceptible de générer une erreur mettant fin à l'exécution du programme. Habituellement, les gestionnaires d'erreur servent à gérer des événements extérieurs qui influencent un programme – par exemple, des événements engendrés par une connexion Internet ou réseau défaillante, un CD, DVD ou une disquette qui n'est pas correctement inséré dans le lecteur, ou une imprimante ou un scanneur déconnectés. Le tableau qui suit présente les problèmes potentiels que les gestionnaires d'erreur peuvent traiter.

Problème	Description
Problèmes réseau/Internet	Serveurs réseau, connexions Internet et autres ressources qui échouent, ou *tombent en panne*, de manière inattendue.
Problèmes de base de données	Impossibilité d'établir une connexion à la base de données, de traiter une requête (par exemple suite à un dépassement de délai), renvoi d'une erreur par la base de donnes, etc.
Problèmes de lecteur de disque	CD, DVD, disquette ou tout autre support mal ou non formatés, mal insérés, avec des secteurs endommagés, pleins, problèmes de lecteur de CD ou DVD, etc.
Problèmes de chemin d'accès	Chemin vers un fichier nécessaire, manquant ou incorrect.
Problèmes d'imprimante	Problèmes d'imprimante déconnectée, manque de papier, manque de mémoire ou indisponibilité.
Logiciel non installé	Fichier ou composant dont dépend votre application, qui n'est pas installé sur l'ordinateur de l'utilisateur ou incompatible avec le système d'exploitation.
Problèmes de sécurité	Une application ou un processus tente de modifier des fichiers du système d'exploitation, accéder de façon non autorisée à l'Internet ou de modifier d'autres programmes ou fichiers.
Problèmes de permissions	Permissions utilisateur inappropriées pour accomplir une tâche.
Erreurs de dépassement de capacité	Activité qui dépasse l'espace de stockage alloué.
Erreurs de manque de mémoire	Espace disponible insuffisant pour les applications ou les ressources dans le schéma de gestion de mémoire de Microsoft Windows.
Problèmes de Presse- papiers.	Problèmes de transfert des données ou avec le Presse-papiers Windows.
Erreurs de logique	Erreurs de syntaxe ou de logique non détectées par le compilateur et par les tests précédents (comme un nom de fichier mal orthographié).

Mettre en place un piège : le bloc de code Try...Catch

Le bloc de code permettant de gérer une erreur d'exécution s'appelle *Try...Catch*. Vous placez l'instruction *Try* dans une procédure événementielle juste avant l'instruction qui pose problème et l'instruction *Catch* est immédiatement suivie d'une liste d'instructions à exécuter en cas d'erreur d'exécution. Vous pouvez inclure plusieurs instructions supplémentaires, comme *Catch When*, *Finally* et *Exit Try*, et des blocs de code *Try...Catch* imbriqués, comme nous le verrons dans les exemples de ce chapitre. Toutefois, la syntaxe de base d'un gestionnaire d'exception *Try...Catch* se résume simplement à :

```
Try
    Instructions susceptibles de générer une erreur d'exécution
Catch
    Instructions à exécuter si une erreur d'exécution se produit
Finally
    Instructions optionnelles à exécuter si une erreur se produit ou non
End Try
```

L'instruction *Try* identifie le début d'un gestionnaire d'erreur dans lequel les mots clés *Try*, *Catch* et *End Try* sont nécessaires, et les instructions *Finally* et suivantes sont optionnelles. Notez que parfois, les programmeurs appellent les instructions situées entre les mots clés *Try* et *Catch,* du *code protégé* car les erreurs d'exécution qui résultent de ces instructions n'entraînent pas l'interruption du programme. À la place, Visual Basic exécute les instructions du gestionnaire d'erreur du bloc de code *Catch*.

Erreurs de chemin d'accès et de lecteur de disque

L'exemple qui suit illustre une erreur d'exécution classique – un problème avec un chemin d'accès, un lecteur de disque ou un périphérique. Pour mener à bien cet exercice, vous allez charger le projet Visual Basic que j'ai créé pour illustrer comment des fichiers image s'ouvrent dans un objet zone d'image sur un formulaire Windows.

Pour préparer cet exercice, insérez un CD ou DVD vierge dans le lecteur D (ou équivalent) et utilisez l'Explorateur Windows ou votre logiciel de création de CD/DVD pour y copier ou y *graver* le fichier ouvrirfichier.bmp. Sinon, vous pouvez copier ce fichier sur une disquette (lecteur A) ou tout autre type de support de stockage amovible, comme un appareil photo numérique, une carte mémoire, une clé USB ou un lecteur Zip Iomega.

Astuce Le fichier ouvrirfichier.bmp, ainsi que le projet Erreur Disque, se trouvent dans le dossier c:\vb08epe\chap09.

Pour réaliser cet exercice, vous devez être en mesure d'extraire le CD/DVD ou de connecter et déconnecter votre périphérique de stockage externe, comme les conditions de test le stipulent. Vous allez devoir modifier le code du programme ci-après en y insérant la lettre du lecteur que vous utilisez. Vous allez employer le CD/DVD et son lecteur ou un support équivalent (comme une clé USB) tout au long de ce chapitre pour forcer des erreurs d'exécution et effectuer des récupérations.

Expérimenter des erreurs de disque

1. Insérez un CD ou DVD vierge dans le lecteur D (ou le lecteur dans lequel vous créez vos CD/DVD) ou connectez un périphérique de stockage externe et copiez-y le fichier ouvrirfichier.bmp.

   Utilisez l'Explorateur Windows ou un autre programme de création de CD/DVD pour copier le fichier et graver le disque. Si vous utilisez un périphérique de stockage externe, copiez-y le fichier ouvrirfichier.bmp et notez la lettre de lecteur que Windows attribue à ce périphérique.

2. Démarrez Visual Studio, puis ouvrez le projet Erreur Disque qui se situe dans le dossier c:\vb08epe\chap09\Erreur Disque.

   Le projet Erreur Disque s'ouvre dans l'environnement de développement.

3. Affichez le formulaire si nécessaire.

   Le projet Erreur Disque est un programme squelette qui affiche le fichier ouvrirfichier.bmp dans une zone d'image lorsque l'utilisateur clique sur le bouton Vérifier lecteur. Ce projet est un moyen pratique de créer et de piéger des erreurs d'exécution. Utilisez-le tout au long de ce chapitre pour construire des gestionnaires d'erreur en utilisant le bloc de code *Try...Catch*.

4. Double-cliquez sur le bouton Vérifier lecteur du formulaire pour afficher la procédure événementielle *Button1_Click*.

   La ligne de code suivante s'affiche entre les instructions *Private Sub* et *End Sub*.

```
PictureBox1.Image = _
     System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
```

   Comme vous l'avez appris dans les chapitres précédents, la méthode *FromFile* ouvre le fichier spécifié. Cet usage particulier de *FromFile* ouvre le fichier ouvrirfichier.bmp sur le lecteur concerné (ici F) et l'affiche dans une zone d'image. Toutefois, si le CD/DVD est manquant, si le plateau du lecteur est ouvert, si le fichier ne se trouve pas sur le CD/DVD ou si autre problème concerne le chemin d'accès ou la lettre de lecteur spécifiés dans le code, l'instruction génère une erreur de type « Fichier introuvable » dans Visual Basic. Il s'agit de l'erreur d'exécution que nous souhaitons piéger.

> **Remarque** Si votre lecteur de CD/DVD ou autre périphérique utilise une lettre de lecteur autre que « D », modifiez-la dans cette instruction pour la remplacer par celle que vous utilisez. Par exemple, le lecteur de disquette exploite habituellement la lettre « A ». Les cartes mémoire, les appareils photo numériques et d'autres supports amovibles utilisent les lettres « E », « F » et suivantes.

5. Avec le CD/DVD dans le lecteur D (ou autre), cliquez sur le bouton Démarrer le débogage de la barre d'outils Standard pour exécuter le programme.

Le formulaire du projet s'affiche, comme suit :

6. Dans le formulaire, cliquez sur le bouton Vérifier lecteur.

Le programme charge le fichier ouvrirfichier.bmp à partir du CD/DVD et l'affiche dans une zone d'image, comme ci-après.

La propriété *SizeMode* de l'objet zone d'image est définie à *StretchImage*. Le fichier remplit donc la totalité de l'objet zone d'image. Voyons maintenant ce qui se produit si le CD/DVD ne se trouve pas dans le lecteur lorsque le programme tente de charger le fichier.

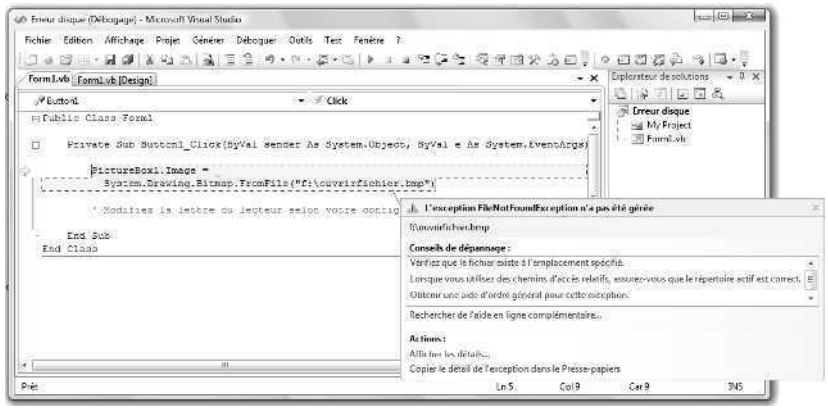
7. Retirez le CD/DVD du lecteur.

Si vous utilisez un type de support différent, retirez-le. Si vous réalisez ce test avec un périphérique de stockage amovible, suivez la procédure habituelle pour l'arrêter en toute sécurité et retirer le support contenant le fichier ouvrirfichier.bmp.

8. Dans le formulaire, cliquez de nouveau sur le bouton Vérifier lecteur.

Le programme ne trouve pas le fichier et Visual Basic émet une erreur d'exécution, ou une *exception non gérée*, qui entraîne la panne du programme. Visual Studio

entre en mode débogage, met en surbrillance l'instruction problématique et affiche la boîte de dialogue suivante :



Remarquez comme Visual Studio tente d'être utile, en proposant des astuces de dépannage pour vous aider à identifier la source de l'exception non gérée qui a arrêté le programme. La liste Actions permet d'en apprendre encore plus sur le message d'erreur spécifique affiché en haut de la boîte de dialogue.

9. Dans la barre d'outils Standard, cliquez sur le bouton Arrêter le débogage pour fermer le programme.

L'environnement de développement réapparaît.

Nous allons maintenant modifier le code pour gérer à l'avenir cet éventuel (et plausible) scénario d'erreur.

Développer un gestionnaire d'erreur pour le lecteur de disque

Le problème du programme Erreur Disque n'est pas qu'il défie les capacités inhérentes de traitement d'erreur de Visual Basic. Nous avons tout simplement omis de spécifier ce que Visual Basic doit faire lorsqu'il rencontre une exception qu'il ne sait pas gérer. Pour résoudre ce problème, il suffit de rédiger un bloc de code *Try...Catch* qui reconnaisse l'erreur et indique à Visual Basic l'action à entreprendre. Vous allez maintenant ajouter ce gestionnaire d'erreur.

Utiliser Try...Catch pour piéger l'erreur

1. Affichez la procédure événementielle *Button1_Click* si elle n'est pas visible dans l'Éditeur de code.

   Vous devez ajouter un gestionnaire d'erreur à la procédure événementielle à l'origine des problèmes. Comme vous allez le voir dans cet exemple, le bloc de code *Try...Catch* se construit autour du code susceptible d'être à la source du problème, protégeant ainsi le programme des erreurs d'exécution qu'il peut engendrer.

2. Modifiez la procédure événementielle de sorte que l'instruction *FromFile* existante se place entre les instructions *Try* et *Catch*, comme dans le bloc de code suivant :

```
Try
     PictureBox1.Image = _
         System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch
     MsgBox("Insérez le disque dans le lecteur ou connectez votre périphérique amovible !")
End Try
```

   Inutile de resaisir l'instruction *FromFile* – il suffit de taper les instructions *Try*, *Catch*, *MsgBox* et *End Try* au-dessus et en dessous. Si Visual Studio ajoute *Catch*, une déclaration de variable ou des instructions *End Try*, au mauvais endroit, supprimez simplement les instructions et retapez-les, comme nous le montrons dans ce livre. En effet, l'Éditeur de code tente de se rendre utile, mais sa fonctionnalité de saisie semi-automatique s'avère parfois gênante.

   Ce code illustre l'utilisation la plus fondamentale d'un bloc de code *Try...Catch*. Il place l'instruction problématique *FromFile* dans un bloc de code *Try* afin que, si le code génère une erreur, les instructions situées dans le bloc de code *Catch* s'exécutent. Le bloc de code *Catch* affiche simplement une boîte de message demandant à l'utilisateur d'insérer le disque ou de connecter le périphérique pour que le programme puisse se poursuivre. Ce bloc de code *Try...Catch* ne contient pas d'instruction *Finally*. Le gestionnaire d'erreur se termine donc par les mots clés *End Try*.

   Encore une fois, si vous utilisez un périphérique de stockage amovible ou un support associé à une lettre de lecteur différente, effectuez ces changements dans les instructions que vous venez de taper.

Test du gestionnaire d'erreur

1. Retirez le CD/DVD du lecteur (ou déconnectez le périphérique amovible) et cliquez sur le bouton Démarrer le débogage pour exécuter le programme.
2. Cliquez sur le bouton Vérifier lecteur.

Au lieu d'interrompre l'exécution du programme, Visual Basic invoque l'instruction *Catch*, qui affiche la boîte de message suivante :

3. Cliquez sur OK, puis cliquez à nouveau sur le bouton Vérifier le lecteur.

   Le programme affiche encore la boîte de message vous demandant d'insérer correctement le disque dans le lecteur D. Chaque fois que vous rencontrerez un problème de chargement du fichier, cette boîte de message s'affichera.

4. Insérez le disque dans le lecteur ou connectez le périphérique amovible, patientez quelques instants pour que le système le reconnaisse(fermez toutes les fenêtres qui s'affichent lorsque vous insérez le disque ou connectez le périphérique), cliquez sur OK, puis cliquez à nouveau sur le bouton Vérifier lecteur.

   L'image apparaît dans la zone appropriée, comme voulu. Le gestionnaire d'erreur a rempli efficacement sa mission. Plutôt que de s'arrêter sans crier gare, le programme sait désormais comment corriger cette erreur : vous pouvez continuer à travailler avec l'application.

5. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Voyons à présent quelques variantes du gestionnaire d'erreur *Try...Catch*.

Utiliser la clause *Finally* pour accomplir des tâches de nettoyage

Comme nous l'avons vu avec la description de la syntaxe *Try...Catch*, précédemment dans ce chapitre, vous pouvez utiliser la clause optionnelle *Finally* avec *Try...Catch* pour exécuter un bloc d'instructions, indépendamment de la manière dont le compilateur exécute les blocs *Try* ou *Catch*. En d'autres termes, que les instructions *Try* aient généré ou non une erreur d'exécution, vous pourriez disposer d'un code à exécuter chaque fois qu'un gestionnaire d'erreur a terminé sa tâche. Par exemple, vous pouvez avoir besoin de mettre à jour des variables ou des propriétés, d'afficher les résultats d'un calcul, de fermer une connexion à une base de données ou d'accomplir des opérations de « nettoyage » en effaçant des variables ou en désactivant des objets inutiles sur un formulaire.

L'exercice suivant illustre le fonctionnement de la clause *Finally* en affichant une deuxième boîte de message, que la méthode *FromFile* génère ou non une erreur d'exécution.

Utiliser *Finally* pour afficher une boîte de message

1. Affichez la procédure événementielle *Button1_Click*, puis modifiez le bloc de code *Try...Catch* pour qu'il contienne deux lignes de code supplémentaires au-dessus de l'instruction *End Try*. Le gestionnaire d'erreur complet présente un résultat similaire à :

```
Try
     PictureBox1.Image = _
         System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch
     MsgBox("Insérez le disque dans le lecteur ou connectez le périphérique amovible !")
Finally
     MsgBox("Gestionnaire d'erreur terminé")
End Try
```

   L'instruction *Finally* indique au compilateur qu'un bloc de code final doit être exécuté, que l'on ait traité ou non une erreur d'exécution. Pour vous aider à comprendre dans le détail comment cela fonctionne, j'ai inséré une fonction *MsgBox* pour afficher un message test, après l'instruction *Finally*. Bien que cette simple utilisation de l'instruction *Finally* soit utile à des fins de test, dans un programme réel, vous utiliserez sans doute le bloc de code *Finally* pour mettre à jour des variables ou des propriétés importantes, afficher des données ou accomplir d'autres opérations de nettoyage.

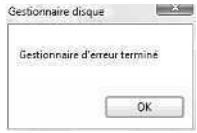
2. Retirez le CD du lecteur ou déconnectez le périphérique amovible, puis cliquez sur le bouton Démarrer le débogage pour exécuter le programme.

3. Cliquez sur le bouton Vérifier lecteur.

   Le gestionnaire d'erreur affiche une boîte de dialogue vous demandant d'insérer le disque dans le lecteur ou de connecter le périphérique.

4. Cliquez sur OK.

   Le programme exécute la clause *Finally* dans le gestionnaire d'erreur et la boîte de message suivante s'affiche :



5. Cliquez sur OK, insérez le disque dans le lecteur ou connectez le périphérique, puis cliquez de nouveau sur le bouton Vérifier lecteur.

   Le fichier apparaît dans la zone d'image, comme voulu. En outre, la clause *Finally* s'exécute et la boîte de message « Gestionnaire d'erreur terminé » s'affiche à nouveau. Comme je l'ai dit précédemment, erreur ou non, les instructions *Finally* s'exécutent à la fin d'un bloc *Try...Catch*.

6. Cliquez sur OK, puis cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Gestionnaires d'erreur Try...Catch plus complexes

Au fur et à mesure que vos programmes se complexifient, vous trouverez utile de rédiger des gestionnaires d'erreur *Try...Catch* plus élaborés pour gérer plusieurs erreurs d'exécution ainsi que des situations de gestion d'erreur inhabituelles. *Try...Catch* autorise une telle complexité en :

- autorisant plusieurs lignes de code dans chaque bloc de code Try, Catch ou Finally ;
- proposant la syntaxe *Catch When*, qui teste des conditions d'erreur spécifiques ;
- autorisant des blocs de code Try...Catch imbriqués, que lon peut utiliser pour construire des gestionnaires derreur sophistiqués et robustes.

En outre, grâce à un objet gestionnaire d'erreur spécial appelé *Err*, vous pouvez identifier et traiter dans votre programme des erreurs d'exécution et des conditions spécifiques. Vous allez étudier chacune de ces fonctionnalités de gestion des erreurs dans la prochaine section.

L'objet *Err*

Héritage des versions antérieures de Visual Basic, un mécanisme précieux de Visual Basic 2008 nommé l'objet *Err* est actualisé avec des informations détaillées de gestion des erreurs chaque fois qu'une erreur d'exécution se produit. Même s'il existe de nouvelles façons de gérer les erreurs à l'aide du Microsoft .NET Framework, comme le très puissant objet *Exception*, nous allons commencer notre travail avec des messages de gestion d'erreur en examinant comment l'objet *Err* propose des informations sur le type d'erreur qui s'est produit dans un programme.

Les propriétés *Err* les plus utiles pour identifier des erreurs d'exécution sont *Err.Number* et *Err.Description*. *Err.Number* contient le numéro de l'erreur d'exécution la plus récente et *Err.Description* contient un court message d'erreur correspondant au numéro de l'erreur d'exécution. En associant ces deux propriétés dans un gestionnaire d'erreur, vous pouvez reconnaître des erreurs spécifiques et y répondre, ainsi que donner à l'utilisateur des informations utiles sur la manière de les résoudre.

Il est possible d'effacer le contenu de l'objet *Err* grâce à la méthode *Err.Clear*, qui supprime les anciennes informations d'erreur. Toutefois, si vous utilisez l'objet *Err* dans un bloc de code *Catch*, il n'est pas nécessaire d'effacer l'objet *Err*, car les blocs *Catch* ne sont saisis que si une erreur d'exécution s'est produite dans le voisinage du bloc de code *Try*.

Le tableau qui suit présente la plupart des erreurs d'exécution que les applications Visual Basic sont susceptibles de rencontrer. Outre ces codes d'erreur, vous verrez que certaines bibliothèques Visual Basic et d'autres composants (comme les bases de données et les composants système) proposent leurs propres messages d'erreur que vous découvrirez en utilisant la documentation de Visual Studio. Notez qu'en dépit des descriptions du message d'erreur, certaines erreurs n'apparaissent pas comme vous vous y attendez. Vous

devez donc tester de manière spécifique les numéros d'erreur (si possible) en observant la manière dont la propriété *Err.Number* change pendant l'exécution du programme. Les numéros d'erreur non utilisés – dans la plage allant de 1 à 1000 – sont réservés par Visual Basic pour un usage ultérieur.

Numéro de l'erreur	Message d'erreur par défaut
5	Argument ou appel de procédure non valide
6	Dépassement de capacité
7	Mémoire insuffisante
9	Indice hors limite
11	Division par zéro
13	Incompatibilité de type
48	Erreur de chargement de la DLL
51	Erreur interne
52	Nom ou numéro de fichier incorrect
53	Le fichier <nomfichier> est introuvable
55	Le fichier est déjà ouvert
57	Erreur d'E/S de périphérique
58	Fichier déjà existant
61	Disque plein
62	L'entrée dépasse la fin du fichier
67	Trop de fichiers
68	Périphérique non disponible
70	Autorisation refusée
71	Disque non prêt
74	Impossible de renommer avec un lecteur différent
75	Erreur dans le chemin d'accès
76	Chemin d'accès introuvable
91	Variable objet ou variable bloc With non définie
321	Le format de fichier n'est pas valide
322	Impossible de créer le fichier temporaire nécessaire
380	Valeur de propriété invalide
381	L'index de tableau de propriétés est non valide
422	Propriété introuvable
423	Propriété ou méthode introuvable
424	Objet requis
429	Impossible de créer le composant ActiveX
430	La classe ne prend pas en charge Automation ou l'interface attendue
438	L'objet ne prend pas en charge cette propriété ou méthode
440	Erreur Automation
460	Format de Presse-papiers invalide
461	Méthode ou données membres introuvable

Numéro de l'erreur	Message d'erreur par défaut
462	Le serveur distant n'existe pas ou n'est pas disponible
463	La classe n'est pas inscrite sur l'ordinateur local
481	Caractère non valide
482	Erreur de l'imprimante

L'exercice suivant utilise les propriétés *Err.Number* et *Err.Description* dans un gestionnaire d'erreur *Try...Catch* pour tester plusieurs conditions d'erreur d'exécution. Cette capacité est rendue possible grâce à la syntaxe *Catch When* que vous allez utiliser pour tester des conditions d'erreur spécifiques dans un bloc de code *Try...Catch*.

Tester plusieurs conditions d'erreur d'exécution

1. Dans la procédure événementielle *Button1_Click*, modifiez le gestionnaire d'erreur *Try...Catch* de sorte qu'il ressemble au bloc de code suivant. (L'instruction *FromFile* d'origine est la même que le code utilisé dans les exercices précédents, mais les instructions *Catch* sont entièrement nouvelles).

```
Try
     PictureBox1.Image = _
         System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch When Err.Number = 53 'si erreur de fichier introuvable
     MsgBox("Vérifiez le nom du chemin d'accès et le lecteur de disque")
Catch When Err.Number = 7 'si erreur de manque de mémoire
     MsgBox("Est-ce réellement un bitmap ?", , Err.Description)
Catch
     MsgBox("Problème de chargement du fichier", , Err.Description)
End Try
```

   La syntaxe *Catch When* est utilisée deux fois dans le gestionnaire d'erreur et chaque fois, cette syntaxe est associée à la propriété *Err.Number* pour tester si le bloc de code *Try* a généré un type d'erreur d'exécution particulier. Si la propriété *Err.Number* correspond au numéro 53, l'erreur d'exécution Fichier introuvable s'est produite pendant la procédure d'ouverture du fichier. Le message « Vérifiez le nom du chemin d'accès et le lecteur de disque » s'affiche dans une boîte de message. Si la propriété *Err.Number* correspond au numéro 7, une erreur Mémoire insuffisante s'est produite – certainement en raison du chargement d'un fichier ne contenant pas d'image. (J'obtiens cette erreur si j'essaie involontairement d'ouvrir un document Microsoft Word dans un objet zone d'image en utilisant la méthode *FromFile*).

   L'instruction *Catch* finale gère toutes les erreurs d'exécution susceptibles de se produire pendant un processus d'ouverture de fichier – il s'agit d'un bloc de code générique « capturer tout » qui affiche un message d'erreur général dans une boîte de message et un message d'erreur spécifique découlant de la propriété *Err.Description* dans la barre de titre de la boîte de message.

2. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.
3. Retirez le CD/DVD du lecteur ou déconnectez le périphérique amovible.

4. Cliquez sur le bouton Vérifier lecteur.

   Le gestionnaire d'erreur affiche le message d'erreur « Vérifiez le nom du chemin d'accès et le lecteur de disque » dans une boîte de message. La première instruction *Check When* fonctionne.

5. Cliquez sur OK, puis cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

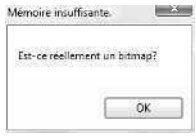
6. Insérez de nouveau le CD/DVD ou connectez le périphérique, puis servez-vous de l'Explorateur Windows ou autre outil pour copier un deuxième fichier qui ne soit pas un fichier image. Par exemple, copiez un document Word ou une feuille de calcul Microsoft Excel.

   Vous n'allez pas ouvrir ce fichier dans Word ou Excel mais vous allez essayer de l'ouvrir (sans succès, je l'espère) dans votre objet zone d'image du programme. Si votre logiciel ou votre lecteur de CD/DVD ne vous permet pas d'ajouter d'autres fichiers sur le CD/DVD après sa gravure, créez un deuxième CD/DVD contenant ces deux fichiers.

7. Dans l'Éditeur de code, remplacez le nom du fichier ouvrirfichier.bmp dans l'instruction *FromFile* par le nom du fichier (Word, Excel, ou autre) que vous avez copié sur le CD/DVD ou le support amovible.

   En utilisant un fichier de format différent, vous avez la possibilité de tester un deuxième type d'erreur d'exécution – une exception de type Manque de mémoire, qui se produit lorsque Visual Basic tente de charger un fichier qui n'est pas un fichier graphique ou qui contient trop d'informations pour une zone d'image.

8. Relancez l'exécution du programme, puis cliquez sur le bouton Vérifier lecteur. Le gestionnaire d'erreur affiche le message d'erreur suivant.

   Remarquez que j'ai utilisé la propriété *Err.Description* pour afficher une brève description du problème (« Mémoire insuffisante ») dans la barre de titre de la boîte de message. En exploitant cette propriété dans votre gestionnaire d'erreur, l'utilisateur a une idée plus claire de ce qui vient de se produire.

9. Cliquez sur OK, puis cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

10. Renommez à nouveau le fichier ouvrirfichier.bmp dans la méthode *FromFile*. Vous allez vous en servir dans le prochain exercice.

L'instruction *Catch When* est très puissante. En l'associant aux propriétés *Err.Number* et *Err.Description*, vous pouvez développer des gestionnaires d'erreur sophistiqués capables de reconnaître et de répondre à plusieurs types d'exceptions.

Déclencher vos propres erreurs

Pour des questions de test ou d'autres utilisations spécialisées, vous pouvez générer artificiellement vos propres erreurs d'exécution dans un programme grâce à une technique appelée *déclencher* ou *lever* des exceptions. Pour ce faire, utilisez la méthode *Err.Raise* avec un des numéros d'erreur du tableau présenté précédemment. Par exemple, la syntaxe suivante exploite la méthode *Raise* pour générer une erreur d'exécution Disque plein, puis gère l'erreur grâce à une instruction *Catch When* :

```
Try
    Err.Raise(61) 'déclenche une erreur Disque plein
Catch When Err.Number = 61
    MsgBox("Erreur: Disque plein")
End Try
```

Lorsque vous saurez rédiger vos propres procédures, vous pourrez générer, grâce à cette technique, vos propres erreurs et les retourner à la routine appelante.

Spécifier la fréquence des tentatives

Vous pouvez exploiter dans un gestionnaire d'erreur une autre stratégie qui consiste à effectuer plusieurs tentatives d'exécution d'une opération, puis à la désactiver si le problème n'est pas résolu. Par exemple, dans l'exercice suivant, un bloc *Try...Catch* emploie une variable compteur appelée *Tentatives* pour suivre le nombre d'affichage du message « Insérez le disque dans le lecteur ou connectez le périphérique amovible ». À l'issue de la deuxième tentative, le gestionnaire d'erreur désactive le bouton Vérifier lecteur. Dans cette technique, l'astuce consiste à déclarer la variable *Tentatives* en haut du code afin qu'elle s'étende à l'ensemble des procédures événementielles du formulaire. La variable *Tentatives* est ensuite incrémentée et testée dans le bloc de code *Catch*. Vous pouvez modifier le nombre de tentatives en remplaçant simplement « 2 » dans l'instruction, comme suit :

```
If Tentatives <= 2
```

Utiliser une variable pour suivre les erreurs d'exécution

1. Dans l'Éditeur de code, remontez vers le haut du code et tapez la déclaration de variable suivante, juste en dessous de l'instruction `Public Class Form1` :

   ```
   Dim Tentatives As Short = 0
   ```

   La variable *Tentatives* est déclarée comme variable entière *Short* car elle ne contiendra pas de nombres élevés. On lui attribue une valeur initiale de 0 afin qu'elle soit correctement redéfinie à chaque exécution du programme.

2. Dans la procédure événementielle *Button1_Click*, modifiez le gestionnaire d'erreur *Try...Catch* de sorte qu'il se présente comme dans le bloc de code suivant :

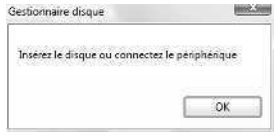
```
Try
     PictureBox1.Image = _
         System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch
     Tentatives += 1
     If Tentatives <= 2 Then
         MsgBox("Insérez le disque ou connectez le périphérique !")
     Else
         MsgBox("Fonctionnalité de chargement de fichier désactivée")
         Button1.Enabled = False
     End If
End Try
```

Le bloc *Try* teste la même procédure d'ouverture de fichier mais cette fois, si une erreur se produit, le bloc *Catch* incrémente la variable *Tentatives* et la teste pour s'assurer qu'elle est bien inférieure ou égale à 2. Il est possible de modifier ce chiffre pour autoriser autant de tentatives que voulu – il n'est ici autorisé que deux erreurs d'exécution. Après deux erreurs, la clause *Else* s'exécute et une boîte de message s'affiche, indiquant que la fonctionnalité de chargement de fichier a été désactivée. Le bouton Vérifier lecteur est alors désactivé – en d'autres termes, il est grisé et rendu inutilisable pour la suite du programme.

Astuce La version révisée du gestionnaire d'erreur que vous avez développé a été renommée Gestionnaire Disque et se trouve dans le dossier c:\vb08epe\chap09\Gestionnaire Disque.

3. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.
4. Retirez le CD/DVD du lecteur ou déconnectez le périphérique amovible.
5. Cliquez sur le bouton Vérifier lecteur.

Le gestionnaire d'erreur affiche le message d'erreur « Insérez le disque ou connectez le périphérique ! » dans une boîte de message, comme ci-après. À l'arrière-plan, la variable *Tentatives* est également incrémentée à 1.

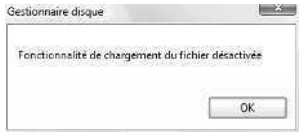


6. Cliquez sur OK, puis cliquez de nouveau sur le bouton Vérifier lecteur.

   La variable *Tentatives* est définie à 2 et le message « Insérez le disque ou connectez le périphérique ! » s'affiche à nouveau.

7. Cliquez sur OK, puis cliquez une troisième fois sur le bouton Vérifier lecteur.

   La variable *Tentatives* est incrémentée à 3 et la clause *Else* est exécutée. Le message « Fonctionnalité de chargement de fichier désactivée » s'affiche, comme ci-après :



8. Cliquez sur OK dans la boîte de message.

   Le bouton Vérifier lecteur est désactivé, comme suit :

   Le gestionnaire d'erreur a répondu au problème de lecteur de disque en autorisant l'utilisateur à effectuer plusieurs tentatives pour corriger le problème, puis il a désactivé le bouton problématique. En d'autres termes, l'utilisateur ne peut plus cliquer sur ce bouton. Cette action de désactivation met fin aux futures erreurs d'exécution, même si le programme ne peut plus fonctionner exactement comme il a été conçu à l'origine.

9. Cliquez sur le bouton Fermer pour arrêter le programme.

Utiliser des blocs Try...Catch imbriqués

Vous pouvez également utiliser des blocs de code *Try...Catch* dans vos gestionnaires d'erreur. Par exemple, le gestionnaire d'erreur de disque suivant exploite un deuxième bloc *Try...Catch* pour effectuer une seule nouvelle tentative d'ouverture de fichier si la première échoue et génère une erreur d'exécution :

```
Try
    PictureBox1.Image = _
        System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch
    MsgBox("Insérez le disque dans le lecteur ou connectez le périphérique, puis cliquez sur OK !")
    Try
        PictureBox1.Image = _
            System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
    Catch
        MsgBox("Fonctionnalité de chargement du fichier désactivée")
        Button1.Enabled = False
    End Try
End Try
```

Si l'utilisateur insère le disque dans le lecteur ou connecte le périphérique à l'invite du message, le deuxième bloc *Try* ouvre le fichier sans erreur. Toutefois, si une erreur d'exécution liée au fichier persiste, le deuxième bloc *Catch* affiche un message indiquant que la fonctionnalité de chargement du fichier est désactivée et le bouton est désactivé.

En général, les gestionnaires d'erreur *Try...Catch* fonctionnent tant que le nombre de tests ou de tentatives est restreint. Si vous devez tenter à plusieurs reprises une opération problématique, utilisez une variable pour suivre les tentatives, ou développez une fonction contenant un gestionnaire d'erreur que vous pourrez appeler de manière répétée depuis vos procédures événementielles. (Pour plus d'informations sur la création de fonction, reportez-vous au chapitre 10, « Créer des modules et des procédures »).

Comparaison des gestionnaires d'erreur à des techniques de programmation défensive

Pour un programme, les gestionnaires d'erreur ne représentent pas les seuls mécanismes de protection contre les erreurs d'exécution. Par exemple, le code qui suit exploite la méthode *File.Exists* dans l'espace de noms *System.IO* de la bibliothèque de classes du .NET Framework pour vérifier l'existence d'un fichier sur un CD ou DVD avant de l'ouvrir :

```
If File.Exists("f:\ouvrirfichier.bmp") Then
    PictureBox1.Image = _
        System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Else
    MsgBox("Impossible de trouver ouvrirfichier.bmp sur le lecteur F.")
End If
```

Cette instruction *If…Then* n'est vraiment pas un gestionnaire d'erreur car elle n'empêche pas une erreur d'exécution d'interrompre un programme. Il s'agit plutôt d'une technique de validation appelée parfois *programmation défensive*. Elle exploite une méthode pratique de la bibliothèque de classes du .NET Framework pour vérifier l'opération de fichier voulue avant qu'elle soit réellement tentée dans le code. Dans ce cas précis, le fait de tester si le fichier existe grâce à la méthode du .NET Framework s'avère en réalité plus rapide que d'attendre que Visual Basic émette une exception et récupère d'une erreur d'exécution avec un gestionnaire d'erreur.

Remarque Pour que cette logique de programme particulière fonctionne, vous devez inclure l'instruction qui suit dans la section des déclarations en haut du code pour y faire référence à la bibliothèque de classes du .NET Framework invoquée :

```
Imports System.IO
```

Pour plus d'informations sur l'utilisation de l'instruction *Imports* permettant d'utiliser des objets, des propriétés et des méthodes des bibliothèques de classes du .NET Framework, reportez-vous au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework ».

Quand utiliser des techniques de programmation défensives et quand utiliser des gestionnaires d'erreur structurés ? La réponse est que le mieux consiste à combiner dans votre code les techniques de programmation défensive et de gestion d'erreur structurée. La logique de la programmation défensive représente d'ordinaire le moyen le plus efficace de gérer les problèmes potentiels. Comme nous l'avons vu précédemment avec le bloc de code *If…Then*, la méthode *File.Exists* est plus rapide qu'un gestionnaire d'erreur *Try…Catch*. Il est donc judicieux d'exploiter une technique de programmation défensive en cas de problèmes de performance. Servez-vous de la programmation défensive pour les erreurs susceptibles de se produire fréquemment dans votre programme. Réservez les gestionnaires d'erreurs structurés pour les erreurs qui ne devraient survenir que rarement. Les gestionnaires d'erreur sont également indispensables si l'on doit tester plusieurs conditions et proposer à l'utilisateur de nombreuses options pour répondre à l'erreur. Ils permettent en outre de gérer en douceur des erreurs auxquelles vous n'avez même pas pensé !

Aller plus loin : L'instruction *Exit Try*

Dans ce chapitre, vous avez beaucoup appris sur les gestionnaires d'erreur. Vous voilà prêt à les mettre en application dans vos propres programmes. Mais avant de passer au chapitre suivant, voici une autre option de syntaxe utile dans les blocs de code *Try...Catch* : l'instruction *Exit Try*. Il s'agit d'une technique rapide et un peu abrupte pour quitter prématurément un bloc de code *Try...Catch*. Si vous avez déjà écrit des programmes Visual Basic, vous remarquerez leur similitude avec les instructions *Exit For* et *Exit Sub*, qui servent à quitter de manière précoce une routine structurée. La syntaxe *Exit Try* vous permet de passer complètement en dehors du bloc de code *Try* ou *Catch* en cours. S'il existe un bloc de code *Finally*, il sera exécuté, mais *Exit Try* vous permet de sauter toutes les autres instructions *Try* ou *Catch* que vous ne souhaitez pas exécuter.

L'exemple de routine suivante illustre le fonctionnement de l'instruction *Exit Try*. Elle vérifie d'abord si la propriété *Enabled* de l'objet *PictureBox1* est définie à False. Ce drapeau peut indiquer que la zone d'image n'est pas prête à recevoir d'entrées. Si la zone d'image n'est pas encore activée, l'instruction *Exit Try* passe directement à la fin du bloc de code *Catch* et l'opération de chargement du fichier n'est pas tentée.

```
Try
    If PictureBox1.Enabled = False Then Exit Try
    PictureBox1.Image = _
        System.Drawing.Bitmap.FromFile("f:\ouvrirfichier.bmp")
Catch
    Tentatives += 1
    If Tentatives <= 2 Then
        MsgBox("Insérez le disque dans le lecteur ou connectez le périphérique !")
    Else
        MsgBox("Fonctionnalité de chargement du fichier désactivée")
        Button1.Enabled = False
    End If
End Try
```

Cet exemple s'appuie sur le dernier gestionnaire d'erreur que vous avez testé dans ce chapitre (le projet Gestionnaire Disque). Pour tester l'instruction *Exit Try* dans ce programme, chargez de nouveau le projet Gestionnaire Disque et saisissez l'instruction *If* contenant *Exit Try* dans l'Éditeur de code. Pour désactiver l'objet zone d'image, vous devrez également faire appel à la fenêtre Propriétés (en d'autres termes, définir sa propriété *Enabled* à False).

Félicitations ! Vous avez appris plusieurs techniques importantes de programmation fondamentales dans Visual Basic, dont la rédaction de gestionnaires d'erreur. Vous voici prêt à accroître votre efficacité en matière de programmation en apprenant à écrire des modules et des procédures Visual Basic.

Rappel du chapitre 9

Pour	Faites ceci

Détecter et traiter des erreurs d'exécution	Développez un gestionnaire d'erreur en utilisant un ou plusieurs blocs de code *Try...Catch*. Par exemple, le code du gestionnaire d'erreur suivant teste des problèmes de chemin d'accès ou de lecteur de disque :
	`Try`
	`  PictureBox1.Image = _`
	`  System.Drawing.Bitmap.FromFile _ ("f:\ouvrirfichier.bmp")`
	`Catch`
	`  MsgBox("Vérifiez le chemin d'accès ou insérez un disque")`
	`Finally`
	`  MsgBox("Gestionnaire d'erreur terminé")`
	`End Try`

Tester des conditions d'erreur spécifiques dans un gestionnaire d'événement	Utilisez la syntaxe *Catch When* et la propriété *Err.Number*. Par exemple :
	`Try`
	`  PictureBox1.Image = _`
	`  System.Drawing.Bitmap.FromFile _ ("f:\ouvrirfichier.bmp")`
	`Catch When Err.Number = 53 'si fichier introuvable`
	`  MsgBox("Vérifiez le chemin d'accès et le lecteur de disque")`
	`Catch When Err.Number = 7 'si mémoire insuffisante`
	`  MsgBox("Est-ce vraiment un bitmap ?", , _ Err.Description)`
	`Catch`
	`  MsgBox("Problème de chargement de fichier", , _ Err.Description)`
	`End Try`

Créer vos propres erreurs dans un programme	Utilisez la méthode *Err.Raise*. Par exemple, le code qui suit génère une erreur de type Disque plein et le gère :
	`Try`
	`  Err.Raise(61) 'déclenche une erreur Disque plein`
	`Catch When Err.Number = 61`
	`  MsgBox("Erreur: Disque plein")`
	`End Try`

Écrire des gestionnaires d'erreur *Try...Catch* imbriqués	Placez un bloc de code *Try...Catch* dans l'autre. Par exemple :
	`Try`
	`  PictureBox1.Image = _`
	`  System.Drawing.Bitmap.FromFile _ ("f:\ouvrirfichier.bmp")`
	`Catch`
	`  MsgBox("Insérer le disque dans le lecteur ou connectez le périphérique !")`
	`  Try`
	`  PictureBox1.Image = _`
	`  System.Drawing.Bitmap.FromFile _ ("f:\ouvrirfichier.bmp")`
	`  Catch`
	`  MsgBox("Fonctionnalité de chargement de fichier désactivée")`
	`  Button1.Enabled = False`
	`  End Try`
	`End Try`

Pour	Faites ceci
Quitter le bloc de code *Try* ou *Catch* en cours	Utilisez l'instruction *Exit Try* dans le bloc de code *Try* ou *Catch*. Par exemple : `If PictureBox1.Enabled = False Then Exit Try`

Chapitre 10

Créer des modules et des procédures

À la fin de ce chapitre, vous saurez :

- Employer des techniques de programmation structurées et créer des modules contenant des variables publiques et des définitions de procédure
- Vous entraîner à utiliser des variables publiques de portée globale
- Augmenter l'efficacité de la programmation en créant des fonctions définies par l'utilisateur et des procédures Sub
- Maîtriser la syntaxe destinée à invoquer et à utiliser des procédures définies par l'utilisateur
- Passer des arguments à des procédures par valeur et par référence

Dans les neuf premiers chapitres de ce livre, vous avez utilisé des procédures événementielles comme *Button1_Click*, *Timer1_Tick* et *Form1_Load* pour gérer des événements et organiser le flot de vos programmes. En programmation Visual Basic, toutes les instructions exécutables doivent être placées dans une procédure ; seules les déclarations et les instructions générales destinées au compilateur peuvent être placées en dehors de la portée d'une procédure. Dans ce chapitre, vous allez continuer à organiser vos programmes d'après des *techniques de programmation structurées* en développant une structure hiérarchique au sein de votre application et en scindant les tâches de calcul en unités logiques distinctes.

Vous allez commencer par apprendre à créer des *modules*. Il s'agit de zones distinctes d'un programme qui contiennent des variables globales ou *publiques* et des procédures *Function* et *Sub*. Vous allez apprendre à déclarer et à utiliser des variables publiques ainsi qu'à développer des procédures à vocation générale qui vous feront gagner du temps et que vous pourrez réutiliser dans d'autres projets. Ces compétences sont particulièrement utiles dans le cadre de projets de programmation de plus grande envergure et d'efforts de développement en équipe.

Travailler avec les modules

Au fur et à mesure que vos programmes s'allongeront, plusieurs de vos formulaires et procédures événementielles utiliseront sans doute des variables et des routines similaires. Par défaut, les variables sont *locales* à une procédure événementielle – elles ne peuvent être lues ou modifiées qu'au sein de la procédure événementielle dans laquelle elles ont été créées. On peut également déclarer des variables au début du code du formulaire et leur accorder une portée plus importante. Toutefois, si vous créez plusieurs formulaires dans un projet, les variables déclarées en en-tête ne sont valides que dans le formulaire dans lequel elles sont déclarées. De même, les procédures événementielles sont par défaut déclarées comme privées et ne sont locales que pour le formulaire dans lequel elles ont été créées. Par exemple, vous ne pouvez pas appeler la procédure événementielle *Button1_Click* depuis un deuxième formulaire appelé Form2 si la procédure événementielle est déclarée comme privée dans Form1. (Vous apprendrez comment ajouter d'autres formulaires à votre projet au chapitre 14, « Gérer les formulaires et les contrôles Windows à l'exécution »).

Pour partager des variables et des procédures entre tous les formulaires et les procédures événementielles d'un projet, il suffit de les déclarer dans un ou plusieurs modules inclus dans le projet. Un module est un fichier spécial qui porte l'extension .vb et qui contient des déclarations de variables et des procédures que l'on peut utiliser partout dans le programme. Dans Visual Basic 6, les modules standards portent l'extension .bas.

À l'instar des formulaires, les modules apparaissent séparément dans l'Explorateur de solutions. Contrairement aux formulaires, les modules ne contiennent que du code et ne possèdent pas d'interface utilisateur. Bien que les modules présentent quelques ressemblances avec les classes (anciennement appelées modules de classes), il ne s'agit pas de classes car les modules ne sont pas orientés objet, ne définissent pas la structure et les caractéristiques des objets et ne peuvent pas être hérités. Vous en saurez plus sur la création des classes au chapitre 16, « Gérer l'héritage de formulaire et créer des classes de base ».

Créer un module

Pour créer un nouveau module dans un programme, cliquez sur le bouton Ajouter un nouvel élément de la barre d'outils Standard ou cliquez sur la commande Ajouter un nouvel élément dans le menu Projet. Vous pouvez également cliquer sur la commande Ajouter un module dans le menu Projet. Une boîte de dialogue vous permet alors de sélectionner le modèle Module et de spécifier son nom. Un nouveau module vierge s'affiche ensuite dans l'Éditeur de code. Par défaut, le premier module d'un programme s'appelle Module1.vb. Vous pouvez toutefois en modifier le nom en effectuant un clic droit sur le module dans l'Explorateur de solutions, en sélectionnant Renommer et en tapant son nouveau nom. Vous pouvez également renommer un module en modifiant la propriété File Name dans la fenêtre Propriété. Vous allez maintenant créer un module vide dans un projet.

Créer et enregistrer un module

1. Démarrez Visual Studio 2008 et créez un nouveau projet Visual Basic Application Windows Forms intitulé Module test.

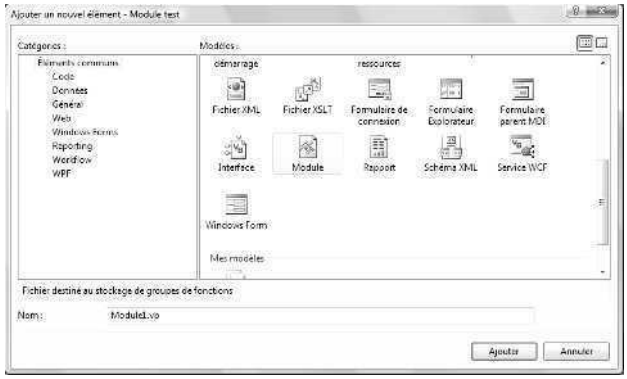
   Visual Basic crée le nouveau projet et un formulaire vierge s'affiche dans la fenêtre conception.

2. Dans le menu Projet, cliquez sur la commande Ajouter un nouvel élément.

   La boîte de dialogue Ajouter un nouvel élément s'affiche.

3. Sélectionnez le modèle Module.

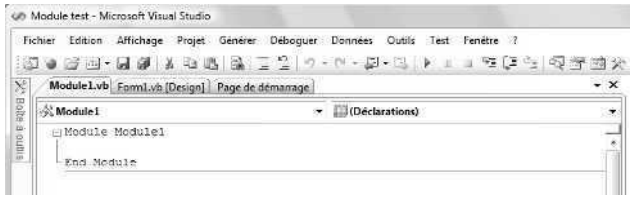
   Le nom par défaut, Module1.vb, s'affiche dans la zone de texte Nom.



Astuce La boîte de dialogue Ajouter un nouvel élément propose plusieurs modèles à utiliser dans vos projets. Chaque modèle présente des caractéristiques différentes et contient le code de démarrage vous permettant de les utiliser. Visual Studio 2008 contient de nouveaux modèles actualisés de formulaires Microsoft Windows, dont Formulaire Explorateur, Écran de démarrage et un formulaire d'ouverture de session, ainsi que de nombreux modèles de classes. Vous utiliserez ces modèles après avoir lu l'introduction à la programmation orientée objet au chapitre 16.

4. Cliquez sur le bouton Ajouter.

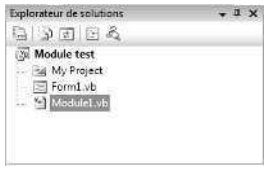
   Visual Basic ajoute Module1 à votre projet. Le module s'affiche dans l'Éditeur de code, comme suit :



La zone de liste Nom de la méthode indique que la section des déclarations générales du module est ouverte. Les variables et les procédures déclarées dans cette section sont disponibles pour l'ensemble du projet. Vous essaierez ultérieurement de déclarer des variables et des procédures.

5. Double-cliquez sur la barre de titre de l'Explorateur de solutions pour afficher la totalité de sa fenêtre, puis sélectionnez Module1.vb s'il n'est pas déjà sélectionné.

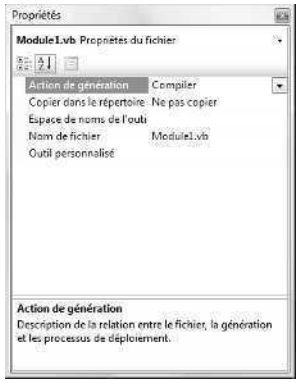
   L'Explorateur de solutions s'affiche, comme suit :



   L'Explorateur de solutions répertorie le module que vous avez ajouté au programme dans la liste des composants de votre projet. Le nom Module1 est le nom du fichier par défaut du module. Dans les prochaines étapes, vous allez modifier ces noms de fichier.

6. Double-cliquez sur la barre de titre de la fenêtre Propriétés pour afficher la fenêtre complète.

   La fenêtre Propriétés affiche les propriétés de Module1.vb, comme suit :



   Comme un modèle contient uniquement du code, il ne possède que quelques propriétés. En utilisant la propriété la plus importante, *File Name*, vous pouvez attribuer un nom de fichier personnalisé qui décrit le but du modèle. Réfléchissez à cette étiquette d'identification car vous aurez à incorporer plus tard votre module à une autre solution. Les autres propriétés du module sont utiles pour des projets plus sophistiqués ; inutile de vous en soucier maintenant.

7. Modifiez la propriété *File Name* par **Fonctions Math.vb** ou tout autre nom de fichier pertinent, puis appuyez sur ENTRÉE. Comme ce projet n'est créé qu'à des fins de test, vous disposez d'une marge de manœuvre considérable – vous n'allez pas réellement créer de fonctions mathématiques ou tout autre « contenu » et l'effacerez plus tard.

   Le nom de fichier de votre module est actualisé dans la fenêtre Propriétés, l'Explorateur de solutions et l'Éditeur de code.

8. Ancrez de nouveau la fenêtre Propriétés et l'Explorateur de solutions en double-cliquant sur leurs barres de titre.

Comme vous pouvez le constater, le travail avec les modules ressemble beaucoup à celui avec les formulaires. Dans le prochain exercice, vous allez ajouter une variable publique à un module.

Astuce Pour supprimer un module d'un projet, cliquez dessus dans l'Explorateur de solutions, puis cliquez sur la commande Exclure du projet du menu Projet. Cette commande ne supprime pas le module de votre disque dur mais supprime le lien entre le module spécifié et le projet en cours. Il est possible d'inverser les effets de cette commande en cliquant sur la commande Ajouter un élément existant dans le menu Projet, en sélectionnant le fichier que vous souhaitez ajouter au projet et en cliquant sur Ajouter.

Travailler avec des variables publiques

Il est très simple de déclarer une variable globale, ou publique, dans un module. Pour ce faire, tapez le mot clé *Public* suivi du nom de la variable et d'une déclaration de type. Après avoir déclaré la variable, vous pouvez la lire, la modifier ou l'afficher dans n'importe quelle procédure de votre programme. Par exemple, l'instruction

```
Public TotalEnCours As Integer
```

déclare une variable publique appelée *TotalEnCours* de type *Integer* (entier).

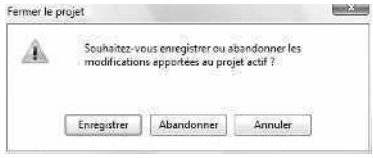
Les exercices suivants montrent comme utiliser une variable publique appelée *Gains* dans un module. Vous allez reprendre Bandit Manchot, le premier programme que vous avez créé dans ce livre, et utiliser la variable *Gains* pour connaître votre taux de réussite à ce jeu.

Remarque Bandit Manchot est le programme de jeu de machine à sous du chapitre 2, « Écrire son premier programme ».

Modifier le projet Bandit Manchot

1. Cliquez sur la commande Fermer le projet du menu Fichier pour fermer le projet Module Test.

   Comme vous avez nommé mais pas encore enregistré votre projet, la boîte de dialogue qui suit s'affiche :



   Inutile de conserver ce projet sur votre disque dur ; il ne s'agit que d'un test. Pour illustrer la fonctionnalité « fermer sans enregistrer » de Visual Studio 2008, vous allez maintenant abandonner le projet.

2. Cliquez sur le bouton Abandonner.

   Visual Studio efface la totalité du projet, en supprimant les fichiers temporaires associés au module de la mémoire de l'ordinateur et du disque dur. Cette fonctionnalité semble évidente, mais j'ai voulu mettre en évidence la possibilité de fermer un projet sans l'enregistrer. En effet, il s'agit d'une amélioration heureuse du logiciel qui répondait à nos besoins pour ce type de test (ne l'employez toutefois pas à la légère !). Vous allez maintenant ouvrir un projet plus substantiel et le modifier.

3. Ouvrez le projet Gagnant dans le dossier c:\vb08epe\chap10\Gagnant\Bandit Manchot.

   Le programme s'exécute dans l'environnement de développement.

4. Affichez le formulaire si ce n'est déjà fait.

   L'interface utilisateur suivante apparaît :

Le projet Gagnant correspond au même programme de machine à sous que celui que vous avez créé au chapitre 2. Grâce à ce programme, l'utilisateur peut cliquer sur un bouton Lancer pour afficher des numéros aléatoires dans trois zones de chiffres. Si le numéro 7 apparaît dans une de ces zones, l'ordinateur émet un bip et affiche une image montrant un vieil homme qui distribue de l'argent. Dans ce chapitre, je me suis contenté de renommer la solution afin que vous ne confondiez pas cette nouvelle version avec la version d'origine.

5. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.
6. Cliquez à six ou sept reprises sur le bouton Lancer, puis cliquez sur le bouton Fin.

   Comme vous vous en souvenez sans doute, le programme utilise la fonction *Rnd* pour générer trois numéros aléatoires chaque fois que vous cliquez sur le bouton Lancer. Si un de ces numéros est 7, la procédure événementielle du bouton Lancer (*Button1_Click*) affiche l'image d'une personne distribuant de l'argent et émet un bip.

Vous allez maintenant modifier le formulaire et ajouter un module pour améliorer le programme.

Ajouter un module

1. Cliquez sur le contrôle Label de la Boîte à outils, puis créez une nouvelle étiquette rectangulaire en dessous de l'étiquette Bandit Manchot.
2. Définissez les propriétés de la nouvelle étiquette comme dans le tableau suivant : pour vous aider à identifier la nouvelle étiquette dans le code, vous allez remplacer le nom de l'objet étiquette par *lblWins*.

Objet	Propriété	Paramètre
Label5	*Font*	Arial, Bold Italic, 12 points
	ForeColor	Vert (sous l'onglet Personnaliser)
	Name	lblWins
	Text	« Gains : 0 »
	TextAlign	MiddleCenter

Une fois que vous avez terminé, votre formulaire ressemble à :

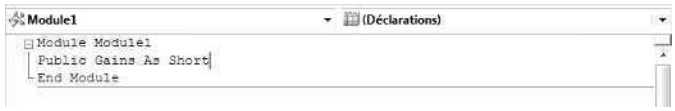
Vous allez maintenant ajouter un nouveau module à votre projet.

3. Dans le menu Projet, cliquez sur la commande Ajouter un nouvel élément, sélectionnez le modèle Module, puis cliquez sur Ajouter.

   Un module appelé Module1.vb s'affiche dans l'Éditeur de code.

4. Placez le point d'insertion sur la ligne blanche située entre les instructions *Module Module 1* et *End Module*, tapez **Public Gains As Short**, puis appuyez sur ENTRÉE.

   Cette instruction déclare une variable publique de type entier *Short* dans votre programme. Celle-ci ressemble à une déclaration de variable normale, sauf que le mot clé *Public* a été remplacé par le mot clé *Dim*. Lorsque votre programme s'exécute, chaque procédure événementielle du programme a accès à cette variable. Votre module présente un résultat similaire à :



5. Dans l'Explorateur de solutions, cliquez sur Gagnant.vb, cliquez sur le bouton Concepteur de vues, puis double-cliquez sur le bouton Lancer.

   La procédure événementielle *Button1_Click* du bouton Lancer s'affiche dans l'Éditeur de code.

6. Tapez les instructions suivantes en dessous de l'instruction *Beep()* dans la procédure événementielle :

```
Gains = Gains + 1
blWins.Text = "Gains : " & Gains
```

Cette partie du code incrémente la variable publique *Gains* si un 7 apparaît pendant un lancement. La deuxième instruction utilise l'opérateur de concaténation (*&*) pour assigner une chaîne à l'objet *lblWins* au format *Gains : X*, où *X* correspond au nombre de gains. Voici à quoi ressemble la procédure événementielle terminée :

```
Button1                                    Click
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.Eve
        PictureBox1.Visible = False    'masque image
        Label1.Text = CStr(Int(Rnd() * 10))   'Sélectionne des nombres aléatoires
        Label2.Text = CStr(Int(Rnd() * 10))
        Label3.Text = CStr(Int(Rnd() * 10))
        ' si l'un des nombres est 7, affiche l'image et émet un signal sonore
        If (Label1.Text = "7") Or (Label2.Text = "7") Or _
              (Label3.Text = "7") Then
            PictureBox1.Visible = True
            Beep()
            Gains = Gains + 1
            lblWins.Text = "Gains : " & Gains
        End If
    End Sub
```

7. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer tous vos changements sur le disque.

La commande Enregistrer tout enregistre les changements apportés à votre module ainsi que ceux de votre formulaire et de vos procédures événementielles.

8. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

9. Cliquez sur le bouton Lancer jusqu'à ce que vous ayez gagné plusieurs fois.

L'étiquette Gains garde une trace de vos jackpots. Chaque fois que vous gagnez, elle incrémente le total de 1. Après 10 lancés, nous obtenons le résultat ci-après :

Remarque Le nombre exact de réussites sera différent chaque fois que vous exécuterez le programme, en raison de l'instruction *Randomize* qui se trouve dans la procédure événementielle *Form1_Load*.

10. Cliquez sur Fin pour quitter le programme.

La variable publique *Gains* était utile dans la procédure précédente car elle maintenait sa valeur *via* plusieurs appels à la procédure événementielle *Button1_Click*. Si vous aviez déclaré localement *Gains* dans la procédure événementielle *Button1_Click*, la variable serait redéfinie à chaque fois, exactement comme le compteur kilométrique de votre véhicule lorsque vous le remettez à zéro. En exploitant une variable publique dans un module, vous vous évitez cette réinitialisation.

Variable publique ou variables de formulaire

Dans l'exercice précédent, vous avez utilisé une variable publique pour suivre le nombre de gains dans le programme qui simule une machine à sous. Vous auriez également pu déclarer la variable *Gains* en début de code du formulaire. Les deux techniques génèrent le même résultat car une variable publique et une variable déclarée dans la zone des déclarations générales d'un formulaire ont la même portée dans l'ensemble du formulaire. Les variables publiques sont toutefois uniques car elles maintiennent leurs valeurs dans *tous* les formulaires et modules utilisés dans un projet – en d'autres termes, dans tous les composants qui appartiennent au même *espace de noms* de projet. Le mot clé de l'espace de noms du projet est défini automatiquement la première fois que vous enregistrez votre projet. Pour afficher ou modifier le nom de l'espace de noms, sélectionnez le projet dans l'Explorateur de solutions, cliquez sur la commande Propriétés de Bandit Manchot dans le menu Projet, puis examinez ou modifiez le texte dans la zone de texte Espace de noms racine de l'onglet Application.

Créer des procédures

Les procédures permettent de grouper un ensemble d'instructions liées pour accomplir une tâche. Visual Basic comprend deux principaux types de procédures :

- Les *procédures Function* sont appelées d'après leur nom à partir de procédures événementielles ou d'autres procédures. Souvent utilisées pour accomplir des calculs, les procédures *Function* peuvent recevoir des arguments et retournent toujours une valeur dans le nom de la fonction.
- Les *procédures Sub* sont appelées d'après leur nom à partir des procédures événementielles ou d'autres procédures. Elles peuvent recevoir des arguments et également

repasser des valeurs modifiées dans une liste d'arguments. Toutefois, contrairement aux fonctions, les procédures *Sub* ne retournent pas de valeurs associées à leur nom de procédure *Sub* particulier. Ces procédures servent habituellement à recevoir ou à traiter des entrées, afficher des sorties ou définir des propriétés.

Il est possible de définir des procédures *Function* et *Sub* dans le code d'un formulaire mais, pour de nombreux utilisateurs, il est plus utile de créer des procédures dans un module car elles ont alors une portée s'étendant à l'ensemble du projet. Cela est particulièrement vrai pour les *procédures généralistes* – des blocs de code souples et suffisamment utiles pour s'adapter à une multitude de contextes de programmation.

Imaginez, par exemple, un programme contenant trois mécanismes pour imprimer une image sur des formulaires différents : une commande de menu appelée Imprimer, un bouton de barre d'outils appelé Imprimer et l'icône d'une imprimante à utiliser par glisser-déposer. Vous pouvez placer les mêmes instructions d'impression dans chacune des trois procédures événementielles ou bien gérer les requêtes d'impression depuis les trois sources grâce à une procédure placée dans un module.

Avantages des procédures généralistes

Voici les avantages des procédures généralistes :

- Elles permettent d'associer un groupe d'instructions souvent utilisées à un nom familier.
- Elles éliminent les répétitions. Vous pouvez définir une procédure une fois et l'exécuter dans votre programme plusieurs fois.
- Les programmes sont plus lisibles. Un programme divisé en une collection de petites parties est plus facile à isoler et à comprendre qu'un programme composé d'une seule grande partie.
- Elles simplifient le développement du programme. Les programmes séparés en unités logiques sont plus simples à concevoir, à rédiger et à déboguer. En outre, si vous rédigez un programme dans une définition de groupe, vous pouvez échanger les procédures et les modules plutôt que des programmes entiers.
- On peut les réutiliser dans d'autres projets et solutions. Il est aisé d'incorporer des procédures composées de modules standard à d'autres projets de programmation.
- Elles étendent le langage Visual Basic. Les procédures peuvent souvent accomplir des tâches impossibles pour des mots clés Visual Basic ou des méthodes Microsoft .Net Framework.

Développer des procédures Function

Une procédure Function est un groupe d'instructions situé entre une instruction *Function* et une instruction *End Function*. Les instructions de la fonction accomplissent le travail utile – traitement du texte, gestion des entrées ou calcul d'une valeur numérique. Dans un programme, vous exécutez, ou *appelez*, une fonction en plaçant le nom de la fonction dans une instruction accompagnée de tous les arguments nécessaires.

Les *arguments* sont des données que l'on utilise pour faire fonctionner les fonctions. Ils doivent être placés entre parenthèses et séparés par des virgules. L'exploitation d'une procédure Function revient exactement au même que celle d'une fonction intégrée ou d'une méthode comme *Int*, *Rnd* ou *FromFile*.

Astuce Par défaut, les fonctions déclarées dans des modules sont publiques. Par conséquent, on peut les utiliser dans n'importe quelle procédure événementielle d'un projet.

Syntaxe d'une fonction

Voici la syntaxe de base d'une fonction :

```
Function NomFonction([arguments]) As Type
    instructions de la fonction
    [Valeur retournée]
End Function
```

Voici les éléments de syntaxe importants :

- *NomFunction* est le nom de la fonction créée.
- *As Type* est une paire de mots clés qui spécifie le type de retour de la fonction. Dans Visual Basic 6, la déclaration de type était optionnelle, mais elle est fortement recommandée dans Visual Basic 2008. En l'absence de type, le type de retour par défaut est *Object*.
- *arguments* est une liste d'arguments optionnels (séparés par des virgules) à utiliser dans la fonction. Chaque argument doit également être déclaré comme étant d'un type spécifique. Par défaut, Visual Basic ajoute le mot clé *ByVal* à chaque argument, indiquant ainsi qu'une copie des données est passée à la fonction *via* cet argument, mais que tout changement apporté aux arguments ne sera pas renvoyé à la routine appelante.

- Les *instructions de la fonction* représentent un bloc d'instructions qui accomplit le travail de la fonction. Les premières instructions déclarent habituellement des variables locales qui seront utilisées dans la fonction tandis que les autres accomplissent le travail de la fonction.
- *Return* est une instruction nouvelle qui n'existe pas dans Visual Basic 6 – elle vous permet d'indiquer à quel endroit du bloc de code de la fonction vous souhaitez retourner une valeur à la procédure appelante ainsi que la nature de cette valeur. À l'exécution d'une instruction *Return*, on quitte la fonction. Ainsi, les instructions de fonction situées après l'instruction *Return* ne sont pas exécutées. Utilisez sinon la syntaxe de Visual Basic 6 et retournez une valeur à la routine appelante en assignant la valeur à *NomFonction*.
- Les crochets ([ ]) entourent les éléments de syntaxe optionnels. Visual Basic n'a pas besoin de placer ces éléments de syntaxe entre crochets.

Les fonctions retournent toujours une valeur à la procédure appelante dans le nom de la fonction (*NomFonction*). C'est pourquoi la dernière instruction d'une fonction est souvent une instruction d'assignation qui place le calcul final de la fonction dans *NomFonction*. Par exemple, la procédure Function *TotalImpôt* calcule les impôts locaux et nationaux d'un élément, puis assigne le résultat au nom *TotalImpôt*, comme suit :

```
Function TotalImpôt(ByVal Prix as Single) As Single
    Dim ImpôtNational, ImpôtLocal As Single
    ImpôtNational = Prix * 0.05 'L'impôt national est de 5%
    ImpôtLocal = Prix * 0.015 'L'impôt local est de 1.5%
    TotalImpôt = ImpôtNational + ImpôtLocal
End Function
```

Vous pouvez alternativement employer la syntaxe Visual Basic 2008 et retournez une valeur à la procédure appelante grâce à une instruction *Return*, comme dans la déclaration de fonction suivante :

```
Function TotalImpôt(ByVal Prix as Single) As Single
    Dim ImpôtNational, ImpôtLocal As Single
    ImpôtNational = Prix * 0.05 'L'impôt national est de 5%
    ImpôtLocal = Prix * 0.015 'L'impôt local est de 1.5%
    Return ImpôtNational + ImpôtLocal
End Function
```

Dans ce livre, j'utilise le plus souvent la syntaxe *Return*, mais vous pouvez exploiter tout autre mécanisme pour retourner des données depuis une fonction.

Appeler une procédure de fonction

Pour appeler la fonction *TotalImpôt* dans une procédure événementielle, utilisez une instruction semblable à :

```
lblImpôts.Text = TotalImpôt(500)
```

Cette instruction calcule les impôts totaux dûs pour un élément coûtant 500, puis assigne le résultat à la propriété *Text* de l'objet *lblImpôts*. La fonction *TotalImpôt* peut également recevoir comme argument une variable, comme le montre les instructions suivantes :

```
Dim PrixTotal, PrixVente As Single
PrixVente = 500
PrixTotal = PrixVente + TotalImpôt(PrixVente)
```

La dernière instruction exploite la fonction *TotalImpôt* pour déterminer les taxes correspondant au chiffre de la variable *PrixVente*, puis ajoute la taxe calculée à *PrixVente* pour obtenir le prix total d'un élément. Le code est bien plus clair avec une fonction, ne trouvez-vous pas ?

Exploiter une fonction pour effectuer un calcul

Dans l'exercice suivant, vous allez ajouter une fonction au programme Gagnant pour calculer le taux de réussite dans le jeu – en d'autres termes, le pourcentage de lancements où un ou plusieurs 7 apparaissent. Pour effectuer ce calcul, vous allez ajouter au module une fonction appelée *TauxRéussite* et une variable publique appelée *Lancements*. Ensuite, vous appellerez la fonction *TauxRéussite* chaque fois que l'on cliquera sur le bouton Lancer. Vous afficherez les résultats dans une nouvelle étiquette que vous allez créer sur le formulaire.

Créer une fonction pour calculer le taux de réussite

1. Affichez le formulaire du programme Gagnant que vous avez modifié. L'interface utilisateur de la machine à sous apparaît.
2. Utilisez le contrôle *Label* pour créer un nouvel objet étiquette en dessous de l'étiquette Gains. Définissez les propriétés suivantes pour l'étiquette :

Objet	Propriété	Paramètres
Label5	*Font*	Arial, Bold Italic, 12 points
	ForeColor	Rouge (sous l'onglet Personnaliser)
	Name	lblRate
	Text	« 0,0 % »
	TextAlign	MiddleCenter

Votre formulaire se présente ainsi :

3. Dans l'Explorateur de solutions, cliquez sur le module Module1.vb, puis cliquez sur le bouton Afficher le code.

Le module Module1 s'affiche dans l'Éditeur de code.

4. Tapez la déclaration de variable publique suivante en dessous de l'instruction `Public Gains As Short` :

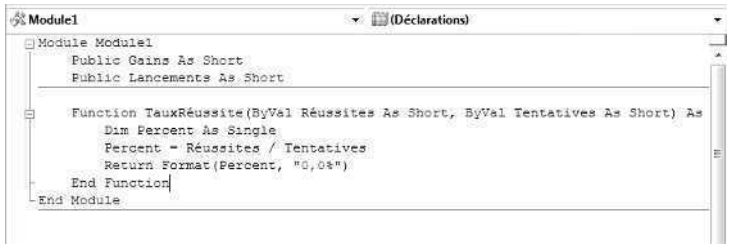
```
Public Lancements As Short
```

Le module contient désormais deux variables publiques, *Gains* et *Lancements*, qui seront disponibles dans toutes les procédures du projet. Vous allez utiliser *Lancements* comme compteur pour suivre le nombre de lancements effectués.

5. Insérez une ligne vierge dans le module, puis tapez la déclaration de fonction suivante :

```
Function TauxRéussite(ByVal Réussites As Short, ByVal Tentatives As Short) As String
    Dim Percent As Single
    Percent = Réussites / Tentatives
    Return Format(Percent, "0,0 %")
End Function
```

Après avoir tapé la première ligne de code de la fonction, Visual Basic ajoute automatiquement une instruction *End Function*. Après avoir tapé le reste du code, votre écran se présente ainsi :



La fonction *TauxRéussite* détermine le pourcentage de gains en divisant l'argument *Réussites* par l'argument *Tentatives*, puis règle l'apparence du résultat en utilisant la fonction *Format*. La fonction *TauxRéussite* est déclarée comme une chaîne car la fonction *Format* retourne une valeur chaîne. Les arguments *Réussites* et *Tentatives* sont des marques de réservation destinées aux deux variables entières ***courtes*** (*short*) qui seront passées à la fonction pendant l'appel de fonction. La fonction *TauxRéussite* est suffisamment généraliste pour être utilisée avec des nombres ou des variables entières plus courtes, et non uniquement avec *Gains* et *Lancements*.

6. Affichez de nouveau le formulaire Gagnant, puis double-cliquez sur le bouton Lancer pour faire apparaître la procédure événementielle *Button1_Click*.

7. En dessous de la quatrième ligne de la procédure événementielle (`Label3.Text = CStr(Int(Rnd() * 10))`), tapez l'instruction suivante :

```
Lancements = Lancements + 1
```

Cette instruction incrémente la variable *Lancements* chaque fois que l'utilisateur clique sur Lancer ; de nouveaux chiffres sont placés dans la fenêtre correspondante.

8. Descendez dans l'Éditeur de code puis, entre les instructions *End If* et *End Sub*, tapez l'instruction suivante en dernière ligne de la procédure événementielle *Button1_Click* :

```
lblRate.Text = TauxRéussite(Gains, Lancements)
```

En tapant la fonction *TauxRéussite*, notez comment Visual Studio affiche automatiquement les noms et les types des arguments de la fonction *TauxRéussite* que vous venez de développer (petit détail agréable).

Cette instruction a pour objet d'appeler la fonction *TauxRéussite* en utilisant les variables *Gains* et *Lancements* comme arguments. Le résultat retourné est un pourcentage au format chaîne et cette valeur est assignée à la propriété *Text* de l'étiquette *lblRate* sur le formulaire après chaque lancement. Supprimez maintenant la fonction *Randomize* de la procédure événementielle *Form1_Load* de sorte que vos résultats suivent un modèle familier lorsque vous testerez le projet.

9. Descendez dans l'Éditeur de code jusqu'à la procédure événementielle *Form1_Load* et supprimez la fonction *Randomize* ou placez-la en commentaire en la faisant précéder d'une apostrophe.

Désormais, chaque fois que vous exécuterez ce programme, les nombres aléatoires générés suivront un schéma prévisible. Cela vous permet de tester votre code. Toutefois, une fois le test terminé, sans doute souhaiterez-vous réactiver la fonction afin que les résultats soient réellement aléatoires.

Vous allez à présent exécuter le programme.

Exécuter le programme Gagnant

1. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme modifié Gagnant.
2. Cliquez sur le bouton Lancer à dix reprises.

   Les cinq premières fois, le taux de réussite reste à 100,0 %. Vous gagnez chaque fois le jackpot. Toutefois, au fur et à mesure que vous cliquez, le taux de réussite chute à 83,3 %, 71,4 %, 75,0 % (un autre gain) et 60,0 % (un total de 6 sur 10). Après 10 lancements, votre écran se présente ainsi :

   Si vous continuez de lancer le jeu, vous remarquerez que le taux de réussite chute à 28 %. La fonction *TauxRéussite* montre que vous êtes assez chanceux au début, mais qu'au bout de quelque temps, la réalité vous rattrape.
3. Une fois que vous avez terminé avec le programme, cliquez sur le bouton Arrêter.

   Le programme s'arrête et vous revenez à l'environnement de développement. Vous pouvez ajouter de nouveau la fonction *Randomize* à la procédure événementielle *Form1_Load* pour voir comment le programme fonctionne de façon véritablement aléatoire. Au bout d'environ 100 lancements (il faut suffisamment d'itérations pour que la variation statistique s'atténue un peu), vous devriez atteindre un taux de réussite de 28 % chaque fois que vous exécutez le programme. Si vous aimez les chiffres, ce test est intéressant.
4. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

Développer des procédures Sub

Une procédure Sub ressemble à une procédure Function, à la différence près qu'elle ne retourne pas de valeur associée à son nom. Les procédures Sub servent généralement à obtenir des entrées de l'utilisateur, à afficher ou imprimer des informations ou à manipuler plusieurs propriétés associées à une condition. Ces procédures permettent également de traiter et de mettre à jour des variables reçues dans une liste d'arguments pendant un appel de procédure, et de repasser une ou plusieurs de ces valeurs au programme appelant.

Syntaxe de la procédure Sub

Voici la syntaxe de base d'une procédure Sub

```
Sub NomProcédure([arguments])
    instructions de la procédure
End Sub
```

Voici les éléments importants :

- *NomProcédure* est le nom de la procédure Sub que vous créez.
- *arguments* est une liste d'arguments optionnels (séparés par des virgules s'il y en a plusieurs) à utiliser dans la procédure Sub. Chaque argument doit également être déclaré comme étant d'un type spécifique. (Par défaut, Visual Studio ajoute le mot clé *ByVal* à chaque argument, indiquant ainsi qu'une copie des données est passée à la fonction *via* cet argument, mais que tout changement apporté aux arguments ne sera pas retourné à la routine appelante).
- les *instructions de la procédure* sont un bloc d'instructions qui accomplissent le travail de la procédure.

Dans l'appel de procédure Sub, le nombre et le type d'arguments envoyés à la procédure doivent correspondre au nombre et au type d'arguments de la déclaration de procédure Sub, et l'ensemble du groupe doit être placé entre parenthèses. Si des variables passées à une procédure Sub sont modifiées pendant la procédure, les variables mises à jour ne sont pas repassées au programme, sauf si la procédure a défini les arguments en utilisant le mot clé *ByRef*. Par défaut, les procédures Sub déclarées dans un module sont publiques. Elles peuvent donc être appelées par n'importe quelle procédure événementielle dans un projet.

Important À partir de Visual Basic .NET 2002, tous les appels à une procédure Sub doivent posséder des parenthèses après le nom de la procédure. Un ensemble de parenthèses vides est nécessaire même en l'absence d'argument. Il s'agit d'une nouveauté par rapport à Visual Basic 6, où les parenthèses ne sont nécessaires que si un argument est passé par valeur à une procédure Sub. Vous en saurez plus sur le passage des variables par référence et par valeur plus loin dans ce chapitre.

Par exemple, la procédure Sub suivante reçoit un argument de chaîne (le nom d'une personne) et utilise une zone de texte pour lui souhaiter un joyeux anniversaire. Si cette procédure Sub est déclarée dans un module, on peut l'appeler depuis n'importe quelle procédure événementielle du programme.

```
Sub VoeuxAnniversaire (ByVal Personne As String)
    Dim Msg As String
    If Personne <> "" Then
        Msg = "Joyeux Anniversaire" & Personne & "!"
    Else
        Msg = "Nom non spécifié."
    End If
    MsgBox(Msg, , "Meilleurs voeux")
End Sub
```

La procédure *VoeuxAnniversaire* reçoit le nom de la personne à féliciter en utilisant l'argument *Personne*, une variable de chaîne reçue par valeur pendant l'appel de procédure. Si la valeur de *Personne* est vide, ou *null*, on utilise le nom spécifié pour construire une chaîne de message qui s'affichera avec une fonction *MsgBox*. Si l'argument est null, la procédure affiche le message « Nom non spécifié ».

Appeler une procédure Sub

Pour appeler une procédure Sub dans un programme, déclarez-la par son nom puis faites-la suivre des arguments requis. Par exemple, pour appeler la procédure *VoeuxAnniversaire*, tapez l'instruction suivante :

```
VoeuxAnniversaire("Robert")
```

Dans cet exemple, la procédure *VoeuxAnniversaire* insère le nom « Robert » dans une chaîne de message et la routine affiche la boîte de message suivante :

Le gain de place obtenu grâce à une procédure devient évident lorsque vous appelez à plusieurs reprises la procédure en utilisant une variable, comme ci-après :

```
Dim NouveauNom As String
Do
    NouveauNom = InputBox("Taper un nom.", "Liste Anniversaire")
    VoeuxAnniversaire(NouveauNom)
Loop Until NouveauNom = ""
```

Ici, l'utilisateur peut saisir autant de noms de personnes à féliciter qu'il le souhaite. Le prochain exercice vous permet d'exploiter une procédure Sub pour vous entraîner à gérer un autre type d'entrée dans un programme.

Exploiter une procédure Sub pour gérer des saisies utilisateur

Les procédures Sub servent souvent à gérer les saisies utilisateur lorsque les informations proviennent de deux sources ou plus et que leur format doit être identique. Dans l'exercice suivant, vous allez créer une procédure Sub appelée *AjouterNom* qui invite l'utilisateur à saisir une entrée et formate le texte de sorte à pouvoir l'afficher sur plusieurs lignes dans une zone de texte. Cette procédure vous fera gagner du temps car vous l'utiliserez dans deux procédures événementielles, associées chacune à une zone de texte différente. Comme la procédure est déclarée dans un module, vous n'aurez à la taper qu'à un seul emplacement. La procédure sera également disponible pour tous les autres formulaires ajoutés au projet.

Créer une procédure Sub de zone de texte

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet. Visual Studio ferme le projet en cours (la machine à sous Gagnant).
2. Créez un nouveau projet Application Windows Forms nommé **MaSubTextBox**.

   Le nouveau projet est créé et un formulaire vierge s'affiche dans le Concepteur.
3. Utilisez le contrôle *TextBox* pour créer côte à côte deux zones de texte au milieu du formulaire.

   Vous allez aujourd'hui faire preuve d'initiative et utiliser ces zones de texte pour accueillir le nom des employés que vous assignerez à deux services.
4. Utilisez le contrôle *Label* pour créer deux étiquettes au-dessus des zones de texte.

   Ces étiquettes contiendront le nom des services.
5. Utilisez le contrôle *Button* pour créer trois boutons : un sous chaque zone de texte et un en haut du formulaire.

   Vous allez utiliser les deux premiers boutons pour assigner des employés à leurs services et le dernier bouton pour quitter le programme.
6. Définissez les propriétés des objets du formulaire comme dans le tableau suivant :

   Comme les zones de texte vont contenir plusieurs lignes, vous allez définir leurs propriétés *Multiline* à True et leurs propriétés *ScrollBars* à Vertical. Ces paramètres sont habituellement utilisés lorsque des zones de texte accueillent plusieurs lignes. Définissez également leurs propriétés *TabStop* à False et leurs propriétés *ReadOnly* à True afin qu'il soit impossible de modifier les informations.

Objet	Propriété	Paramètres
TextBox1	*Multiline*	True
	Name	txtSales
	ReadOnly	True
	ScrollBars	Vertical
	TabStop	False
TextBox2	*Multiline*	True
	Name	txtMkt
	ReadOnly	True
	ScrollBars	Vertical
	TabStop	False
Label1	*Font*	Bold
	Name	lblSales
	Text	« Ventes »
Label2	*Font*	Bold
	Name	lblMkt
	Text	« Marketing »
Button1	*Name*	btnSales
	Text	« Ajouter Nom »
Button2	*Name*	btnMkt
	Text	« Ajouter Nom »
Button3	*Name*	btnQuit
	Text	« Quitter »
Form1	*Text*	« Assigner les équipes du service »

7. Redimensionnez et positionnez les objets afin que votre formulaire se présente ainsi :

Vous allez maintenant ajouter un module et créer une procédure Sub *AjouterNom* générique.

8. Dans le menu Projet, cliquez sur la commande Ajouter un nouvel élément, sélectionnez le modèle Module, puis cliquez sur Ajouter.

   Un nouveau module s'affiche dans l'Éditeur de code.

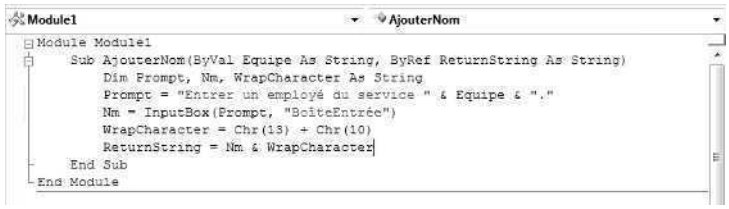
9. Tapez la procédure *AjouterNom* suivante entre les instructions *Module Module1* et *End Module*.

```
Sub AjouterNom(ByVal Equipe As String, ByRef ReturnString As String)
     Dim Invite, Nm, WrapCharacter As String
     Invite = "Entrer un employé. " & Equipe & ""
     Nm = InputBox(Invite, "BoîteEntrée")
     WrapCharacter = Chr(13) + Chr(10)
     ReturnString = Nm & WrapCharacter
End Sub
```

   La procédure Sub générique a recours à la fonction *InputBox* pour demander à l'utilisateur le nom d'un employé. Elle reçoit deux arguments pendant l'appel de procédure : *Equipe*, une chaîne contenant le nom du service, et *ReturnString*, une variable chaîne vide qui contient le nom de l'employé formaté. *ReturnString* est déclarée avec le mot clé *ByRef* afin que tout changement apporté à cet argument dans la procédure soit repassé à la routine appelante via l'argument.

   Avant que le nom de l'employé ne soit renvoyé, des retours chariot et des sauts de ligne sont ajoutés à la chaîne afin que chaque nom s'affiche sur sa propre ligne dans la zone de texte. Vous pouvez utiliser cette technique générale dans n'importe quelle chaîne pour créer une nouvelle ligne.

   Votre Éditeur de code présente un résultat similaire à :



10. Affichez de nouveau le formulaire, puis double-cliquez sur le premier bouton Ajouter Nom sur le formulaire (le bouton en dessous de la zone de texte *txtVentes*). Tapez les instructions suivantes dans la procédure événementielle *btnSales_Click* :

```
Dim PositionVentes As String = ""
AjouterNom("Ventes", PositionVentes)
txtSales.Text = txtSales.Text & PositionVentes
```

L'appel vers la procédure Sub *AjouterNom* comprend un argument passé par valeur (« *Ventes* ») et un argument passé par référence (*PositionVentes*). Cette dernière ligne se sert de l'argument passé par référence pour ajouter du texte dans la zone de texte *txtSales*. L'opérateur de concaténation (*&*) ajoute le nouveau nom à la fin du texte dans la zone de texte.

11. Dans l'Éditeur de code, cliquez sur la flèche Nom de la classe et cliquez sur l'objet *btnMkt* dans la liste. Cliquez ensuite sur la flèche Nom de la méthode, puis cliquez sur l'événement *Click*.

La procédure événementielle *btnMkt_Click* s'affiche dans l'Éditeur de code. Recourir aux zones de liste Nom de la calsse et Nom de la méthode est une autre façon pratique d'ajouter une procédure événementielle.

12. Tapez les instructions suivantes dans la procédure événementielle :

```
Dim MktPosition As String = ""
AjouterNom("Marketing", MktPosition)
txtMkt.Text = txtMkt.Text & MktPosition
```

Cette procédure événementielle est identique à *btnSales_Click*, sauf qu'elle envoie « *Marketing* » à la procédure *AjouterNom* et met à jour la zone de texte *txtMkt*. La variable de retour locale *MktPosition* a été renommée pour la rendre plus intuitive.

13. Cliquez sur la flèche Nom de la classe et cliquez sur l'objet *btnQuit* dans la liste. Cliquez ensuite sur la flèche Nom de la méthode, puis cliquez sur l'événement *Click*.

La procédure événementielle *btnQuit_Click* s'affiche dans l'Éditeur de code.

14. Tapez **End** dans la procédure événementielle *btnQuit_Click*.

15. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout et choisissez le dossier de destination c:\vb08epe\chap10.

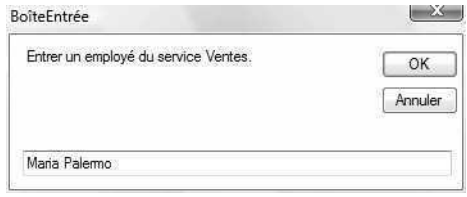
Et voilà ! Vous allez à présent exécuter le programme SubTextBox.

Exécuter le programme SubTextBox

> **Astuce** Le programme SubTextBox complet se trouve dans le dossier c:\vb08epe\chap10\SubTextBox.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.
2. Cliquez sur le bouton Ajouter Nom sous la zone de texte Ventes, puis tapez **Maria Palermo** (ou n'importe quel nom) dans la zone de saisie.

   Votre zone de texte se présente ainsi :



3. Cliquez sur le bouton OK pour ajouter le nom à la zone de texte Ventes.

   Le nom s'affiche dans la première zone de texte.
4. Cliquez sur le bouton Ajouter Nom sous la zone de texte Marketing, tapez **Abraham Asante** dans la zone de saisie correspondante, puis appuyez sur Entrée.

   Le nom s'affiche dans la zone de texte Marketing. Votre écran présente un résultat similaire à :

5. Saisissez d'autres noms dans chaque zone de texte. Vous avez la possibilité de créer le service de vos rêves.

   Chaque nom s'affiche sur sa propre ligne dans les zones de texte. Les zones de texte ne défilent pas automatiquement. Vous ne verrez donc pas chaque nom saisi si vous avez tapé plus de noms que ce que la zone de texte peut accueillir. Utilisez les barres de défilement pour accéder aux noms qui ne sont pas visibles.

6. Cliquez ensuite sur le bouton Quitter pour arrêter le programme.

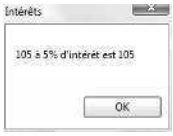
Vous venez de prouver qu'une seule procédure Sub est capable de gérer des tâches de saisie à partir de deux procédures événementielles ou plus. Grâce à ce concept de base comme point de départ, vous pouvez désormais créer des programmes plus sophistiqués qui utilisent les procédures Function et Sub comme outils d'organisation et placent les tâches courantes dans des unités logiques que l'on peut appeler à l'envi.

Aller plus loin : Passer des arguments par valeur et par référence

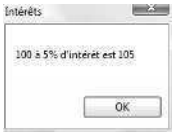
Dans le débat sur les procédures Sub et Function, vous avez appris que les arguments sont passés aux procédures soit par valeur, soit par référence. L'utilisation du mot clé *ByVal* indique que les variables doivent être passées à la procédure par valeur (par défaut). Tout changement apporté à une variable passée par valeur n'est pas renvoyé à la procédure appelante. Toutefois, comme vous l'avez appris dans le programme SubTextBox, l'utilisation du mot clé *ByRef* indique que les variables doivent être passées à une procédure par référence, ce qui signifie que tout changement apporté à la variable dans la procédure est renvoyé à la routine appelante. Le passage par référence présente des avantages significatifs tant que vous veillez à ne pas modifier involontairement une variable dans une procédure. Considérez par exemple la déclaration de procédure Sub et l'appel suivants :

```
Sub CoûtPlusIntérêt(ByRef Coût As Single, ByRef Total As Single)
    Coût = Coût * 1.05 'ajouter 5% au coût...
    Total = Int(Coût) 'arrondit à l'entier et retourne
End Sub
.
.
.
Dim Prix, TotalPrix As Single
Prix = 100
TotalPrix = 0
CoûtPlusIntérêt(Prix, TotalPrix)
MsgBox(Prix & " à 5% d'intérêt est " & TotalPrix)
```

Dans cet exemple, le programmeur passe deux variables en simple précision par référence à la procédure *CoûtPlusIntérêt* : *Prix* et *TotalPrix*. Le programmeur prévoie d'utiliser la variable mise à jour *TotalPrix* dans l'appel *MsgBox* suivant mais a malheureusement oublié que la variable *Prix* a également été mise à jour au cours d'une étape intermédiaire dans la procédure *CoûtPlusIntérêt*. Comme *Prix* a été transmise par référence, les changements apportés à *Coût* se répercutent automatiquement à *Prix*. Cela génère le résultat erroné suivant lorsque le programme s'exécute :



Le programmeur voulait toutefois probablement afficher le message suivant :



Comment donc corriger la procédure *CoûtPlusIntérêt* pour produire le résultat voulu ? La manière la plus simple consiste à déclarer l'argument *Coût* en utilisant le mot clé *ByVal*, comme le montre l'instruction qui suit :

```
Sub CoûtPlusIntérêt(ByVal Coût As Single, ByRef Total As Single)
```

En déclarant *Coût* avec *ByVal*, vous pouvez modifier en toute sécurité *Coût* dans la procédure *CoûtPlusIntérêt* sans renvoyer les changements à la procédure appelante. En conservant la déclaration de *Total* avec *ByRef*, vous pouvez modifier la variable qui est passée mais seuls ces changements seront renvoyés à la procédure appelante. En règle générale, si vous utilisez *ByRef* uniquement lorsque cela est nécessaire, vos programmes ont plus de chance d'être exempts de tout défaut.

Voici quelques lignes directrices pour savoir quand utiliser *ByVal* et *ByRef* :

- Utilisez *ByVal* lorsque vous ne souhaitez pas qu'une procédure modifie une variable passée à la procédure *via* un argument.
- Utilisez *ByRef* lorsque vous souhaitez autoriser une procédure à modifier une variable passée à la procédure.
- Dans le doute, utilisez le mot clé *ByVal*.

Rappel du chapitre 10

Pour	Faites ceci
Créer un nouveau module	Dans la barre d'outils Standard, cliquez sur le bouton Ajouter un nouvel élément, puis sélectionnez le modèle Module. *ou* Dans le menu Projet, cliquez sur la commande Ajouter un nouvel élément, puis sélectionnez le modèle Module.
Renommer un module	Sélectionnez le modèle dans l'Explorateur de solution. Dans la fenêtre Propriétés, spécifiez un nouveau nom dans la propriété *File Name*. *ou* effectuez un clic droit sur le module dans l'Explorateur de solution, sélectionnez Renommer et spécifiez un nouveau nom.
Supprimer un module d'un programme	Sélectionnez le module dans l'Explorateur de solution, puis cliquez sur la commande Exclure du projet dans le menu Projet.
Ajouter un module existant à un projet	Dans le menu Projet, cliquez sur la commande Ajouter un élément existant.
Créer une variable publique	Déclarez la variable en utilisant le mot clé *Public* entre les mots clés *Module* et *End Module* dans un module. Par exemple : `Public VentesTotales As Integer`

Créer une fonction publique	Placez les instructions de la fonction entre les mots clés *Function* et *End Function* dans un module. Par défaut, les fonctions sont publiques. Par exemple :
	`Function TauxRéussite(ByVal Réussites As Short, ByVal _`
	`Tentatives As Short) As String`
	`  Dim Pourcentage As Single`
	`  Pourcentage = Réussites / Tentatives`
	`  Return Format(Pourcentage, "0.0%")`
	`End Function`

Appeler une procédure de fonction	Tapez le nom de la fonction et tous les arguments nécessaires dans une instruction et assignez-la à une variable ou à une propriété du type de retour approprié. Par exemple : `lblRate.Text = TauxRéussite(Gains, Lancements)`

Créer une procédure publique Sub	Placez les instructions de la procédure entre les mots clés *Sub* et *End Sub* dans un module. Par défaut, les procédures Sub sont publiques. Par exemple :
	`Sub CoûtPlusIntérêt(ByVal Coût As Single, _ ByRef Total As Single)`
	`  Coût = Coût * 1.05`
	`  Total = Int(Coût)`
	`End Sub`

Appeler une procédure Sub	Tapez le nom de la procédure et tous les arguments nécessaires dans une instruction. Par exemple : `CoûtPlusIntérêt(Prix, TotalPrix)`

Pour	Faites ceci
Passer un argument par valeur	Utilisez le mot clé *ByVal* dans la déclaration de la procédure. Par exemple : `Sub PersonneFélicitée(ByVal Nom As String)`
Passer un argument par référence	Utilisez le mot clé *ByRef* dans la déclaration de la procédure. Par exemple : `Sub PersonneFélicitée(ByRef Nom As String)`

Chapitre 11

Utiliser les tableaux pour gérer les données numériques et les chaînes

À la fin de ce chapitre, vous saurez :

- Organiser des données dans des tableaux à taille fixe et dynamiques
- Conserver les données d'un tableau pendant son redimensionnement
- Utiliser des tableaux dans votre code pour gérer des volumes de données importants
- Utiliser les méthodes *Sort* et *Reverse* de la classe *Array* pour réordonner les tableaux
- Utiliser le contrôle *ProgressBar* dans vos programmes pour représenter visuellement la durée d'une tâche

Dans une application Microsoft Visual Basic, la gestion des informations est une tâche importante. Plus vos programmes seront denses et plus vous aurez besoin d'outils supplémentaires pour stocker et traiter les données. Pour gérer des données dans des programmes, une approche classique consiste à stocker et à extraire des informations dans des fichiers texte auxiliaires, comme vous le verrez au chapitre 13, « Explorer le traitement des fichiers texte et des chaînes ». La méthode la plus efficace consiste toutefois à stocker et à récupérer les informations dans une base de données. Vous apprendrez à combiner des programmes Visual Basic et des bases de données à partir du chapitre 18, « Introduction à ADO.NET ».

Dans ce chapitre, vous allez apprendre à organiser des variables et d'autres informations dans des conteneurs appelés *tableaux*. Vous verrez comment rationaliser les tâches de gestion des données grâce à des tableaux à taille fixe et dynamiques et comment exploiter les tableaux dans votre code pour gérer des volumes de données importants. Vous apprendrez à redimensionner des tableaux et à préserver leurs données lorsque vous déciderez de modifier leur taille. Pour montrer comment traiter des tableaux de grande envergure, vous allez utiliser les méthodes *Sort* et *Reverse* de la classe *Array* du Microsoft .NET Framework pour réordonner un tableau contenant des valeurs entières aléatoires à six chiffres. Enfin, vous apprendrez à utiliser le contrôle *ProgressBar* pour fournir aux utilisateurs une indication visuelle concernant la durée d'un processus (lié aux tableaux ou autre). Les techniques que vous allez apprendre constituent une introduction solide aux techniques de programmation de bases de données que vous explorerez plus loin dans ce livre.

Travailler avec des tableaux de variables

Cette section vous entraîne à la découverte des tableaux, qui constituent une méthode utile pour stocker n'importe quelle quantité de données pendant l'exécution d'un programme. Les tableaux représentent un mécanisme puissant utilisé depuis longtemps pour stocker des valeurs logiquement apparentées dans un programme. Les développeurs en BASIC, Pascal, C, et d'autres langages de programmation courants intègrent des tableaux aux versions les plus récentes de ces produits pour faire référence à un groupe de valeurs en utilisant un nom et pour traiter ces valeurs individuellement ou collectivement.

Les tableaux vous permettent de suivre un petit jeu de valeurs d'une manière qui serait peu réalisable avec des variables traditionnelles. Prenons par exemple, un tableau de scores pour une partie de base-ball en neuf manches. Pour enregistrer et mémoriser les scores de chaque manche, vous pouvez créer deux groupes de neuf variables (soit un total de 18 variables) dans le programme. Vous les appellerez *EquipeDomicileManche1*, *EquipeVisiteurManche1*, et ainsi de suite, pour les organiser. Le temps et l'espace nécessaires dans votre programme pour travailler individuellement avec ces variables sont considérables. Visual Basic vous permet donc d'organiser des groupes de variables similaires dans un tableau portant un nom commun, avec un index simple d'utilisation. Vous pouvez par exemple créer un tableau bidimensionnel (de hauteur 2 et de largeur 9) appelé *TableauDeScores* contenant les scores du match de base-ball. Voyons comment cela fonctionne.

Créer un tableau

Vous créez, ou *déclarez*, des tableaux dans le code de la même manière que vous déclarez des variables simples. Comme d'habitude, la place à laquelle vous déclarez le tableau détermine l'endroit où l'on peut l'utiliser, ou sa *portée*, comme suit :

- Un tableau déclaré localement dans une procédure ne peut être utilisé que dans cette procédure.
- Un tableau déclaré en haut d'un formulaire peut être utilisé dans l'ensemble du formulaire.
- Un tableau est déclaré publiquement dans un module peut être utilisé partout dans le projet.

Lorsque vous déclarez un tableau, votre instruction de déclaration contient habituellement les informations présentées dans le tableau suivant.

Informations contenues dans l'instruction de déclaration d'un tableau	Description
Nom du tableau	Nom que vous allez utiliser pour représenter votre tableau dans le programme. En général, les noms de tableau suivent les mêmes règles que les noms de variable. (Reportez-vous au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework »).
Types de données	Type des données que vous allez stocker dans le tableau. Dans la plupart des cas, toutes les variables d'un tableau sont du même type. Vous pouvez spécifier un des types de données fondamentaux ou, si le type n'est pas encore arrêté ou si vous allez stocker plusieurs types, vous pouvez spécifier le type *Object*.
Nombre de dimensions	Nombre de dimensions que votre tableau va contenir. La plupart des tableaux sont unidimensionnels (une liste de valeurs) ou bidimensionnels (un tableau de valeurs). Toutefois, il est possible de définir des dimensions supplémentaires si vous travaillez avec un modèle mathématique complexe, comme une forme tridimensionnelle.
Nombre d'éléments	Nombre d'éléments que votre tableau va contenir. Les éléments de votre tableau correspondent directement à l'index du tableau. Dans Visual Basic 2008, le premier index du tableau est toujours 0 (zéro).

Astuce Les tableaux qui contiennent un nombre défini d'éléments sont appelés *tableaux à taille fixe*. Ceux qui contiennent un nombre variable d'éléments (les tableaux qui peuvent s'agrandir pendant l'exécution du programme) sont appelés *tableaux dynamiques*.

Déclarer un tableau à taille fixe

Voici la syntaxe de base d'un tableau à taille fixe

```
Dim NomTableau(Dim1Index, Dim2Index, ...) As DataType
```

Voici les arguments importants :

- *Dim* est le mot clé qui déclare le tableau ; utilisez plutôt *Public* si vous placez le tableau dans un module.
- *NomTableau* est le nom variable du tableau.
- *Dim1Index* est la limite supérieure de la première dimension du tableau, qui correspond au nombre d'éléments moins 1.
- *Dim2Index* est la limite supérieure de la deuxième dimension du tableau, qui correspond au nombre d'éléments moins 1. On peut inclure des dimensions supplémentaires à condition de les séparer par des virgules.
- *DataType* est un mot clé correspondant au type de données inclus dans le tableau.

Par exemple, pour déclarer un tableau de chaîne unidimensionnel appelé *Employés* pouvant contenir 10 noms d'employés (numérotés de 0 à 9), tapez ce qui suit dans une procédure événementielle :

```
Dim Employés(9) As String
```

Voici la même déclaration de tableau dans un module :

```
Public Employés(9) As String
```

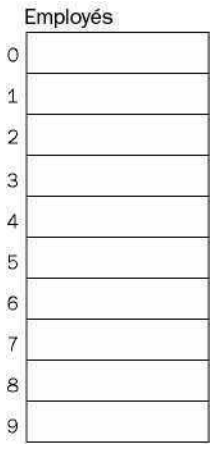
Grâce à la nouvelle syntaxe prise en charge par Visual Basic 2005 et 2008 (mais pas par Microsoft Visual Basic .NET 2002 ou 2003), il est également possible de spécifier explicitement la limite inférieure zéro du tableau avec le code suivant dans une procédure événementielle :

```
Dim Employés(0 To 9) As String
```

Cette syntaxe « 0 à 9 » contribue à améliorer la lisibilité de votre code – une première lecture révèle immédiatement que le tableau *Employés* possède 10 éléments numérotés de 0 à 9. Toutefois, la limite inférieure du tableau doit toujours être zéro. Vous ne pouvez pas utiliser cette syntaxe pour créer une limite inférieure différente.

Définir une mémoire annexe

Lorsque vous créez un tableau, Visual Basic définit à part un espace mémoire qui lui est destiné. La figure qui suit illustre conceptuellement comment s'organise le tableau *Employés* à 10 éléments. Ceux-ci sont numérotés de 0 à 9 et non de 1 à 10 car les index commencent toujours par 0. Encore une fois, l'instruction *Option Base* de Visual Basic 6, qui permet d'indexer des tableaux en commençant au numéro 1, n'est plus prise en charge.



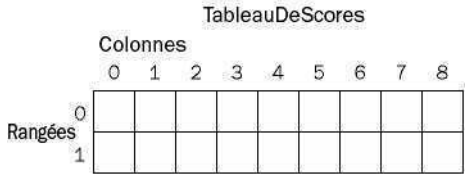
Pour déclarer un tableau public bidimensionnel appelé *TableauDeScores* de deux rangées et neuf colonnes de données entières *Short*, tapez cette instruction dans une procédure événementielle en haut du formulaire :

```
Dim TableauDeScores(1, 8) As Short
```

Grâce à la syntaxe de Visual Basic 2008 qui met l'accent sur la limite inférieure (zéro), vous pouvez également déclarer le tableau comme suit :

```
Dim TableauDeScores(0 To 1, 0 To 8) As Short
```

Après avoir déclaré ce tableau bidimensionnel et une fois que Visual Basic a réservé à cet effet de l'espace mémoire, utilisez ce tableau dans votre programme comme s'il s'agissait d'un tableau de valeurs, comme le montre la figure suivante. Dans ce cas, les éléments du tableau sont numérotés de 0 à 1 et de 0 à 8.



Travailler avec des éléments de tableau

Pour faire référence à un élément d'un tableau, utilisez le nom du tableau suivi d'un index entre parenthèses. L'index doit être un entier ou une expression qui résulte d'un entier. Par exemple, l'index peut être un nombre comme 5, une variable entière comme *num* ou une expression comme *num-1*. On utilise souvent la variable compteur d'une boucle *For…Next*. Par exemple, l'instruction suivante assigne la valeur « Leslie » à l'élément avec un index de 5 dans l'exemple de tableau *Employés* utilisé dans la section précédente.

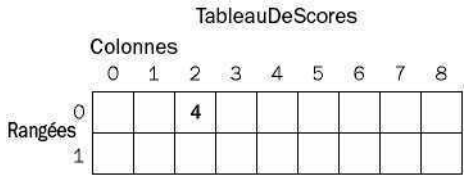
```
Employés(5) = "Leslie"
```

Cette instruction génère le résultat suivant dans notre tableau *Employés* :

De même, l'instruction suivante assigne le numéro 4 à la rangée 0, colonne 2 (en haut de la troisième manche) dans l'exemple de tableau *TableauDeScores* utilisé de la section précédente.

```
TableauDeScores(0, 2) = 4
```

Cette instruction génère le résultat suivant dans notre *TableauDeScores* :



Utilisez ces techniques d'indexation pour assigner ou extraire un élément de tableau.

Créer un tableau à taille fixe destiné à des températures

L'exercice qui suit a recours à un tableau unidimensionnel appelé *Températures* pour enregistrer les températures journalières les plus élevées sur une semaine de sept jours. Le programme montre comment utiliser un tableau pour stocker et traiter un groupe de valeurs liées sur un formulaire. La variable du tableau *Températures* est déclarée en haut du formulaire, puis les températures sont assignées au tableau grâce à une fonction *InputBox* et une boucle *For…Next* que vous avez vues au chapitre 7, « Utiliser les boucles et les minuteurs ». Le compteur boucle sert à référencer chaque élément du tableau. Le contenu du tableau s'affiche ensuite sur le formulaire grâce à une boucle *For…Next* et à un objet zone de texte. On calcule et on affiche également la température moyenne.

Les fonctions *LBound* et *UBound*

Pour simplifier le travail avec le tableau, le programme Tableau Fixe utilise la fonction *UBound* pour vérifier la limite supérieure, ou valeur supérieure d'index, du tableau. *UBound* est un des premiers mots clés Visual Basic, qui reste très utile. Grâce à lui, vous pouvez gérer des tableaux sans faire référence aux instructions de déclaration qui définissent exactement le nombre de valeurs qu'il peut contenir. Étroitement liée, la fonction *LBound*, qui confirme la limite inférieure d'un tableau, est toujours valide dans Visual Basic. En revanche, comme tous les tableaux Visual Basic possèdent désormais une limite inférieure de zéro (0), cette fonction retourne simplement une valeur de 0. La syntaxe des fonctions *UBound* et *LBound* est :

```
LBound(NomTableau)
UBound(NomTableau)
```

où *NomTableau* est le nom d'un tableau déclaré dans le projet.

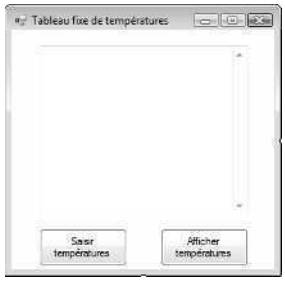
Utiliser un tableau à taille fixe

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Tableau fixe**.
2. Dessinez un objet zone de texte sur le formulaire.

3. Définissez la propriété *Multiline* de l'objet *TextBox1* à True afin de pouvoir redimensionner l'objet.
4. Redimensionnez l'objet zone de texte afin qu'il occupe la quasi-totalité du formulaire.
5. Dessinez côte à côte deux grands objets bouton en dessous de l'objet zone de texte.
6. Définissez les propriétés suivantes pour le formulaire et ses objets :

Objet	Propriété	Paramètres
TextBox1	*ScrollBars*	Vertical
Button1	*Text*	« Saisir températures »
Button2	*Text*	« Afficher températures »
Form1	*Text*	« Tableau fixe de températures »

Voici à quoi ressemble votre formulaire.



7. Dans l'Explorateur de solutions, cliquez sur le bouton Afficher le code pour afficher l'Éditeur de code.
8. Défilez vers le haut du code et tapez la déclaration de tableau suivante, juste sous l'instruction `Public Class Form1` :

```
Dim Températures(0 To 6) As Single
```

Cette instruction crée un tableau appelé *Températures* (de type Single) contenant sept éléments numérotés de 0 à 6. Comme le tableau a été déclaré en haut du code du formulaire, il est disponible dans toutes les procédures événementielles du formulaire.

9. Affichez de nouveau le formulaire et double-cliquez sur le bouton Saisir températures (*Button1*).

La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

10. Tapez les instructions suivantes pour inviter l'utilisateur à saisir des températures et pour charger les entrées dans le tableau :

```
Dim Invite, Titre As String
Dim i As Short
Invite = "Tapez la température du jour."
For i = 0 To UBound(Températures)
     Titre = "Jour " & (i + 1)
     Températures(i) = InputBox(Invite, Titre)
Next
```

La boucle *For…Next* utilise la variable compteur entière courte *i* comme un index de tableau pour charger les températures dans des éléments de tableau de 0 à 6. Au lieu d'utiliser la syntaxe de boucle *For* simplifiée

```
For i = 0 to 6
```

pour traiter le tableau, j'ai choisi une syntaxe un peu plus complexe comprenant la fonction *UBound* pour une meilleure souplesse future. La construction de boucle *For*

```
For i = 0 To UBound(Températures)
```

détermine la limite supérieure du tableau en utilisant l'instruction *UBound*. Cette technique est plus souple car la boucle *For* s'adapte automatiquement à la nouvelle dimension du tableau si celui-ci est ultérieurement agrandi ou réduit.

Pour remplir le tableau avec des températures, la procédure événementielle exploite une fonction *InputBox*, qui affiche le jour en cours grâce au compteur de boucle *For*.

11. Affichez de nouveau le formulaire et double-cliquez sur le bouton Afficher températures (*Button2*).

12. Tapez les instructions suivantes dans la procédure événementielle *Button2_Click* :

```
Dim Résultat As String
Dim i As Short
Dim Total As Single = 0
Résultat = "Températures les plus hautes de la semaine:" & vbCrLf & vbCrLf
For i = 0 To UBound(Températures)
     Résultat = Résultat & "Jour " & (i + 1) & vbTab & _
         Températures(i) & vbCrLf
     Total = Total + Températures(i)
Next
Résultat = Résultat & vbCrLf & _
     "Température moyenne:  " & Format(Total / 7, "0.0")
TextBox1.Text = Résultat
```

Cette procédure événementielle se sert d'une boucle *For…Next* pour parcourir les éléments du tableau et ajoute chaque élément du tableau à une variable chaîne appelée *Résultat*, déclarée en haut de la procédure événementielle. J'ai utilisé plusieurs chaînes littérales, des constantes et des opérateurs de concaténation de chaîne (*&*) pour remplir et formater la chaîne en utilisant des retours chariot (*vbCrLf*), des tabulations (*vbTab*) et des en-têtes. La constante *vbCrLf*, utilisée ici pour la première fois, contient les caractères retour chariot et saut de ligne. Elle représente un moyen efficace de créer de nouvelles lignes. La constante *vbTab* apparaît également pour la première fois. Elle permet de séparer les valeurs de jour

et de température de la chaîne *Résultat*. À la fin de la procédure événementielle, on détermine une moyenne des températures et la chaîne finale est assignée à la propriété *Text* de l'objet zone de texte, comme le montre cette instruction :

```
TextBox1.Text = Résultat
```

13. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer le projet. Désignez le dossier c:\vb08epe\chap11 comme emplacement.

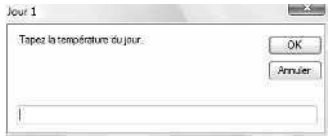
Vous allez à présent exécuter le programme.

Astuce Le programme Tableau fixe complet se trouve dans le dossier c:\vb08epe\chap11\Tableau fixe.

14. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

15. Cliquez sur le bouton Saisir températures et, à l'invite de la fonction *InputBox*, saisissez sept températures différentes. Prenez par exemple les températures de vos dernières vacances.

Voici la boîte de dialogue de la fonction *InputBox* :



16. Après avoir saisi les températures, cliquez sur le bouton Afficher températures.

Grâce au tableau, Visual Basic affiche chaque température dans la zone de texte et présente en bas une moyenne. Votre écran présente un résultat similaire à :

17. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Créer un tableau dynamique

Comme vous pouvez le voir, les tableaux sont très pratiques pour travailler avec des listes de nombres, en particulier si vous les traitez avec des boucles *For…Next*. Mais comment faire si vous n'êtes pas certain de l'espace nécessaire pour votre tableau avant d'exécuter votre programme ? Par exemple, que se passe-t-il si vous laissez le soin à l'utilisateur de choisir le nombre de températures saisies dans le programme Tableau fixe ?

Visual Basic gère efficacement ce problème grâce à un conteneur « élastique » spécial appelé *tableau dynamique*. Les tableaux dynamiques sont dimensionnés à l'exécution, soit si l'utilisateur a défini la taille du tableau, soit si la logique de programmation détermine la taille du tableau en fonction de conditions spécifiques. Le dimensionnement d'un tableau dynamique se fait en plusieurs étapes car même si la taille du tableau n'est pas spécifiée avant l'exécution du programme, vous devez effectuer des « réservations » pour le tableau au moment de la conception. Voici les étapes qui jalonnent la création d'un tableau dynamique :

1. Indiquez le nom et le type du tableau dans le programme à la conception, en omettant le nombre d'éléments dans le tableau. Par exemple, pour créer un tableau dynamique appelé *Températures*, tapez

   ```
   Dim Températures() As Single
   ```

2. Ajoutez du code pour définir le nombre d'éléments du tableau à l'exécution. Vous pouvez inviter l'utilisateur à utiliser une fonction *InputBox* ou un objet zone de texte, ou bien calculer les besoins de stockage du programme grâce à des propriétés ou toute autre logique. Par exemple, les instructions suivantes récupèrent la taille du tableau saisie par l'utilisateur et l'assignent à la variable *Jours* de type *Short* :

   ```
   Dim Jours As Short
   Jours = InputBox("Combien de jours ?", "Créer tableau")
   ```

3. Utilisez la variable dans une instruction *ReDim* pour dimensionner le tableau, en enlevant 1 car les tableaux sont indexés à partir de zéro. Par exemple, l'instruction suivante définit la taille du tableau *Températures* à l'exécution en utilisant la variable *Jours* :

   ```
   ReDim Températures(Jours - 1)
   ```

Important Avec *ReDim*, vous ne devez pas tenter de modifier le nombre de dimensions d'un tableau déjà déclaré.

4. Utilisez la fonction *UBound* pour déterminer la limite supérieure dans une boucle *For…Next* et traiter les éléments du tableau le cas échéant, comme suit :

```
For i = 0 to UBound(Températures)
     Températures(i) = InputBox(Invite, Titre)
Next
```

Dans le prochain exercice, vous allez vous servir de ces étapes pour revoir le programme Tableau fixe afin qu'il traite autant de températures que voulu grâce à un tableau dynamique.

Utiliser un tableau dynamique pour héberger des températures

1. Ouvrez l'Éditeur de code pour afficher le code du projet Tableau fixe.
2. Défilez vers le haut du code où vous avez à l'origine déclaré le tableau à taille fixe *Températures*.
3. Supprimez `0 To 6` de la déclaration du tableau *Températures* afin de rendre le tableau dynamique.

   Voici l'instruction correspondante :

```
Dim Températures() As Single
```

4. Ajoutez la déclaration de variable suivante juste en dessous de la déclaration du tableau *Températures* :

```
Dim Jours As Integer
```

   La variable entière *Jours* sert à recevoir les entrées de l'utilisateur et à dimensionner le tableau dynamique à l'exécution.

5. Défilez vers le bas dans l'Éditeur de code pour afficher la procédure événementielle *Button1_Click* et modifiez le code afin qu'il présente un résultat similaire à ce qui suit. (Les éléments modifiés ou ajoutés sont présentés en gras).

```
Dim Invite, Titre As String
Dim i As Short
Invite = " Tapez la température du jour."
Jours = InputBox("Combien de jours ?", "Créer tableau")
If Jours > 0 Then ReDim Températures(Jours - 1)
For i = 0 To UBound(Températures)
     Titre = "Jour " & (i + 1)
     Températures(i) = InputBox(Invite, Titre)
Next
```

Les quatrième et cinquième lignes invitent l'utilisateur à saisir le nombre de températures qu'il souhaite enregistrer, puis cette entrée sert à dimensionner un tableau dynamique. La structure de décision *If…Then* permet de vérifier que le nombre de jours est supérieur à 0. Le dimensionnement d'un tableau avec un nombre inférieur ou égal à 0 génère une erreur. Comme l'index 0 du tableau sert à stocker la température du premier jour, la variable *Jours* est décrémentée de 1 au moment du dimensionnement du tableau. La variable *Jours* n'est pas nécessaire pour déterminer la limite supérieure de la boucle *For…Next* – comme dans l'exemple précédent, on utilise à la place la fonction *UBound*.

6. Défilez vers le bas dans l'Éditeur de code pour afficher la procédure événementielle *Button2_Click*. Modifiez le code de sorte à ce qu'il ressemble à la routine suivante, où les éléments modifiés apparaissent en gras.

```
Dim Résultat As String
Dim i As Short
Dim Total As Single = 0
Résultat = "Températures les plus élevées :" & vbCrLf & vbCrLf
For i = 0 To UBound(Températures)
     Résultat = Résultat & "Jour " & (i + 1) & vbTab & _
         Températures(i) & vbCrLf
     Total = Total + Températures(i)
Next
Résultat = Résultat & vbCrLf & _
     "Température moyenne:  " & Format(Total / Jours, "0.0")
TextBox1.Text = Résultat
```

La variable *Jours* remplace le numéro 7 dans le calcul de la température moyenne en bas de la procédure événementielle. J'ai également ajouté l'en-tête « Températures les plus élevées : » qui s'affichera dans la zone de texte.

7. Remplacez la propriété *Text* de *Form1* par **Tableau dynamique**.
8. Enregistrez vos changements sur le disque.

Astuce Ce projet porte un nom différent dans les fichiers d'exercices afin de le distinguer du projet Tableau fixe. Le projet Tableau dynamique complet se trouve dans le dossier c:\vb08epe\ chap11\Tableau dynamique.

9. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.
10. Cliquez sur le bouton Saisir températures.
11. Tapez **5** lorsque vous serez invité à saisir le nombre de jours à enregistrer, puis cliquez sur OK.
12. Tapez cinq températures à l'invite.
13. Après avoir saisi les températures, cliquez sur le bouton Afficher températures.

Le programme affiche cinq températures sur le formulaire ainsi que la température moyenne. Votre écran présente un résultat similaire à

14. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Vous venez de vous entraîner à utiliser les deux types de tableau les plus courants en programmation Visual Basic. Lorsque vous développerez vos propres programmes, vous exploiterez rapidement des tableaux de plus grande envergure, mais les concepts restent les mêmes. Vous serez même surpris de constater la rapidité à laquelle Visual Basic effectue des calculs sur des tableaux.

Conserver le contenu d'un tableau en utilisant *ReDim Preserve*

Dans l'exercice précédent, vous avez utilisé l'instruction *ReDim* pour spécifier la taille d'un tableau dynamique à l'exécution. Toutefois, cette instruction présente un inconvénient potentiel : si vous redimensionnez un tableau contenant déjà des données, toutes les données existantes sont irrémédiablement perdues. Une fois l'instruction *ReDim* exécutée, le contenu d'un tableau dynamique est défini à sa valeur par défaut, comme zéro ou null. Selon vos objectifs, vous pouvez utiliser cette fonctionnalité pour vider le contenu de vos tableaux ou devrez faire en sorte que cela n'arrive pas.

Visual Basic 2008 propose heureusement le même mot clé *Preserve* que Visual Basic 6, très utile pour redimensionner les tableaux. Il vous permet de préserver les données d'un tableau lorsque vous modifiez ses dimensions. Voici la syntaxe du mot clé *Preserve* :

```
ReDim Preserve NomTableau(Dim1Eléments, Dim2Eléments, ...)
```

Dans une instruction *ReDim* de ce type, le tableau doit toujours avoir le même nombre de dimensions et contenir le même type de données. En outre, vous ne pouvez redimensionner que la dernière dimension du tableau. Par exemple, si votre tableau contient deux dimensions ou plus, il n'est possible de modifier que la taille de la dernière dimension tout en préservant le contenu du tableau. Les tableaux unidimensionnels passent automatiquement ce test. Vous pouvez donc en toute liberté étendre la taille des tableaux dynamiques grâce au mot clé *Preserve*.

Les exemples suivants montrent comment utiliser ce mot clé pour augmenter la taille de la dernière dimension d'un tableau dynamique sans effacer les données existantes du tableau.

Si vous avez au départ déclaré un tableau chaîne dynamique appelé *Philosophes* avec la syntaxe

```
Dim Philosophes() As String
```

vous pouvez redimensionner le tableau et y ajouter des données en utilisant un code comme celui-ci :

```
ReDim Philosophes(200)
Philosophes(200) = "Steve Harrison"
```

La syntaxe qui suit permet d'étendre la taille du tableau *Philosophes* à 301 éléments (de 0 à 300) tout en préservant son contenu :

```
ReDim Preserve Philosophes(300)
```

Tableaux tridimensionnels

Un exemple plus complexe concernant un tableau tridimensionnel exploite une syntaxe similaire. Imaginez que vous souhaitiez utiliser un tableau tridimensionnel, à simple précision et à virgule flottante appelé *monCube*. Voici la syntaxe qui permet de déclarer le tableau *monCube* :

```
Dim monCube(,,) As Single
```

Redimensionnez ensuite le tableau et ajoutez-y des données grâce au code suivant :

```
ReDim monCube(25, 25, 25)
monCube(10, 1, 1) = 150.46
```

après quoi, étendez la taille de la troisième dimension du tableau (tout en préservant son contenu) grâce à la syntaxe :

```
ReDim Preserve monCube(25, 25, 50)
```

Toutefois, dans cet exemple, seule la troisième dimension peut être étendue – il n'est possible de modifier ni la première, ni la deuxième dimension si vous redimensionnez le tableau avec le mot clé *Preserve*. Si vous vous y risquez, vous générez une erreur d'exécution lorsque l'instruction *ReDim Preserve* s'exécute.

Entraînez-vous un peu à utiliser *ReDim Preserve* et voyez comment l'exploiter pour rendre vos propres tableaux flexibles et opérationnels.

Aller plus loin : Traitement des grands tableaux grâce aux méthodes de la classe *Array*

Dans les sections précédentes, vous avez appris à utiliser des tableaux pour stocker des informations pendant l'exécution du programme. Dans cette section, vous allez apprendre à exploiter les méthodes de la classe *Array* du .NET Framework, qui permet de trier, rechercher et inverser rapidement les éléments d'un tableau, ainsi que d'accomplir d'autres fonctions. Le programme que j'ai créé montre à quel point ces fonctionnalités sont particulièrement adaptées aux tableaux de grande taille. Vous allez également apprendre à utiliser le contrôle *ProgressBar*.

La classe *Array*

Lorsque vous créez des tableaux dans Visual Basic, vous utilisez une classe de base définie par Visual Basic pour mettre en œuvre des tableaux dans des programmes créés par l'utilisateur. La classe *Array* propose également une collection de méthodes qui servent à manipuler des tableaux actifs dans des programmes. Les méthodes les plus utiles sont *Array.Sort*, *Array.Find*, *Array.Reverse*, *Array.Copy* et *Array.Clear*. Vous pouvez localiser d'autres méthodes intéressantes en exploitant la classe *Array* dans l'Éditeur de code (avec Microsoft IntelliSense) et en consultant la documentation. Les méthodes de la classe *Array* s'apparentent aux méthodes .NET Framework que vous avez utilisées dans ce livre ; elles sont appelées par nom et (dans ce cas) requièrent un nom de tableau valide comme argument. Par exemple, pour trier un tableau de températures (comme le tableau *Températures* créé dans le dernier exercice), utilisez la syntaxe suivante :

```
Array.Sort(Températures)
```

Vous effectuez ce type d'appel après que le tableau *Températures* a été déclaré et rempli avec les données dans le programme. Lorsque Visual Basic exécute la méthode *Array.Sort*, il crée en mémoire un emplacement de stockage temporaire destiné au tableau et utilise une routine de tri pour réorganiser le tableau par ordre alphabétique. Une fois le tri accompli, le tableau d'origine est organisé par ordre croissant, avec la valeur la plus petite à l'emplacement 0 et la valeur la plus élevée au dernier emplacement du tableau. Dans l'exemple précédent, le tri génère un tableau des températures journalières classées de la plus froide à la plus élevée.

Dans l'exercice suivant, vous allez observer comment les méthodes *Array.Sort* et *Array.Reverse* peuvent servir à réordonner rapidement un tableau de grande taille contenant des nombres à six chiffres sélectionnés de manière aléatoire entre 0 et 1 000 000. Vous apprendrez également à utiliser le contrôle *ProgressBar*, un outil de l'interface utilisateur intéressant qui offre un retour d'information visuel utile pour l'utilisateur pendant les tris de longue durée. Le contrôle *ProgressBar* se trouve sur l'onglet Contrôles communs de la Boîte à outils ; nous allons l'utiliser pour la première fois.

Utiliser les méthodes *Array* pour trier un tableau de 3 000 éléments

1. Dans le menu Fichier, cliquez sur Ouvrir un projet, puis ouvrez le projet Tri de tableau qui se trouve dans le dossier c:\vb08epe\chap11.
2. Affichez le formulaire s'il n'est pas visible.

   Votre écran présente un résultat similaire à :

   Ce formulaire ressemble aux précédents projets de ce chapitre et présente une zone de texte pour afficher les données du tableau. Il contient toutefois trois boutons destinés à manipuler des tableaux de grande envergure et un objet barre de progression qui offre à l'utilisateur un retour d'information pendant les opérations de longue durée. Le retour d'information visuel est utile dans le cas de calculs d'une durée de plusieurs secondes ; si vous utilisez ce code pour trier un tableau de 3 000 éléments, un léger temps d'attente est inévitable.
3. Cliquez sur la barre de progression du formulaire.

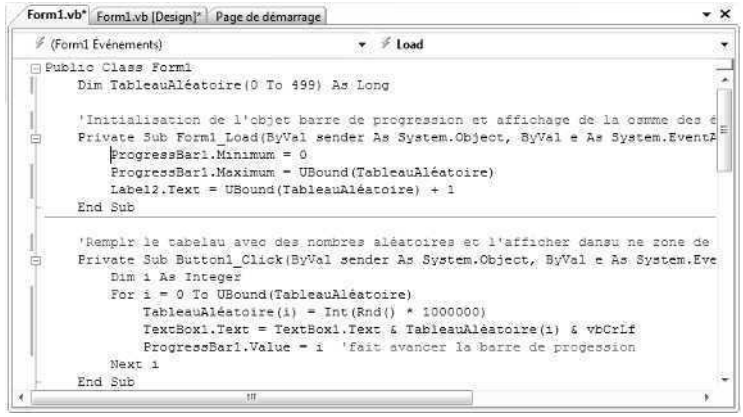
   L'objet *ProgressBar1* est sélectionné sur le formulaire et apparaît dans la fenêtre Propriétés. J'ai créé l'objet barre de progression en utilisant le contrôle *ProgressBar*, dans l'onglet Contrôles communs de la Boîte à outils. Une barre de progression est conçue pour afficher la progression d'un calcul en affichant un nombre approprié de rectangles colorés organisés en une barre de progression horizontale. Avec Windows Vista, un effet de diffusion est employé et la barre de progression affiche une bande de couleur continue : un effet visuel séduisant. Vous avez probablement déjà pu observer cette barre à de nombreuses reprises lors du téléchargement de fichiers et de l'installation de programmes dans Microsoft Windows. Vous allez maintenant en créer une dans vos propres programmes !

   Les propriétés importantes qui font fonctionner une barre de progression sont les propriétés *Minimum*, *Maximum* et *Value*, habituellement manipulées dans le code. Les autres propriétés de barre de progression, qui se trouvent dans la fenêtre Pro-

priétés, contrôlent son aspect et ses fonctions. Pour observer comment ces propriétés sont définies, examinez la procédure événementielle *Form1_Load* de ce programme.

4. Double-cliquez sur le formulaire pour afficher la procédure événementielle *Form1_Load*.

Le code se présente ainsi :



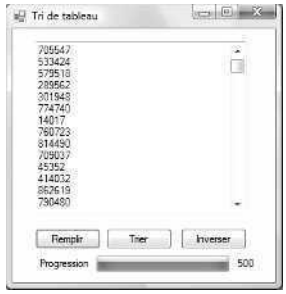
Pour qu'une barre de progression affiche une indication précise sur la durée d'une tâche de calcul, vous devez définir des mesures relatives au début et à la fin de la barre. Pour cela, vous disposez des propriétés *Minimum* et *Maximum* qui sont définies pour correspondre au premier et au dernier élément du tableau que nous avons élaboré. Comme je l'ai dit, le premier élément du tableau est toujours 0, tandis que le dernier élément de tableau dépend de sa taille. J'ai donc utilisé la fonction *UBound* pour récupérer ce nombre et définir la propriété *Maximum* de la barre de progression en conséquence. Le tableau que nous manipulons dans cet exercice est *TableauAléatoire*, un tableau entier *Long* déclaré au départ pour contenir 500 éléments (de 0 à 499).

5. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

Le programme s'exécute et le formulaire Tri de tableau s'affiche. Dans cette procédure événementielle *Form1_Load*, le programme déclarait un tableau appelé *TableauAléatoire*, dimensionné avec 500 éléments. Un objet barre de progression était calibré pour suivre un calcul de 500 unités (la taille du tableau) et le nombre 500 s'affichait à droite de la barre de progression (le travail d'un objet étiquette et de la fonction *UBound*).

6. Cliquez sur le bouton Remplir.

Le programme charge *TableauAléatoire* avec 500 nombres aléatoires (dérivés par la fonction *Rnd*) et affiche les nombres dans la zone de texte. Au fur et à mesure que le programme traite le tableau et remplit l'objet zone de texte avec des données, la barre de progression se remplit lentement de vert. Une fois le processus terminé, votre écran se présente ainsi :



Le code qui génère ce résultat est la procédure événementielle *Button1_Click*, qui contient les instructions suivantes :

```
'Remplit le tableau avec des nombres aléatoires et l'affiche dans la zone de texte
Private Sub Button1_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button1.Click
         Dim i As Integer
         For i = 0 To UBound(TableauAléatoire)
             TableauAléatoire        TableauAléatoire(i) = Int(Rnd() * 1000000)
             TextBox1.Text = TextBox1.Text & TableauAléatoire(i) & vbCrLf
             ProgressBar1.Value = i  'fait avancer la barre de progression
     Next i
End Sub
```

Pour obtenir des nombres aléatoires entiers, j'ai utilisé ensemble les fonctions *Int* et *Rnd*, comme au chapitre 2, « Écrire son premier programme ». J'ai multiplié le nombre aléatoire généré par *Rnd* par 1 000 000 pour obtenir l'ensemble des nombres à six chiffres ou moins. L'assignation de ces nombres au tableau est facilitée par l'utilisation d'une boucle *For…Next* avec un index de tableau mis en correspondance avec le compteur de boucle (*i*). Le remplissage du tableau est une opération extrêmement rapide ; le ralentissement (et la nécessité d'une barre de progression) est provoqué par l'assignation individuelle des éléments du tableau à l'objet zone de texte. Il faut donc mettre à jour 500 fois un composant de l'interface utilisateur et ce processus met quelques secondes à s'accomplir. Ce délai est intéressant car il me permet de mettre en avant le contrôle *ProgressBar*. Comme l'objet barre de pro-

gression a été calibré pour utiliser le nombre d'éléments comme son maximum, l'assignation d'un compteur de boucle (*i*) à la propriété *Value* de la barre de progression permet à la barre d'afficher exactement la quantité de calcul effectué.

7. Cliquez sur le bouton Trier.

Le programme suit un processus similaire pour trier *TableauAléatoire*, en utilisant cette fois la méthode *Array.Sort* pour réordonner le tableau par ordre croissant (les 500 éléments sont classés du plus bas au plus haut).

Votre écran présente un résultat similaire à

Le code qui a généré ce résultat est la procédure événementielle *Button2_Click*, qui contient les instructions suivantes :

```
'Trie le tableau avec la méthode Array.Sort et l'affiche
Private Sub Button2_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button2.Click
         Dim i As Integer
         TextBox1.Text = ""
         Array.Sort(TableauAléatoire)
         For i = 0 To UBound(TableauAléatoire)
             TextBox1.Text = TextBox1.Text & TableauAléatoire(i) & vbCrLf
             ProgressBar1.Value = i  'fait avancer la barre de progression
     Next i
End Sub
```

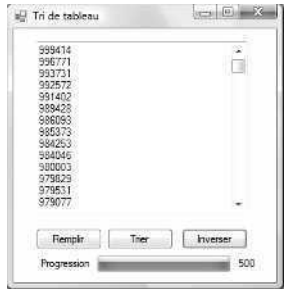
Cette procédure événementielle efface l'objet zone de texte lorsque l'utilisateur clique sur le bouton Trier le tableau, puis trie le tableau en utilisant la méthode *Array.Sort* décrite précédemment. Le processus de tri est très rapide. Encore une fois, le seul ralentissement est provoqué par la reconstruction de l'objet zone de texte ligne par ligne dans la boucle *For…Next*, un processus qui est suivi par l'objet *ProgressBar1* et sa propriété *Value*. La méthode *Array.Sort* est très simple à utiliser, n'est-ce pas ?

8. Cliquez sur le bouton Inverser.

Le programme utilise la méthode *Array.Reverse* pour manipuler *TableauAléatoire*, qui réordonne le tableau dans un sens ou dans l'autre ; dans ce cas, le premier élément devient le dernier et le dernier devient le premier.

> **Remarque** Cette méthode ne génère pas toujours une liste triée ; les éléments du tableau sont triés par ordre décroissant uniquement parce que *TableauAléatoire* a déjà été trié par ordre croissant par la méthode *Array.Sort*. Pour examiner cette liste de plus près, utilisez les barres de défilement ou les flèches directionnelles.

Votre écran présente un résultat similaire à :



Le code qui a généré ce résultat est la procédure événementielle *Button3_Click*, qui contient les instructions suivantes :

```
'Inverse l'ordre des éléments du tableau avec Array.Reverse
Private Sub Button3_Click(ByVal sender As System.Object, _
     ByVal e As System.EventArgs) Handles Button3.Click
         Dim i As Integer
         TextBox1.Text = ""
         Array.Reverse(TableauAléatoire)
         For i = 0 To UBound(TableauAléatoire)
             TextBox1.Text = TextBox1.Text & TableauAléatoire(i) & vbCrLf
             ProgressBar1.Value = i  'fait avancer la barre de progression
     Next i
End Sub
```

Cette procédure événementielle est identique à la procédure événementielle *Button2_Click*, à l'exception de :

```
Array.Sort(TableauAléatoire)
```

qui est devenu

```
Array.Reverse(TableauAléatoire)
```

9. Cliquez sur le bouton Arrêter le débogage pour arrêter le programme.
10. Défilez vers le haut de l'Éditeur de code et localisez l'instruction qui déclare le tableau *TableauAléatoire* :

```
Dim TableauAléatoire(0 To 499) As Long
```

11. Remplacez 499 dans l'instruction de la déclaration du tableau par 2999. L'instruction se présente alors ainsi :

```
Dim TableauAléatoire(0 To 2999) As Long
```

12. Exécutez de nouveau le programme pour observer comment la déclaration et le remplissage d'un tableau avec 3 000 éléments affectent la performance du programme.

    Comme le traitement de 3 000 éléments représente une quantité de travail bien supérieure, Visual Basic prend un peu de temps pour remettre à jour l'objet zone de texte et remplir, trier et inverser de nouveau *TableauAléatoire*. Toutefois, la barre de progression vous tient au courant et vous pouvez voir qu'avec un seul changement mineur, vous pouvez transposer ce que vous avez appris dans ce chapitre à d'autres situations. Le secret consiste à utiliser la fonction *UBound* pour transmettre la taille du tableau aux procédures événementielles du programme plutôt que de « coder en dur » la limite supérieure à 499.

Vous pouvez vous entraîner à ajouter une instruction *Randomize* à la procédure événementielle *Form1_Load* (pour que les résultats soient réellement aléatoires à chaque exécution du programme) ou à tester d'autres tailles et types de tableaux. (Essayez avec une taille de 100, 800, 2 000 ou 5 000 éléments, par exemple). Avec des nombres plus élevés, vous allez peut-être dépasser la quantité de données que l'objet zone de texte peut afficher, mais il en faudrait bien plus pour dépasser la taille de tableau maximum autorisée par Visual Basic.

Pour se concentrer sur les opérations de tableau sans afficher les résultats, placez un caractère de commentaire (') avant chaque ligne de code qui manipule un objet zone de texte pour « commenter » les portions de zone de texte (mais pas la barre de progression) du programme. Vous serez surpris par la rapidité d'exécution des opérations sur les tableaux lorsque les résultats n'ont pas besoin de s'afficher sur le formulaire. (Un tableau de 100 000 éléments se charge en quelques secondes).

Rappel du chapitre 11

Pour	Faites ceci
Créer un tableau	Dimensionnez le tableau grâce au mot clé *Dim*. Par exemple : `Dim Employés(9) As String`
Créer un tableau public	Dimensionnez le tableau grâce au mot clé *Public* dans un module. Par exemple : `Public Employés(9) As String`
Créer un tableau public en spécifiant des limites supérieures et inférieures	Dimensionnez le tableau comme précédemment en utilisant aussi le mot clé *To*. Par exemple : `Public Employés(0 To 9) As String` Remarque : la limite inférieure du tableau doit toujours être zéro (0). Par conséquent, cette syntaxe est destinée essentiellement à une bonne lisibilité du code (et n'est pas prise en charge dans Visual Basic .NET 2002 et 2003).
Assigner une valeur à un tableau	Spécifiez le nom du tableau, l'index de l'élément de tableau et la valeur. Par exemple : `Employés(5) = "Leslie"`
Formater des chaînes textuelles avec des retours chariot et des tabulations	Utilisez les constantes *vbCrLf* et *vbTab* dans votre code. Pour ajouter et tabuler ces valeurs dans des chaînes, utilisez l'opérateur (*&*).
Créer un tableau dynamique	Spécifiez le nom et le type du tableau à la conception, en omettant le nombre d'éléments. Si le tableau possède plusieurs dimensions, insérez des virgules sans chiffre entre les dimensions. Pendant l'exécution du programme, spécifiez la taille du tableau en utilisant l'instruction *ReDim*. Par exemple : `ReDim Températures(10)`

Traiter les éléments dans un tableau	Rédigez une boucle *For…Next* qui exploite la variable compteur de boucle pour traiter chaque élément du tableau. Par exemple :
	`Dim i As Short Dim Total As Single`
	`For i = 0 To UBound(Températures)`
	`Total = Total + Températures(i) Next`

Redimensionner un tableau en préservant ses données	Utilisez le mot clé *Preserve* dans votre instruction *ReDim*. Par exemple : `ReDim Preserve monCube(25, 25, 50)`

Réordonner le contenu d'un tableau	Utilisez les méthodes de la classe *Array* du .NET Framework. Pour trier un tableau appelé *TableauAléatoire* par ordre croissant, utilisez la méthode *Array.Sort* comme suit :
	`Array.Sort(TableauAléatoire)`
	Pour inverser l'ordre d'un tableau appelé *TableauAléatoire*, utilisez la méthode *Array.Reverse* comme suit :
	`Array.Reverse(TableauAléatoire)`

Pour offrir à l'utilisateur un retour d'information visuel pendant des calculs de longue durée	Ajoutez un contrôle *ProgressBar* à votre formulaire. Vous le trouverez sur l'onglet Contrôles communs de la Boîte à outils. Définissez les propriétés *Minimum*, *Maximum* et *Value* du contrôle en utilisant le code. La variable compteur d'une boucle *For…Next* est souvent un bon moyen de définir la propriété *Value*.

Chapitre 12

Travailler avec les collections et l'espace de noms *System.Collections*

À la fin de ce chapitre, vous saurez :

- Manipuler la collection *Controls* dans le cadre d'un formulaire
- Utiliser une boucle *For Each...Next* pour parcourir les objets d'une collection
- Créer vos propres collections pour gérer des URL de site web et d'autres informations
- Utiliser des collections VBA dans Microsoft Office

Dans ce chapitre, vous allez apprendre à utiliser des groupes d'objets appelés *collections* dans un programme Visual Basic. Vous verrez comment gérer des informations avec des collections, traiter des objets collection grâce à des boucles *For Each...Next* et explorer de nouveaux objets dans l'espace de noms *System.Collections*. En combinant vos compétences en matière de traitement des collections et ce que vous avez appris sur les tableaux au chapitre 11 « Utiliser les tableaux pour gérer les données numériques et les chaînes », vous en saurez assez pour gérer efficacement des données dans un programme, et vous aurez pris un bon départ quant à la manipulation des collections d'objets exposées par Microsoft Visual Studio 2008 et les applications Windows classiques.

Travailler avec les collections d'objets

Dans cette section, vous allez découvrir les collections, un mécanisme puissant destiné à contrôler les objets et d'autres données dans un programme Visual Basic. Vous savez déjà que les objets d'un formulaire sont stockés dans le même fichier. Mais savez-vous que Visual Basic considère ces objets comme appartenant au même groupe ? Dans la terminologie Visual Studio, le jeu complet des objets d'un formulaire s'appelle la *collection Controls*, qui appartient à l'espace de noms *System.Collections* du .NET Framework. La collection *Controls* est créée automatiquement à l'ouverture d'un nouveau formulaire. Si vous y ajoutez des objets, ils viennent grossir cette collection. En outre, Visual Studio maintient plusieurs collections d'objets standards que vous pouvez utiliser lors de la rédaction de vos programmes. Le reste de ce chapitre vous apportera les connaissances dont vous avez besoin pour travailler avec les collections.

Dans un programme, chaque collection possède son propre nom afin que vous puissiez la référencer comme une unité distincte dans le code. Par exemple, comme vous venez de l'apprendre, la collection contenant tous les objets d'un formulaire est la collection *Controls*. Cette méthode de regroupement ressemble à la manière dont les tableaux regrou-

pent une liste d'éléments sous un même nom. À l'instar des tableaux Visual Basic, la collection *Controls* est fondée sur zéro.

Si un projet contient plusieurs formulaires, vous pouvez créer des variables publiques associées aux noms des formulaires et les exploiter pour différencier une collection *Controls* d'une autre. Vous apprendrez comment exploiter les variables publiques pour stocker des données de formulaire au chapitre 14 « Gérer les formulaires et les contrôles Windows à l'exécution ». Il est même possible d'ajouter, par programmation, des contrôles à la collection *Controls* dans un formulaire.

Outre le travail avec les collections et les objets dans vos propres programmes, Visual Studio permet de parcourir votre système pour rechercher et exploiter d'autres objets application.

Référencer des objets dans une collection

Pour référencer des objets dans une collection, ou des membres de la collection isolément, spécifiez la position d'index de l'objet dans le groupe. Visual Basic stocke des objets collection dans l'ordre inverse à celui de leur création. Vous pouvez donc utiliser « l'ordre de création » d'un objet pour référencer individuellement l'objet, ou exploiter une boucle pour parcourir plusieurs objets. Par exemple, pour identifier le dernier objet créé sur un formulaire, spécifiez l'index 0 (zéro), comme dans cet exemple :

```
Controls(0).Text = "Société"
```

Cette instruction définit la propriété *Text* du dernier objet du formulaire « Société ». L'avant-dernier objet créé possède un index de 1, le précédent un index de 2, et ainsi de suite. Suivant cette logique, il est important de ne pas toujours associer un objet particulier du formulaire à une valeur d'index car tout nouvel objet ajouté à la collection prend l'emplacement d'index 0 et les autres index sont incrémentés de 1.

La boucle *For…Next* suivante exploite une boîte de message pour afficher les noms des quatre derniers contrôles ajoutés au formulaire.

```
Dim i As Integer
For i = 0 To 3
    MsgBox(Controls(i).Nom)
Next i
```

Notez que cette boucle parcourt la plage de 0 à 3, car le dernier objet contrôle ajouté au contrôle se trouve à l'emplacement 0. Dans la section à venir, vous allez découvrir une méthode plus efficace pour rédiger une telle boucle.

Développer des boucles For Each…Next

Bien que l'on puisse référencer individuellement les membres d'une collection, la manière la plus utile de travailler avec les objets d'une collection consiste à les traiter en tant que

groupe. En effet, les collections ont été créées pour pouvoir traiter efficacement des groupes d'objets. Par exemple, pour afficher, déplacer, trier, renommer ou redimensionner une collection complète d'objets en une seule étape, vous pouvez utiliser une boucle spéciale appelée *For Each…Next* qui parcourt un à un les objets d'une collection. Une boucle *For Each...Next* ressemble à une boucle *For…Next*. Voici sa configuration dans le cadre d'une collection *Controls* :

```
Dim CtrlVar As Control

For Each CtrlVar In Controls
    traiter objet
Next CtrlVar
```

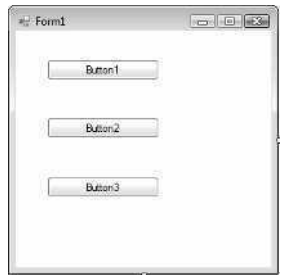
La variable *CtrlVar* est déclarée comme un type *Control* et représente l'objet en cours dans la boucle *For Each…Next*. *Controls* (remarquez le « s » final) est la classe de collections introduite précédemment, qui représente tous les objets contrôle sur le formulaire en cours. Le corps de la boucle sert à traiter les objets individuels de la collection. Par exemple, vous pouvez choisir de modifier les propriétés *Enabled*, *Left*, *Top*, *Text* ou *Visible* des objets de la collection, ou afficher le nom de chaque objet dans une zone de liste.

Exploiter des objets dans la collection *Controls*

Dans les exercices à venir, vous allez utiliser du code pour manipuler des objets sur un formulaire en utilisant la collection *Controls*. Votre projet possèdera trois objets boutons. Vous allez créer des procédures événementielles qui modifient les propriétés *Text* de chaque objet, déplacer des objets vers la droite et accorder un traitement spécial à un objet du groupe. Le programme exploitera trois boucles *For Each...Next* pour manipuler les objets chaque fois que l'utilisateur cliquera sur un des boutons.

Utiliser une boucle *For Each…Next* pour modifier des propriétés *Text*

1. Créez un nouveau projet Visual Basic Application Windows Forms intitulé **Ma Collection Controls**.
2. Utilisez le contrôle *Button* pour dessiner trois objets bouton à gauche de l'objet bouton, puis élargissez-les à environ 124 pixels, comme suit :



3. Utilisez la fenêtre Propriétés pour définir la propriété *Name* du troisième objet bouton (*Button3*) à « btnMoveObjects ».
4. Dans le formulaire, double-cliquez sur le premier objet bouton (*Button1*). La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.
5. Tapez les instructions suivantes :

```
For Each ctrl In Controls
ctrl.Text = "Cliquez sur moi !"
Next
```

Cette boucle *For Each...Next* parcourt la collection *Controls* contrôle par contrôle et définit la propriété *Text* de chaque contrôle à « Cliquez sur moi ! ». La boucle utilise *ctrl* comme variable d'objet dans la boucle, que vous déclarerez à l'étape suivante.

6. Défilez vers le haut du code et tapez le commentaire et la déclaration de variable suivante, juste en dessous de l'instruction `Public Class Form1` :

```
'Déclare une variable de type Control pour représenter les contrôles du formulaire
Dim ctrl As Control
```

Cette déclaration de variable globale crée une variable dans le type de la classe *Control* qui représente les contrôles du formulaire en cours dans le programme. Vous déclarez cette variable dans la zone des déclarations générales du formulaire afin qu'elle soit valide dans toutes les procédures événementielles du formulaire.

Vous voilà prêt à exécuter le programme et à modifier la propriété *Text* de chaque bouton du formulaire.

7. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.
8. Dans le formulaire, cliquez sur le premier bouton (*Button1*).

La procédure événementielle *Button1_Click* modifie la propriété *Text* de chaque contrôle dans la collection *Controls*. Votre formulaire présente un résultat similaire à :

9. Cliquez sur le bouton Fermer du formulaire. Le programme s'interrompt.

Remarque Les changements apportés à la propriété *Text* par le programme n'ont pas été répliqués dans le formulaire au niveau du Concepteur. Les changements effectués à l'exécution n'affectent pas les principaux paramètres de propriété du programme.

10. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap12 comme emplacement.

Vous voilà prêt à tester différemment la collection *Controls*. Vous allez utiliser la propriété *Left* pour déplacer chaque contrôle de la collection *Controls* vers la droite.

Utiliser une boucle *For Each...Next* pour déplacer des contrôles

1. Affichez le formulaire, puis double-cliquez sur le deuxième objet bouton (*Button2*).
2. Tapez le code suivant dans la procédure événementielle *Button2_Click* :

```
For Each ctrl In Controls
     ctrl.Left = ctrl.Left + 25
Next
```

Chaque fois que l'utilisateur clique sur le deuxième bouton, cette boucle *For Each...Next* parcourt les objets de la collection *Controls* un à un et les déplace de 25 pixels vers la droite (pour déplacer des objets de 25 pixels vers la gauche, il suffit de soustraire 25). Un *pixel* est une unité de mesure indépendante de tout périphérique qui permet de positionner avec précision des objets sur un formulaire.

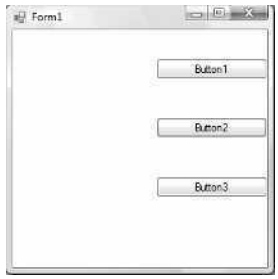
Astuce Dans Visual Basic 6, on utilise normalement des twips et non des pixels comme unité de mesure. Un twip est une unité typographique équivalente à un vingtième de point.

Comme dans la précédente procédure événementielle, la variable *ctrl* est une « doublure » de l'objet en cours dans la collection, qui contient les mêmes paramètres de propriété que l'objet qu'elle représente. Dans cette boucle, vous réglez la propriété *Left*, qui détermine l'emplacement d'un objet par rapport au côté gauche du formulaire.

3. Cliquez sur le bouton Démarrer le débogage. Le programme s'exécute et les trois boutons s'affichent à gauche du formulaire.

4. Cliquez sur le deuxième bouton à plusieurs reprises.

   Chaque fois que vous cliquez sur le bouton, les objets du formulaire se déplacent progressivement vers la droite. Voici à quoi ressemble votre écran après cinq clics :



5. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.
6. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

   Il ne sera pas toujours indiqué de déplacer tous les objets d'un formulaire comme un groupe. Avec Visual Basic, vous pouvez traiter individuellement les membres d'une collection. Dans le prochain exercice, vous allez voir comment maintenir le troisième objet bouton à sa place tout en déplaçant les deux autres boutons vers la droite.

Exploiter la propriété *Name* dans une boucle *For Each...Next*

Pour traiter différemment un ou plusieurs membres d'une collection, utilisez la propriété *Name*, qui identifie de manière unique chaque objet du formulaire. Dans ce livre, vous avez régulièrement défini cette propriété pour améliorer la lisibilité du code. On peut également l'utiliser par programmation pour identifier des objets spécifiques dans un programme.

Pour ce faire, choisissez les objets qui subiront un traitement spécial, puis notez leurs propriétés *Name*. Lorsque par la suite, vous parcourrez les objets du formulaire en utilisant une boucle *For Each...Next*, utilisez une ou plusieurs instructions *If* pour tester les propriétés *Name* importantes et gérer ces objets différemment. Par exemple, supposons que vous souhaitiez construire une boucle *For Each...Next* qui déplace un objet plus lentement sur le formulaire que les autres objets. Utilisez une instruction *If...Then* pour repérer la propriété *Name* de l'objet le plus lent, puis déplacez cet objet d'une distance plus courte en incrémentant moins sa propriété *Left* que pour les autres objets.

Astuce Pour accorder à plusieurs objets un traitement spécial dans une boucle *For Each...Next*, utilisez des instructions *ElseIf* avec l'instruction *If…Then* ou servez-vous d'une structure de décision *Select Case*.

Dans l'exercice suivant, vous allez tester la propriété *Name* du troisième objet bouton (*btnMoveObjects*) pour accorder un traitement spécial à ce bouton dans une boucle *For Each...Next*. Il en résultera une procédure événementielle qui déplacera les deux boutons supérieurs vers la droite et maintiendra le bouton du bas à un emplacement fixe.

Astuce Outre la propriété *Name*, la plupart des objets prennent en charge la propriété *Tag*. Au même titre que la propriété *Name*, il s'agit d'un emplacement dans lequel vous pouvez stocker des données chaîne à propos de l'objet. Par défaut, la propriété *Tag* est vide mais vous pouvez lui assigner des informations et la tester pour identifier de manière unique les objets de votre programme que vous souhaitez traiter différemment.

Utiliser la propriété *Name* pour traiter un objet de la collection *Controls*

1. Affichez le formulaire et double-cliquez sur le troisième objet bouton.

   La procédure événementielle *btnMoveObjects_Click* s'affiche dans l'Éditeur de code. Souvenez-vous que, dans l'exercice précédent, vous avez modifié la propriété *Name* de cet objet en remplaçant « *Button1* » par « *btnMoveObjects* ».

2. Tapez le code suivant dans la procédure événementielle :

```
For Each ctrl In Controls
     If ctrl.Name <> "btnMoveObjects" Then
         ctrl.Left = ctrl.Left + 25
     End If
Next
```

   La nouvelle fonctionnalité de cette boucle *For Each...Next* est l'instruction *If…Then*, qui vérifie si chaque membre de la collection possède une propriété *Name* appelée « *btnMoveObjects* ». Si la boucle rencontre ce marqueur, elle passe l'objet sans le déplacer. Notez que, comme dans les exemples précédents, la variable *ctrl* a été déclarée en haut du formulaire en tant que variable de type *Control* avec une portée sur l'ensemble du formulaire.

3. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

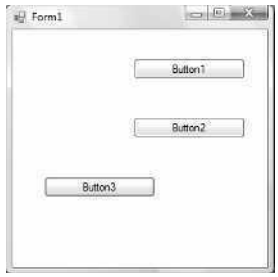
Astuce Le programme Collection Controls complet se trouve dans le dossier c:\vb08epe\chap12\Collection Controls.

4. Cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute et les trois objets bouton s'affichent sur le formulaire.

5. Cliquez trois ou quatre fois sur le troisième objet bouton.

   Au fur et à mesure que vous cliquez, les deux objets du haut se déplacent sur l'écran. Toutefois, le troisième bouton conserve sa place, comme ci-après :



6. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Il peut s'avérer très utile d'accorder un traitement spécial à un objet d'une collection. Dans ce cas, l'utilisation de la propriété *Name* dans la boucle *For Each...Next* a amélioré la lisibilité du code, et suggère de nombreuses applications potentielles pour un programme de jeu ou de traitement graphique. Lorsque vous utiliserez d'autres types de collections dans Visual Basic, gardez bien à l'esprit cette propriété.

Créer vos propres collections

Visual Basic permet également de créer vos propres collections pour suivre les données d'un programme et les manipuler de manière méthodique. Bien que les collections servent souvent à contenir des objets, comme des contrôles de l'interface, il est également possible de les utiliser pour stocker des valeurs numériques ou de chaîne pendant l'exécution d'un programme. De cette manière, les collections viennent parfaitement compléter les capacités des tableaux, abordées au chapitre précédent.

Déclarer de nouvelles collections

Les nouvelles collections sont déclarées comme des variables dans un programme et l'emplacement où vous les déclarez détermine leur portée, c'est-à-dire l'étendue sur laquelle les valeurs qui leur sont assignées vont persister. L'utilité des collections est telle que je les déclare habituellement en haut des formulaires ou dans un module.

Voici la syntaxe pour déclarer une nouvelle collection

```
Dim NomCollection As New Collection()
```

où *NomCollection* correspond au nom de votre collection. Si vous placez la déclaration de collection dans un module, utilisez le mot clé *Public* au lieu de *Dim*. Après avoir créé une collection, ajoutez-lui des membres en utilisant la méthode *Add*. Vous pouvez examiner chaque membre en utilisant une boucle *For Each...Next*.

L'exercice suivant montre comment créer une collection contenant des données chaîne qui représentent les adresses Internet (URL, ou *Uniform Resource Locators*) récemment utilisées lors de votre dernier passage sur le web. Pour se connecter au web, le programme utilisera la méthode Visual Basic *System.Diagnostics.Process.Start* et votre Navigateur Web par défaut, une technique que j'ai déjà introduite au chapitre 3 « Travailler avec les contrôles de la Boîte à outils ».

Suivre des URL en utilisant une nouvelle collection

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet.
2. Créez un nouveau projet Application Windows Forms appelé **Ma Collection URL**.
3. Dessinez un grand objet zone de texte centré en haut du formulaire.
4. Dessinez l'un en dessous de l'autre deux larges objets bouton sous l'objet zone de texte.
5. Définissez les propriétés suivantes pour le formulaire et ses objets :

Objet	Propriété	Paramètre
TextBox1	*Text*	« http://www.microsoft.com/learning/books/ »
Button1	*Text*	« Visiter le site »
Button2	*Text*	« Afficher les sites récents »
Form1	*Text*	« Collection URL »

Votre formulaire présente un résultat similaire à

6. Dans l'Explorateur de solutions, cliquez sur le bouton Afficher le code pour afficher l'Éditeur de code.

7. Placez le point d'insertion en haut du code du formulaire, tapez la déclaration de variable suivante juste en dessous de l'instruction `Public Class Form1` et appuyez sur ENTRÉE :

```
Dim URLVisitées As New Collection()
```

Cette instruction crée une nouvelle collection et lui attribue le nom de variable *URLVisitées*. Comme vous avez placé la déclaration dans la zone de déclaration du formulaire, la portée de la collection s'étend à l'ensemble des procédures événementielles du formulaire.

8. Affichez le formulaire, double-cliquez sur le bouton « Visiter le site » et tapez le code suivant dans la procédure événementielle *Button1_Click* :

```
URLVisitées.Add(TextBox1.Text)
System.Diagnostics.Process.Start(TextBox1.Text)
```

Ce code utilise la méthode *Add* pour remplir la collection de membres. Lorsque l'utilisateur clique sur l'objet *Button1*, le programme suppose qu'une adresse Internet valide a été placée dans l'objet *TextBox1*. Chaque fois que l'on clique sur l'objet *Button1*, l'URL en cours dans *TextBox1* est copiée dans la collection *URLVisitées* sous forme de chaîne. Ensuite, la méthode *System.Diagnostics.Process.Start* est appelée avec l'URL comme paramètre. Comme le paramètre est une URL, la méthode *Start* tente de l'ouvrir en utilisant le Navigateur Web par défaut du système. Si l'URL est invalide ou que la connexion Internet n'a pu être établie, le Navigateur Web gère l'erreur.

Remarque Les seules URL ajoutées par le programme à la collection *URLVisitées* sont celles que vous avez spécifiées dans l'objet *TextBox1*. Si vous naviguez vers d'autres sites web en utilisant le Navigateur Web, ces sites ne seront pas ajoutés à la collection.

9. Affichez de nouveau le formulaire puis double-cliquez sur le bouton Afficher les sites récents.

10. Tapez le code suivant en utilisant l'Éditeur de code :

```
Dim NomURL As String = "", ToutesURL As String = ""
For Each NomURL In URLVisitées
     ToutesURL = ToutesURL & NomURL & vbCrLf
Next NomURL
MsgBox(ToutesURL, MsgBoxStyle.Information, "Sites web visités")
```

Cette procédure événementielle affiche la collection complète en utilisant une boucle *For Each...Next* et une fonction *MsgBox*. Cette routine déclare une variable chaîne appelée *NomURL* pour héberger chaque membre de la collection au fur et à mesure qu'ils sont traités et initialise la variable à vide (""). Cette valeur est ajoutée à une chaîne appelée *ToutesURL* grâce à l'opérateur de concaténation (*&*) et la constante *vbCrLf* sert à placer chaque URL sur sa propre ligne.

Enfin, la chaîne *ToutesURL*, qui représente l'ensemble du contenu de la collection *URLVisitées*, s'affiche dans une boîte de message. J'ai ajouté l'argument *MsgBoxStyle.Information* dans la fonction *MsgBox* pour mettre en évidence que le texte à afficher est une information générale et non un avertissement. *MsgBoxStyle.Information* est également une constante Visual Basic prédéfinie.

11. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap12 comme emplacement.

Remarque Pour exécuter le programme Collection URL, votre ordinateur doit établir une connexion Internet et être équipé d'un Navigateur Web, comme Internet Explorer.

Exécuter le programme Collection URL

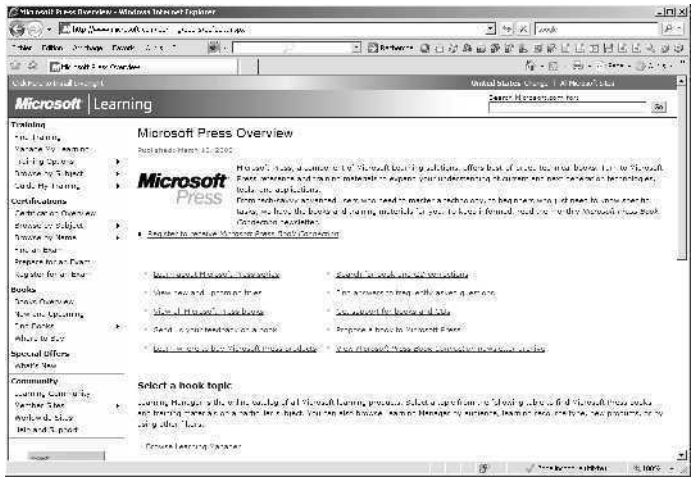
Astuce Le programme Collection URL complet se trouve à l'adresse c:\vb08epe\chap12\Collection URL.

1. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

Le programme affiche un site web par défaut dans la zone d'URL. Il est donc inutile de taper d'emblée votre propre adresse Internet.

2. Cliquez sur le bouton Visiter le site.

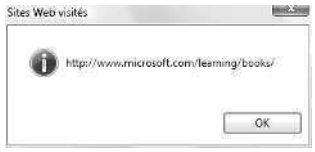
Visual Basic ajoute le site web de Microsoft Press (*http://www.microsoft.com/learning/books/*) à la collection *URLVisitées*, ouvre le Navigateur Web par défaut de votre système et charge la page web demandée, comme ci-après.



3. Cliquez de nouveau sur le formulaire (vous aurez peut-être à cliquer sur l'icône du formulaire dans la barre des tâches Windows).

4. Cliquez sur le bouton Afficher les sites récents.

Visual Basic exécute la procédure événementielle de l'objet *Button2*. Une boîte de message s'affiche :

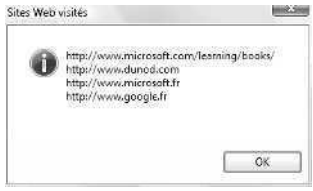


5. Cliquez sur OK dans la boîte de message, tapez un site web différent dans la zone de texte du formulaire, puis cliquez sur le bouton Visiter le site.

   Pour en savoir plus sur Visual Basic, vous pourriez visiter le site Microsoft Visual Basic Developer Center à l'adresse *http://msdn.microsoft.com/vbasic/*.

6. Visitez d'autres sites web en utilisant le formulaire Collection URL, puis cliquez sur le bouton Afficher les sites récents.

   Chaque fois que vous cliquez sur ce bouton, la fonction *MsgBox* s'étend pour afficher l'historique croissant des URL, comme suit :



   Si vous visitez plus d'une dizaine de sites web, vous devrez remplacer la fonction *MsgBox* par une zone de texte multiligne sur le formulaire. Mais vous savez déjà écrire le code correspondant…

7. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire, puis fermez votre Navigateur Web.

Félicitations ! Vous venez d'apprendre à utiliser la collection *Controls* et à gérer les collections en utilisant une boucle *For Each...Next*. Ces compétences vous seront utiles dès que vous travaillerez avec des collections dans l'espace de noms *System.Collections*. Lorsque vous vous serez familiarisé avec les structures de données informatiques classiques et les algorithmes liés à la gestion des listes (les piles, les files d'attente, les dictionnaires, les tables de hachage et d'autres listes structurées), vous découvrirez que *System.Collections* offre à Visual Studio des équivalents pour gérer des informations de manière extrêmement innovante. Vous trouverez quelques idées de livres traitant des structures de données et des algorithmes dans l'Annexe « Où trouver d'autres informations », dans la section « Ouvrages généraux sur la programmation et l'informatique ».

Aller plus loin : collections VBA

Si vous prévoyez d'écrire des macros Visual Basic pour des applications Microsoft Office, vous découvrirez que les collections jouent un rôle très important dans les modèles objets de Microsoft Word, Excel, Access, PowerPoint et de nombreuses applications autres qui prennent en charge le langage de programmation Visual Basic pour Applications (VBA). Par exemple, dans Word, tous les documents ouverts sont stockés dans la collection *Documents* et chaque paragraphe du document en cours est stocké dans la collection *Paragraphs*. Vous pouvez manipuler ces collections au moyen d'une boucle *For Each...Next* exactement comme pour les collections des exercices précédents. Office 2003 et le système Microsoft Office 2007 procurent une large base d'installation pour des solutions fondées sur VBA.

Astuce En tant que développeur logiciel, vous devez être conscient de ce que chacun ne possède pas encore le système Office 2007. Vous devrez parfois proposer des solutions VBA pour plusieurs versions d'Office, car une entreprise typique a recours simultanément à plusieurs versions d'Office.

Le prochain exemple de code provient d'une macro VBA Word qui emploie une boucle *For Each...Next* pour rechercher chaque document ouvert dans la collection *Documents* pour un fichier appelé MaLettre.doc. Si l'on trouve ce fichier dans la collection, la macro l'enregistre grâce à la méthode *Save*. Dans le cas contraire, la macro tente d'ouvrir le fichier depuis le dossier c:\vb08epe\chap12.

```
Dim aDoc As Document
Dim docFound As Boolean
Dim docLocation As String
docFound = False
docLocation = "c:\vb08epe\chap12\MaLettre.doc"
For Each aDoc In Documents
    If InStr(1, aDoc.Name, "MaLettre.doc", 1) Then
        docFound = True
        aDoc.Save
        Exit For
    End If
Next aDoc
If docFound = False Then
Documents.Open FileName:=docLocation
End If
```

La macro commence par déclarer trois variables. La variable d'objet *aDoc* représente l'élément de collection en cours dans la boucle *For Each...Next*. La variable booléenne *docFound* assigne une valeur booléenne True si l'on trouve un document dans la collection *Documents*. La variable de chaîne *docLocation* contient le chemin du fichier MaLettre.doc sur le disque (cette routine suppose que le fichier MaLettre.doc se trouve dans vos fichiers d'exercices, dans le dossier c:\vb08epe\chap12).

La boucle *For Each...Next* parcourt chaque document dans la collection *Documents* à la recherche du fichier MaLettre. Si le fichier est détecté par la fonction *IntStr* (qui détecte

une chaîne dans une autre), le fichier est enregistré. Dans le cas contraire, la macro tente de l'ouvrir en utilisant la méthode *Open* de l'objet *Documents*.

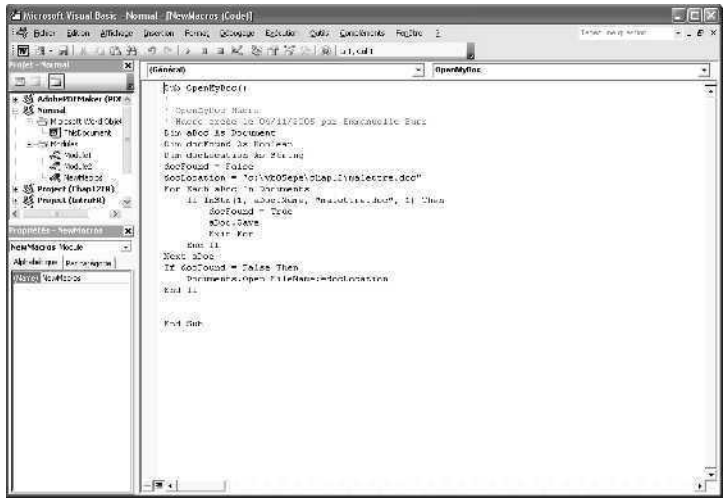
Notez également l'instruction *Exit For* que j'utilise pour quitter la boucle *For Each...Next* une fois que le fichier MaLettre a été trouvé et enregistré. *Exit For* est une instruction spéciale que vous pouvez utiliser pour quitter une boucle *For...Next* ou une boucle *For Each...Next* dont la poursuite de l'exécution pourrait entraîner des résultats inattendus. Dans cet exemple, si le fichier MaLettre.doc se trouve dans la collection, la poursuite de la recherche est infructueuse et l'instruction *Exit For* propose un moyen élégant d'interrompre la boucle dès qu'elle a accompli sa tâche.

Bienvenue dans les macros Word

J'ai inséré cet exemple de macro Word pour vous montrer comment utiliser des collections dans Visual Basic pour Applications, mais le code source est conçu pour Word et non pour l'EDI de Visual Studio. Si vous ne travaillez pas sous Word, la collection *Documents* n'aura aucune signification pour le compilateur.

Les étapes à suivre pour tester la macro dépendent de la version de Word que vous employez. Avec Word 2003, démarrez Word. Dans le menu Outils, cliquez sur la commande Macros du sous-menu Macro, donnez un nouveau nom de macro (j'ai utilisé OuvrirMonDoc), puis entrez dans le code *via* l'éditeur Visual Basic de Word. Avec Word 2007, démarrez Word, cliquez sur l'onglet Développeur, cliquez sur la commande Macros, créez un nouveau nom pour la macro puis saisissez le code à l'aide de l'Éditeur Visual Basic. Si l'onglet Développeur n'est pas visible, vous devez l'activer dans la boîte de dialogue Options Word.

Dans l'Éditeur Visual Basic, la macro achevée ressemble à la fenêtre Word suivante. Pour l'exécuter, cliquez sur le bouton Exécuter Sub/UserForm de la barre d'outils, comme si vous exécutiez un programme dans l'EDI de Visual Studio.



Astuce Les macros Word sont généralement compatibles entre versions, même si la mise à niveau de macros VBA ou la prise en charge de plusieurs versions d'Office peut parfois poser quelques problèmes. Si vous avez employé une version différente de Word, il pourrait être nécessaire de modifier légèrement l'exemple de code montré ici.

Rappel du chapitre 12

Pour	Faites ceci

Traiter les objets d'une collection	Écrivez une boucle *For Each...Next* qui traite individuellement chaque membre de la collection. Par exemple :
	`Dim ctrl As Control`
	`For Each ctrl In Controls`
	`ctrl.Text = "Cliquez sur moi !"`
	`Next`

Déplacer des objets dans la collection *Controls* de gauche à droite à l'écran	Modifiez la propriété *Contrl.Left* de chaque objet de la collection dans une boucle *For Each...Next*. Par exemple :
	`Dim ctrl As Control`
	`For Each ctrl In Controls`
	`Ctrl.Left = Ctrl.Left + 25`
	`Next Ctrl`

Accorder un traitement spécial à un objet dans une collection	Testez la propriété *Name* des objets de la collection en utilisant une boucle *For Each...Next*. Par exemple :
	`Dim ctrl As Control`
	`For Each ctrl In Controls`
	`If ctrl.Name <> "btnMoveObjects" Then`
	`ctrl.Left = ctrl.Left + 25`
	`End If`
	`Next`

Créer une nouvelle collection et y ajouter des membres	Déclarez une variable en utilisant la syntaxe *New Collection*. Utilisez la méthode *Add* pour ajouter des membres. Par exemple :
	`Dim URLVisitées As New Collection()`
	`URLVisitées.Add(TextBox1.Text)`

Utiliser des collections Visual Basic pour Applications dans Word	Si vous employez Word 2003, démarrez le programme, cliquez sur la commande Macros du sous-menu Macro, donnez un nom à la macro, cliquez sur Créer, puis saisissez un code de macro VBA en utilisant l'éditeur Visual Basic. Avec Word 2007, démarrez Word, cliquez sur l'onglet Développeur, cliquez sur la commande Macros, créez un nouveau nom pour la macro puis saisissez le code à l'aide de l'Éditeur Visual Basic. Word expose de nombreuses collections utiles, dont *Documents* et *Paragraphs*.

Chapitre 13

Explorer le traitement des fichiers texte et des chaînes

À la fin de ce chapitre, vous saurez :

- Afficher un fichier texte en utilisant un objet zone de texte, la fonction *LineInput* et la classe *StreamReader*
- Utiliser l'objet *My*, une nouvelle fonctionnalité « d'appel rapide » de Microsoft Visual Studio 2008
- Enregistrer des notes dans un fichier texte en utilisant la fonction *PrintLine* et le contrôle *SaveFileDialog*
- Utiliser des techniques de traitement de chaînes pour comparer, combiner et trier des chaînes

La gestion des documents numériques est une fonction cruciale pour les entreprises modernes. Microsoft Visual Basic 2008 propose de nombreux mécanismes pour travailler avec plusieurs types de document différents et manipuler des informations dans des documents. Le *fichier texte* est le type de document le plus rudimentaire. Il est composé de mots, de paragraphes, de lettres, de chiffres et de plusieurs caractères et symboles spéciaux.

Dans ce chapitre, vous allez apprendre à travailler avec les informations stockées dans des fichiers texte sur votre système. Vous verrez comment ouvrir un fichier texte, afficher son contenu dans un objet zone de texte et créer un nouveau fichier texte sur le disque. Vous approfondirez également vos connaissances en matière de gestion des chaînes dans vos programmes et utiliserez des méthodes des classes *String* et *StreamReader* du Microsoft .NET Framework pour combiner, trier et afficher des mots, des lignes et des fichiers texte entiers.

Afficher des fichiers texte grâce à un objet zone de texte

La manière la plus simple d'afficher un fichier texte dans un programme consiste à utiliser un objet zone de texte. Comme nous l'avons vu, vous pouvez créer des objets zone de texte de toute taille. Si le contenu du fichier texte excède la capacité de la zone de texte, vous pouvez ajouter des barres de défilement afin que l'utilisateur puisse avoir accès à l'ensemble du fichier. Pour utiliser Visual Basic dans le but de charger le contenu d'un fichier texte dans une zone de texte, il vous faut recourir à quatre fonctions. Celles-ci sont décrites dans le tableau qui suit et sont employées dans le premier exercice de ce chapitre. Comme je l'ai dit précédemment, plusieurs de ces fonctions remplacent des mots clés plus anciens du langage Visual Basic.

Fonction	Objet
FileOpen	Ouvre un fichier texte pour y ajouter des entrées ou afficher des résultats.
LineInput	Lit une ligne d'entrées depuis le fichier texte.
EOF	Vérifie la fin du fichier texte.
FileClose	Ferme le fichier texte.

Ouvrir un fichier pour y ajouter des entrées

Un *fichier texte* est composé d'une ou de plusieurs lignes de chiffres, de mots ou de caractères. Les fichiers texte sont différents des *fichiers de documents* et des *pages web*, qui contiennent des codes de mise en forme, ainsi que des *fichiers exécutables*, qui contiennent des instructions destinées au système d'exploitation. Sur votre système, les fichiers texte classiques sont identifiés par l'Explorateur Windows comme des « documents texte » et portent l'extension .txt, .ini, .log ou .inf. Comme les fichiers texte contiennent uniquement des caractères ordinaires reconnaissables, vous pouvez aisément les afficher grâce à des objets zone de texte.

En utilisant un contrôle *OpenFileDialog* pour inviter l'utilisateur à saisir le chemin d'un fichier, vous autorisez à l'utilisateur à choisir le fichier texte qu'il souhaite ouvrir dans un programme. Ce contrôle contient la propriété *Filter*, qui contrôle le type de fichier affiché, la méthode *ShowDialog*, qui affiche la boîte de dialogue Ouvrir, et la propriété *FileName*, qui retourne le chemin spécifié par l'utilisateur. Le contrôle *OpenFileDialog* n'ouvre pas le fichier, mais se contente d'en obtenir le chemin.

La fonction *FileOpen*

Après avoir obtenu le chemin de l'utilisateur, vous ouvrez le fichier dans le programme en utilisant la fonction *FileOpen*. Voici la syntaxe abrégée de cette fonction.

```
FileOpen(numérofichier, nomchemin, mode)
```

Vous trouverez la liste complète des arguments dans la documentation de Visual Basic. Voici les plus importants :

- *numérofichier* est un entier compris entre 1 et 255.
- *nomchemin* est un chemin Microsoft Windows valide.
- *mode* est un mot clé indiquant comment utiliser le fichier. Dans ce chapitre, vous allez utiliser les modes *OpenMode.Input* et *OpenMode.Output*.

Le numéro de fichier est associé au fichier lors de son ouverture. Vous emploierez ensuite ce numéro de fichier dans votre code dès que vous aurez besoin de faire référence au fichier ouvert. Hormis cette association, il ne reste rien de particulier à souligner ; Visual Basic utilise les numéros de fichier pour suivre les différents fichiers ouverts dans votre programme.

Voici comment se présente une fonction *FileOpen* classique qui utilise un objet *OpenFileDialog* :

```
FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
```

Ici, la propriété *OpenFileDialog1.FileName* représente le chemin, *OpenMode.Input* est le mode et 1 correspond au numéro du fichier.

Astuce Les fichiers texte ouverts avec cette syntaxe sont appelés *fichiers séquentiels* car vous devez travailler avec leur contenu de manière séquentielle. À l'opposé, vous pouvez accéder aux informations d'une base de données dans n'importe quel ordre. Vous en saurez plus sur les bases de données au chapitre 18 « Démarrer avec ADO.NET ».

L'exercice qui suit montre comment utiliser le contrôle *OpenFileDialog* et la fonction *FileOpen* pour ouvrir un fichier texte. Vous verrez également comment exploiter les fonctions *LineInput* et *EOF* pour afficher le contenu d'un fichier texte dans une zone de texte ainsi que la fonction *FileClose* pour fermer un fichier. Pour plus d'informations sur l'utilisation des contrôles sur l'onglet Boîtes de dialogue de la Boîte à outils pour créer des boîtes de dialogue standards, reportez-vous au chapitre 4 « Travailler avec les menus, les barres d'outils et les boîtes de dialogue ».

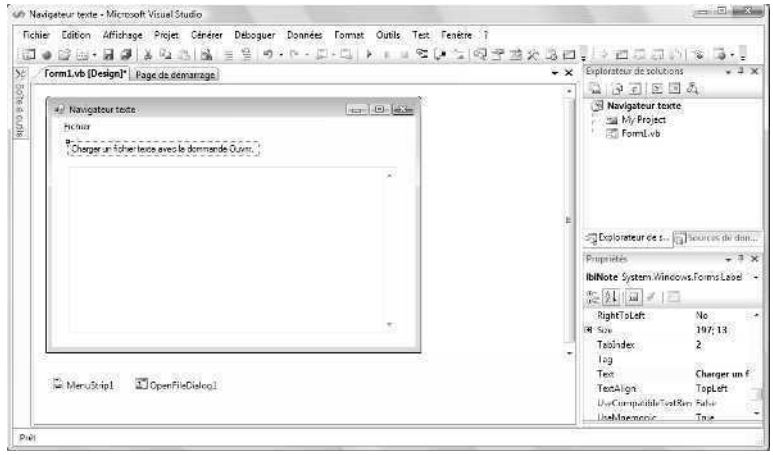
Exécuter le programme Navigateur texte

1. Démarrez Microsoft Visual Studio et ouvrez le projet Navigateur texte qui se trouve dans le dossier c:\vb08epe\chap13\Navigateur texte.

   Le programme s'exécute dans l'environnement de développement.

2. Affichez le formulaire si ce n'est déjà fait.

   Le formulaire Navigateur texte s'affiche, comme suit :



Le formulaire comprend un grand objet zone de texte équipé de barres de défilement. Il contient également un objet menu déroulant qui place les commandes Ouvrir, Fermer et Quitter dans le menu Fichier, un objet *Openfiledialog*, et une étiquette qui propose des instructions de fonctionnement. J'ai également créé les paramètres de propriétés présentés dans le tableau suivant (notez en particulier les paramètres de la zone de texte) :

Objet	Propriété	Paramètre
txtNote	*Enabled*	False
	Multiline	True
	Name	txtNote
	ScrollBars	Both
CloseToolStripMenuItem	*Enabled*	False
lblNote	*Text*	« Charger un fichier texte avec la commande Ouvrir »
	Name	lblNote
Form1	*Text*	« Navigateur texte »

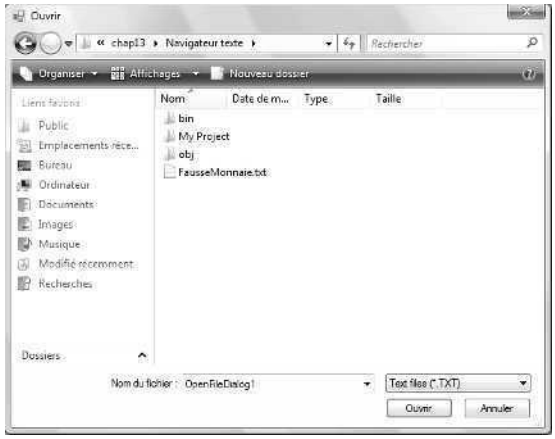
3. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le programme Navigateur texte s'exécute.

4. Dans le menu Fichier du programme Navigateur texte, cliquez sur la commande Ouvrir.

La boîte de dialogue Ouvrir s'affiche.

5. Ouvrez le dossier c:\vb08epe\chap13\Navigateur texte. Voici son contenu :



6. Double-cliquez sur le fichier FausseMonnaie.txt.

   FausseMonnaie, un fichier texte contenant un article écrit en 1951 au États-Unis sur les dangers de la fausse monnaie, s'affiche dans la zone de texte, comme suit :

7. Utilisez les barres de défilement pour parcourir l'ensemble du document. Retenez le numéro 5.
8. Une fois que vous avez terminé, cliquez sur la commande Fermer du menu Fichier pour fermer le fichier, puis sur la commande Quitter pour quitter le programme.

   Le programme s'arrête et vous revenez à l'environnement de développement.

Vous allez maintenant observer deux procédures événementielles importantes de ce programme.

Examiner le code du programme Navigateur texte

1. Dans le menu Fichier du formulaire Navigateur texte, double-cliquez sur la commande Ouvrir.

   La procédure événementielle *OpenToolStripMenuItem_Click* s'affiche dans l'Éditeur de code.
2. Redimensionnez l'Éditeur de code si nécessaire.

Voici le code de la procédure événementielle *OpenToolStripMenuItem_Click* :

```
Dim AllText As String = "", LineOfText As String = ""
OpenFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog() 'affiche la boîte de dialogue Ouvrir
If OpenFileDialog1.FileName <> "" Then
     Try 'ouvre le fichier et piège toutes les erreurs avec un gestionnaire
         FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
         Do Until EOF(1) 'lit les lignes du fichier
             LineOfText = LineInput(1)
             'ajoute chaque ligne à la variable AllText
             AllText = AllText & LineOfText & vbCrLf
         Loop
         lblNote.Text = OpenFileDialog1.FileName 'met à jour l'étiquette
         txtNote.Text = AllText 'affiche le fichier
         txtNote.Enabled = True 'autorise un curseur texte
         CloseToolStripMenuItem.Enabled = True  'active la commande Fermer
         OpenToolStripMenuItem.Enabled = False  'désactive la commande Ouvrir
     Catch
         MsgBox("Erreur d'ouverture de fichier.")
     Finally
         FileClose(1) 'ferme le fichier
     End Try
End If
```

Voici les actions que réalise cette procédure événementielle :

- Elle déclare des variables et assigne une valeur à la propriété *Filter* de l'objet *OpenFileDialog*.
- Elle invite l'utilisateur à saisir un chemin en utilisant l'objet *OpenFileDialog1*.
- Elle piège les erreurs grâce au bloc de code *Try…Catch*.
- Elle ouvre le fichier spécifié en vue d'y saisir des entrées grâce à la fonction *FileOpen*.
- Elle exploite la fonction *LineInput* pour copier une ligne à la fois depuis le fichier dans une chaîne appelée *AllText*.
- Elle copie les lignes jusqu'à la fin du fichier (EOF, *End Of File*) ou jusqu'à ce qu'il n'y ait plus de place dans la chaîne. La chaîne *AllText* peut contenir un fichier de très grande taille mais, si une erreur se produit pendant le processus de copie, la clause *Catch* affiche l'erreur.
- Elle affiche la chaîne *AllText* dans la zone de texte et active les barres de défilement et le curseur texte.
- Elle met à jour les commandes du menu Fichier et ferme le fichier en utilisant la fonction *FileClose*.

Prenez quelques instants pour observer comment fonctionnent les instructions de la procédure événementielle *OpenToolStripMenuItem_Click* et en particulier *File-*

Open, *LineInput*, *EOF* et *FileClose*. Le gestionnaire d'erreur de la procédure affiche un message et abandonne le processus de chargement en cas d'erreur.

> **Astuce** Pour plus d'informations sur les instructions et les fonctions, mettez en surbrillance le mot clé qui vous intéresse et appuyez sur F1 pour afficher une explication dans la documentation de Visual Basic.

3. Affichez la procédure événementielle *CloseToolStripMenuItem_Click* qui s'exécute lors d'un clic sur la commande du menu Fermer.

   La procédure événementielle se présente ainsi :

```
txtNote.Text = "" 'efface la zone de texte
lblNote.Text = "Charge un fichier texte avec la commande Ouvrir."
CloseToolStripMenuItem.Enabled = False  'désactive la commande Fermer
OpenToolStripMenuItem.Enabled = True   'active la commande Ouvrir
```

   Cette procédure efface la zone de texte, met à jour l'étiquette *lblNote*, désactive la commande Fermer et active la commande Ouvrir.

Vous pouvez maintenant utiliser ce programme simple comme modèle dans des programmes plus avancés qui traitent des fichiers texte. Dans la prochaine section, vous allez apprendre à taper votre propre texte dans une zone de texte et à l'enregistrer dans un fichier sur votre disque dur.

Exploiter la classe *StreamReader* et *My.Computer.FileSystem* pour ouvrir des fichiers texte

Outre les commandes Visual Basic qui ouvrent et affichent des fichiers texte, il existe d'autres techniques pour ouvrir des fichiers texte dans un programme Visual Studio : la classe *StreamReader* et l'espace de noms *My*. Comme ces techniques exploitent des objets du .NET Framework disponibles dans tous les langages de programmation Visual Studio, je les préfère aux fonctions qui ne s'appliquent qu'à Visual Basic. Toutefois, pour des questions esthétiques et de compatibilité, Microsoft a pris soin de conserver plusieurs mécanismes de traitement des fichiers. Vous restez donc maître du choix final.

La classe *StreamReader*

La classe *StreamReader* de la bibliothèque de classes du .NET Framework permet d'ouvrir et afficher des fichiers texte dans vos programmes. Je reprends cette technique à plusieurs reprises dans ce livre pour travailler sur des fichiers texte (par exemple, au chapitre 16 « Gérer l'héritage de formulaire et créer des classes de base »). Pour accéder à la classe *StreamReader*, ajoutez l'instruction *Imports* suivante en début de code (comme abordé au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework »). :

```
Imports System.IO
```

Ensuite, si votre code contient un objet zone de texte, vous pouvez afficher un fichier texte dans celui-ci grâce au code suivant. Le fichier texte ouvert dans cet exemple est FausseMonnaie.txt et ce code suppose qu'un objet appelé *TextBox1* a été créé sur le formulaire.

```
Dim StreamToDisplay As StreamReader
StreamToDisplay = New StreamReader("C:\vb08epe\chap13\Navigateur texte\FausseMonnaie.txt")
TextBox1.Text = StreamToDisplay.ReadToEnd
StreamToDisplay.Close()
TextBox1.Select(0, 0)
```

StreamReader est une alternative du .NET Framework pour ouvrir un fichier texte en utilisant la fonction Visual Basic *FileOpen*. Dans cet exemple, j'ai déclaré une variable appelée *StreamToDisplay* de type *StreamReader* pour héberger le contenu du fichier texte, puis j'ai spécifié le chemin valide du fichier que je souhaite ouvrir. Ensuite, j'ai lu le contenu du fichier texte dans la variable *StreamToDisplay via* la méthode *ReadToEnd*, qui extrait l'ensemble du texte du fichier depuis l'emplacement en cours (le début du fichier texte) jusqu'à la fin du fichier texte, et l'assigne à la propriété *Text* de l'objet zone de texte. Les dernières instructions ferment le fichier texte et utilisent la méthode *Select* pour supprimer la sélection dans la zone de texte.

L'espace de noms *My*

Pour ouvrir des fichiers texte, la deuxième solution réside dans une utile fonctionnalité de Visual Studio qui emploie l'espace de noms *My*. Il s'agit d'une fonctionnalité d'accès rapide conçue pour simplifier l'accès au .NET Framework en vue d'accomplir des tâches courantes, comme la manipulation de formulaires, l'exploration de l'ordinateur hôte et son système de fichiers, l'affichage des informations sur l'application en cours ou sur son utilisateur, et l'accès aux services web. La plupart de ces fonctionnalités étaient déjà disponibles *via* la bibliothèque de classes de base du .NET Framework. Toutefois, en raison de sa complexité, bon nombre de programmeurs éprouvent quelques difficultés à les localiser et à les utiliser. L'espace de noms *My* a été ajouté à Visual Basic 2005 afin de faciliter la programmation.

L'espace de noms *My* propose plusieurs catégories de fonctionnalités, comme le montre le tableau suivant.

Objet	Description
My.Application	Information liée à l'application en cours, dont son titre, son dossier et son numéro de version.
My.Computer	Informations sur le matériel, le logiciel et les fichiers situés sur l'ordinateur (local) en cours. *My.Computer* comprend *My.Computer.FileSystem*, que vous pouvez utiliser pour ouvrir des fichiers texte et des fichiers encodés sur le système.
My.Forms	Informations sur les formulaires de votre projet Visual Studio en cours. Le chapitre 16 montre comment utiliser *My.Forms* pour basculer entre des formulaires à l'exécution.

Objet	Description
My.Resources	Informations sur les ressources de l'application (en lecture seule). Permet de récupérer dynamiquement les ressources de votre application.
My.Settings	Informations sur les paramètres de votre application. Permet de stocker et de récupérer dynamiquement les réglages de propriétés et autres informations sur votre application.
My.User	Informations sur l'utilisateur en cours actif sur *My.Computer*.
My.WebServices	Informations sur les services web actif sur *My.Computer* et un mécanisme d'accès aux nouveaux services web.

L'espace de noms *My* est une véritable fonctionnalité « appel rapide », que vous pouvez explorer pleinement *via* la fonctionnalité Microsoft IntelliSense de l'Éditeur de code. Par exemple, pour utiliser une boîte de message destinée à afficher le nom de l'ordinateur suivi du nom de l'utilisateur en cours dans un programme, tapez simplement :

```
MsgBox(My.User.Name)
```

Cela génère une sortie semblable à :

L'objet *My.Computer* peut afficher plusieurs catégories d'informations sur votre ordinateur et ses fichiers. Par exemple, l'instruction suivante affiche l'heure système en cours (la date et l'heure locale) de l'ordinateur :

```
MsgBox(My.Computer.Clock.LocalTime)
```

Vous pouvez utiliser l'objet *My.Computer.FileSystem* avec la méthode *ReadAllText* pour ouvrir un fichier texte et afficher son contenu dans un objet zone de texte. Voici la syntaxe à utiliser si vous possédez un objet zone de texte sur votre formulaire appelé *txtNote* (comme dans le dernier programme exemple) et que vous prévoyez d'utiliser un objet *OpenFileDialog* appelé *OpenFileDialog1* pour obtenir le nom du fichier texte de l'utilisateur :

```
Dim AllText As String = ""
OpenFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog() 'affiche une boîte de dialogue Ouvrir
If OpenFileDialog1.FileName <> "" Then
    AllText = My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)
    txtNote.Text = AllText 'affiche le fichier
End If
```

La méthode *ReadAllText* copie l'ensemble du contenu du fichier texte donné dans une variable de chaîne ou un objet (dans ce cas, une variable de chaîne appelée *AllText*). Donc,

en termes de performance et de temps de codage, la méthode *ReadAllText* est plus rapide que la lecture du fichier ligne par ligne avec la fonction *LineInput*.

En raison de cette rapidité, l'espace de noms *My* représente un excellent raccourci vers de nombreuses tâches de programmation courantes. Il est important de prendre note de cette fonctionnalité et de ses utilisations possibles. Toutefois, l'espace de noms *My* est utile ici car nous lisons la totalité du fichier texte. La fonction *LineInput* et la classe *StreamReader* proposent plus de fonctionnalités que la mise en œuvre actuelle de l'espace de noms *My*, et en particulier la possibilité de traiter des fichiers ligne par ligne (une capacité vitale pour des tâches de tri et d'analyse, comme nous le verrons bientôt). Il est donc préférable de maîtriser chacune des trois méthodes traitées dans ce chapitre pour ouvrir des fichiers texte. Celle que vous choisirez dans la pratique dépendra de la tâche à accomplir et de la manière dont vous envisagez d'utiliser votre code.

Créer un nouveau fichier texte sur le disque

Pour créer un nouveau fichier texte sur le disque en utilisant Visual Basic, vous pouvez exploiter bon nombre de fonctions et de mots clés issus de l'exemple précédent. La création et la sauvegarde de données sur de nouveaux fichiers texte sont utiles si vous prévoyez de générer des rapports ou des journaux personnalisés, que vous sauvegardez des calculs ou des valeurs importantes ou que vous créez un traitement de texte ou un éditeur de texte spécialisé. Voici un aperçu des étapes à suivre dans le programme :

1. Récupérez des entrées utilisateur, effectuez des calculs mathématiques, ou les deux.
2. Assignez les résultats de votre calcul à une ou plusieurs variables. Par exemple, attribuez le contenu d'une zone de texte à une variable de chaîne appelée *InputForFile*.
3. Invitez l'utilisateur à saisir un chemin en utilisant un contrôle *SaveFileDialog*. Utilisez la méthode *ShowDialog* pour afficher la boîte de dialogue.
4. Utilisez le chemin obtenu dans la boîte de dialogue pour ouvrir le fichier destiné aux résultats.
5. Utilisez la fonction *PrintLine* pour enregistrer une ou plusieurs valeurs destinées au fichier ouvert.
6. Fermez le fichier lorsque vous avez terminé en utilisant la fonction *FileClose*.

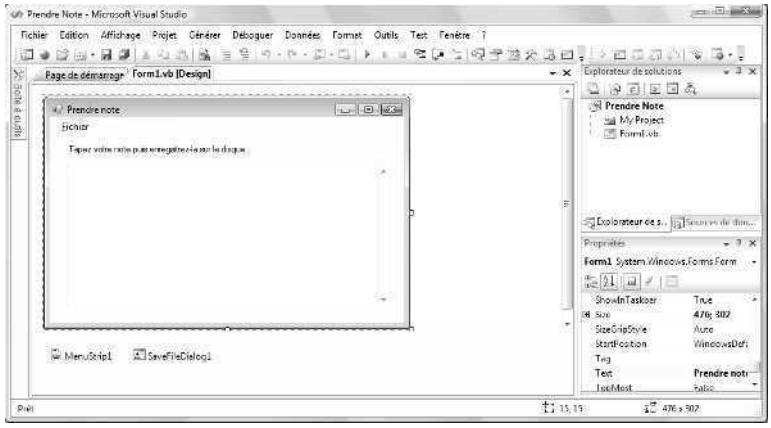
L'exercice suivant montre comment exploiter les contrôles *TextBox* et *SaveFileDialog* pour créer un utilitaire simple de prise de note. Le programme utilise la fonction *FileOpen* pour ouvrir un fichier, la fonction *PrintLine* pour y stocker les données chaîne et la fonction *FileClose* pour fermer le fichier. Vous pouvez utiliser ce programme pour prendre des notes à votre domicile ou sur votre lieu de travail, puis y apposer le tampon d'heure et la date en cours.

Exécuter le programme Prendre Note

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet.
2. Ouvrez le projet Prendre Note dans le dossier c:\vb08epe\chap13\Prendre Note.

   Le programme s'ouvre dans l'environnement de développement.
3. Affichez le formulaire si ce n'est déjà fait.

   Le formulaire Prendre Note s'affiche, comme le montre l'illustration suivante. Celui-ci ressemble au formulaire Navigateur Texte. Toutefois, le contrôle *OpenFileDialog* a été remplacé par le contrôle *SaveFileDialog*. Le menu Fichier contient les commandes Enregistrer sous, Insérer date et Quitter.



   Voici les propriétés définies dans le projet :

Objet	Propriété	Paramètre
txtNote	*Multiline*	True
	Name	txtNote
	ScrollBars	Vertical
lblNote	*Text*	« Tapez votre note puis enregistrez-la sur le disque »
Form1	*Text*	« Prendre Note »

4. Cliquez sur le bouton Démarrer le débogage.
5. Tapez le texte suivant ou celui de votre choix dans la zone de texte :

   Comment détecter de fausses monnaies

   1. **Laisser tomber les pièces sur une surface dure. Les pièces de monnaie véritables résonnent comme une cloche : les contrefaçons font un bruit sourd.**

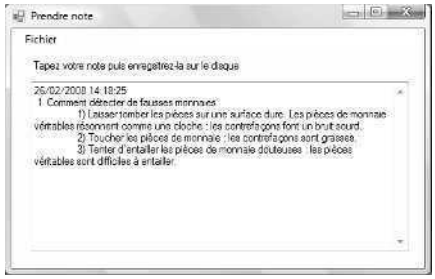
2. **Toucher les pièces de monnaie : les contrefaçons sont grasses.**
3. **Tenter d'entailler les pièces de monnaie douteuses : les pièces véritables sont difficiles à entailler.**

Une fois que vous avez terminé, votre formulaire ressemble à :

> **Astuce** Pour coller du texte du Presse-papiers Windows dans la zone de texte, appuyez sur CTRL+V ou ALT+INSER. Pour copier du texte de la zone de texte vers le Presse-papiers Windows, sélectionnez le texte puis appuyez sur CTRL+C.

Essayez maintenant les commandes du menu Fichier.

6. Dans le menu Fichier, cliquez sur la commande Insérer date. La date et l'heure en cours s'affichent dans la première ligne de la zone de texte, comme suit :

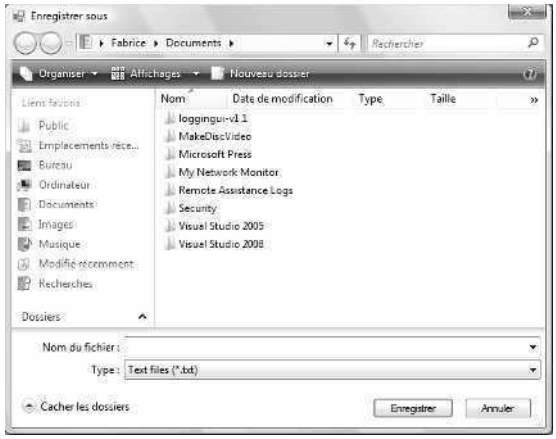


Cette commande offre un moyen pratique d'inclure le tampon de l'heure en cours sur un fichier, ce qui est utile si vous créez un journal ou un registre.

7. Dans le menu Fichier, cliquez sur la commande Enregistrer sous.

Le programme affiche une boîte de dialogue Enregistrer sous avec toutes les fonctionnalités attendues. Le type de fichier par défaut est défini à .txt.

Votre écran présente un résultat similaire à



8. Dans la boîte de dialogue Enregistrer sous, ouvrez le dossier c:\vb08epe\chap13\Prendre Note s'il n'est pas déjà ouvert. Tapez **FaussesMonnaies.txt** dans la zone de texte Nom du fichier puis cliquez sur Enregistrer.

Le texte de votre document est enregistré dans le nouveau fichier texte Fausses-Monnaies.txt.

9. Dans le menu Fichier, cliquez sur la commande Quitter.

Le programme s'arrête et vous revenez à l'environnement de développement.

Vous allez maintenant observer les procédures événementielles de ce programme.

Examiner le code du programme Prendre Note

1. Dans le menu Fichier du formulaire Prendre Note, double-cliquez sur la commande Insérer date.

La procédure événementielle *InsertDateToolStripMenuItem_Click* s'affiche dans l'Éditeur de code. Le code se présente ainsi :

```
txtNote.Text = My.Computer.Clock.LocalTime & vbCrLf & txtNote.Text
txtNote.Select(1, 0) 'efface la sélection
```

Cette procédure événementielle ajoute l'heure et la date en cours dans la zone de texte en liant, ou en *concaténant*, la date en cours (générée par l'objet *My.Computer.Clock* et la propriété *LocalTime*), un retour chariot (ajouté par la constante *vbCrLf*) et la propriété *Text*. Vous pouvez utiliser une technique similaire pour ajouter uniquement la date en cours (grâce à *DateString*) ou toute autre information dans le texte de la zone de texte.

2. Prenez quelques instants pour observer comment les instructions de concaténation fonctionnent, puis examinez la procédure événementielle *SaveAsToolStripMenuItem_Click* dans l'Éditeur de code.

   Voici le code qui s'affiche :

```
SaveFileDialog1.Filter = "Text files (*.txt)|*.txt"
SaveFileDialog1.ShowDialog()
If SaveFileDialog1.FileName <> "" Then
     FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
     PrintLine(1, txtNote.Text)  'copie le texte sur le disque
     FileClose(1)
End If
```

   Ce bloc d'instructions exploite un objet *SaveFileDialog* pour afficher une boîte de dialogue Enregistrer sous, vérifie si l'utilisateur a sélectionné un fichier, ouvre le fichier destiné aux résultats sous la forme de fichier numéro 1, inscrit la valeur dans la propriété *txtNote.Text* sur le disque en utilisant la fonction *PrintLine*, puis ferme le fichier texte. Prêtez une attention particulière à l'instruction

```
PrintLine(1, txtNote.Text)  'copie le texte sur le disque
```

   qui assigne le contenu complet de la zone de texte au fichier ouvert. *PrintLine* ressemble aux anciennes instructions Visual Basic *Print* et *Print #* ; elle oriente la sortie vers le fichier spécifié plutôt que vers l'écran ou vers l'imprimante. Ici, il est important de relever que le fichier complet est stocké dans la propriété *txtNote.Text*.

3. Revoyez les fonctions *FileOpen*, *PrintLine* et *FileClose*, puis fermez le programme en utilisant la commande Fermer le projet du menu Fichier.

Vous avez terminé votre travail avec le programme Prendre Note.

Traiter des chaînes textuelles avec le code du programme

Comme vous l'avez appris dans les exercices précédents, le contrôle *TextBox* ainsi que des instructions judicieusement choisies, permettent d'ouvrir, de modifier et d'enregistrer rapidement des fichiers texte sur le disque. Visual Basic propose également plusieurs instructions et fonctions puissantes conçues spécialement pour traiter les éléments textuels de vos programmes. Dans cette section, vous allez apprendre à extraire des informations utiles d'une chaîne textuelle et à copier et trier une liste de chaînes dans un tableau.

Cette capacité de tri est extrêmement utile lorsque l'on travaille avec des éléments textuels. Les concepts de base du tri sont très simples. Vous constituez une liste d'éléments à trier, puis vous les comparez un à un jusqu'à ce que la liste soit triée selon un ordre alphabétique croissant ou décroissant.

Dans Visual Basic, vous comparez un élément à un autre avec les mêmes opérateurs relationnels que ceux qui servent à comparer des valeurs numériques. La partie délicate (sujet de discussions sans fin parmi les informaticiens) concerne l'algorithme de tri spécifique utilisé pour comparer les éléments d'une liste. Dans ce chapitre, nous ne parlerons pas des avantages et des inconvénients des différents algorithmes de tri. (Le principal différent repose sur la rapidité qui n'est importante que dans un tri de plusieurs milliers d'éléments). Nous allons plutôt explorer comment s'accomplissent les comparaisons de chaîne fondamentales dans un tri. Vous allez acquérir en cours de route les connaissances nécessaires pour trier vos propres zones de texte, zones de liste, fichiers et bases de données.

Classe String et méthodes et mots clés utiles

Jusqu'à présent, vous vous êtes limités à concaténer des chaînes grâce à l'opérateur de concaténation ((&). Par exemple, l'instruction qui suit concatène trois expressions de chaîne littérales et assigne le résultat « Vive le cirque ! » à la variable de chaîne Slogan :

```
Dim Slogan As String
Slogan = "Vive" & " le " & "cirque!"
```

Vous pouvez également concaténer et manipuler des chaînes grâces aux méthodes de la classe *String* (de la bibliothèque de classes du .NET Framework. Par exemple, la méthode *String.Concat* autorise une concaténation de chaîne équivalente avec la syntaxe :

```
Dim Slogan As String
Slogan = String.Concat("Vive", " le ", "cirque !")
```

Visual Basic 2008 contient deux méthodes destinées à la concaténation de chaînes et à de nombreuses autres tâches de traitement de chaîne : vous pouvez utiliser des opérateurs et des fonctions d'anciennes versions de Visual Basic (*Mid*, *UCase*, *LCase*, et ainsi de suite) ou bien des méthodes plus récentes du .NET Framework (*Substring*, *ToUpper*, *ToLower*, et ainsi de suite). Aucune « pénalité » n'est appliquée si l'on utilise une de ces techniques de traitement de chaîne, même si les méthodes les plus anciennes continuent d'exister essentiellement pour des questions de compatibilité. (En prenant en charge ces deux types de méthodes, Microsoft permet aux utilisateurs qui se sont mis à niveau d'apprendre de nouvelles fonctionnalités à leur propre rythme). Dans le reste de ce chapitre, je vais me concentrer sur les fonctions de traitement de chaînes les plus récentes issues de la classe *String* du .NET Framework. Toutefois, vous pouvez exploiter les deux méthodes de traitement de chaînes ou une combinaison des deux.

Le tableau suivant répertorie plusieurs méthodes du .NET Framework qui apparaissent dans les exercices à venir et leurs équivalents les plus proches dans Visual Basic. La quatrième colonne du tableau propose un exemple de code pour les méthodes de la classe *String* du .NET Framework.

Méthode du .NET Framework	Fonction Visual Basic	Description	Exemple .NET Framework
ToUpper	*UCase*	Remplace les lettres d'une chaîne par des majuscules.	`Dim Name, NewName As String Name = "Kim" NewName = Name.ToUpper 'NewName = "KIM"`
ToLower	*LCase*	Remplace les lettres d'une chaîne par des minuscules.	`Dim Name, NewName As String Name = "Kim" NewName = Name.ToLower 'NewName = "kim"`
Length	*Len*	Détermine le nombre de caractères dans une chaîne.	`Dim River As String Dim Size As Short River = "Mississippi" Size = River.Length 'Size = 11`
Substring	*Mid*	Retourne un nombre fixe de caractères dans une chaîne depuis un point de départ donné. (Note : le premier élément d'une chaîne prend l'index 0).	`Dim Cols, Middle As String Cols = "First Second Third" Middle = Cols.SubString(6, 6) 'Middle = "Second"`
IndexOf	*InStr*	Trouve le point de départ d'une chaîne dans une chaîne de plus grande taille.	`Dim Name As String Dim Start As Short Name = "Abraham" Start = Name.IndexOf("h") 'Start = 4`
Trim	*Trim*	Supprime les premiers espaces et les espaces suivants dans une chaîne.	`Dim Spacey, Trimmed As String Spacey = " Hello " Trimmed = Spacey.Trim 'Trimmed = "Hello"`
Remove		Supprime les caractères au milieu d'une chaîne.	`Dim RawStr, CleanStr As String RawStr = "Hello333 there!" CleanStr = RawStr.Remove(5, 3) 'CleanStr = "Hello there!"`
Insert		Ajoute des caractères au milieu d'une chaîne.	`Dim Oldstr, Newstr As String Oldstr = "Hi Felix" Newstr = Oldstr.Insert(3, "there ") 'Newstr = "Hi there Felix"`
StrComp		Compare des chaînes sans tenir compte des différences de casse.	`Dim str1 As String = "Soccer" Dim str2 As String = "SOCCER" Dim Match As Short Match = StrComp(str1, _ str2, CompareMethod.Text) 'Match = 0 [strings match]`

Trier du texte

Avant que Visual Basic puisse comparer un caractère à un autre dans un tri, il doit convertir chaque caractère en numéro grâce à un tableau de conversion appelé *jeu de caractères ASCII* (également appelé *jeu de caractères ANSI*). ASCII est l'acronyme de *American Standard Code for Information Interchange*. Chacun des principaux symboles qui s'affichent sur votre ordinateur possède un code ASCII différent. Ces codes comprennent le jeu de base des caractères d'une « machine à écrire » (codes 32 à 127) et des caractères de « contrôle » spéciaux, comme les tabulations, les sauts de ligne et les retours chariot (codes 0 à 31). Par exemple, la lettre minuscule « a » correspond au code ASCII 97 et la lettre majuscule « A » correspond au code ASCII 65. En conséquence, Visual Basic traite ces deux caractères différemment au cours d'un tri ou d'autres tâches de comparaison.

Dans les années 1980, IBM a étendu le code ASCII de 128 à 255, qui représente les caractères accentués, l'alphabet grec, des caractères graphiques, ainsi que divers symboles. ASCII et ces autres caractères et symboles sont connus sous le nom de *jeu de caractères étendu d'IBM*.

Astuce Pour un aperçu du tableau des codes du jeu de caractères ASCII, recherchez « ASCII Character Codes » dans la documentation de Visual Studio.

Le jeu de caractères ASCII demeure le code numérique le plus important à apprendre pour les programmeurs débutants.Il n'est toutefois pas le seul. La globalisation du marché des ordinateurs et des applications logicielles a donné naissance à une représentation de caractères standard plus détaillée appelée *Unicode*. Celle-ci peut contenir jusqu'à 65 536 symboles, chiffre suffisant pour représenter les symboles traditionnels du jeu de caractères ASCII ainsi que la plupart des langages et symboles (écrits) internationaux. Un organisme de normalisation gère le jeu de caractères Unicode et y ajoute régulièrement de nouveaux symboles. Windows Server 2003, Windows XP, Windows Vista et Visual Studio ont été spécialement conçus pour gérer les jeux de caractères ASCII et Unicode. Pour plus d'informations sur la relation entre les types de données Unicode, ASCII et Visual Basic, reportez-vous à la section « Travailler avec différents types de données » au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework ».

Dans les sections à venir, vous en saurez plus sur l'utilisation du jeu de caractères ASCII pour traiter des chaînes dans vos programmes. Lorsque vos applications se feront plus sophistiquées et que vous commencerez à planifier la distribution globale de votre logiciel, vous aurez besoin d'en savoir plus sur l'Unicode et les autres paramètres internationaux.

Travailler avec les codes ASCII

Pour connaître le code ASCII d'une lettre particulière, utilisez la fonction Visual Basic *Asc*. Par exemple, l'instruction suivante assigne le numéro 122 (le code ASCII de la lettre minuscule « z ») à la variable entière courte *AscCode*.

```
Dim AscCode As Short
AscCode = Asc("z")
```

Inversement, vous pouvez convertir un code ASCII en lettre avec la fonction *Chr*. Par exemple, cette instruction assigne la lettre « z » à la variable de caractère lettre :

```
Dim letter As Char
letter = Chr(122)
```

Vous obtenez le même résultat avec la variable *AscCode* déclarée comme suit :

```
letter = Chr(AscCode)
```

Comment peut-on comparer une chaîne textuelle ou un code ASCII à un autre ? Il suffit d'utiliser un des six opérateurs relationnels fournis avec Visual Basic pour travailler avec des éléments textuels et numériques. Ceux-ci sont présentés dans le tableau qui suit.

Opérateur	Signification
<>	Différent de
=	Égal
<	Inférieur à
>	Supérieur à
<=	Inférieur ou égale à
>=	Supérieur ou égale à

Un caractère est « supérieur à » un autre caractère si son code ASCII est supérieur. Par exemple, la valeur ASCII de la lettre « B » est supérieure à la valeur ASCII de la lettre « A ». Donc, l'expression

```
"A" < "B"
```

est vraie et l'expression

```
"A" > "B"
```

est fausse.

Lorsque l'on compare deux chaînes contenant chacune plusieurs caractères, Visual Basic commence par comparer le premier caractère de la première chaîne au premier caractère de la deuxième chaîne, puis continue caractère par caractère dans la chaîne jusqu'à ce qu'il rencontre une différence. Par exemple, les chaînes Mike et Michael sont identiques jusqu'au troisième caractère (« k » et « c »). Comme la valeur ASCII de « k » est supérieure à celle de « c », l'expression

```
"Mike" > "Michael"
```

est vraie.

Si l'on ne trouve aucune différence entre les chaînes, elles sont égales. Si deux chaînes sont égales sur plusieurs caractères mais qu'une d'entre elles continue tandis que l'autre est terminée, la chaîne la plus longue est supérieure à la chaîne la plus courte. Par exemple, l'expression

```
"AAAAA" > "AAA"
```

est vraie.

Trier des chaînes dans une zone de texte

L'exercice suivant montre comment utiliser des opérateurs relationnels et plusieurs méthodes de chaîne et fonctions pour trier des lignes de texte dans une zone de texte. Le programme est une révision de l'utilitaire Prendre Note et contient une commande Ouvrir qui ouvre un fichier existant et une commande Fermer qui ferme le fichier. Le menu Fichier propose également une commande Trier le texte qui permet de trier le texte affiché dans la zone de texte.

Comme le contenu complet d'une zone de texte est stocké dans une chaîne, le programme doit d'abord diviser cette longue chaîne en chaînes individuelles plus courtes. On peut ensuite les trier grâce à la procédure Sub *ShellSort*, une routine de tri fondée sur un algorithme créé par Donald Shell en 1959. Pour simplifier ces tâches, j'ai créé un module qui définit un tableau de chaînes dynamique destiné à contenir chaque ligne de la zone de texte. J'ai également placé la procédure Sub *ShellSort* dans le module de sorte à pouvoir l'appeler depuis n'importe quelle procédure événementielle du projet. (Pour plus d'informations sur les modules, reportez-vous au chapitre 10, « Créer des modules et des procédures »). Bien que vous ayez appris comment utiliser la puissante méthode *Array.Sort* au chapitre 11, « Utiliser les tableaux pour gérer les données numériques et les chaînes », la procédure *ShellSort* est un outil plus flexible et personnalisable. La construction de la routine à partir de rien enrichit votre expérience sur le traitement des valeurs textuelles : il s'agit d'un des objectifs importants de ce chapitre.

La routine qui détermine le nombre de lignes dans l'objet zone de texte représente un autre aspect intéressant de ce programme. Il n'existe pas de fonction Visual Basic qui calcule automatiquement cette valeur. J'ai souhaité que le programme puisse trier une zone de texte de n'importe quelle taille ligne par ligne. Pour ce faire, j'ai créé le code qui suit. Il exploite la méthode *Substring* pour examiner une lettre à la fois dans l'objet zone de texte, puis utilise la fonction *Chr* pour rechercher le caractère de retour chariot, le code ASCII 13, à la fin de chaque ligne. (Notez en particulier comment la méthode *Substring* est utilisée comme partie de la propriété *Text* de l'objet *txtNote*. La classe *String* fournit automatiquement cette méthode, et bien d'autres, pour toutes les propriétés ou les variables déclarées dans le type *String*).

```
Dim ln, curline, letter As String
Dim i, charsInFile, lineCount As Short

'détermine le nombre de lignes dans l'objet zone de texte (txtNote)
lineCount = 0 'cette variable contient le nombre total de lignes
charsInFile = txtNote.Text.Length 'récupère le total des caractères
For i = 0 To charsInFile - 1 'se déplace d'un caractère à la fois
    letter = txtNote.Text.Substring(i, 1) 'récupère la lettre
    If letter = Chr(13) Then 'si retour chariot trouvé
        lineCount += 1 'va à la ligne suivante (ajoute au compteur)
        i += 1 'saute le caractère de saut de ligne (suit habituellement cr sur un PC)
    End If
Next i
```

Le nombre total de lignes dans la zone de texte est assignée à la variable entière courte *lineCount*. J'ai utilisé cette valeur un peu plus tard pour dimensionner un tableau dynamique destiné à contenir chaque chaîne textuelle individuelle. Le tableau de chaînes résultant est ensuite passé à la procédure Sub *ShellSort* en vue du tri. *ShellSort* retourne le tableau de chaînes par ordre alphabétique. Une fois que le tableau est trié, je peux le recopier simplement dans la zone de texte en utilisant une boucle *Loop*.

Exécuter le programme Tri de texte

1. Ouvrez le projet Tri de texte situé dans le dossier c:\vb08epe\chap13\Tri de texte.
2. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.
3. Tapez le texte suivant ou celui de votre choix dans la zone de texte :

   Zèbre

   Gorille

   Lune

   Banane

   Pomme

   Tortue

   Assurez-vous d'avoir appuyé sur ENTRÉE après avoir tapé « Tortue » (ou votre dernière ligne personnalisée) de sorte que Visual Basic puisse calculer correctement le nombre de lignes.

4. Dans le menu Fichier, cliquez sur la commande Trier le texte. Le texte tapé est trié et affiché de nouveau dans la zone de texte, comme suit :

5. Dans le menu Fichier, cliquez sur la commande Ouvrir, ouvrez le fichier abc.txt dans le dossier c:\vb08epe\chap13, comme suit :

Le fichier abc.txt contient 36 lignes de texte. Chaque ligne commence par une lettre ou un chiffre de 1 à 10.

6. Dans le menu Fichier, cliquez sur la commande Trier texte pour trier le contenu du fichier abc.txt.

Le programme Tri de texte trie le fichier par ordre croissant et affiche la liste triée des lignes dans la zone de texte, comme suit :

7. Défilez dans le fichier pour voir les résultats du tri alphabétique.

   Notez que même si la portion alphabétique du tri s'est parfaitement accomplie, le tri a produit un résultat étrange pour une des entrées numériques – la ligne commençant par le numéro 10 apparaît en deuxième position au lieu de la dixième. En fait, Visual Basic a lu le 1 et le 0 du chiffre 10 comme s'il s'agissait de deux caractères indépendants, et non en tant que chiffre. Comme nous comparons les codes ASCII de ces chaînes de gauche à droite, le programme génère un tri purement alphabétique. Pour trier uniquement des nombres dans ce programme, vous devez interdire les entrées textuelles, modifier le code de sorte que les entrées numériques soient stockées dans des variables numériques, puis comparer les variables numériques et non les chaînes.

Aller plus loin : Examiner le code du programme Tri de texte

Pour étoffer vos compétences en matière de programmation grâce à d'autres outils et revoir certains concepts déjà abordés au cours des précédents chapitres, nous allons examiner de plus près dans cet exercice le code du programme Tri de texte.

Examiner le programme Tri de texte

1. Dans le menu Fichier du programme Tri de texte, cliquez sur la commande Quitter pour arrêter le programme.
2. Ouvrez l'Éditeur de code pour *Form1* et affichez le code de la procédure événementielle *SortTextToolStripMenuItem_Click*.

   Nous avons déjà discuté de la première routine de cette procédure événementielle, qui compte le nombre de lignes dans la zone de texte en utilisant la méthode *Substring* pour rechercher des codes de retour chariot. Le reste de la procédure événementielle dimensionne un tableau de chaînes, copie chaque ligne de texte dans le tableau, appelle une procédure pour trier le tableau et affiche la liste réordonnée dans la zone de texte.

   La procédure événementielle *SortTextToolStripMenuItem_Click* complète se présente ainsi :

```
Dim ln, curline, letter As String
Dim i, charsInFile, lineCount As Short

'détermine le nombre de lignes dans l'objet zone de texte (txtNote)
lineCount = 0 'cette variable contient le nombre total de lignes
charsInFile = txtNote.Text.Length 'récupère le nombre total de caractères
For i = 0 To charsInFile - 1 'se déplace d'un caractère à la fois
     letter = txtNote.Text.Substring(i, 1) 'récupère une lettre
```

```
    If letter = Chr(13) Then 'si le retour chariot est trouvé
        lineCount += 1 'passe à la ligne suivante (ajoute au compteur)
        i += 1 'skip linefeed char (suit habituellement cr sur un PC)
    End If
Next i

'construit un tableau pour héberger le texte de la zone de texte
ReDim strArray(lineCount) 'crée un tableau de la taille appropriée
curline = 1
ln = "" 'utilise ln pour construire des lignes caractère par caractère
For i = 0 To charsInFile - 1 'boucle de nouveau dans le texte
    letter = txtNote.Text.Substring(i, 1) 'récupère la lettre
    If letter = Chr(13) Then 'si le retour chariot est trouvé
        curline = curline + 1 'incrémente le compteur de ligne
        i += 1 'saute le caractère de saut de ligne
        ln = "" 'efface la ligne et passe à la suite
    Else
        ln = ln & letter 'ajoute la lettre à la ligne
        strArray(curline) = ln 'et la met dans un tableau
    End If
Next i

'trie le tableau
ShellSort(strArray, lineCount)
'puis affiche le tableau trié dans la zone de texte
txtNote.Text = ""
curline = 1
For i = 1 To lineCount
    txtNote.Text = txtNote.Text & _
        strArray(curline) & vbCrLf
    curline += 1
Next i
txtNote.Select(1, 0) 'supprime la sélection de texte
```

Le tableau *strArray* était déclaré dans un module (Module1.vb) qui fait également partie de ce programme (chapitre 10). Grâce à l'instruction *ReDim* (chapitre 11), je dimensionne *strArray* sous forme de tableau dynamique avec la variable *lineCount*. Cette instruction crée un tableau avec autant d'éléments que de lignes de texte dans la zone de texte (une exigence de la procédure Sub *ShellSort*). Grâce à une boucle *For* (voir chapitre 7, « Utiliser les boucles et les minuteurs ») et à la variable *ln*, j'analyse de nouveau la zone de texte à la recherche des caractères de retour chariot et je copie chaque ligne complète trouvée dans *strArray*. Une fois que le tableau est rempli de texte, j'appelle la procédure *ShellSort* située dans le module Module1.vb, étudié précédemment dans ce chapitre.

3. Affichez le code du module Module1.vb dans l'Éditeur de code.

Ce module déclare la variable de tableau publique *strArray* (chapitre 11), puis définit le contenu de la procédure *ShellSort*. Cette dernière exploite une instruction *If* et l'opérateur relationnel <= (chapitres 6, 8 et ce chapitre) pour comparer les élé-

ments du tableau et changer de place ceux qui ne sont pas dans le bon ordre. La procédure se présente ainsi :

```
Sub ShellSort(ByRef sort() As String, ByVal numOfElements As Short)
     Dim temp As String
     Dim i, j, span As Short
     'La procédure ShellSort trie les éléments du tableau sort()
     'par ordre décroissant et le retourne à la procédure
     'appellante

     span = numOfElements \ 2
     Do While span > 0
         For i = span To numOfElements - 1
             For j = (i - span + 1) To 1 Step -span
                 If sort(j) <= sort(j + span) Then Exit For
                 'change de place les éléments du tableau qui ne sont pas dans le bon ordre
                 temp = sort(j)
                 sort(j) = sort(j + span)
                 sort(j + span) = temp
             Next j
         Next i
         span = span \ 2
     Loop
End Sub
```

La méthode de tri consiste à diviser constamment par deux la liste principale des éléments en sous-listes. Le tri compare ensuite le haut et le bas des sous-listes pour voir si des éléments sont mal classés. Si le haut et le bas de la liste sont mal classés, on les échange. Il en résulte un tableau appelé *sort()* trié par ordre alphabétique selon un ordre décroissant. Pour modifier la direction du tri, il suffit d'inverser l'opérateur relationnel (remplacer <= par >=).

Les autres procédures événementielles (*OpenToolStripMenuItem_Click*, *CloseToolStripMenuItem_Click*, *SaveAsToolStripMenuItem_Click*, *InsertDateToolStripMenuItem_Click* et *ExitToolStripMenuItem_Click*) sont toutes similaires aux procédures que vous avez étudiées dans les programmes Navigateur texte et Prendre Note. (Pour plus d'informations, reportez-vous aux explications fournies dans ce chapitre).

4. Dans le menu Fichier, cliquez sur la commande Fermer le projet.

   Votre travail sur les chaînes, les tableaux et les fichiers texte est terminé pour l'instant.

Félicitations ! Si vous avez étudié les chapitres 5 à 13, vous avez achevé la partie correspondant aux principes fondamentaux de la programmation de ce livre. Vous voici prêt à vous concentrer plus particulièrement sur la création d'interfaces utilisateur de qualité professionnelle dans vos programmes. Vous avez accompli une grande étape dans votre étude de la programmation Visual Basic et dans votre utilisation de l'environnement de développement Visual Studio. Faites une petite pause et je vous retrouverai dans la partie 3, « Concevoir l'interface utilisateur ».

Rappel du chapitre 13

Pour	Faites ceci

Ouvrir un fichier texte	Utilisez la fonction *FileOpen*. Par exemple :
	`FileOpen(1, OpenFileDialog1.FileName, _`
	`OpenMode.Input)`

Récupérer une ligne d'entrées depuis le fichier texte	Utilisez la fonction *LineInput*. Par exemple :
	`Dim LineOfText As String`
	`LineOfText = LineInput(1)`

Rechercher la fin d'un fichier	Utilisez la fonction *EOF*. Par exemple :
	`Dim LineOfText, AllText As String`
	`Do Until EOF(1)`
	`  LineOfText = LineInput(1)`
	`  AllText = AllText & LineOfText & _vbCrLf`
	`Loop`

Fermer un fichier ouvert	Utilisez la fonction *FileClose*. Par exemple : `FileClose(1)`

Afficher un fichier texte en utilisant *LineInput*	Utilisez la fonction *LineInput* pour copier du texte d'un fichier ouvert dans une variable de chaîne, puis assignez la variable de chaîne à un objet zone de texte. Par exemple :
	`Dim AllText, LineOfText As String`
	`Do Until EOF(1) 'lit les lignes depuis le fichier`
	`  LineOfText = LineInput(1)`
	`  AllText = AllText & LineOfText & _ vbCrLf`
	`Loop`
	`txtNote.Text = AllText 'afficher le fichier`

Afficher un fichier texte en utilisant la classe *StreamReader*	Ajoutez l'instruction *Imports System.IO* dans la section de déclaration de votre formulaire, puis utilisez *StreamReader*. Par exemple, pour afficher le fichier dans un objet zone de texte appelé *TextBox1* :
	`Dim StreamToDisplay As StreamReader`
	`StreamToDisplay = New StreamReader( _`
	`  "c:\vb05epe\chap13\Navigateur texte\faussesmonnaie.txt")`
	`TextBox1.Text = StreamToDisplay.ReadToEnd`
	`StreamToDisplay.Close()`
	`TextBox1.Select(0, 0)`

Afficher un fichier texte en utilisant l'espace de noms *My*	Utilisez l'objet *My.Computer.FileSystem* et la méthode *ReadAllText*. Par exemple, en supposant que vous utilisiez également un objet *OpenFileDialog* appelé *ofd* et un objet zone de texte appelé *txtNote* :
	`Dim AllText As String = ""`
	`ofd.Filter = "Text files (*.TXT)|*.TXT"`
	`ofd.ShowDialog()`
	`If ofd.FileName <> "" Then`
	`  AllText = _`
	`  My.Computer.FileSystem.ReadAllText(ofd.FileName)`
	`  txtNote.Text = AllText 'afficher le fichier`
	`End If`

Pour	Faites ceci
Afficher une boîte de dialogue Ouvrir	Ajoutez un contrôle *OpenFileDialog* à votre formulaire, puis utilisez la méthode *ShowDialog* de l'objet *OpenFileDialog*. Par exemple : `OpenFileDialog1.ShowDialog()`
Créer un nouveau fichier texte	Utilisez la fonction *FileOpen*. Par exemple : `FileOpen(1, SaveFileDialog1.FileName, _ OpenMode.Output)`
Afficher une boîte de dialogue Enregistrer sous	Ajoutez un contrôle *SaveFileDialog* à votre formulaire, puis utilisez la méthode *ShowDialog* de l'objet OpenFileDialog. Par exemple : `SaveFileDialog1.ShowDialog()`
Enregistrer du texte dans un fichier	Utilisez la fonction *Print* ou *PrintLine*. Par exemple : `PrintLine(1, txtNote.Text)`

Convertir des caractères textuels en codes ASCII	Utilisez la fonction *Asc*. Par exemple :
	`Dim Code As Short`
	`Code = Asc("A") 'Code égal 65`

Convertir des codes ASCII en caractères textuels	Utilisez la fonction *Chr*. Par exemple :
	`Dim Letter As Char`
	`Letter = Chr(65) 'Lettre égal "A"`

Extraire des caractères au milieu d'une chaîne	Utilisez la méthode *Substring* ou la fonction *Mid*. Par exemple :
	`Dim Cols, Middle As String`
	`Cols = "First Second Third"`
	`Middle = Cols.SubString(6, 6)`
	`'Middle = "Second"`

Partie III

Concevoir l'interface utilisateur

Dans cette partie :

Dans la Partie II, nous avons étudié la plupart des techniques de développement essentielles servant de base à la création d'applications Microsoft Visual Basic. Vous avez appris à employer les variables, les opérateurs, les structures de décision et le Microsoft .NET Framework. Nous avons appris à gérer le flux du code avec les boucles, les horloges, les procédures et les gestionnaires d'erreur structurés, à déboguer les programmes, organiser les informations dans des tableaux, des collections, des fichiers texte et découvert des techniques de traitement des chaînes.

Tous les exercices que vous avez réalisés jusqu'à présent se concentraient sur une ou plusieurs de ces techniques de base dans un simple programme autonome. Les programmes de production sont rarement aussi simples. Ils exigent généralement de combiner ces techniques de diverses manières avec diverses améliorations. Vos programmes nécessiteront souvent plusieurs formulaires, employés comme boîtes de dialogue, des formulaires d'entrée et de sortie, des rapports et ainsi de suite. Dans Visual Basic, chaque formulaire est traité comme un objet séparé. Ils peuvent donc être considérés comme des blocs de construction qui peuvent être combinés pour créer des programmes puissants.

Dans la Partie III, nous nous concentrons à nouveau sur l'interface utilisateur. Nous verrons comment ajouter des projets à plusieurs formulaires, des effets d'animation, l'héritage visuel et un support d'impression à vos applications Visual Basic.

Chapitre 14

Gérer les formulaires et les contrôles Windows à l'exécution

À la fin de ce chapitre, vous saurez :

- Ajouter de nouveaux formulaires à un programme et basculer entre plusieurs formulaires
- Changer la position d'un formulaire sur le bureau Microsoft Windows
- Ajouter des contrôles à un formulaire pendant l'exécution
- Changer l'alignement des objets au sein d'un formulaire pendant l'exécution
- Utiliser la boîte de dialogue Propriétés pour spécifier le formulaire de démarrage

Dans ce chapitre, vous allez apprendre à ajouter des formulaires à une application pour gérer l'entrée, la sortie et les messages spéciaux. Vous étudierez également comment utiliser les objets *Me* et *My.Forms* pour basculer entre les formulaires, exploiter la propriété *DesktopBounds* pour redimensionner un formulaire, ajouter des contrôles de la Boîte à outils à un formulaire pendant l'exécution, changer l'alignement des objets au sein d'un formulaire et désigner le formulaire à exécuter au démarrage d'un programme.

Ajouter de nouveaux formulaires à un programme

Tous les programmes que vous avez écrits jusqu'à présent utilisent un formulaire et une série de boîtes de dialogue polyvalentes pour l'entrée et la sortie. Des boîtes de dialogue et un formulaire suffisent souvent à communiquer avec l'utilisateur. Pour échanger davantage d'informations et ce de manière plus personnalisée, il est possible d'ajouter des formulaires aux programmes. Chaque nouveau formulaire est considéré comme un objet qui hérite ses fonctionnalités de la classe *System.Windows.Forms.Form*. Le premier formulaire d'un programme s'intitule Form1.vb. Les formulaires suivants sont intitulés Form2.vb, Form3.vb et ainsi de suite (pour modifier le nom d'un formulaire, servez-vous de la boîte de dialogue Ajouter un nouvel élément ou de l'Explorateur de solutions). Chaque nouveau formulaire possède un nom unique et son propre jeu d'objets, de propriétés, de méthodes et de procédures événementielles.

Le tableau suivant présente plusieurs utilisations pratiques de formulaires supplémentaires dans les programmes.

Formulaire ou formulaires	Description
Formulaire d'introduction	Un formulaire qui présente un message de bienvenue, une illustration ou des informations de droits d'auteur au démarrage du programme.
Instructions	Un formulaire qui présente des informations et des astuces relatives au fonctionnement du programme.
Boîtes de dialogue	Des boîtes de dialogue personnalisées qui acceptent des entrées et affichent des sorties dans le programme.
Contenu du document	Un formulaire qui affiche le contenu d'un ou de plusieurs fichiers et illustrations employés dans le programme.

Comment utiliser les formulaires

Visual Basic est particulièrement souple quant à l'utilisation des formulaires. Vous pouvez choisir d'afficher simultanément tous les formulaires d'un programme ou de charger et décharger les formulaires à mesure que le programme les exige. En cas d'affichage simultané de plusieurs formulaires, l'utilisateur peut être autorisé à basculer entre les formulaires ou devoir respecter un ordre d'utilisation précis. Un formulaire auquel il faut répondre lorsqu'il s'affiche à l'écran est appelé *boîte de dialogue*. Les boîtes de dialogue (appelées *formulaires modaux* dans Visual Basic 6) conservent le focus jusqu'à ce que l'utilisateur clique sur OK, sur Annuler ou les organisent différemment. Pour afficher un formulaire existant sous forme de boîte de dialogue dans Visual Basic 2008, il suffit de l'ouvrir en se servant de la méthode *ShowDialog*.

Pour afficher un formulaire à partir duquel l'utilisateur peut basculer, faites appel à la méthode *Show* à la place de la méthode *ShowDialog*. Dans Visual Basic 6, les formulaires qui ne sont pas contraints de conserver le focus de l'application sont appelés *formulaires non modaux* ou *formulaires sans mode*, termes toujours employés dans cette nouvelle version de Visual Basic. La plupart des applications Windows exploitent des formulaires classiques, non modaux, pour afficher des informations en raison de leur grande souplesse d'utilisation. Il s'agit donc du style par défaut de chaque nouveau formulaire créé dans Visual Studio. Les formulaires étant de simples membres de la classe *System.Windows.Forms.Form*, vous pouvez également les créer et les afficher avec du code.

Exploiter plusieurs formulaires

Les prochains exercices montrent comment utiliser un deuxième formulaire pour afficher des informations d'Aide pour le programme Bandit manchot sur lequel vous avez travaillé dans les chapitres 2, « Écrire son premier programme » et 10, « Créer des modules et des procédures ». Vous allez ajouter un deuxième formulaire avec la commande Ajouter un

formulaire Windows du menu Projet et l'afficher dans le code en vous servant de l'espace de noms *My* et de la méthode *ShowDialog*. Le deuxième formulaire présentera un petit fichier lisez-moi.txt qui présente des informations d'aide et de copyright du programme (le type d'informations que l'on trouve habituellement dans les boîtes de dialogue À propos et Aide).

Ajouter un deuxième formulaire

1. Démarrez Visual Studio et ouvrez le projet Aide Bandit Manchot qui se trouve dans le dossier c:\vb08epe\chap14\Aide Bandit Manchot.

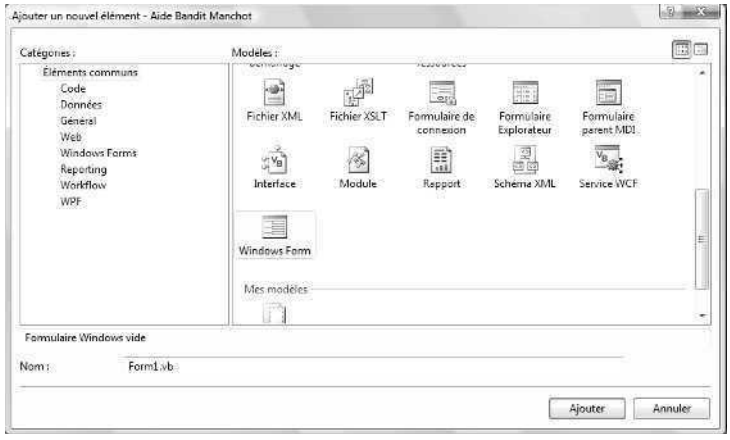
   Le projet Aide Bandit Manchot correspond au jeu du Bandit Manchot sur lequel vous avez travaillé au chapitre 10. Le programme utilise un module et une fonction pour calculer le taux de réussite de l'affichage du chiffre 7.

2. Affichez le formulaire principal (BanditManchot.vb) dans le Concepteur, s'il ne l'est pas déjà.

3. Dans le menu Projet, choisissez la commande Ajouter un formulaire Windows pour ajouter un deuxième formulaire au projet.

4. Servez-vous de la barre de défilement de la boîte de dialogue pour repérer le modèle sélectionné par défaut, Windows Form.

   La boîte de dialogue se présente comme suit :

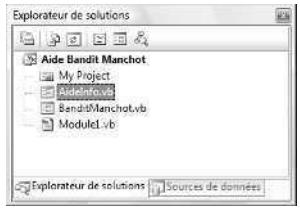


La boîte de dialogue Ajouter un nouvel élément permet d'ajouter des formulaires, des classes, des modules et d'autres composants au projet Visual Basic. Même si vous avez sélectionné la commande Ajouter un formulaire Windows, les formulaires ne sont pas les seuls composants qu'elle présente (même si le modèle Windows Form est sélectionné par défaut). La boîte de dialogue Ajouter un nouvel élément est suffisamment polyvalente pour vous permettre de choisir d'autres composants de projet si vous changez d'avis.

Astuce Je vous recommande tout particulièrement de tester le modèle Formulaire Explorateur, qui permet d'ajouter un navigateur de style Internet Explorer à l'application, avec ses menus, sa barre d'outils et le volet de hiérarchie des dossiers.

5. Tapez **AideInfo.vb** dans la zone de texte Nom et cliquez sur Ajouter.

   Un deuxième formulaire intitulé AideInfo.vb est ajouté au projet Aide Bandit Manchot et s'affiche dans l'Explorateur de solutions, comme le montre la figure suivante :



Astuce Pour renommer ou supprimer les fichiers de formulaire, servez-vous de l'Explorateur de solutions. Pour renommer un fichier, cliquez droit sur le fichier et choisissez la commande Renommer. Pour supprimer un fichier du projet et le supprimer définitivement de l'ordinateur, sélectionnez-le et appuyez sur la touche SUPPR.

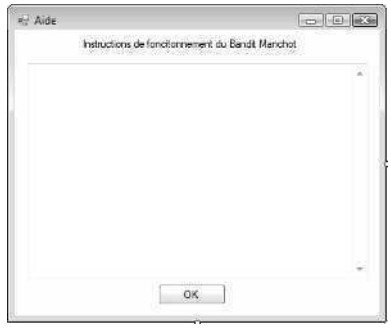
Vous pouvez maintenant ajouter quelques contrôles au formulaire AideInfo.vb.

6. Servez-vous du contrôle *Label* pour créer une étiquette dans la partie supérieure du formulaire AideInfo.vb. Placez-la à proximité du bord gauche du formulaire tout en laissant de la place pour ajouter un libellé descriptif.
7. Servez-vous du contrôle *TextBox* pour créer un objet zone de texte.
8. Attribuez la valeur *True* à la propriété *Multiline* de l'objet zone de texte pour vous permettre de redimensionner l'objet.
9. Redimensionnez l'objet zone de texte de sorte qu'il recouvre une grande partie du formulaire.
10. Servez-vous du contrôle *Button* pour créer un bouton dans la partie inférieure du formulaire.

11. Définissez les propriétés suivantes pour les objets du formulaire AideInfo.vb :

Objet	Propriété	Paramètre
Label1	*Text*	« Instructions de fonctionnement du Bandit Manchot »
TextBox1	*ScrollBars*	Vertical
Button1	*Text*	« OK »
AideInfo.vb	*Text*	« Aide »

Le formulaire AideInfo.vb est similaire à celui-ci :



Vous allez maintenant saisir une ligne de code pour la procédure événementielle *Button1_Click* du formulaire AideInfo.vb.

12. Double-cliquez sur le bouton OK pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.

13. Tapez l'instruction suivante :

```
Me.DialogResult = DialogResult.OK
```

Le formulaire AideInfo.vb fait office de boîte de dialogue dans ce projet puisque le formulaire BanditManchot.vb s'ouvre par le biais de la méthode *ShowDialog*. Après que l'utilisateur a lu les informations d'aide affichées par la boîte de dialogue, il clique sur le bouton OK, qui fixe la propriété *DialogResult* du formulaire en cours à *DialogResult.OK* (le mot clé *Me* sert ici à faire référence au formulaire AideInfo.vb ; vous rencontrerez cette syntaxe abrégée de temps en temps lors d'une référence à une *instance* en cours d'une classe ou d'une structure dans laquelle le code s'exécute).

DialogResult.OK est une constante Visual Basic qui indique que la boîte de dialogue a été fermée et doit renvoyer la valeur « OK » à la procédure appelante. Une boîte de dialogue plus élaborée peut autoriser le retour de valeurs *via* des procédures événementielles de boutons parallèles, comme *DialogResult.Cancel*, *DialogResult.No* et *DialogResult.Yes*. Lorsque l'on définit la propriété *DialogResult*, cependant, le formulaire se ferme automatiquement.

14. Dans la partie supérieure de l'Éditeur de code, tapez l'instruction *Imports* suivante au-dessus de la déclaration *Public Class* :

```
Imports System.IO
```

Cette instruction facilite les références à la classe *StreamReader* dans votre code. La classe *StreamReader* n'est pas spécifiquement liée à la définition ou l'utilisation de formulaires supplémentaires : je l'emploie ici comme moyen plus rapide d'ajouter des informations textuelles au nouveau formulaire que je crée.

15. Affichez à nouveau le formulaire AideInfo.vb et double-cliquez sur son arrière-plan.

La procédure événementielle *AideInfo_Load* s'affiche dans l'Éditeur de code. Il s'agit de la procédure événementielle qui s'exécute lorsque le formulaire est chargé dans en mémoire et affiché à l'écran pour la première fois.

16. Tapez les instructions suivantes :

```
Dim TexteAAfficher As StreamReader
TexteAAfficher = New StreamReader _
    ("c:\vb08epe\chap14\Aide Bandit Manchot\Lisezmoi.txt")
TextBox1.Text = TexteAAfficher.ReadToEnd
TexteAAfficher.Close()
TextBox1.Select(0, 0)
```

Au lieu de saisir le contenu du fichier d'Aide dans la propriété *Text* de l'objet zone de texte (ce qui serait long), j'ai employé la classe *StreamReader* pour ouvrir, lire et afficher un fichier Lisezmoi.txt dans l'objet zone de texte. Ce fichier contient des informations de fonctionnement et des informations de contact générales.

Nous avons fait connaissance avec la classe *StreamReader* au chapitre 13, « Explorer le traitement des fichiers texte et des chaînes », mais vous ne l'avez peut-être pas encore testée. Comme vous l'avez appris, *StreamReader* constitue une alternative .NET Framework pour ouvrir un fichier texte avec l'objet *My.Computer.FileSystem* ou la fonction Visual Basic *FileOpen*. Pour faciliter l'emploi de *StreamReader* dans le code, vous incluez l'espace de noms *System.IO* en haut du code du formulaire. Vous déclarez ensuite une variable *TexteAAfficher* de type *StreamReader* pour héberger le contenu du fichier texte et ouvrez le fichier texte à l'aide d'un chemin d'accès spécifique. Enfin, vous lisez le contenu du fichier texte dans la variable *TexteAAfficher* en faisant appel à la méthode *ReadToEnd* qui lit tout le texte du fichier à partir de l'emplacement actuel (le début du fichier texte) jusqu'à la fin du fichier texte et vous l'affectez à la propriété *Text* de l'objet zone de texte.

L'instruction *StreamReader.Close* ferme le fichier texte et la méthode *Select* supprime la sélection du texte dans l'objet zone de texte.

Vous en avez terminé avec le formulaire AideInfo.vb. Vous allez à présent ajouter un objet bouton et quelques lignes de code au premier formulaire.

Afficher le deuxième formulaire avec une procédure événementielle

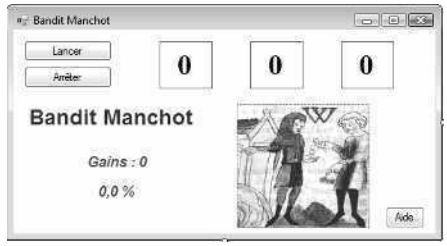
1. Dans l'Explorateur de solutions, cliquez sur BanditManchot.vb puis sur le bouton Concepteur de vues.

   Le formulaire s'affiche dans le Concepteur. Vous allez à présent ajouter un bouton Aide à l'interface utilisateur.

2. Servez-vous du contrôle *Button* pour dessiner un petit bouton dans l'angle inférieur droit du formulaire.

3. Servez-vous de la fenêtre Propriétés pour attribuer la valeur **Aide** à la propriété *Text* de l'objet bouton.

   Votre formulaire présente un résultat similaire à ceci :

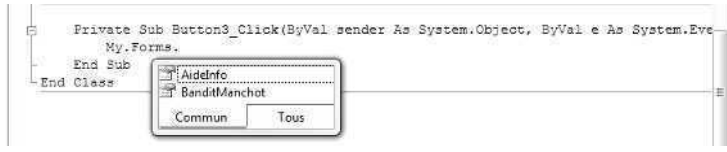


4. Double-cliquez sur le bouton Aide pour afficher la procédure événementielle *Button3_Click* dans l'Éditeur de code.

5. Tapez l'instruction suivante :

```
My.Forms.AideInfo.ShowDialog()
```

   Cette instruction utilise l'espace de noms *My* (présenté au chapitre 13) pour accéder aux formulaires actifs dans le projet en cours. Lorsque vous saisissez l'instruction, la fonctionnalité Microsoft IntelliSense de Visual Studio présente la liste des formulaires disponibles dans la collection *Forms*, comme le montre la figure suivante :



Contrairement à Visual Basic .NET 2003, qui exigeait la déclaration spécifique d'une variable du type du formulaire avant d'utiliser un deuxième formulaire, l'espace de noms *My* de Visual Basic 2005 et 2008 rend tous les formulaires du projet disponibles sans déclaration spécifique.

Remarquez qu'il est également possible d'ouvrir et de manipuler les formulaires directement (à l'instar de Visual Basic 6) en se servant de la syntaxe suivante :

```
AideInfo.ShowDialog()
```

Cette instruction ouvre le formulaire AideInfo sous forme de boîte de dialogue *via* la méthode *ShowDialog*.

6. Vous pouvez également utiliser la méthode *Show* pour ouvrir le formulaire, mais dans ce cas, Visual Basic ne considère pas le formulaire AideInfo.vb comme une boîte de dialogue. Ce formulaire est alors un formulaire non modal à partir duquel et vers lequel l'utilisateur peut basculer à loisir. De surcroît, la propriété *DialogResult* de la procédure événementielle *Button1_Click* du formulaire AideInfo.vb ne ferme pas ce dernier. À la place, il faut ajouter l'instruction *Me.Close*.

Astuce Lors de la conception de vos formulaires, rappelez-vous les différences entre les formulaires modaux et non modaux. Chaque type de formulaire est différent et chaque style présente des avantages particuliers pour l'utilisateur.

Exécutons à présent le programme pour observer le fonctionnement d'une application à plusieurs formulaires.

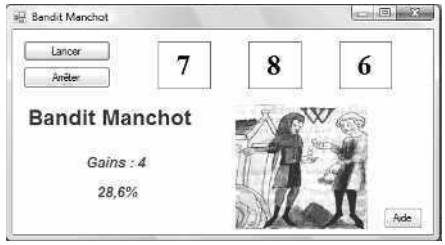
Exécuter le programme

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Le premier formulaire du projet Bandit Manchot s'affiche.

2. Cliquez sept ou huit fois sur le bouton Lancer pour jouer.

   Votre écran est similaire à



3. Cliquez sur le bouton Aide.

   Visual Basic ouvre le deuxième formulaire du projet, AideInfo.vb, et affiche le texte Lisezmoi.txt dans l'objet zone de texte. Voici à quoi ressemble le formulaire :

4. Servez-vous de la barre de défilement verticale pour lire l'intégralité du fichier Lisezmoi.
5. Cliquez sur le bouton OK pour fermer le formulaire AideInfo.vb. Le formulaire se ferme et le premier formulaire est à nouveau actif.
6. Cliquez sur le bouton Lancer quelques fois puis cliquez à nouveau sur le bouton Aide.

   Le formulaire AideInfo.vb s'affiche à nouveau, entièrement fonctionnel. Remarquez qu'il est impossible d'activer le premier formulaire pendant que le deuxième est actif (pour faire un test, essayez de cliquer sur le bouton Lancer du premier formulaire alors que le deuxième est actif). Le deuxième formulaire étant une boîte de dialogue (un formulaire modal), vous devez y répondre avant de pouvoir continuer le programme.
7. Cliquez sur le bouton OK puis sur le bouton Arrêter du premier formulaire.

   Le programme s'arrête et vous revenez à l'environnement de développement.

Utiliser la propriété *DialogResult* dans le formulaire appelant

Bien que je ne l'aie pas montré dans l'exemple de programme, il est possible d'utiliser la propriété *DialogResult* que vous avez assignée à la boîte de dialogue pour bénéficier d'un effet intéressant dans un programme Visual Basic. Comme je l'ai déjà mentionné, une boîte de dialogue plus élaborée pourrait proposer d'autres boutons à l'utilisateur : Annuler, Oui, Non, Abandonner et ainsi de suite. Chaque bouton de la boîte de dialogue peut être associé à un type différent d'action dans le programme principal. En outre, dans les procédures événementielles de chaque bouton de la boîte de dialogue, il est possible d'assigner la propriété *DialogResult* au formulaire qui correspond au nom du bouton, comme dans l'instruction suivante :

```
Me.DialogResult = DialogResult.Cancel    'l'utilisateur a cliqué sur le bouton Annuler
```

Dans la procédure événementielle appelante (autrement dit dans la procédure événementielle *Button3_Click* du formulaire BanditManchot.vb), vous pouvez ajouter du code pour détecter sur quel bouton l'utilisateur a cliqué dans la boîte de dialogue. Cette information est alors stockée dans la propriété *DialogResult* du formulaire, qui peut être évaluée à l'aide d'une structure de décision de base comme *If…Then* ou *Select…Case*. Par exemple, le code suivant placé dans la procédure événementielle *Button3_Click* permet de vérifier si l'utilisateur a cliqué sur OK, Annuler ou tout autre bouton de la boîte de dialogue (la première ligne n'est pas nouvelle, mais vous rappelle le nom du formulaire AideInfo que vous utilisez dans cet exemple).

```
My.Forms.AideInfo.ShowDialog()

If AideInfo.DialogResult = DialogResult.OK Then
    MsgBox("L'utilisateur a cliqué sur OK !")
ElseIf AideInfo.DialogResult = DialogResult.Cancel Then
    MsgBox("L'utilisateur a cliqué sur Annuler !")
Else
    MsgBox("L'utilisateur a cliqué sur un autre bouton")
End If
```

En faisant appel à des procédures événementielles inventives qui déclarent, ouvrent et traitent les choix faits dans les boîtes de dialogue, vous pouvez ajouter autant de formulaires que nécessaires aux programmes et créer une interface utilisateur d'aspect professionnel, polyvalente et conviviale.

Positionner les formulaires sur le bureau Windows

Vous savez maintenant ajouter des formulaires à un projet Visual Basic ainsi qu'ouvrir et fermer des formulaires avec le code. Mais quel outil ou paramètre permet-il de déterminer le placement des formulaires sur le bureau Windows pendant l'exécution des programmes ? Comme vous l'aurez remarqué, le placement des formulaires à l'écran

pendant l'exécution ne correspond pas au placement au sein de l'environnement de développement Visual Studio pendant la conception. Dans cette section, vous allez apprendre à positionner les formulaires à l'emplacement de votre choix pendant l'exécution de sorte que les utilisateurs voient ce que vous voulez qu'ils voient.

Dans Visual Basic 6, l'outil graphique Disposition du formulaire contrôle le placement des formulaires pendant l'exécution. Vous faites glisser une minuscule icône de formulaire dans la fenêtre Disposition du formulaire jusqu'à l'endroit où le formulaire doit s'afficher pendant l'exécution et Visual Basic enregistre les coordonnées d'écran indiquées. Dans Visual Basic 2008, la fenêtre Disposition du formulaire n'existe pas, mais il reste possible de positionner précisément les formulaires sur le bureau Windows.

L'outil utilisé n'est pas une fenêtre de mise en place graphique, mais une propriété appelée *DesktopBounds*, conservée pour chaque formulaire du projet. La propriété *DesktopBounds* peut être lue ou modifiée pendant l'exécution. Elle prend comme argument les dimensions d'un rectangle : deux paires de points qui indiquent les coordonnées l'angle supérieur gauche et de l'angle inférieur droit de la fenêtre. Les points de coordonnées s'expriment en pixels et les distances par rapport à l'angle supérieur gauche et à l'angle inférieur droit se mesurent à partir de l'angle supérieur gauche de l'écran (nous étudierons le système de coordonnées de Visual Basic en détail au prochain chapitre). Comme la propriété *DesktopBounds* prend une structure rectangulaire comme argument, elle permet de définir la taille et l'emplacement du formulaire sur le bureau Windows.

Outre la propriété *DesktopBounds*, il existe un mécanisme plus simple, dont les fonctionnalités sont moindres, pour définir l'emplacement d'un formulaire au moment de la conception. Ce mécanisme, la propriété *StartPosition*, positionne un formulaire sur le bureau Windows en se servant de l'un des paramètres de propriété suivants : *Manual*, *CenterScreen*, *WindowsDefaultLocation*, *WindowsDefaultBounds* ou *CenterParent*. Le paramètre par défaut de la propriété *StartPosition*, *WindowsDefaultLocation*, permet à Windows de positionner le formulaire sur le bureau à l'emplacement de son choix, généralement dans l'angle supérieur gauche de l'écran.

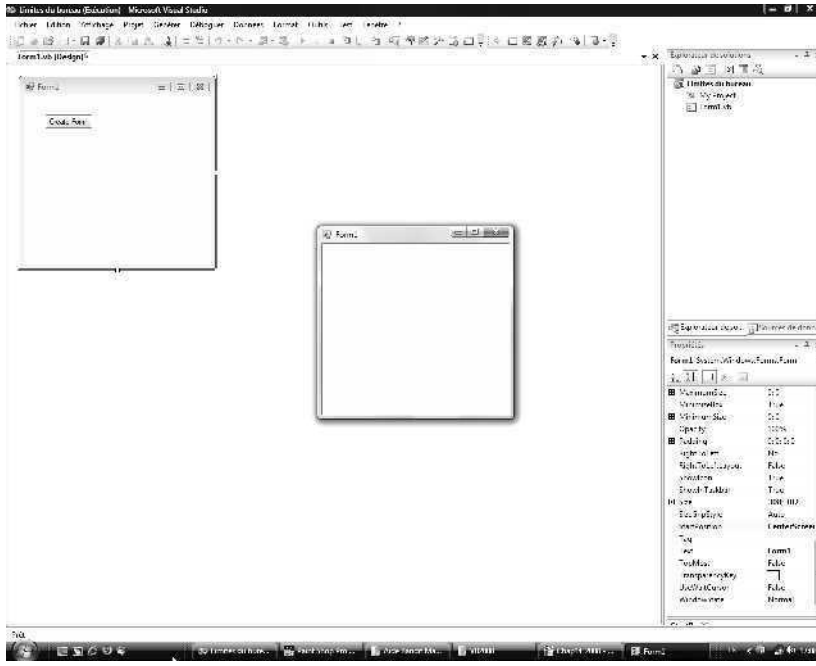
Si vous positionnez *StartPosition* sur *Manual*, vous pouvez définir manuellement l'emplacement du formulaire en vous servant de la propriété *Location*, dans laquelle le premier nombre (*x*) représente la distance à partir du bord gauche de l'écran et le deuxième (*y*) la distance à partir du bord supérieur de l'écran (nous étudierons la propriété *Location* en détail au prochain chapitre). Si vous fixez *StartPosition* à *CenterScreen*, le formulaire s'ouvre au centre du bureau Windows (c'est mon paramètre favori). Si vous la fixez à *WindowsDefaultBounds*, le formulaire est redimensionné pour s'adapter à la taille de la fenêtre standard d'une application Windows, puis il est ouvert à l'emplacement par défaut d'un nouveau formulaire Windows. Si vous attribuez la valeur *CenterParent* à *StartPosition*, le formulaire est centré au sein du formulaire parent. Ce dernier paramètre est particulièrement intéressant dans les applications dites interface à documents multiples (MDI, *multiple document interface*) dans lesquelles les fenêtres parent et enfant possèdent une relation spéciale.

Les prochains exercices montrent comment définir les propriétés *StartPosition* et *DesktopBounds* pour positionner un formulaire Visual Basic. Vous pouvez employer l'une ou l'autre technique pour positionner les formulaires sur le bureau Windows pendant l'exécution.

Utiliser la propriété *StartPosition* pour positionner le formulaire

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms nommé **Mes Limites du bureau**.
2. Si le formulaire du projet n'est pas visible, affichez-le.
3. Cliquez sur le formulaire pour afficher ses propriétés dans la fenêtre Propriétés.
4. Positionnez sa propriété *StartPosition* sur *CenterScreen*.

   Ce faisant, vous indiquez à Visual Basic d'afficher le formulaire au centre du bureau Windows au moment de l'exécution du programme.
5. Cliquez sur le bouton Démarrer le débogage pour exécuter l'application. Visual Basic charge le formulaire et l'affiche au centre de l'écran :



6. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme. L'environnement de développement s'affiche à nouveau.
7. Positionnez sa propriété *StartPosition* sur *Manual*.

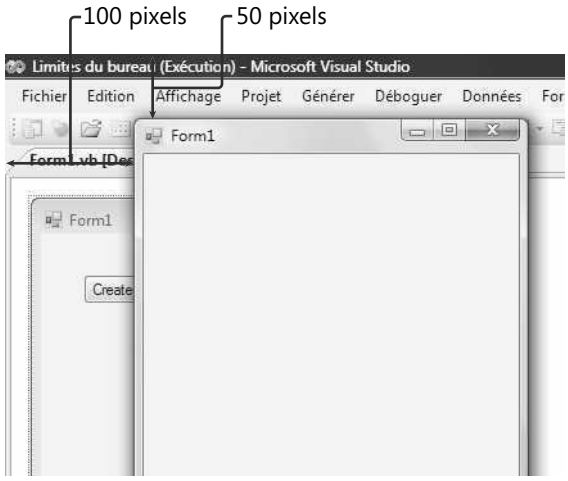
Le paramètre de la propriété *Manual* indique à Visual Basic de positionner le formulaire en fonction des valeurs de la propriété *Location*.

8. Fixez la propriété *Location* sur 100; 50.

   La propriété *Location* indique la position, en pixels, de l'angle supérieur gauche du formulaire.

9. Cliquez sur le bouton Démarrer le débogage pour exécuter l'application.

   Visual Basic charge le formulaire et l'affiche sur le bureau Windows à 100 pixels du bord gauche et 50 pixels du bord supérieur, comme ceci :



10. Cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Vous avez testé quelques paramètres *StartPosition* de base permettant de positionner un formulaire pendant l'exécution. Vous allez maintenant utiliser la propriété *DesktopBounds* pour dimensionner et positionner la fenêtre d'un deuxième formulaire pendant l'exécution du programme. Vous apprendrez également à créer un nouveau formulaire pendant l'exécution sans utiliser la commande Ajouter un formulaire Windows du menu Projet.

Définir la propriété *DesktopBounds*

1. Servez-vous du contrôle *Button* pour ajouter un objet bouton au formulaire et attribuez la valeur « Créer un formulaire » à la propriété *Text* de l'objet bouton.
2. Double-cliquez sur le bouton Créer un formulaire pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.
3. Tapez le code suivant :

```
'Créer un deuxième formulaire intitulé form2
Dim form2 As New Form
```

```
'Définit la propriété Text et le style de bordure du formulaire
form2.Text = "Nouveau formulaire"
form2.FormBorderStyle = FormBorderStyle.FixedDialog

'Indique que la position du formulaire est définie manuellement
form2.StartPosition = FormStartPosition.Manual

'Déclare une structure rectangulaire pour contenir les dimensions du formulaire
'Angle supérieur gauche du formulaire (200; 100)
'Largeur et hauteur du formulaire (300; 250)
Dim form2Rect As New Rectangle(200, 100, 300, 250)

'Définit les limites du formulaire en utilisant l'objet Rectangle
form2.DesktopBounds = form2Rect

'Affiche le formulaire sous forme de boîte de dialogue modale
form2.ShowDialog()
```

Lorsque l'utilisateur clique sur le bouton Créer un formulaire, cette procédure événementielle crée un nouveau formulaire portant le titre « Nouveau formulaire » et un style de bordure fixe. Pour créer un nouveau formulaire avec le code, on se sert d'une instruction *Dim* et on indique un nom de variable pour le formulaire et la classe *Form*, qui est automatiquement incluse dans les projets dans le cadre de l'espace de noms *System.Windows.Forms*. Vous pouvez définir les propriétés comme *Text*, *FormBorderStyle*, *StartPosition* et *DesktopBounds*. La propriété *StartPosition* prend la valeur *FormStartPosition.Manual* pour indiquer que la position va être définie manuellement. La propriété *DesktopBounds* dimensionne et positionne le formulaire et exige un argument de type *Rectangle*. Ce dernier est une structure qui définit une zone rectangulaire, automatiquement incluse dans les projets Visual Basic. L'instruction *Dim* permet de déclarer la variable *form2Rect* de type *Rectangle* et de l'initialiser avec les valeurs de position et de taille du formulaire. Dans la partie inférieure de la procédure événementielle, le nouveau formulaire s'ouvre en tant que boîte de dialogue par le biais de la méthode *ShowDialog*.

Je recommande généralement de rassembler les instructions *Dim* dans la partie supérieure du formulaire, mais dans ce cas, j'en ai placé une un peu plus loin dans le code pour simplifier la compréhension du contexte et l'utilisation de la variable.

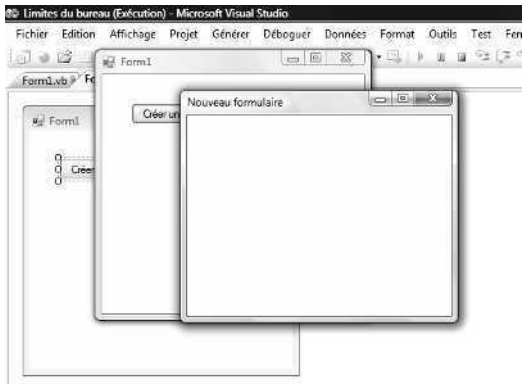
Astuce Le programme Limites du bureau complet est disponible dans le dossier c:\vb08epe\chap14\Limites du bureau.

4. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

   Visual Basic affiche le premier formulaire sur le bureau.

5. Cliquez sur le bouton Créer un formulaire.

Visual Basic affiche la boîte de dialogue Nouveau formulaire dimensionnée et positionnée selon les spécifications du code, comme l'illustre la figure suivante :



Remarquez qu'il est possible de redimensionner le deuxième formulaire, puisque *FormBorderStyle* est positionnée sur *FixedDialog*.

6. Fermez le deuxième formulaire puis le premier.

   L'exécution du programme s'arrête et l'environnement de développement s'affiche à nouveau.

7. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap14.

Réduire, agrandir et restaurer les fenêtres

Outre la définition de la taille et de l'emplacement d'un formulaire Visual Basic, vous pouvez le réduire dans la barre des tâches Windows, l'agrandir de sorte qu'il occupe l'ensemble de l'écran ou restaurer sa taille d'origine. Vous modifiez ces paramètres au moment de la conception ou pendant l'exécution, selon les conditions du programme.

Pour autoriser l'agrandissement et la réduction d'un formulaire, il vous faut d'abord vérifier que les boutons Réduire et Agrandir sont disponibles. En vous servant de la fenêtre Propriétés ou du code, précisez les paramètres suivants :

```
form2.MaximizeBox = True
form2.MinimizeBox = True
```

Ensuite, dans le code ou la fenêtre Propriétés, attribuez la valeur *Minimized*, *Maximized* ou *Normal* à la propriété *WindowState* du formulaire (dans le code, vous devez ajouter la constante *FormWindowState*, comme dans l'exemple suivant). Par exemple, l'instruction suivante réduit le formulaire *form2* dans la barre des tâches Windows :

```
form2.WindowState = FormWindowState.Minimized
```

Pour contrôler la taille maximale ou minimale d'un formulaire, définissez les propriétés *MaximumSize* ou *MinimumSize* au moment de la conception, dans la fenêtre Propriétés.

Pour définir *MaximumSize* ou *MinimumSize* dans le code, vous devez faire appel à une structure *Size* (similaire à la structure *Rectangle* employée dans l'exercice précédent), en procédant comme suit :

```
Dim TailleForm As New Size(400, 300)
MaximumSize = TailleForm
```

Ajouter des contrôles à un formulaire pendant l'exécution

Dans ce livre, vous avez ajouté des objets aux formulaires en vous servant de la Boîte à outils et du Concepteur. Toutefois, le dernier exercice a montré qu'il est également possible de créer des objets Visual Basic sur les formulaires pendant l'exécution, soit pour réduire le temps de développement (si vous copiez des routines employées auparavant) soit pour répondre à un besoin immédiat dans le programme. Vous pouvez, par exemple, générer une boîte de dialogue simple contenant des objets qui traitent l'entrée uniquement sous certaines conditions.

Il est simple de créer des objets puisque les classes fondamentales qui définissent les contrôles dans la Boîte à outils sont disponibles dans tous les programmes. Les objets sont déclarés et instanciés (ou créés) par le biais des mots clé *Dim* et *New*. L'instruction suivante montre comment ce processus fonctionne lorsqu'un nouvel objet bouton intitulé *bouton1* est créé sur le formulaire :

```
Dim bouton1 As New Button
```

Après avoir créé un objet pendant l'exécution, vous pouvez également utiliser le code pour le personnaliser avec les paramètres de ses propriétés. Il est particulièrement intéressant de préciser un nom et un emplacement pour l'objet. En effet, ces propriétés n'ont pas été spécifiées manuellement avec le Concepteur. Par exemple, les instructions suivantes configurent les propriétés *Text* et *Location* pour le nouvel objet *bouton1* :

```
bouton1.Text = "Cliquez sur moi"
bouton1.Location = New Point(20, 25)
```

Pour finir, le code doit ajouter le nouvel objet suivant à la collection *Controls* du formulaire où il a été créé. On rend l'objet visible et actif dans le programme :

```
form2.Controls.Add(bouton1)
```

Si vous ajoutez un nouveau bouton au formulaire en cours (c'est-à-dire si vous ajoutez un bouton à *Form1* et que le code se trouve dans la procédure événementielle *Form1*), vous pouvez faire appel l'objet *Me* à la place. Par exemple,

```
Me.Controls.Add(bouton1)
```

ajoute l'objet *bouton1* à la collection *Controls* du formulaire en cours. Vérifiez que l'objet *bouton1* n'existe pas déjà sur le formulaire auquel vous l'ajoutez (chaque objet doit posséder un nom unique).

Servez-vous de ce processus pour ajouter n'importe quel contrôle de la Boîte à outils à un formulaire Visual Basic. Le nom de la classe employé pour déclarer et instancier le contrôle est une variation du nom qui s'affiche dans la propriété *Name* du contrôle.

Le prochain exercice montre comment ajouter un contrôle *Label* et un contrôle *Button* à un nouveau formulaire pendant l'exécution. Le nouveau formulaire agit comme une boîte de dialogue qui présente la date en cours.

Créer des nouveaux contrôles Label et Button

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms intitulé **Mon Ajouter des contrôles**.
2. Affichez le formulaire (Form1.vb).
3. Servez-vous du contrôle *Button* pour ajouter un objet bouton au formulaire et attribuez la valeur « Afficher la date » à la propriété *Text* de l'objet bouton.
4. Double-cliquez sur le bouton Afficher la date pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.
5. Tapez le code suivant :

```
'Déclare de nouveaux objets formulaire et contrôle
Dim form2 As New Form
Dim lblDate As New Label
Dim btnAnnuler As New Button

'Définit les propriétés de l'étiquette
lblDate.Text = "La date actuelle est : " & DateString
lblDate.Size = New Size(200, 50)
lblDate.Location = New Point(80, 50)

'Définit les propriétés du bouton
btnAnnuler.Text = "Annuler"
btnAnnuler.Location = New Point(110, 100)

'Définit les propriétés du formulaire
form2.Text = "Date"
form2.CancelButton = btnAnnuler
form2.StartPosition = FormStartPosition.CenterScreen

'Ajoute de nouveaux objets à la collection Controls
form2.Controls.Add(lblDate)
form2.Controls.Add(btnAnnuler)

'Affiche le formulaire sous forme de boîte de dialogue
form2.ShowDialog()
```

Cette procédure événementielle affiche à l'écran un nouveau formulaire contenant un objet étiquette et un objet bouton. L'objet étiquette contient la date en cours telle qu'enregistrée dans l'horloge système de l'ordinateur (retournée *via DateString*). La propriété *Text* de l'objet bouton prend la valeur « Annuler ».

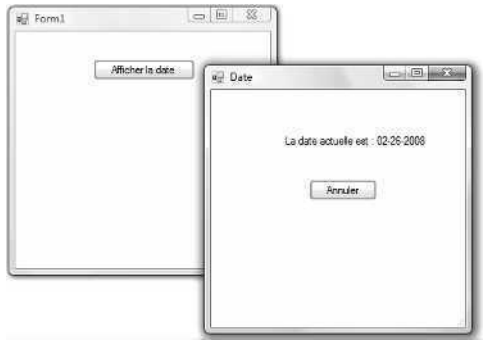
Comme mentionné précédemment, vous ajoutez des contrôles à un formulaire en déclarant une variable qui contiendra le contrôle, en paramétrant les propriétés de l'objet et en ajoutant des objets à la collection *Controls*. Dans cet exercice, j'ai également introduit pour la première fois les propriétés *Size* et *CancelButton*. La propriété *Size* exige une structure *Size*. Le mot clé *New* sert à créer immédiatement la structure *Size*. La propriété *CancelButton* permet à l'utilisateur de fermer la boîte de dialogue en appuyant sur ECHAP ou en cliquant sur le bouton Annuler (ces deux actions sont équivalentes).

6. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap14.

Astuce Le programme Ajouter des contrôles complet est disponible dans le dossier c:\vb08epe\chap14\Ajouter des contrôles.

7. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme. Visual Basic affiche le premier formulaire sur le bureau.
8. Cliquez sur le bouton Afficher la date.

   Visual Basic affiche le deuxième formulaire sur le bureau. Ce formulaire contient les objets étiquette et bouton que vous avez définis dans le code. L'objet étiquette contient la date en cours, comme dans l'exemple suivant :



9. Cliquez sur le bouton Annuler pour fermer le nouveau formulaire.
10. Cliquez à nouveau sur le bouton Afficher la date.

    Le nouveau formulaire s'affiche comme la première fois.
11. Appuyez sur ECHAP pour fermer le formulaire.

    La propriété *CancelButton* étant fixée à l'objet *btnAnnuler*, un clic sur le bouton Annuler ou une pression sur la touche ECHAP produisent le même résultat.
12. Cliquez sur le bouton Fermer du formulaire pour terminer le programme.

    Le programme s'arrête et vous revenez à l'environnement de développement.

Organiser les contrôles sur un formulaire

Lorsque vous ajoutez des contrôles à un formulaire par programmation, plusieurs tentatives sont souvent nécessaires avant de parvenir à positionner les nouveaux objets de sorte qu'ils s'alignent correctement et produisent un aspect organisé. Après tout, le Concepteur Visual Studio n'est pas là pour vous aider. Vous disposez uniquement des coordonnées (x, y) des propriétés *Location* et *Size*, des valeurs peu pratiques à exploiter sauf si vous avez un talent caché pour la pensée bidimensionnelle ou le temps d'exécuter de nombreuses fois le programme pour vérifier le placement des objets.

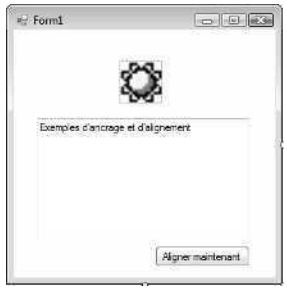
Heureusement, Visual Basic propose plusieurs paramètres de propriétés permettant d'organiser les objets sur un formulaire pendant l'exécution. Citons, par exemple, la propriété *Anchor* qui oblige un objet du formulaire à demeurer à une distance constante des bords spécifiés du formulaire et la propriété *Dock* qui oblige un objet à rester attacher à un bord du formulaire. Vous pouvez exploiter ces propriétés à l'heure de la conception, mais je trouve qu'elles sont également intéressantes pour aligner les objets par programmation pendant l'exécution. L'exercice suivant montre comment fonctionne ces propriétés.

Ancrer et aligner les objets pendant l'exécution

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms intitulé **Mon Ancrer et aligner**.
2. Affichez le formulaire.
3. Cliquez sur le contrôle *PictureBox* et ajoutez un objet zone d'image au centre de la partie supérieure du formulaire.
4. Servez-vous du contrôle *TextBox* pour créer un objet zone de texte.
5. Attribuez la valeur *True* à la propriété *Multiline* de l'objet zone de texte pour vous permettre de redimensionner l'objet.
6. Redimensionnez l'objet zone de texte de sorte qu'il recouvre la moitié inférieure du formulaire.
7. Cliquez sur le contrôle *Button* et ajouter un objet bouton dans l'angle inférieur droit du formulaire.
8. Définissez les propriétés suivantes pour le formulaire et les objets qu'il héberge (vous allez utiliser un fichier image du prochain chapitre. Saisissez exactement le nom du chemin d'accès ou sélectionnez Tous les fichiers dans la zone de liste Type de fichiers pour y trouver soleil.ico).

Objet	Propriété	Paramètre
PictureBox1	*Image*	« c:\vb08epe\chap15\soleil.ico »
	SizeMode	StretchImage
Button1	*Text*	« Aligner maintenant »
TextBox1	*Text*	« Exemples d'ancrage et d'alignement »

Voici à quoi ressemble votre formulaire :



9. Double-cliquez sur le bouton Aligner maintenant pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.

10. Tapez le code suivant :

```
PictureBox1.Dock = DockStyle.Top
TextBox1.Anchor = AnchorStyles.Bottom Or _
     AnchorStyles.Left Or AnchorStyles.Right Or _
     AnchorStyles.Top
Button1.Anchor = AnchorStyles.Bottom Or _
   AnchorStyles.Right
```

Lorsque cette procédure événementielle s'exécute, la propriété *Dock* de l'objet *PictureBox1* ancre la zone d'image sur le bord supérieure du formulaire. Par conséquent, le bord supérieur de l'objet zone d'image touche et adhère au bord supérieur du formulaire, à l'instar de la fonctionnalité d'ancrage dans l'environnement de développement de Visual Studio. Le seul comportement étonnant dans ce cas est le changement de taille de l'objet zone d'image : ses côtés adhèrent aux bords gauche et droit du formulaire.

On fait ensuite appel à la propriété *Anchor* des objets *TextBox1* et *Button1*. Celle-ci maintient la distance actuelle avec les bords spécifiés du formulaire, même si l'on en modifie la taille. Notez que la propriété *Anchor* maintient la distance actuelle de l'objet avec les bords indiqués : elle ne l'attache pas aux bords, sauf s'il s'y trouve déjà. Dans notre exemple, nous avons indiqué que l'objet *TextBox1* doit être ancré aux quatre bords du formulaire (*Bottom*, *Left*, *Right* et *Top*, soit bas, gauche, droite et haut). J'ai fait appel à l'opérateur *Or* pour combiner les sélections des bords. J'ai ancré l'objet *Button1* aux bords bas et droit du formulaire.

11. Enregistrez le projet et choisissez le dossier de destination c:\vb08epe\chap14.

Astuce Le programme Ancrer et aligner complet est disponible dans le dossier c:\vb08epe\chap14\Ancrer et aligner.

12. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme. Le formulaire s'affiche tel que vous l'avez conçu.

13. Placez le curseur sur l'angle inférieur droit du formulaire jusqu'à ce qu'il prenne la forme du pointeur Redimensionner et agrandissez le formulaire.

Remarquez que la taille et la position des objets sur le formulaire ne change pas.

14. Redonnez au formulaire sa taille d'origine.

15. Dans le formulaire, double-cliquez sur le bouton Aligner maintenant.

L'objet zone d'image est à présent ancré au bord supérieur du formulaire. La zone d'image est également redimensionnée de sorte que ses côtés adhèrent aux bords gauche et droit du formulaire :

Notez que l'icône du Soleil dans la zone d'image est déformée, en résultat du processus d'ancrage.

16. Agrandissez à nouveau le formulaire.

À mesure que vous redimensionnez le formulaire, les objets zone d'image et zone de texte changent également de taille. La zone de texte étant ancrée sur les quatre côtés, la distance entre les bords du formulaire et ceux de la zone de texte restent constante. Pendant le changement de taille, il devient également apparent que l'objet bouton est repositionné. Bien que la distance entre l'objet bouton et les bords supérieur et gauche du formulaire changent, la distance que le sépare des bords inférieur et droit demeure constante :

17. Faites plusieurs essais avec les propriétés *Dock* et *Anchor* et essayez différentes images bitmap, si vous le souhaitez. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire pour terminer le programme.

Vous disposez à présent des techniques nécessaires pour ajouter de nouveaux formulaires à un projet, les positionner sur le bureau Windows, les peupler avec de nouveaux contrôles et aligner ces derniers *via* le code. Vous avez appris un certain nombre de techniques utiles dans le cadre des formulaires Windows dans un programme.

Aller plus loin : Spécifier l'objet de démarrage

Dans un projet qui contient plusieurs formulaires, lequel est chargé et affiché en premier lieu lorsque l'on exécute l'application ? Visual Basic charge le premier formulaire créé dans le projet (Form1.vb), mais rien ne vous empêche d'en choisir un autre. Pour ce faire, il vous faut ajuster un paramètre dans le Concepteur de projet Visual Studio, un outil pratique que je vous présente ici pour la première fois.

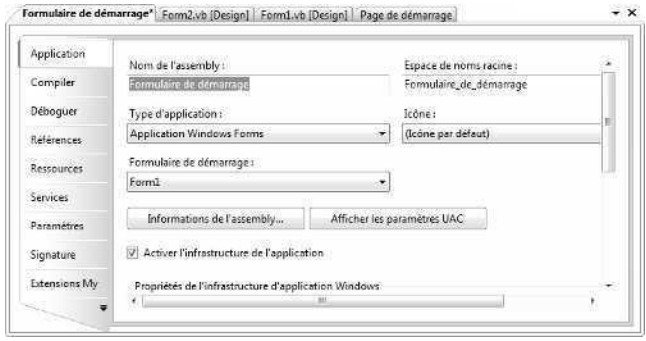
L'exercice suivant montre comment changer le premier formulaire, ou *formulaire de démarrage*, en se servant du Concepteur de projet Visual Studio.

Basculer le formulaire de démarrage de Form1 à Form2

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms intitulé **Formulaire de démarrage**.
2. Affichez Form1.vb, s'il n'est pas déjà visible.
3. Dans le menu Projet, choisissez la commande Ajouter un formulaire Windows.

   Vous allez ajouter un nouveau formulaire au projet pour constater le fonctionnement du changement de formulaire de démarrage.
4. Cliquez sur Ajouter pour ajouter un deuxième formulaire (Form2.vb) à l'Explorateur de solutions.
5. Dans le menu Projet, choisissez Propriétés de Formulaire de démarrage.

   Le Concepteur de projet s'affiche :



Le Concepteur de projet, préalablement intitulé boîte de dialogue « pages des propriétés » en raison de ses multiples écrans de propriétés du projet, permet d'ajuster les paramètres qui s'appliquent à l'ensemble d'un projet à partir d'un emplacement centralisé. Nous allons utiliser l'onglet Application et la liste déroulante Formulaire de démarrage pour désigner le nouveau formulaire de démarrage.

6. Dans l'onglet Application, cliquez sur la flèche de la liste déroulante Formulaire de démarrage et choisissez Form2.

   Visual Basic remplace le formulaire de démarrage Form1 dans le projet par Form2. Lorsque le programme s'exécute, Form2 s'affiche et Form1 ne s'affiche que s'il est ouvert avec la méthode *Show* ou la méthode *ShowDialog*.

7. Cliquez sur le bouton Fermer pour fermer le Concepteur de projet.

8. Cliquez sur le bouton Démarrer le débogage.

   Le programme s'exécute dans l'environnement de développement et Form2 s'affiche.

9. Cliquez sur le bouton Fermer sur le formulaire pour terminer le programme.

10. Fermez le projet et annulez vos changements : il n'est pas nécessaire d'enregistrer ce projet de démonstration. Vous avez terminé la gestion des formulaires pour l'instant.

Bien que cet exercice de démonstration soit relativement simple, vous noterez que Visual Basic offre une souplesse certaine quant au démarrage des programmes. Il est possible de concevoir le formulaire de démarrage et de placer du code dans la procédure événementielle *Load* du formulaire pour configurer le programme ou ajuster ces paramètres avant le chargement réel du premier formulaire.

Applications de console

Pour écrire une application Visual Basic qui ne présente pas du tout d'interface graphique, envisagez la création d'une *application de console*. Ce type de projet Visual Studio traite l'entrée et la sortie par le biais d'une console en ligne de commandes (une fenêtre basée sur les caractères également appelée *invite de commandes*).

Pour ce faire, à l'heure de créer le projet, choisissez la commande Nouveau Projet dans le menu Fichier et sélectionnez le modèle Application console. Vous pouvez également convertir un projet existant en application de console en affichant le Concepteur de projet, et en sélectionnant Application console dans la liste déroulante Type d'application de l'onglet Application. Les applications de console démarrent leur exécution au sein d'une procédure *Sub Main* dans un module de code, puisqu'il n'existe pas de formulaire à afficher. Pour plus d'informations sur le sujet, recherchez « Building Console Applications » dans la documentation Visual Studio.

Rappel du chapitre 14

Pour	Faites ceci
Ajouter un nouveau formulaire à un programme	Dans le menu Projet, choisissez Ajouter un formulaire Windows et cliquez sur Ajouter.

Basculer entre formulaires dans le projet ou ouvrir des formulaires cachés avec le code	Servez-vous de la méthode *Show* ou de la méthode *ShowDialog*. Par exemple :
	`form2.ShowDialog()`
	Vous pouvez également employer l'objet *My.Forms* pour afficher un formulaire. Par exemple :
	`My.Forms.HelpInfo.ShowDialog()`
	Pour masquer le formulaire en cours servez-vous de l'objet *Me*. Par exemple :
	`Me.Visible = False`
	Pour afficher le formulaire masqué, servez-vous également de l'objet *Me*. Par exemple :
	`Me.ShowDialog()`
	Notez que pour utiliser l'objet *Me*, le code doit se trouver au sein du formulaire que vous manipulez.

Créer un nouveau formulaire avec le code et définir ses propriétés	Créez un formulaire en vous servant des mots clé *Dim* et *New* et de la classe *Form*, puis définissez toutes les propriétés nécessaires. Par exemple : `Dim form2 As New Form form2.Text = "Nouveau formulaire"`
Positionner un formulaire de démarrage sur le bureau Windows	Attribuez à la propriété *StartPosition* l'une des valeurs disponibles comme *CenterScreen* ou *CenterParent*.

Dimensionner et positionner un formulaire de démarrage sur le bureau Windows avec le code	Positionnez la propriété *StartPosition* sur *Manual*, déclarez une structure *Rectangle* qui définit la taille et la position du formulaire puis servez-vous de la propriété *DesktopBounds* pour dimensionner et positionner le formulaire sur le bureau. Par exemple :
	`form2.StartPosition = FormStartPosition.Manual`
	`Dim Form2Rect As New Rectangle _`
	`(200, 100, 300, 250)`
	`form2.DesktopBounds = form2Rect`

Agrandir, réduire ou restaurer la taille d'un formulaire pendant l'exécution	Positionnez les propriétés *MaximumSize* et *MinimumSize* du formulaire sur *True* en mode conception pour autoriser les opérations Agrandir et Réduire. Dans le code, attribuez à la propriété *WindowState* du formulaire la valeur *FormWindowState.Minimized*, *FormWindowState.Maximized* ou *FormWindowState.Normal* lorsque vous voulez changer l'état de la fenêtre du formulaire.

Pour	Faites ceci

Ajouter des contrôles à un formulaire pendant l'exécution	Créez un contrôle du type souhaité, définissez ses propriétés et ajoutez-le à la collection *Controls* du formulaire. Par exemple :
	`Dim bouton1 As New Button`
	`bouton1.Text = "Cliquez sur moi"`
	`bouton1.Location = New Point(20, 25)`
	`form2.Controls.Add(bouton1)`

Ancrer un objet à une distance spécifique de bords spécifiques d'un formulaire	Définissez la propriété *Anchor* de l'objet et désignez les bords qui doivent rester à une distance constante de ces bords. Servez-vous de l'opérateur *Or* si vous indiquez plusieurs bords. Par exemple : `Button1.Anchor = AnchorStyles.Bottom Or AnchorStyles.Right`
Aligner un objet sur l'un des bords du formulaire	Définissez la propriété *Dock* de l'objet et désignez les bords auxquels l'objet doit être attaché. Par exemple : `PictureBox1.Dock = DockStyle.Top`
Spécifier le formulaire de démarrage dans un projet	Dans le menu Projet, choisissez la commande Propriétés *Nom du projet* pour ouvrir le Concepteur de projet. Pour un projet Application Windows, vous pouvez désigner n'importe quel formulaire du projet comme formulaire de démarrage en cliquant sur le nom du formulaire dans la liste déroulante Formulaire de démarrage.
Créer un programme Visual Basic sans interface utilisateur (ou uniquement une interface en ligne de commandes)	Créez un projet d'application de console en choisissant la commande Nouveau Projet dans le menu Fichier avant de sélectionner le modèle Application console et de cliquer sur OK. Ajoutez ensuite le code à un ou plusieurs modules, et non à des formulaires. L'exécution démarre par une procédure intitulée *Sub Main*.

Chapitre 15

Ajouter des images et des effets d'animation

À la fin de ce chapitre, vous saurez :

- Utiliser l'espace de noms *System.Drawing* pour ajouter des images à vos formulaires
- Créer des effets d'animation sur vos formulaires
- Élargir ou rétrécir des objets sur un formulaire à l'exécution
- Modifier la transparence d'un formulaire

Pour de nombreux développeurs, l'ajout d'images et d'effets spéciaux dans une application est la partie la plus intéressante – et la plus prenante – de la programmation. Microsoft Visual Basic 2008 permet de créer simplement des effets graphiques à la fois impressionnants et utiles.

Dans ce chapitre, vous allez apprendre à ajouter plusieurs fonctionnalités visuelles intéressantes à vos programmes. Vous verrez comment créer une image attrayante sur un formulaire à l'aide de l'espace de noms *System.Drawing*, créer des effets d'animation simples en utilisant les contrôles *PictureBox* et *Timer* et comment élargir ou rétrécir des objets à l'exécution en utilisant les propriétés *Height* et *Width*. Vous verrez également comment modifier la transparence ainsi que la couleur et l'image de fond du formulaire. Une fois que vous aurez terminé, vous aurez acquis les connaissances dont vous avez besoin pour créer une interface utilisateur visuellement attrayante.

Qu'êtes-vous capable d'accomplir par vous-même ? C'est là que votre imagination doit prendre le dessus. L'idée qui m'a le plus séduite est venue d'un lecteur de la version précédente de ce livre qui a utilisé ce qu'il avait appris sur Visual Basic et l'infographie pour construire son propre électrocardiographe, avec un ensemble de circuits et un formulaire Windows pour afficher les données numériques de sa machine faite maison. Si vous ne partagez pas le même sens de l'humour, vous pouvez améliorer plus modestement la page d'accueil de votre application grâce à une image personnalisée et des effets visuels, associés à une ou plusieurs photographies numériques chargées dans des objets zone d'image sur un formulaire.

Même les programmeurs de jeux peuvent s'amuser avec des images dans Visual Basic et Microsoft Visual Studio. Toutefois, si vous prévoyez de créer la prochaine version de Microsoft Zoo Tycoon ou de Halo, les sorties visuelles ne suffiront pas. Les jeux vidéo modernes contiennent des bibliothèques énormes d'objets et de formules complexes destinés à restituer des images graphiques, qui dépassent de loin la portée de ce livre. Cela nous laisse tout de même une marge considérable pour l'expérimentation et le divertissement !

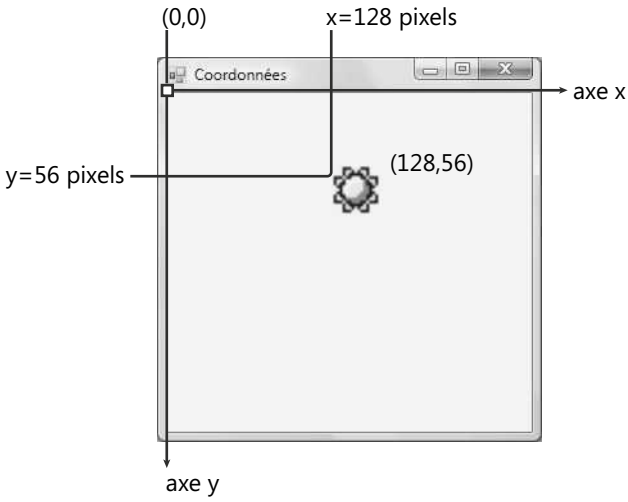
Ajouter des images avec l'espace de noms *System.Drawing*

Dans Visual Basic, il est très simple d'ajouter une image prête à l'emploi. Dans ce livre, vous avez déjà ajouté des images bitmap et des icônes à un formulaire grâce aux objets zone d'image. Vous allez à présent apprendre à créer une image originale grâce aux fonctions GDI+ de l'espace de noms *System.Drawing*, une API (*Application Programming Interface*) du .NET Framework visant à la création de graphiques bidimensionnels, le traitement des images et la typographie au sein du système d'exploitation Windows. Les effets que vous créez servent à ajouter de la couleur, des formes et des textures à vos formulaires.

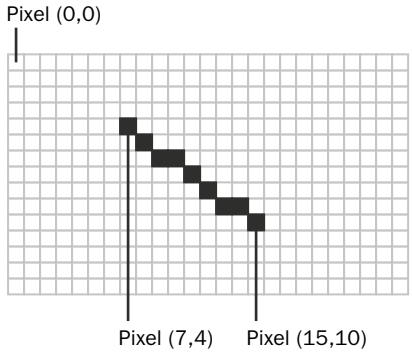
Utiliser un système de coordonnées

En matière de création de graphiques, la première chose à connaître est la présentation du système de coordonnées prédéfini du formulaire. Dans Visual Basic, chaque formulaire possède son propre système de coordonnées. Le point de départ du système, ou *origine*, est situé dans l'angle supérieur gauche du formulaire. Le système de coordonnées par défaut est composé de rangées et de colonnes d'éléments graphiques indépendants, ou *pixels*, qui représentent les plus petits points que l'on peut localiser, ou *gérer*, sur un formulaire Visual Basic.

Dans le système de coordonnées Visual Basic, les rangées de pixels sont alignées sur l'axe des abscisses (l'axe horizontal) et les colonnes de pixels sont alignées sur l'axe des ordonnées (l'axe vertical). Dans le système de coordonnées, on définit les emplacements en identifiant l'intersection d'une rangée et d'une colonne avec la notation (x,y). Les coordonnées (x,y) de l'angle supérieur gauche d'un formulaire est toujours (0,0). La figure qui suit montre comment l'on décrit l'emplacement d'un objet zone d'image dans le système de coordonnées Visual Basic :



Visual Basic fonctionne avec le logiciel d'affichage vidéo de votre ordinateur pour déterminer l'affichage des pixels sur le formulaire ainsi que le rendu de formes tels que les traits, les rectangles, les courbes et les cercles. Il arrive parfois d'activer plusieurs pixels pour afficher une forme particulière, comme le tracé de la figure suivante. La logique qui gère ce type de rendu n'est pas de votre ressort – elle est gérée par votre carte vidéo et les routines de dessin de la bibliothèque graphique GDI+. La figure qui suit montre une vue agrandie de la distorsion ou des bordures en escalier que vous pouvez observer parfois dans Visual Basic et dans des applications Windows :



La classe *System.Drawing.Graphics*

L'espace de noms *System.Drawing* comprend de nombreuses classes pour créer des images et des effets spéciaux dans vos programmes. Dans cette section, vous allez découvrir une partie de la classe *System.Drawing.Graphics*, qui propose des méthodes et des propriétés pour dessiner des formes sur vos formulaires. La documentation de Visual Studio vous permet de découvrir les autres classes.

Qu'il s'agisse de simples illustrations ou de dessins complexes, il est important de pouvoir restituer la plupart des formes géométriques standard dans vos programmes. Le tableau qui suit présente plusieurs formes fondamentales et les méthodes de la classe *System.Drawing.Graphics* employées pour les créer.

Forme	Méthode	Description
Ligne	*DrawLine*	Ligne simple reliant deux points.
Rectangle	*DrawRectangle*	Rectangle ou carré reliant quatre points.
Arc	*DrawArc*	Ligne courbe reliant deux points (portion d'une ellipse).
Cercle/Ellipse	*DrawEllipse*	Forme elliptique "inscrite" dans un rectangle.
Polygone	*DrawPolygon*	Forme complexe avec un nombre variable de points et de côtés (stockés dans un tableau).
Courbe	*DrawCurve*	Ligne courbe qui passe par un nombre variable de points (stockés dans un tableau) ; cette méthode permet également de dessiner des courbes complexes appelées *splines cardinaux*.
Courbe de Bézier	*DrawBezier*	Courbe dessinée grâce à quatre points. Les points deux et trois sont des points de « contrôle ».

Outre ces méthodes, qui créent des formes vides ou « non remplies », il existe plusieurs méthodes pour dessiner des formes remplies de couleur. Elles sont souvent précédées du préfixe « Fill », comme *FillRectangle*, *FillEllipse* et *FillPolygon*.

Lorsque vous utilisez une méthode graphique de la classe *System.Drawing.Graphics*, vous devez créer un objet *Graphics* dans votre code qui représente la classe, et un objet *Pen* ou *Brush* pour indiquer les attributs de la forme que vous souhaitez dessiner, comme la largeur du trait et la couleur de remplissage. L'objet *Pen* est passé comme un des arguments des méthodes qui ne sont pas remplies avec de la couleur. On passe l'objet *Brush* comme argument si l'on souhaite obtenir une couleur de remplissage. Par exemple, l'appel suivant vers la méthode *DrawLine* exploite un objet *Pen* et quatre valeurs entières pour dessiner un trait qui démarre au pixel (20,30) et qui s'achève au pixel (100,80). L'objet *Graphics* est déclaré en utilisant le nom *GraphicsFun*, et l'objet *Pen* est déclaré avec le nom *PenColor*.

```
Dim GraphicsFun AsGraphics
Dim PenColor As NewPen(Color.Red)
GraphicsFun = Me.CreateGraphics
GraphicsFun.DrawLine(PenColor, 20, 30, 100, 80)
```

La syntaxe de la méthode *DrawLine* est importante, mais observez également les trois lignes qui la précèdent ; elles sont nécessaires pour utiliser une méthode de la classe *System.Drawing.Graphics*. Vous devez créer des variables pour représenter les objets *Graphics* et *Pen*, et il faut instancier la variable *Graphics* en utilisant la méthode *CreateGraphics* pour le formulaire Windows. Notez que l'espace de noms *System.Drawing.Graphics* s'inscrit dans votre projet automatiquement – vous n'avez pas besoin d'instruction *Imports* pour référencer la classe.

Utiliser l'événement *Paint* du formulaire

Si vous testez la précédente méthode *DrawLine* dans un programme, vous remarquerez que le trait que vous avez créé dure, ou *persiste*, sur le formulaire tant qu'aucun autre élément ne vient le recouvrir. Si une boîte de dialogue s'affiche momentanément et couvre le trait, il n'est plus visible lorsque le formulaire complet réapparaît. Le trait disparaît également si vous réduisez la fenêtre du formulaire, puis que vous l'agrandissez de nouveau. Pour résoudre ce problème, vous devez placer votre code graphique dans la procédure événementielle *Paint* du formulaire afin qu'à chaque rafraîchissement du formulaire, le graphique soit également redessiné.

Dans l'exercice suivant, vous allez créer trois formes grâce à la procédure événementielle *Paint* du formulaire. Les formes que vous dessinez continuent de persister même si le formulaire est recouvert ou réduit.

Créer des traits, des rectangles et des ellipses

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Dessiner des formes**.

2. Redimensionnez le formulaire pour l'élargir et augmenter sa taille par défaut.

   Vous aurez besoin d'un peu plus de place pour créer ces formes graphiques. Toutefois, vous n'allez pas utiliser les contrôles de la Boîte à outils. Vous allez créer des formes en plaçant du code dans la procédure événementielle *Form1_Paint*.

3. Définissez la propriété *Text* de Form1 à « Dessiner des formes ».

4. Dans l'Explorateur de solutions, cliquez sur le bouton Afficher le code pour afficher l'Éditeur de code.

5. Dans la zone de liste Nom de la classe, cliquez sur Form1 Événements.

   Il s'agit de la liste des événements de votre projet associés à l'objet *Form1*.

6. Dans la zone de liste Nom de la méthode, cliquez sur l'événement *Paint*.

7. La procédure événementielle *Form1_Paint* s'affiche dans l'Éditeur de code.

   Elle correspond à l'endroit où vous allez placer le code qui doit s'exécuter lorsque Visual Basic rafraîchit le formulaire.

8. Tapez le code suivant :

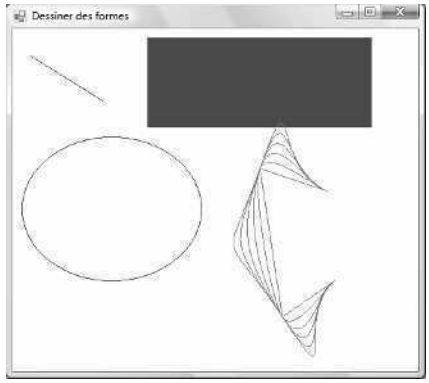
```
'Prépare la variable GraphicsFun pour les appels graphiques
Dim GraphicsFun As Graphics
GraphicsFun = Me.CreateGraphics

'Utilise une couleur de pinceau rouge pour dessiner un trait et une ellipse
Dim PenColor As New Pen(Color.Red)
GraphicsFun.DrawLine(PenColor, 20, 30, 100, 80)
GraphicsFun.DrawEllipse(PenColor, 10, 120, 200, 160)

'Utilise une couleur de pinceau verte pour créer un rectangle plein
Dim BrushColor As New SolidBrush(Color.Green)
GraphicsFun.FillRectangle(BrushColor, 150, 10, 250, 100)

'Crée un spline cardinal bleu à quatre points
Dim Points() As Point = {New Point(358, 280), _
     New Point(300, 320), New Point(275, 155), New Point(350, 180)}
For tension As Single = 0 To 2.5 Step 0.5
     GraphicsFun.DrawCurve(Pens.DodgerBlue, Points, tension)
Next
```

   Cet exemple de procédure événementielle dessine quatre formes graphiques sur votre formulaire : un trait rouge, une ellipse rouge, un rectangle plein vert et un spline cardinal bleu (une courbe complexe composées de cinq traits). Pour activer la programmation de graphiques, la routine déclare une variable appelée *GraphicsFun* dans le code et exploite la méthode *CreateGraphics* pour activer ou instancier la variable. La variable *PenColor* de type *Pen* sert à définir la couleur du trait et de l'ellipse et la variable *BrushColor* de type *SolidBrush* permet de définir la couleur de remplissage du rectangle. Ces exemples ne représentent bien évidemment que la partie visible de l'iceberg – il existe de nombreuses autres formes, couleurs et variations que vous pouvez créer en utilisant des méthodes de la classe *System.Drawing.Graphics*.

Astuce Le programme Dessiner des formes complet se trouve dans le dossier c:\vb08epe\chap15\Dessiner des formes.

9. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Visual Basic charge le formulaire et exécute l'événement *Paint*. Votre formulaire présente un résultat similaire à :



10. Réduisez le formulaire puis restaurez-le de nouveau.

L'événement *Paint* s'exécute de nouveau et les formes graphiques sont rafraîchies sur le formulaire.

11. Cliquez sur le bouton Fermer pour arrêter le programme.

12. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer le projet et choisissez le dossier de destination c:\vb08epe\chap15.

Vous voilà prêt à passer à des effets d'animation simples.

Ajouter une animation à un programme

L'affichage d'images bitmap et de formes graphiques ajoute un intérêt visuel à un programme, mais pour les programmeurs, l'animation a de tout temps été la reine des effets graphiques. L'animation est la simulation de mouvements générée par l'affichage rapide à l'écran d'une série d'images liées. Elle englobe le déplacement d'objets par programmation ainsi qu'une modification de taille ou de forme des images en cours de route.

Dans cette section, vous allez apprendre à ajouter une animation simple à vos programmes. Vous verrez comment mettre à jour les propriétés *Top* et *Left* d'une zone d'image, contrôler la vitesse de l'animation en utilisant un objet minuteur et détecter la bordure de la fenêtre du formulaire.

Déplacer des objets sur le formulaire

Dans Visual Basic 6, une méthode spéciale appelée *Move* permet de déplacer des objets dans le système de coordonnées. Cette méthode n'est plus prise en charge par les contrôles Microsoft Visual Basic 2008. Toutefois, vous pouvez exploiter en lieu et place la méthode et les propriétés présentées dans le tableau suivant.

Mot clé	Description
Top	Cette propriété permet de déplacer verticalement un objet (vers le haut ou vers le bas).
Location	Cette propriété permet de déplacer un objet vers un emplacement spécifique.
SetBounds	Cette méthode définit les frontières d'un objet selon l'emplacement et la taille spécifiés.

Les sections suivantes indiquent comment utiliser les propriétés *Left*, *Top* et *Location* pour déplacer des objets.

Pour déplacer un objet horizontalement, utilisez la propriété *Left*, qui utilise la syntaxe

```
objet.Left = horizontal
```

où *objet* est le nom de l'objet que vous souhaitez déplacer et *horizontal* est la nouvelle coordonnée horizontale, ou coordonnée de l'axe des abscisses, de la bordure gauche de l'objet, exprimée en pixels. Par exemple, l'instruction suivante déplace un objet zone d'image vers un emplacement situé à 300 pixels à droite de la bordure gauche de la fenêtre :

```
PictureBox1.Left = 300
```

Pour se déplacer d'une distance relative vers la droite ou vers la gauche, ajoutez ou soustrayez des pixels du paramètre de propriété *Left* en cours. Par exemple, pour déplacer un objet de 50 pixels vers la droite, ajoutez 50 à la propriété *Left*, comme suit :

```
PictureBox1.Left = PictureBox1.Left + 50
```

De la même manière, vous pouvez modifier l'emplacement vertical d'un objet en définissant la propriété *Top*, qui prend la syntaxe

```
objet.Top = vertical
```

où *objet* est le nom de l'objet que vous souhaitez déplacer et *vertical* est la nouvelle coordonnée verticale, ou coordonnée de l'axe des ordonnées, de la bordure supérieure de l'objet, exprimée en pixels. Par exemple, l'instruction suivante déplace un objet zone d'image à un emplacement situé à 150 pixels en dessous de la barre de titre de la fenêtre :

```
PictureBox1.Top = 150
```

Il est aisé d'effectuer des mouvements relatifs vers le haut ou vers le bas en ajoutant ou en soustrayant des pixels du paramètre de propriété *Top* en cours. Par exemple, pour se déplacer de 30 pixels vers le bas, ajoutez 30 à la propriété *Top* en cours, comme suit :

```
PictureBox1.Top = PictureBox1.Top + 30
```

Propriété *Location*

Pour déplacer un objet à la fois verticalement et horizontalement, utilisez une combinaison des paramètres de propriété *Left* et *Top*. Par exemple, pour repositionner l'angle supérieur gauche d'un objet zone d'image aux coordonnées (x,y) (300,200), tapez le code suivant :

```
PictureBox1.Left = 300
PictureBox1.Top = 200
```

Toutefois, les concepteurs de Visual Studio ne recommandent pas l'utilisation de deux instructions pour repositionner un objet si vous prévoyez d'effectuer de nombreux mouvements dans un programme (par exemple, si vous prévoyez de déplacer un objet des centaines ou des milliers de fois au cours d'un effet d'animation élaboré). Il est préférable d'employer la propriété *Location* avec la syntaxe

```
objet.Location = New Point(horizontal, vertical)
```

où *objet* est le nom de l'objet, *horizontal* est la coordonnée horizontale sur l'axe des abscisses, *vertical* est la coordonnée verticale sur l'axe des ordonnées et *Point* est une structure qui identifie l'emplacement du pixel dans l'angle supérieur gauche de l'objet. Par exemple, l'instruction suivante déplace un objet zone d'image en une coordonnée (x,y) de (300,200) :

```
PictureBox1.Location = New Point(300, 200)
```

Pour effectuer un mouvement relatif grâce à la propriété *Location*, les propriétés *Location.X* et *Location.Y* sont nécessaires. Par exemple, l'instruction

```
PictureBox1.Location = New Point(PictureBox1.Location.X - 50, _
    PictureBox1.Location.Y - 40)
```

déplace l'objet zone d'image de 50 pixels vers la gauche et de 40 pixels vers le haut sur le formulaire. Bien que cette construction puisse sembler un peu lourde, il s'agit du moyen recommandé pour repositionner des objets en mouvements relatifs sur un formulaire à l'exécution.

Créer une animation avec un objet *Timer*

Pour créer une animation dans un programme, l'astuce consiste à placer une ou plusieurs propriétés *Location* mises à jour dans une procédure événementielle de minuteur afin qu'à intervalles donnés, le minuteur entraîne le déplacement d'un ou plus objets à l'écran. Au chapitre 7, « Utiliser les boucles et les minuteurs », vous avez appris à utiliser un objet minuteur pour mettre à jour toutes les secondes une horloge simple afin qu'elle affiche l'heure exacte. Lorsque vous créez une animation, vous définissez la propriété *Interval* du

minuteur à une vitesse bien plus rapide, de l'ordre de 1/5 de seconde (200 millisecondes), 1/10 de seconde (100 millisecondes), voire moins. La vitesse exacte choisie dépend de la vitesse à laquelle vous souhaitez que l'animation s'exécute.

Une autre astuce consiste à utiliser les propriétés *Top* et *Left* et la taille du formulaire pour « détecter » les bordures du formulaire. En utilisant ces valeurs dans une procédure événementielle, vous pouvez interrompre l'animation (désactiver le compteur) lorsqu'un objet atteint la bordure du formulaire. En utilisant la propriété *Top*, la propriété *Left*, les propriétés de taille du formulaire et une structure de décision *If…Then* ou *Select Case*, vous pouvez faire rebondir un objet sur une ou plusieurs bordures du formulaire.

L'exercice qui suit montre comment animer une zone d'image contenant l'icône d'un soleil (Soleil.ico) grâce à la propriété *Location* et un objet minuteur. Dans cet exercice, vous allez utiliser la propriété *Top* pour détecter la bordure supérieure du formulaire et la propriété *Size.Height* pour détecter la bordure inférieure. L'icône Soleil fera des allers-retours entre ces deux extrêmes chaque fois que vous cliquerez sur un bouton.

Animer une icône Soleil sur votre formulaire

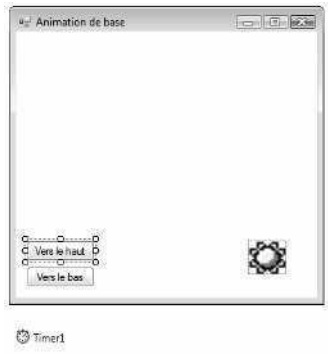
1. Dans le menu Fichier, cliquez sur la commande Fermer le projet, puis créez un nouveau projet Application Windows Forms appelé **Mon Icône animée**.
2. Grâce au contrôle *Button*, dessinez deux objets bouton dans l'angle inférieur gauche du formulaire.
3. À l'aide du contrôle *PictureBox*, dessinez un objet zone d'image rectangulaire de petite taille dans l'angle inférieur droit du formulaire.

   Il s'agit de l'objet que vous allez animer dans le programme.
4. Double-cliquez sur le contrôle *Timer* sur l'onglet Composants de la Boîte à outils pour l'ajouter dans la zone des composants située en dessous du formulaire.

   L'objet minuteur est le mécanisme qui contrôle la vitesse de l'animation. Souvenez-vous que l'objet minuteur lui-même n'est pas visible sur le formulaire ; il apparaît donc dans la zone des composants réservée aux objets invisibles.
5. Définissez les propriétés suivantes pour le bouton, la zone d'image, le minuteur et les objets du formulaire. Pour définir la propriété *Image* de l'objet *PictureBox1*, dans la zone de liste Type de fichiers, sélectionnez Tous les fichiers (par défaut, les fichiers de type .ico ne s'affichent pas).

Objet	Propriété	Paramètres
Button1	*Text*	« Vers le haut »
Button2	*Text*	« Vers le bas »
PictureBox1	*Image*	« c:\vb08epe\chap15\Soleil.ico »
	SizeMode	StretchImage
Timer1	*Interval*	75
Form1	*Text*	« Animation de base »

Une fois ces propriétés définies, votre formulaire se présente ainsi :



6. Double-cliquez sur le bouton Vers le haut pour modifier sa procédure événementielle. La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.
7. Tapez le code suivant :

```
GoingUp = True
Timer1.Enabled = True
```

Cette simple procédure événementielle définit la variable *GoingUp* à True et active l'objet minuteur. Le code qui déplace véritablement l'objet zone d'image et détecte la direction appropriée est stocké dans la procédure événementielle *Timer1_Tick*. La variable *GoingUp* est soulignée d'une ligne dentelée car vous ne l'avez pas encore déclarée.

8. Vers le haut du code, tapez la déclaration de variable suivante, juste en dessous de l'instruction `Public Class Form1` :

```
Dim GoingUp As Boolean 'GoingUp stocke la direction actuelle
```

Cette déclaration de variable rend la variable *GoingUp* disponible pour toutes les procédures événementielles du formulaire. Le soulignement dans la procédure événementielle *Button1_Click* disparaît. J'ai utilisé une variable booléenne car il n'existe que deux directions de mouvement possibles dans ce programme – vers le haut et vers le bas.

9. Affichez le formulaire, double-cliquez sur le bouton Vers le haut et tapez le code suivant dans la procédure événementielle *Button2_Click* :

```
GoingUp = False
Timer1.Enabled = True
```

Cette routine ressemble beaucoup à la procédure événementielle *Button1_Click*, sauf qu'elle change la direction du haut vers le bas.

10. Affichez de nouveau le formulaire, double-cliquez sur l'objet *Timer1* et tapez le code suivant dans la procédure événementielle *Timer1_Tick* :

```
If GoingUp = True Then
     'déplace la zone d'image vers le haut
     If PictureBox1.Top > 10 Then
         PictureBox1.Location = New Point _
         (PictureBox1.Location.X - 10, _
     PictureBox1.Location.Y - 10)
     End If
Else
     'déplace la zone d'image vers le bas
     If PictureBox1.Top < (Me.Size.Height - 75) Then
         PictureBox1.Location = New Point _
             (PictureBox1.Location.X + 10, _
             PictureBox1.Location.Y + 10)
     End If
End If
```

Tant que le minuteur est actif, cette structure de décision *If…Then* s'exécute toutes les 75 millisecondes. La première ligne de la procédure vérifie que la variable booléenne *GoingUp* est définie à True, indiquant ainsi que l'icône se déplace vers le haut du formulaire. Dans le cas contraire, la procédure déplace l'objet zone d'image vers une position relative de 10 pixels plus proche à la fois de la bordure supérieure et gauche du formulaire.

Si la variable *GoingUp* est actuellement définie à False, la structure de décision déplace alors l'icône vers le bas. Dans ce cas, l'objet zone d'image se déplace jusqu'à ce que la bordure du formulaire soit détectée. La hauteur du formulaire peut être déterminée grâce à la propriété *Me.Size.Height*. Je soustrais 75 de la hauteur afin que l'icône s'affiche toujours sur le formulaire. Dans cet exemple, l'objet *Me* représente le formulaire (*Form1*).

Comme vous allez le voir lorsque vous exécuterez le programme, ce mouvement octroie à l'animation de l'icône une qualité de déplacement constante. Pour que l'icône se déplace plus vite, il faut diminuer le paramètre *Interval* de l'objet minuteur. Pour que l'icône se déplace plus lentement, il faut l'augmenter.

Exécuter le programme Icône animée

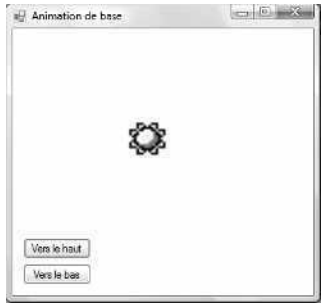
Astuce Le programme Icône animée complet se trouve dans le dossier c:\vb08epe\chap15\Icône animée.

1. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

Le programme Icône animée s'exécute dans l'environnement de développement.

2. Cliquez sur le bouton Vers le haut.

L'objet zone d'image se déplace dans le formulaire en diagonale, comme suit :



Après quelques instants, le bouton se stabilise sur la bordure supérieure du formulaire.

Remarque Si vous avez placé l'objet zone d'image dans l'angle inférieur droit du formulaire, comme indiqué à l'étape 3 de l'exercice précédent, vous obtiendrez un résultat similaire à cette figure. Toutefois, si vous avez placé l'objet zone d'image à un autre emplacement, ou créé un formulaire plus petit, l'image aurait pu glisser en dehors de l'écran si vous aviez cliqué sur le bouton Vers le haut ou Vers le bas. Pouvez-vous dire pourquoi ?

3. Cliquez sur le bouton Vers le bas.

   La zone d'image se déplace de nouveau vers le bas vers l'angle inférieur droit de l'écran.

4. Cliquez sur les deux boutons à plusieurs reprises et réfléchissez aux effets d'animation.

   Notez que vous n'avez pas besoin d'attendre la fin d'un effet d'animation avant de cliquer sur le bouton suivant. La procédure événementielle *Timer1_Tick* utilise immédiatement la variable *GoingUp* pour gérer les requêtes de direction. Peu importe alors que la zone d'image ait terminé sa course dans une direction donnée. Observez quelques instants cet effet, et imaginez comment utiliser un type de logique similaire pour construire vos propres jeux vidéo Visual Basic. La vitesse de l'animation augmente ou diminue en fonction de conditions spécifiques, ou « collisions ». Il est possible de contraindre le déplacement des objets animés dans plusieurs directions. Vous pouvez également modifier l'image affichée par l'objet zone d'image en fonction de l'emplacement de l'icône sur l'écran ou les conditions qu'elle rencontre.

5. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire pour arrêter la démonstration.

6. Cliquez sur le bouton Enregistrer tout pour enregistrer le projet et choisissez le dossier de destination c:\vb08epe\chap15.

Élargir et réduire des objets pendant l'exécution d'un programme

Outre les propriétés *Top* et *Left*, Visual Basic maintient une propriété *Height* et une propriété *Width* pour la plupart des objets d'un formulaire. Vous pouvez utiliser intelligemment ces propriétés pour élargir et rétrécir des projets pendant l'exécution d'un programme. L'exercice suivant indique comment procéder.

Élargir une zone d'image à l'exécution

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet.
2. Créez un nouveau projet Application Windows Forms appelé Mon **Zoomer**.
3. Affichez le formulaire. Dans la Boîte à outils, cliquez sur le contrôle *PictureBox*, puis dessinez un objet zone d'image de petite taille près de l'angle supérieur gauche du formulaire.
4. Définissez les propriétés suivantes pour la zone d'image et le formulaire. Lorsque vous définissez les propriétés de la zone d'image, notez les valeurs en cours dans les propriétés *Height* et *Width* au sein de la propriété *Size*. Il est également possible de les définir à la conception. Comme il s'agit d'une image de l'espace, nous employons un fond noir pour le formulaire et une image .jpg d'étoiles comme arrière-plan. Ces deux propriétés de formulaire, *BackColor* et *BackGroundImage*, sont employées ici pour la première fois dans ce livre.

Objet	Propriété	Paramètre
PictureBox1	*Image*	« c:\vb08epe\chap15\Terre.jpg »
	SizeMode	StretchImage
Form1	*Text*	« Arrimage terre »
	BackColor	Black
	BackgroundImage	« c:\vb08epe\chap15\space.jpg »

5. Double-cliquez sur l'objet *PictureBox1*.

   La procédure événementielle *PictureBox1_Click* s'affiche dans l'Éditeur de code.

6. Tapez le code suivant dans la procédure événementielle *PictureBox1_Click* :

```
PictureBox1.Height = PictureBox1.Height + 15
PictureBox1.Width = PictureBox1.Width + 15
```

7. Ces deux lignes augmentent la hauteur et la largeur de l'icône Terre de 15 pixels chaque fois que l'utilisateur clique sur la zone d'image. En faisant un peu appel à votre imagination, cet effet donne l'impression d'approcher de la terre comme si vous étiez à bord d'une navette spatiale.

8. Cliquez sur le bouton Enregistrer tout, puis enregistrez le projet dans le dossier c:\vb08epe\chap15.

Astuce Le programme Zoomer complet se trouve dans le dossier c:\vb08epe\chap15\Zoomer.

9. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

   L'icône Terre s'affiche seule sur le formulaire.

   Vous voyez des étoiles à l'arrière-plan parce que vous avez chargé le fichier space.jpg dans le formulaire à l'aide de la propriété *BackImage*. Toute zone non couverte par la propriété *BackImage* sur le formulaire est noire parce que vous avez employé la propriété *BackColor* pour simuler la mélancolie tranquille de l'espace profond.

10. Cliquez à plusieurs reprises sur l'icône Terre pour l'élargir.

    Après 10 ou 11 clics, votre écran se présente ainsi :

   Comme l'image possède une résolution relativement faible, elle va devenir quelque peu brouillée si vous l'agrandissez encore. Vous pourriez éviter cela en enregistrant une image plus petite avec une résolution supérieure. Les nuages présents sur l'image atténuent toutefois un peu dans cet exemple cet effet de brouillage. L'image imprimée ne rend pas très bien : essayez ce programme sur votre ordinateur !

11. Une fois que vous avez atteint une orbite standard, cliquez sur le bouton Fermer pour quitter le programme.

    Le programme s'arrête et vous revenez à l'environnement de développement.

Aller plus loin : Modifier la transparence d'un formulaire

Vous êtes intéressé par un dernier effet spécial ? GDI+ vous permet d'accomplir des tâches difficiles, voire impossibles avec les précédentes versions de Visual Basic. Par exemple, vous pouvez rendre un formulaire partiellement transparent afin d'y voir à travers. Supposons que vous conceviez un programme de diaporama contenant un formulaire distinct avec plusieurs options de manipulation des photos. Vous pouvez rendre le formulaire d'options partiellement transparent afin que l'utilisateur puisse visualiser toutes les photos en dessous tout en ayant accès aux options.

Dans l'exercice suivant, vous allez modifier la transparence d'un formulaire en changeant la valeur de la propriété *Opacity*.

Définir la propriété *Opacity*

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet.
2. Créez un nouveau projet Application Windows Forms appelé **Mon Formulaire transparent**.
3. Affichez le formulaire, cliquez sur le contrôle *Button* dans le Boîte à outils, puis dessinez deux boutons.
4. Définissez les propriétés suivantes pour les deux boutons et le formulaire :

Objet	Propriété	Paramètre
Button1	*Text*	« Définir opacité »
Button2	*Text*	« Restaurer »
Form1	*Text*	« Formulaire transparent »

5. Dans le formulaire, double-cliquez sur le bouton Définir opacité.
6. Tapez le code suivant dans la procédure événementielle *Button1_Click* :

```
Me.Opacity = 0.75
```

   Opacity est spécifiée sous forme de pourcentage. Elle est donc située dans une plage allant de 0 à 1. Cette ligne définit la propriété *Opacity* de Form1 (*Me*) à 75 pour cent.

7. Affichez de nouveau le formulaire, double-cliquez sur le bouton Restaurer et tapez le code suivant dans la procédure événementielle *Button2_Click* :

```
Me.Opacity = 1
```

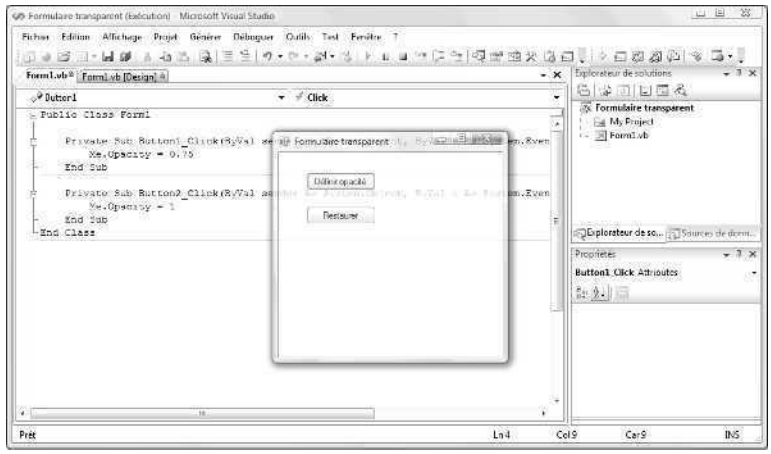
   Cette ligne restaure l'opacité à 100 pour cent.

8. Cliquez sur le bouton Enregistrer tout, puis enregistrez le projet dans le dossier c:\vb08epe\chap15.

> **Astuce** Le programme Formulaire transparent complet se trouve dans le dossier c:\vb08epe\chap15\Formulaire transparent.

9. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.
10. Cliquez sur le bouton Définir opacité.

    Remarquez que vous pouvez voir à travers le formulaire, comme suit :



11. Cliquez sur le bouton Restaurer.

    L'effet de transparence est supprimé.
12. Une fois que vous avez testé la transparence, cliquez sur le bouton Fermer pour quitter le programme.

    Le programme s'arrête et vous revenez à l'environnement de développement.

Rappel du chapitre 15

Pour	Faites ceci

Créer des traits ou des formes sur un formulaire	Utilisez les méthodes de l'espace de noms *System.Drawing.Graphics*. Par exemple, les instructions suivantes dessinent une ellipse sur le formulaire :
	`Dim GraphicsFun As Graphics`
	`GraphicsFun = Me.CreateGraphics`
	`Dim PenColor As New Pen(Color.Red)`
	`GraphicsFun.DrawEllipse(PenColor, 10, 120, 200, 160)`

Créer des traits ou des formes persistantes sur le formulaire pendant le rafraîchissement de la fenêtre	Placez les méthodes graphiques dans la procédure événementielle *Paint* du formulaire.
Déplacer un objet sur un formulaire	Repositionnez l'objet en utilisant la propriété *Location*, le mot clé *New* et la structure *Point*. Par exemple : `PictureBox1.Location = New Point(300, 200)`
Animer un objet	Utilisez une procédure événementielle de compteur pour modifier les propriétés *Left*, *Top* ou *Location* d'un objet sur le formulaire. La propriété *Interval* du compteur contrôle la vitesse de l'animation.
Élargir ou rétrécir un objet à l'exécution	Modifiez la propriété *Height* ou *Width* de l'objet.
Définir la couleur de fond d'un formulaire	Modifiez la propriété *BackColor* du formulaire.
Définir l'image d'arrière-plan d'un formulaire	Modifiez la propriété *BackgroundImage* du formulaire.
Modifier la transparence d'un formulaire	Modifiez la propriété *Opacity* du formulaire.

Chapitre 16

Gérer l'héritage de formulaire et créer des classes de base

À la fin de ce chapitre, vous saurez :

- Utiliser le Sélecteur d'héritage pour incorporer des formulaires existants à vos projets
- Créer vos propres classes de base avec des propriétés et des méthodes personnalisées
- Faire dériver de nouvelles classes de classes de base à l'aide de l'instruction *Inherits*

La compréhension et l'utilisation des techniques de programmation orientée objet (OOP, *object-oriented programming*) doivent aujourd'hui faire impérativement partie des connaissances de tout développeur de logiciels professionnels. Les modifications associées à la programmation orientée objet se sont accentuées dans les dernières versions de Visual Basic. Bien que Microsoft Visual Basic 6 propose plusieurs fonctionnalités de programmation orientée objet, les experts s'accordent à dire que l'absence d'*héritage*, un mécanisme qui permet à une classe d'acquérir les caractéristiques de l'interface et du comportement d'une autre classe, le laisse à la traîne des « vrais » langages de programmation orientée objet, comme Microsoft Visual C++.

À partir de Microsoft Visual Basic .NET 2002, le langage et l'environnement de développement Visual Basic *ont* pris en charge l'héritage : il est possible de construire un formulaire dans l'environnement de développement et de passer ses caractéristiques et fonctionnalités à d'autres formulaires. Il est, de surcroît, possible de construire ses propres classes et d'en hériter les propriétés, méthodes et événements. Ces capacités ont été renforcées dans Microsoft Visual Studio 2008.

Dans ce chapitre, nous étudierons deux types d'héritage. Vous apprendrez à intégrer des formulaires existants dans vos projets en vous servant de la boîte de dialogue Sélecteur d'héritage qui fait partie de Visual Studio 2008 et verrez comment créer vos propres classes et en faire dériver de nouvelles *via* l'instruction *Inherits*. Ces techniques vous permettront d'utiliser la majorité des formulaires et routines que vous avez déjà développés, rendant la programmation Visual Basic plus rapide et plus souple. Avec ces optimisations, vous créerez des interfaces utilisateur convaincantes et étendrez les tâches réalisées dans les autres projets de programmation.

Hériter un formulaire avec le Sélecteur d'héritage

Dans la syntaxe de la programmation orientée objet, l'*héritage* signifie qu'une classe reçoit les objets, propriétés, méthodes et autres attributs d'une autre classe. Comme je l'ai mentionné dans la section « Ajouter de nouveaux formulaires à un programme » du chapitre 14, « Gérer les formulaires et les contrôles Windows à l'exécution », Visual Basic suit ce processus lorsqu'il crée un nouveau formulaire dans l'environnement de développement. Le premier formulaire d'un projet (Form1) se fonde sur la classe *System.Windows.Forms.Form* pour sa définition et ses valeurs par défaut. En fait, cette classe est identifiée dans la fenêtre Propriétés si vous sélectionnez un formulaire dans le Concepteur, comme dans l'illustration suivante :

Même si vous ne l'avez pas encore réalisé, vous avez utilisé l'héritage depuis le début pour définir les formulaires Windows qui ont servi à créer vos applications Visual Basic. Bien qu'il soit possible d'hériter les formulaires existants par le biais du code, les concepteurs de Microsoft Visual Studio ont considéré que la tâche était suffisamment importante pour lui consacrer une boîte de dialogue spéciale dans l'environnement de développement et simplifier ainsi le processus. Vous accédez à cette boîte de dialogue, appelée Sélecteur d'héritage, à l'aide de la commande Ajouter un nouvel élément du menu Projet. Dans le prochain exercice, vous utiliserez le Sélecteur d'héritage pour créer un deuxième exemplaire d'une boîte de dialogue dans un projet.

Hériter une boîte de dialogue simple

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms nommé **Mon Héritage de formulaire**.
2. Affichez le formulaire dans le projet et servez-vous du contrôle *Button* pour ajouter deux objets bouton dans la partie inférieure du formulaire, placés côte à côte.
3. Affectez aux propriétés *Text* des boutons *Button1* et *Button2* respectivement les valeurs « OK » et « Annuler ».
4. Double-cliquez sur le bouton OK pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.
5. Tapez l'instruction suivante :

```
MsgBox("Vous avez cliqué sur OK")
```

6. Affichez à nouveau le formulaire, double-cliquez sur le bouton Annuler et tapez l'instruction suivante dans la procédure événementielle *Button2_Click* :

```
MsgBox("Vous avez cliqué sur Annuler")
```

7. Affichez à nouveau le formulaire et attribuez la valeur « Boîte de dialogue » à la propriété *Text* du formulaire.

   Vous disposez à présent d'un formulaire simple qui peut servir de base à une boîte de dialogue dans un programme. Avec quelques personnalisations, vous pourrez exploiter ce formulaire basique pour traiter diverses tâches : il suffit d'ajouter les contrôles spécifiques à votre application.

8. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap16.

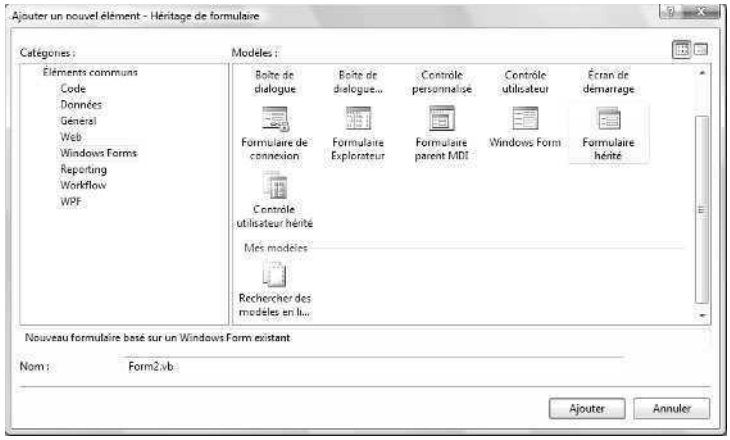
   Testons à présent l'héritage de formulaire. La première phase du processus consiste à créer, ou *compiler*, le projet. En effet, vous ne pouvez hériter que de formulaires compilés en fichiers .exe ou .dll. A chaque recompilation du formulaire de base, les modifications qui lui sont apportées sont transmises au formulaire dérivé (hérité).

9. Dans le menu Générer, choisissez la commande Générer Mon Héritage de formulaire.

   Visual Basic compile le projet et crée un fichier .exe.

10. Dans le menu Projet, choisissez la commande Ajouter un nouvel élément. Cliquez sur la catégorie Windows Forms dans la gauche de la boîte de dialogue puis sur le modèle Formulaire hérité, dans le volet droit de la boîte de dialogue.

    La boîte de dialogue Ajouter un nouvel élément se présente comme suit.

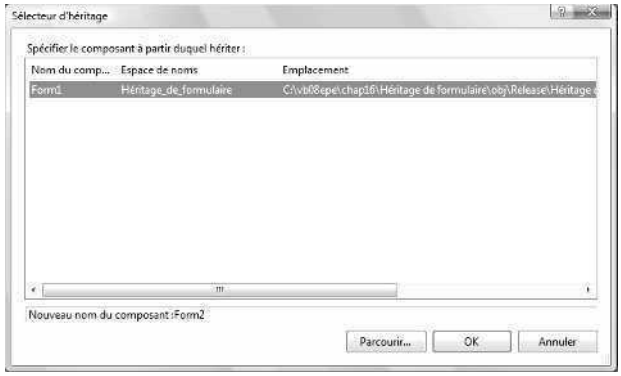


Comme à l'accoutumée, Visual Studio présente tous les modèles qu'il est possible d'inclure dans les projets et pas uniquement ceux relatifs à l'héritage. Le modèle Formulaire hérité donne accès à la boîte de dialogue Sélecteur d'héritage.

Vous pouvez également vous servir de la zone de texte Nom qui se trouve dans la partie inférieure de la boîte de dialogue pour assigner un nom au formulaire hérité, même si cela est superflu dans cet exemple. Ce nom apparaîtra dans l'Explorateur de solutions et dans le nom de fichier du formulaire sur le disque.

11. Cliquez sur Ajouter pour accepter les paramètres par défaut du nouveau formulaire hérité.

Visual Studio affiche la boîte de dialogue Sélecteur d'héritage :

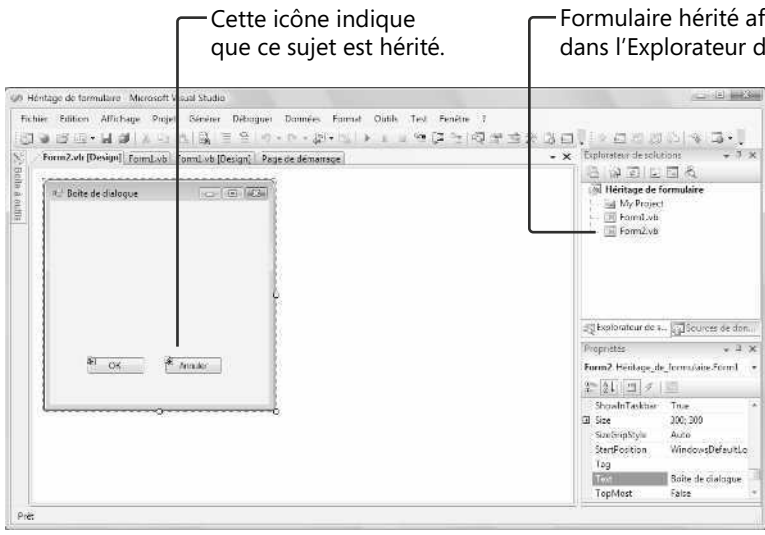


Cette boîte de dialogue présente tous les formulaires héritables du projet en cours. Pour rechercher un autre formulaire compilé, cliquez sur le bouton Parcourir et localisez le fichier .dll sur votre système.

Remarque Pour hériter d'un formulaire qui n'est pas un composant du projet en cours, le formulaire doit être compilé en tant que fichier .dll.

12. Dans la boîte de dialogue Sélecteur d'héritage, cliquez sur Form1 puis sur OK.

Visual Studio crée l'entrée Form2.vb dans l'Explorateur de solutions et affiche le formulaire hérité dans le Concepteur. Dans la figure suivante, remarquez que le formulaire semble identique au Form1 créé précédemment, excepté que les deux boutons contiennent de petites icônes qui indiquent que les objets proviennent d'une source héritée.



Il peut être difficile de différencier un formulaire hérité du formulaire de base (les petites icônes d'héritage ne sont pas claires), mais vous pouvez vous servir de l'Explorateur de solutions et des onglets de l'environnement de développement pour faire la distinction entre les formulaires.

Vous allez à présent ajouter quelques nouveaux éléments au formulaire hérité.

Personnaliser le formulaire hérité

1. Servez-vous du contrôle *Button* pour ajouter un troisième objet bouton à Form2 (le formulaire hérité).
2. Attribuez la valeur « Cliquez sur moi ! » à sa propriété *Text*.
3. Double-cliquez sur le bouton Cliquez sur moi !
4. Dans la procédure événementielle *Button3_Click*, tapez l'instruction suivante :

   ```
   MsgBox("Voici le formulaire hérité !")
   ```

5. Affichez à nouveau Form2 puis essayez de double-cliquer sur les boutons OK et Annuler du formulaire.

   Vous ne pouvez pas afficher ou modifier les procédures événementielles ou propriétés de ces objets hérités sans procéder à des actions qui sortent de la portée de ce chapitre (les minuscules icônes « verrous » indiquent que les objets hérités sont en lecture seule). Il est toutefois possible d'ajouter de nouveaux objets au formulaire pour le personnaliser.

6. Agrandissez le formulaire.

   Vous pouvez également modifier d'autres caractéristiques du formulaire, comme sa taille et son emplacement. Si vous utilisez la fenêtre Propriétés pour personnaliser un formulaire, la zone de liste Objet indique le formulaire duquel celui-ci est dérivé.

   Désignons à présent Form2 comme objet de démarrage.

7. Dans le menu Projet, choisissez la commande Propriétés Mon Héritage de formulaire.

   Le Concepteur de projet, présenté au chapitre 14 « Gérer les formulaires et les contrôles Windows à l'exécution », s'affiche.

8. Dans l'onglet Application, cliquez sur la zone de liste déroulante Formulaire de démarrage, choisissez Form2 et fermez le Concepteur de projet.

   Exécutez le projet.

9. Cliquez sur le bouton Démarrer le débogage.

   Le formulaire hérité s'ouvre, comme le montre la figure suivante (ma version est montrée un peu agrandie suite à l'étape 6 de cet exercice) :

10. Cliquez sur le bouton OK.

    Le formulaire hérité exécute la procédure événementielle dont il a hérité de Form1, puis la procédure événementielle affiche la boîte de message suivante :

11. Cliquez sur OK puis sur le bouton Cliquez sur moi !

Form2 affiche le message du formulaire hérité.

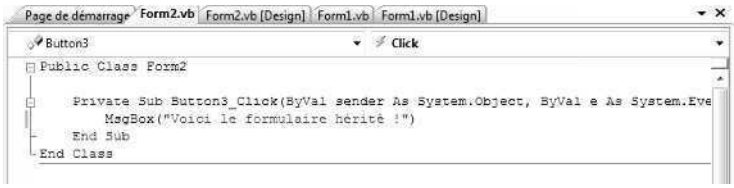
Ceci démontre que Form2 (le formulaire hérité) possède ses propres caractéristiques (un nouveau bouton Cliquez sur moi ! et une taille augmentée). Form2 se sert également de deux boutons (OK et Annuler) hérités de Form1 et renferme le code de Form1, ainsi que la représentation visuelle exacte des boutons. Cela signifie que vous pouvez redéployer les caractéristiques de l'interface utilisateur et du code précédemment créés sans de laborieux copier-coller. En d'autres termes, vous venez de découvrir un des intérêts majeurs de la programmation orientée objet : le réemploi et l'extension de fonctionnalités existantes. Vous avez également appris à employer la boîte de dialogue Sélecteur d'héritage de Visual Studio, qui offre une méthode simple pour sélectionner les objets que vous voulez réemployer.

12. Cliquez sur OK pour fermer la boîte de message puis cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Le programme s'arrête et vous revenez à l'environnement de développement.

Créer vos propres classes de base

Dans l'exercice précédent, le Sélecteur d'héritage gère le processus d'héritage en créant une nouvelle classe dans le projet, intitulée *Form2*. Pour créer la classe *Form2*, le Sélecteur d'héritage établit un lien entre la classe *Form1* du projet Mon Héritage de formulaire et le nouveau formulaire. Voici à quoi ressemble la classe *Form2* dans l'Éditeur de code :



La procédure événementielle *Button3_Click* que vous avez ajoutée est également membre de la nouvelle classe. Mais rappelez-vous que la classe *Form1* est elle-même fondée sur la classe *System.Windows.Forms.Form* pour son comportement et ses caractéristiques fondamentaux. L'exercice précédent a ainsi montré qu'une classe dérivée (*Form2*) peut hériter ses fonctionnalités d'une autre classe dérivée (*Form1*), qui à son tour hérite sa fonctionnalité principale d'une classe de base d'origine (*Form*), membre de l'espace de noms *System.Windows.Forms.Form* de la bibliothèque Microsoft .NET Framework.

Astuce Outre le Sélecteur d'héritage, Visual Studio propose l'instruction *Inherits* par laquelle la classe en cours hérite des propriétés, procédures et variables d'une autre classe. Pour utiliser l'instruction *Inherits* pour hériter un formulaire, placez cette instruction en tête du formulaire en tant que première instruction de la classe. Même si vous optez pour le Sélecteur d'héritage pour exploiter ce type de formulaires, il est intéressant de connaître *Inherits*. Cette instruction peut, en effet, servir dans les classes et les interfaces autres que les formulaires et vous l'exécuterez sans doute de temps à autre dans le code de vos collègues. Nous en verrons un exemple à la fin de ce chapitre.

Reconnaissant l'importance des classes dans les programmes Visual Basic, vous pourriez vous demander comment les nouvelles classes sont créées et comment des classes dérivées ultérieurement peuvent en hériter. Pour réfléchir à ces possibilités, j'ai consacré le reste de ce chapitre à la syntaxe permettant de créer des classes dans Visual Basic 2008 et à expliquer comment d'autres classes peuvent ensuite hériter de ces classes définies par l'utilisateur. Au cours de cette étude, vous découvrirez combien il est intéressant de créer ses propres classes.

Alerte terminologique

Il existe un danger potentiel de surcharge terminologique lors de l'étude de la création et de l'héritage de classes. Un certain nombre de très savants chercheurs en informatique réfléchissent depuis plusieurs années à ces concepts de programmation orientée objet. Il en résulte un nombre important de termes et de définitions employés dans le cadre des concepts que j'ai prévu de développer ici. Toutefois, si vous vous en tenez à mes explications, vous découvrirez que la création de classes et leur héritage sont des concepts simples dans Visual Basic 2008 et qu'ils permettent d'accomplir un grand nombre de tâches en ajoutant simplement quelques lignes de code à vos projets. Comprendre la terminologie orientée objet vous aidera également à comprendre certaines des fonctionnalités avancées de Visual Basic 2008, comme LINQ (*Language-Integrated Query*), les types anonymes, les méthodes d'extension et les expressions lambda. Ces techniques facilitent l'emploi des classes, objets et méthodes et sont parfois mises en avant dans les annonces publicitaires et les listes de nouveaux dispositifs.

Ajouter une nouvelle classe au projet

Pour faire simple, dans Visual Basic, une *classe* est une représentation ou un *plan* qui définit la structure d'un ou de plusieurs objets. En créant une nouvelle classe, vous définissez vos propres objets dans un programme : des objets qui possèdent leurs propriétés, méthodes, champs et événements, à l'instar des objets créés à l'aide des contrôles de la Boîte à outils sur des formulaires Windows. Pour ajouter une nouvelle classe à un projet, dans le menu Projet, cliquez sur la commande Ajouter une classe puis définissez la classe à l'aide du code et de quelques mots clé Visual Basic.

Dans le prochain exercice, vous allez créer un programme qui demande à un nouvel employé son prénom, son nom et sa date de naissance. Vous stockerez des informations dans les propriétés d'une nouvelle classe intitulée *Personne* et créerez une méthode dans la classe pour calculer l'âge actuel du nouvel employé. Ce projet vous apprend à créer vos propres classes et explique comment utiliser les classes dans les procédures événementielles des programmes.

Générer le projet Classe Personne

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms intitulé **Ma Classe Personne**.
2. Servez-vous du contrôle *Label* pour ajouter un objet étiquette dans la partie supérieure de Form1.
3. Servez-vous du contrôle *TextBox* pour dessiner deux larges objets zone de texte sous l'objet étiquette.
4. Servez-vous du contrôle *DateTimePicker* pour dessiner un objet sélecteur de date et d'heure sous les deux objets zone de texte.

   Vous avez utilisé le contrôle *DateTimePicker* pour la dernière fois au chapitre 3, « Travailler avec les contrôles de la Boîte à outils ». Reportez-vous à ce chapitre pour réviser les méthodes et les propriétés de base de ce contrôle.
5. Servez-vous du contrôle *Button* pour dessiner un objet bouton sous l'objet sélecteur de date et d'heure.
6. Définissez les propriétés suivantes pour les objets du formulaire :

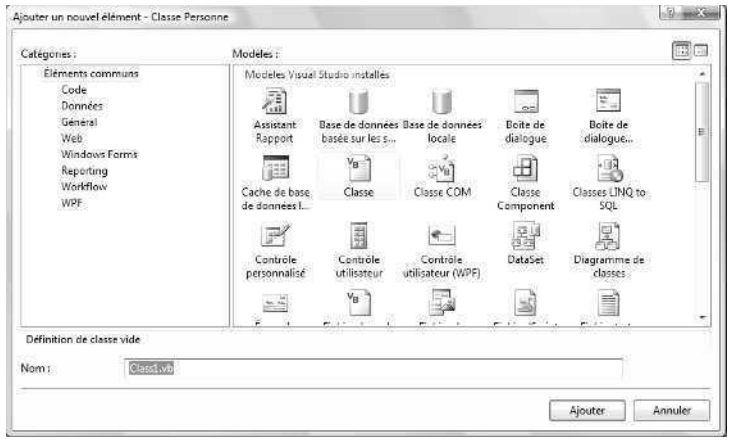
Objet	Propriété	Paramètre
Label1	*Text*	« Saisissez le prénom, le nom et la date de naissance de l'employé »
TextBox1	*Text*	« Prénom »
TextBox2	*Text*	« Nom »
Button1	*Text*	« Afficher l'enregistrement »
Form1	*Text*	« Classe Personne »

Votre formulaire présente un résultat similaire à ceci :

Vous disposez à présent de l'interface utilisateur de base pour un formulaire qui définit l'enregistrement d'un nouvel employé dans une entreprise. Le formulaire n'est pas connecté à une base de données : vous ne pouvez donc stocker qu'un enregistrement à la fois. Vous apprendrez toutefois à réaliser une telle connexion au chapitre 18, « Démarrer avec ADO.NET ». Vous allez maintenant ajouter une classe au projet pour stocker les informations de l'enregistrement.

7. Dans le menu Projet, choisissez la commande Ajouter une classe.

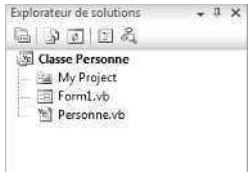
Visual Studio affiche la boîte de dialogue Ajouter un nouvel élément, avec le modèle Classe sélectionné :



La boîte de dialogue Ajouter un nouvel élément permet de nommer la classe. Dans la mesure où vous pouvez stocker plusieurs classes dans un nouveau module de classe, il est préférable de lui affecter un nom polyvalent.

8. Tapez **Personne.vb** dans la zone de texte Nom et cliquez sur Ajouter.

   Visual Studio ouvre un module de classe vide dans l'Éditeur de code et ajoute un fichier intitulé Personne.vb dans l'Explorateur de solutions :



Vous allez maintenant saisir la définition de votre classe dans le module de classe et apprendre quelques mots clé Visual Basic. Vous suivrez quatre étapes : déclarer les variables de classe, créer les propriétés, créer une méthode et pour finir, créer un objet fondé sur la nouvelle classe.

Étape 1 : Déclarer les variables de la classe

- Sous l'instruction *Public Class Personne*, tapez les déclarations de variables suivantes :

```
Private Nom1 As String
Private Nom2 As String
```

  Vous venez de déclarer deux variables qui serviront exclusivement au sein du module de classe pour stocker les valeurs de deux paramètres de propriété de chaîne. J'ai déclaré les variables en me servant du mot clé *Private* puisque, par convention, les programmeurs Visual Basic conservent les variables de classe internes privées, autrement dit, non disponibles en dehors du module de classe.

Étape 2 : Créer les propriétés

1. Sous les déclarations de variables, tapez l'instruction suivante et appuyez sur ENTRÉE :

```
Public Property Prénom() As String
```

   Cette instruction crée une propriété intitulée *Prénom*, du type *String*, dans la classe. Lorsque l'on appuie sur ENTRÉE, Visual Studio fournit automatiquement une structure de code pour les éléments restant dans la déclaration de la propriété. Les éléments obligatoires sont : un bloc *Get* qui détermine ce que les programmeurs voient lorsqu'ils consultent la propriété *Prénom*, un bloc *Set* qui détermine ce qui se produit lorsque l'on définit ou modifie la propriété *Prénom* et une instruction *End Property* qui marque la fin de la procédure de la propriété.

2. Remplissez la structure de la procédure de propriété de sorte qu'elle ressemble au code qui suit (les éléments à saisir sont en gras).

```
Public Property Prénom() As String
    Get
        Return Nom1
    End Get
    Set(ByVal value As String)
        Nom1 = value
    End Set
End Property
```

Le mot clé *Return* indique que la variable de chaîne *Nom1* est renvoyée lorsque l'on référence la propriété *Prénom*. Le bloc *Set* assigne une valeur chaîne à la variable *Nom1* lorsque l'on définit la propriété. Notez la variable *value* employée dans les procédures de propriétés pour représenter la valeur assignée à la classe lorsque l'on définit la propriété. Même si cette syntaxe peut sembler étrange, croyez-moi pour l'instant : c'est ainsi que l'on crée les paramètres de propriété dans les contrôles. Des propriétés plus élaborées pourraient contenir une logique de programme pour tester les valeurs ou faire des calculs.

3. Sous l'instruction *End Property*, tapez une deuxième procédure de propriété pour la propriété *Nom*. Elle doit ressembler au code ci-après (les lignes à saisir sont en gras).

```
Public Property Nom() As String
    Get
        Return Nom2
    End Get
    Set(ByVal value As String)
        Nom2 = value
    End Set
End Property
```

Cette procédure de propriété est similaire à celle du premier code, excepté qu'elle utilise une deuxième variable chaîne (*Nom2*) que vous avez déclarée dans la partie supérieure de la classe.

Vous avez terminé la définition des deux propriétés de la classe. Intéressons-nous à présent à une méthode appelée *Age* qui va déterminer l'âge actuel de l'employé en fonction de sa date de naissance.

Étape 3 : Créer une méthode

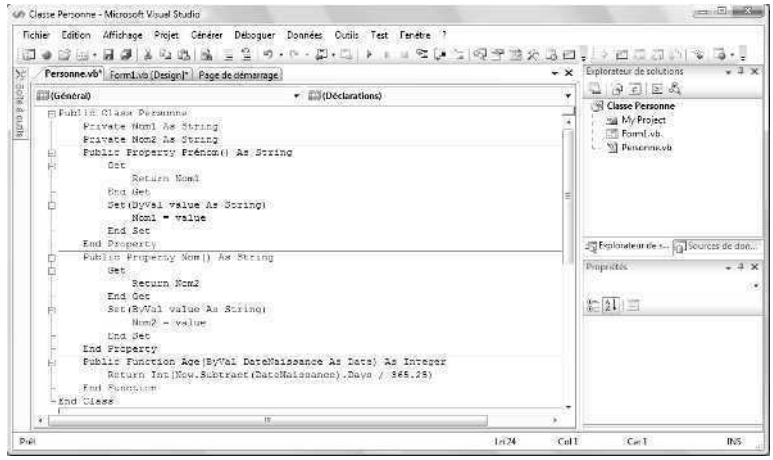
- Sous la procédure de propriété *Nom*, tapez la définition de fonction suivante :

```
Public Function Age(ByVal DateNaissance As Date) As Integer
    Return Int(Now.Subtract(DateNaissance).Days / 365.25)
End Function
```

Pour créer une méthode dans la classe qui effectue une action spécifique, vous ajoutez une fonction ou une procédure *Sub* à la classe. Bien que la majorité des

méthodes n'exigent pas d'argument pour accomplir leur tâche, la méthode *Age* que nous définissons requiert l'argument *DateNaissance* de type *Date* pour effectuer son calcul. Cette méthode fait appel à la méthode *Substract* pour soustraire la date de naissance du nouvel employé de la date actuelle du système et retourne la valeur exprimée en jours divisée par 365,25 (la longueur approximative en jours d'une année). La fonction *Int* convertit la valeur en un entier. Ce nombre est ensuite retourné à la procédure appelante *via* l'instruction *Return*, à l'instar de toute autre fonction classique (pour plus d'informations sur les définitions de fonctions, reportez-vous au chapitre 10, « Créer des modules et des procédures »).

La définition de classe est terminée et, dans l'Éditeur de code, la classe *Personne* ressemble à ceci :



Retournons maintenant à Form1 pour utiliser la nouvelle classe dans une procédure événementielle.

Astuce Même si nous ne l'avons pas fait dans cet exemple, il est généralement sage d'ajouter une logique de vérification de type aux modules de classe dans les projets réels de sorte que les propriétés ou les méthodes mal employées ne déclenchent pas pendant l'exécution d'erreurs qui arrêtent le programme.

Étape 4 : Créer un objet fondé sur la nouvelle classe

1. Dans l'Explorateur de solutions, cliquez sur Form1.vb puis sur le bouton Concepteur de vues. L'interface utilisateur de Form1 s'affiche.
2. Double-cliquez sur le bouton Afficher l'enregistrement pour afficher la procédure événementielle *Button1_Click* dans l'Éditeur de code.

3. Tapez les instructions suivantes :

```
Dim Employé As New Personne
Dim DA As Date

Employé.Prénom = TextBox1.Text
Employé.Nom = TextBox2.Text
DA = DateTimePicker1.Value.Date

MsgBox(Employé.Prénom & " " & Employé.Nom _
    & " a " & Employé.Age(DA) & " ans.")
```

Cette routine stocke les valeurs saisies par l'utilisateur dans un objet appelé *Employé* déclaré de type *Personne*. Le mot clé *New* indique que vous voulez créer immédiatement une nouvelle instance de l'objet *Employé*. Vous avez souvent déclaré des variables dans ce livre, mais cette fois vous en déclarez une fondée sur une classe que vous avez créée ! La routine déclare ensuite une variable *Date* intitulée *DA* pour stocker la date saisie par l'utilisateur. Elle attribue alors le prénom et le nom retournés par les deux objets zone de texte du formulaire aux propriétés *Prénom* et *Nom* de l'objet *Employé*. La valeur retournée par l'objet sélecteur de date et d'heure est stockée dans la variable *DA* et la dernière instruction du programme affiche une boîte de message qui contient les propriétés *Prénom* et *Nom* plus l'âge du nouvel employé tel que déterminé par la méthode *Age*, qui retourne une valeur entière lorsque la variable *DA* lui est passée. Après avoir défini une classe dans un module de classe, il n'y a rien de plus simple que de l'employer dans une procédure événementielle, comme le montre cette routine.

4. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap16.
5. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

   L'interface utilisateur s'affiche dans l'environnement de développement, en attente d'une saisie.
6. Tapez un prénom dans la zone de texte Prénom et un nom dans la zone de texte Nom.
7. Cliquez sur la flèche de l'objet sélecteur de date et d'heure et faites défiler la liste jusqu'à une date d'anniversaire (j'ai retenu le 12 juillet 1970).

Astuce Pour faire défiler la liste plus rapidement, cliquez sur le champ de l'année lorsque la boîte de dialogue du sélecteur de date et d'heure est ouverte. De petites flèches de défilement s'affichent et permettent de passer d'une année à l'autre. Pour atteindre rapidement le mois de votre choix, cliquez sur le champ du mois et sur le mois dans le menu contextuel.

Voici à quoi ressemble votre formulaire :

8. Cliquez sur le bouton Afficher l'enregistrement.

Le programme place les valeurs de prénom et de nom dans les paramètres de la propriété et utilise la méthode *Age* pour calculer l'âge actuel du nouvel employé. Une boîte de message affiche le résultat :

9. Cliquez sur OK pour fermer la boîte de message. Testez quelques valeur de dates différentes en cliquant sur Afficher l'enregistrement chaque fois que vous modifiez le champ de la date de naissance.

10. Lorsque vous avez terminé de tester votre nouvelle classe, cliquez sur le bouton Fermer du formulaire.

L'environnement de développement s'affiche à nouveau.

Aller plus loin : Hériter d'une classe de base

Comme je l'ai promis au début de ce chapitre, j'ai encore une astuce relative aux classes définies par l'utilisateur et à l'héritage à vous montrer. À l'instar des classes de formulaire, les formulaires peuvent hériter de classes que vous avez définies. Pour ce faire, vous employez la commande Ajouter une classe et un module de classe. Le mécanisme d'héritage d'une classe de base (parent) fait appel à l'instruction *Inherits* pour inclure la classe préalablement définie dans une nouvelle classe. Vous pouvez alors ajouter des propriétés ou des méthodes à la classe dérivée (enfant) pour la différencier de la classe de base. Tout ceci pouvant paraître quelque peu abstrait, le mieux est d'essayer un exemple.

Dans le prochain exercice, vous allez modifier le projet Classe Personne de sorte qu'il stocke les informations sur les nouveaux enseignants et les matières qu'ils enseignent. Vous commencerez par ajouter une deuxième classe définie par l'utilisateur, appelé *Enseignant*, au module de classe *Personne*. Cette nouvelle classe héritera la propriété *Prénom*, la propriété *Nom* et la méthode *Age* de la classe *Personne* et contiendra une propriété supplémentaire intitulée *Matière* pour contenir la matière enseignée.

Utiliser le mot clé *Inherits*

1. Dans l'Explorateur de solutions, cliquez sur Personne.vb puis sur le bouton Afficher le code.
2. Rendez-vous à la fin de l'Éditeur de code pour placer le point d'insertion sous l'instruction *End Class*.

   Comme je l'ai mentionné précédemment, il est possible d'inclure plusieurs classes dans un module de classe, tant que chaque classe est délimitée par des instructions *Public Class* et *End Class*. Vous allez créer une classe intitulée *Enseignant* dans ce module de classe et vous utiliserez le mot clé *Inherits* pour incorporer la méthode et les propriétés que vous avez définies dans la classe *Personne*.
3. Saisissez la définition de classe suivante dans l'Éditeur de code (saisissez les instructions en gras : Visual Studio ajoute automatiquement les autres instructions).

```
Public Class Enseignant
    Inherits Personne
    Private Niveau As String
    Public Property Matière() As String
        Get
            Return Niveau
        End Get
        Set(ByVal value As String)
            Niveau = value
        End Set
    End Property
End Class
```

   L'instruction *Inherits* lie la classe *Personne* à cette nouvelle classe, incluant toutes ses variables, propriétés et méthodes. Si la classe *Personne* se trouvait dans un autre module ou projet, vous pourriez identifier son emplacement à l'aide d'une désignation d'espace de nom, comme pour identifier les classes lors de l'emploi de l'instruction *Imports* dans la partie supérieure d'un programme qui utilise des classes des bibliothèques de classes .NET Framework. Pour faire simple, nous avons défini la classe *Enseignant* comme type spécial de la classe *Personne*. Outre les propriétés *Prénom* et *Nom*, la classe *Enseignant* possède la propriété *Matière* qui enregistre le niveau auquel l'enseignant fait cours.

   Nous allons maintenant utiliser la nouvelle classe dans la procédure événementielle *Button1_Click*.

4. Dans Form1, affichez la procédure événementielle *Button1_Click*.

   Au lieu de créer une nouvelle variable pour contenir la classe *Enseignant*, nous utilisons la variable *Employé* telle quelle : la seule différence est que nous pouvons maintenant définit la propriété *Matière* pour le nouvel employé.

5. Dans Form1, modifiez la procédure événementielle *Button1_Click* comme suit (changez les lignes en gras).

```
Dim Employé As New Enseignant
Dim DA As Date

Employé.Prénom = TextBox1.Text
Employé.Nom = TextBox2.Text
DA = DateTimePicker1.Value.Date
Employé.Matière = InputBox("Quelle matière enseignez-vous ?")

MsgBox("Matière enseignée par " & Employé.Prénom & " " & Employé.Nom _
   & " :" & Employé.Matière)
```

   Dans cet exemple, j'ai supprimé le calcul relatif à l'âge (nous n'utilisons pas la méthode *Age*), mais je l'ai uniquement fait pour minimiser les informations affichées dans la boîte de message. Lorsque vous définissez des propriétés et des méthodes dans une classe, il n'est pas indispensable de les utiliser dans le code.

   Exécutons le programme.

> **Astuce** Le programme Classe Personne révisé complet est disponible dans le dossier c:\vb08epe\chap16\Classe Personne.

6. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme. Le formulaire s'affiche.

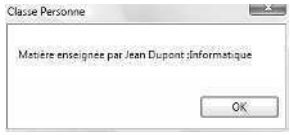
7. Tapez votre prénom dans la zone de texte Prénom et votre nom dans la zone de texte Nom.
8. Cliquez sur l'objet sélecteur de date et d'heure et faites défiler jusqu'à votre date de naissance.

9. Cliquez sur le bouton Afficher l'enregistrement.

   Votre programme stocke les valeurs de prénom et de nom dans les paramètres des propriétés puis affiche la boîte de saisie suivante, qui invite le nouvel enseignant à indiquer la matière qu'il enseigne :

10. Tapez **Informatique** et cliquez sur OK pour fermer la boîte de saisie.

    L'application stocke la chaîne « Informatique » dans la nouvelle propriété *Matière* et se sert des propriétés *Prénom*, *Nom* et *Matière* pour afficher les informations relatives au nouvel employé dans une boîte de message de confirmation. Voici le message :



11. Testez différentes valeurs, si vous le souhaitez, puis cliquez sur le bouton Fermer du formulaire.

    Le programme s'arrête et vous revenez à l'environnement de développement. Vous avez terminé de travailler sur les classes et l'héritage dans ce chapitre. Beau travail !

Approfondissez vos expériences avec la programmation orientée objet

Si vous avez apprécié cette incursion dans les techniques de codage orientées objet, Visual Basic 2008, un réel langage de programmation orientée objet, vous réserve d'autres surprises. En particulier, il est possible d'ajouter des événements aux définitions de classes, de créer des valeurs de propriétés par défaut, de déclarer et d'employer des types anonymes et nommés et d'explorer la fonctionnalité polymorphique appelée *surcharge de méthode*. Vous découvrirez ces fonctionnalités et d'autres fonctionnalités de programmation orientée objet dans la documentation de Visual Studio ou en lisant attentivement un ouvrage avancé sur la programmation Visual Basic (reportez-vous à l'Annexe A, « Où trouver d'autres informations », pour une liste de lectures). Pour plus d'informations sur la programmation orientée objet et les bases de données dans Visual Basic, reportez-vous à la Partie IV, « Programmation web et de base de données ».

Rappel du chapitre 16

Pour	Faites ceci
Hériter l'interface et les fonctionnalités d'un formulaire existant	Cliquez sur la commande Ajouter un nouvel élément du menu Projet, sélectionnez le modèle Formulaire hérité, indiquez le nom du formulaire hérité et cliquez sur Ajouter. Servez-vous du Sélecteur d'héritage pour sélectionner le formulaire dont vous voulez hériter et cliquez sur OK. Pour pouvoir être hérités, les formulaires de base doivent être compilés en fichiers .exe ou .dll. Pour hériter d'un formulaire qui n'est pas un composant du projet en cours, le formulaire doit être compilé en tant que fichier .dll.
Personnaliser un formulaire hérité	Ajoutez des contrôles de la Boîte à outils au formulaire et définissez les paramètres des propriétés. Vous ne pouvez pas définir les propriétés des objets hérités sur le formulaire. On identifie ces objets, qui sont inactifs, grâce à de petites icônes.
Créer vos propres classes de base	Cliquez sur la commande Ajouter une classe dans le menu Projet, précisez le nom de la classe et cliquez sur Ouvrir. Définissez la classe dans un module de classe en vous servant du code.
Masquer les variables déclarées dans une classe	Servez-vous du mot clé Private pour masquer les variables de classes aux autres programmeurs qui examinent votre classe. Par exemple : `Private Nom1 As String`

Créer une nouvelle propriété dans la classe	Définissez une procédure de propriété publique dans la classe. Par exemple :
	`Public Property Prénom() As String`
	`  Get`
	`  Return Nom1`
	`  End Get`
	`  Set(ByVal value As String)`
	`  Nom1 = value`
	`  End Set`
	`End Property`

Créer une nouvelle méthode dans la classe	Définissez une procédure Sub ou Function dans la classe. Par exemple :
	`Public Function Age(ByVal DateNaissance As Date) As Integer`
	`  Return Int(Now.Subtract(DateNaissance).Days / 365.25)`
	`End Function`

Déclarer une variable objet à utiliser dans une classe	Servez-vous des mots clés Dim et New, d'un nom de variable et d'une classe définie par l'utilisateur dans une instruction. Par exemple : `Dim Employé As New Personne`
Définir les propriétés d'une variable objet	Servez-vous de la syntaxe classique pour définir les propriétés de l'objet. Par exemple : `Employé.Prénom = TextBox1.Text`

Pour	Faites ceci

Hériter de la classe de base dans une nouvelle classe	Créez une nouvelle classe et utilisez le mot clé Inherits pour incorporer les définitions de classe de la classe de base. Par exemple :
	`Public Class Enseignant`
	`  Inherits Personne`
	`  Private Matière As String`
	
	`  Public Property Matière() As String`
	`  Get`
	`  Return Niveau`
	`  End Get`
	`  Set(ByVal value As String)`
	`  Niveau = value`
	`  End Set`
	`  End Property`
	`End Class`

Chapitre 17

Travailler avec les imprimantes

À la fin de ce chapitre, vous saurez :

- Imprimer des images à partir d'un programme Visual Basic
- Imprimer du texte à partir d'un programme Visual Basic
- Imprimer des documents de plusieurs pages
- Insérer des boîtes de dialogue Imprimer, Mise en page et Aperçu avant impression dans vos programmes

Dans les prochaines sections, vous achèverez votre étude de la conception de l'interface utilisateur et de ses composants en apprenant à inclure la prise en charge de l'impression dans vos applications Windows. Microsoft Visual Basic 2008 prend en charge l'impression grâce à la classe *PrintDocument* et à ses nombreux objets, méthodes et propriétés qui gèrent l'envoi de texte et de graphismes aux imprimantes.

Dans ce chapitre, nous verrons comment imprimer des images et du texte à partir de programmes Visual Basic, gérer des tâches d'impressions de plusieurs pages et ajouter des boîtes de dialogue d'impression à l'interface utilisateur. Ce chapitre est à mon avis l'un des plus utiles du livre. Il présente des exemples pratiques de code que vous pouvez incorporer directement dans des projets de programmation de production. La prise en charge de l'impression n'est pas évidente dans Visual Basic 2008, mais les routines de ce chapitre vous aideront à imprimer des documents textuels plus longs et à afficher des boîtes de dialogue pratiques comme Mise en page, Imprimer et Aperçu avant impression au sein de vos programmes. Ce chapitre commence par deux routines extrêmement simples qui présentent les bases avant de passer à des projets bien plus élaborés.

Utiliser la classe *PrintDocument*

La majorité des applications Windows permettent aux utilisateurs d'imprimer des documents après qu'ils les ont créés. Vous vous demandez sans doute comment fonctionne l'impression dans les programmes Visual Basic. En fait, il s'agit de l'un des domaines ayant bénéficié de l'une des plus considérables optimisations entre Visual Basic 6 et Visual Basic 2008. Ces améliorations ne sont cependant pas proposées sans contrepartie. La production d'une sortie imprimée à partir des programmes Visual Basic 2005 ne constitue pas un processus aisé et la technique employée dépend du type et de la quantité de sortie imprimée à générer. Dans tous les cas, la classe *PrintDocument* constitue le mécanisme de base qui régule l'impression dans Visual Basic 2008. Pour la créer dans un projet, vous disposez de deux méthodes :

- Ajouter le contrôle *PrintDocument* à un formulaire ;
- La définir par programmation avec quelques lignes de code Visual Basic.

La classe *PrintDocument* appartient à l'espace de noms *System.Drawing.Printing*. Celui-ci propose plusieurs objets intéressants qui permettent d'imprimer du texte et des images, parmi lesquels l'objet *PrinterSettings* qui contient les paramètres d'impression par défaut d'une imprimante, l'objet *PageSettings* qui contient les paramètres d'impression d'une page particulière et l'objet *PrintPageEventArgs* qui contient les informations événementielles relatives à la page à imprimer. L'espace de noms *System.Drawing.Printing* est automatiquement incorporé à vos projets. Pour faciliter les références aux objets d'impression et autres valeurs importantes de cet espace de noms, ajoutez l'instruction *Imports* suivante en haut de votre formulaire :

```
Imports System.Drawing.Printing
```

Pour apprendre à utiliser la classe *PrintDocument* dans un programme, réalisez l'exercice suivant, qui montre comment ajouter un contrôle *PrintDocument* à un projet et l'utiliser pour imprimer un fichier graphique provenant du système.

Utiliser le contrôle *PrintDocument*

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Imprimer une image**.

   Un formulaire vierge s'affiche dans l'environnement de développement Visual Studio.

2. Servez-vous du contrôle *Label* pour créer un objet étiquette dans la partie supérieure du formulaire.

3. Servez-vous du contrôle *TextBox* pour créer un objet zone de texte sous l'objet étiquette.

   L'objet zone de texte va servir à saisir le nom du fichier graphique à ouvrir. Une zone d'une seule ligne suffit.

4. Servez-vous du contrôle *Button* pour dessiner un objet bouton sous la zone de texte. Cet objet bouton imprimera le fichier graphique. Vous allez maintenant ajouter un contrôle *PrintDocument*.

5. Faites défiler la Boite à outils jusqu'à voir l'onglet Impression, puis double-cliquez sur le contrôle *PrintDocument*.

   À l'instar du contrôle *Timer*, le contrôle *PrintDocument* est invisible pendant l'exécution. Il est donc placé dans la zone des composants, sous le formulaire. Le projet a maintenant accès à la classe *PrintDocument* et à ses utiles objets d'impression.

6. Définissez les propriétés suivantes pour les objets du formulaire :

Objet	Propriété	Paramètre
Label1	*Text*	« Saisissez le nom d'un fichier graphique à imprimer »
TextBox1	*Text*	« c:\vb08epe\chap15\soleil.ico »
Button1	*Text*	« Imprimer l'image »
Form1	*Text*	« Imprimer une image »

Voici à quoi ressemble votre formulaire :

Ajoutons à présent le code nécessaire pour imprimer un fichier graphique (bitmap, icône, métafichier, fichier JPEG et ainsi de suite).

7. Double-cliquez sur le bouton Imprimer l'image.

La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

8. Déplacez le point d'insertion au début du code du formulaire et saisissez l'instruction suivante :

```
Imports System.Drawing.Printing
```

Cette instruction *Imports* déclare l'espace de noms *System.Drawing.Printing*, ce qui facilite les références aux classes d'impression.

9. Placez maintenant le point d'insertion dans la procédure événementielle *Button1_Click* et saisissez le code suivant :

```
'Imprime en utilisant un gestionnaire d'erreurs pour intercepter les problèmes
Try
    AddHandler PrintDocument1.PrintPage, AddressOf Me.ImprimerImage
        PrintDocument1.Print()    'Imprime l'image
Catch ex As Exception    'Intercepte les exceptions d'impression
    MessageBox.Show("Désolé, il y a un problème d'impression", ex.ToString())
End Try
```

Remarque Après avoir saisi ce code, vous verrez une ligne dentelée sous *Me.ImprimerImage*. Ne vous inquiétez pas, nous ajouterons la procédure ImprimerImage à l'étape suivante.

Ce code fait appel à une instruction *AddHandler* qui spécifie que le gestionnaire d'événements *ImprimerImage* doit être appelé lorsque l'événement *PrintPage* de l'objet *PrintDocument1* se déclenche. Nous avons étudié les *gestionnaires d'erreurs* dans les précédents chapitres : un *gestionnaire d'événements* est un mécanisme étroitement lié gérant les événements système qui ne sont pas techniquement des erreurs, mais représentent des actions indispensables dans le cycle de vie d'un objet.

Dans ce cas, le gestionnaire d'événements spécifié est lié aux services d'impression. La requête s'accompagne d'informations spécifiques relatives à la page à imprimer, les paramètres de l'imprimante en cours et d'autres attributs de la classe *PrintDocument*. Techniquement, l'opérateur *AddressOf* identifie le gestionnaire d'événements *ImprimerImage* en déterminant son adresse interne et en la stockant. L'opérateur *AddressOf* crée implicitement un objet appelé *délégué* qui transfère les appels au gestionnaire d'événements approprié lorsqu'un événement se produit.

La troisième ligne du code que vous venez de saisir utilise la méthode *Print* de l'objet *PrintDocument1* pour envoyer une requête d'impression à la procédure événementielle *ImprimerImage*, une routine que vous allez créer à la prochaine étape. Cette requête d'impression se trouve dans le bloc de code *Try* pour intercepter les problèmes d'impression qui peuvent se produire pendant l'activité d'impression. Notez que la syntaxe utilisée dans le bloc *Catch* est légèrement différente de celle présentée au chapitre 9, « Gérer les erreurs avec la gestion structurée des exceptions ». Dans cet exemple, la variable *ex* est déclarée de type *Exception* pour obtenir un message détaillé sur toute erreur qui se produirait. L'utilisation du type *Exception* constitue une autre méthode pour récupérer la condition d'erreur sous-jacente qui a créé le problème.

10. Dans l'Éditeur de code, placez le point d'insertion dans l'espace de déclaration général, au-dessus de la procédure événementielle *Button1_Click* et au-dessous de l'instruction *Public Class Form1*. Saisissez ensuite la déclaration de procédure *Sub* suivante :

```
'Sub pour imprimer des images
Private Sub ImprimerImage(ByVal sender As Object, _
    ByVal ev As PrintPageEventArgs)
       'Crée l'image avec DrawImage
       ev.Graphics.DrawImage(Image.FromFile(TextBox1.Text), _
           ev.Graphics.VisibleClipBounds)
       'Indique que ceci est la dernière page à imprimer
       ev.HasMorePages = False
End Sub
```

Cette routine gère l'événement d'impression généré par la méthode *PrintDocument1.Print*. Nous avons déclaré la procédure *Sub* au sein du code du formulaire, mais il est également possible de la déclarer en tant que procédure polyvalente dans un module. Remarquez la variable *ev* qui se trouve dans la liste d'arguments pour la procédure *ImprimerImage*. Cette variable est l'indispensable porteur d'informations

sur la page en cours d'impression. Elle est déclarée avec comme type *PrintPageEvent-Args*, un objet de l'espace de noms *System.Drawing.Printing*.

Pour imprimer réellement l'image, la procédure emploie la méthode *Graphics.DrawImage* associée à la page à imprimer pour charger un fichier graphique en utilisant le nom de fichier stocké dans la propriété *Text* de l'objet *TextBox1* (par défaut, j'ai positionné cette propriété sur c:\vb08epe\chap15\soleil.ico, la même icône Soleil que celle employée au chapitre 15, « Ajouter des images et des effets d'animation », mais vous pouvez remplacer cette valeur pendant l'exécution et imprimer le fichier graphique de votre choix). Pour finir, j'ai attribué la valeur *False* à la propriété *ev.HasMorePages* de sorte que Visual Basic comprenne que la tâche d'impression ne comporte qu'une page.

11. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap17.

Vous êtes prêt à exécuter le programme. Avant de le faire, vous voudrez éventuellement localiser quelques fichiers graphiques à imprimer sur votre système (pour l'instant, contentez-vous de noter les chemins d'accès et saisissez-les).

Exécuter le programme Imprimer une image

Astuce Le programme Imprimer une image complet est disponible dans le dossier c:\vb08epe\chap17\Imprimer une image.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. Le programme s'exécute dans l'environnement de développement. Voici le formulaire :

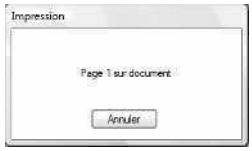
2. Allumez votre imprimante et vérifiez qu'elle est connectée et qu'elle contient du papier.

3. Si vous avez installé les fichiers d'exemple dans le dossier par défaut c:\vb08epe, cliquez sur le bouton Imprimer l'image pour imprimer l'icône Soleil.ico.

Si vous n'avez pas utilisé l'emplacement par défaut ou pour imprimer un autre fichier graphique, modifiez le chemin d'accès qui se trouve dans la zone de texte en conséquence et cliquez sur le bouton Imprimer l'image.

La méthode *DrawImage* agrandit l'image à la taille maximale que l'imprimante peut placer sur une page puis envoie l'image à l'imprimante (cette « fonction d'expansion » remplit la page et permet de mieux apprécier l'image). Cette fonction n'est pas nécessairement intéressante, mais nous allons l'améliorer sous peu (pour modifier l'emplacement ou la taille de la sortie, recherchez la rubrique « *Graphics.DrawImage Method* » dans la documentation de Visual Studio, étudiez les différentes variations possibles de l'argument puis modifiez le code).

Si vous regardez attentivement, vous verrez la boîte de dialogue suivante apparaître lorsque Visual Basic envoie la tâche d'impression à l'imprimante :



Cette boîte d'état est également un produit de la classe *PrintDocument*. Elle propose à l'utilisateur une interface d'impression d'aspect professionnel, comprenant le numéro de chaque page imprimée.

4. Si vous le souhaitez, saisissez d'autres chemins d'accès puis cliquez sur le bouton Imprimer l'image.
5. Lorsque vous avez terminé de tester le programme, cliquez sur le bouton Fermer du formulaire.

   Le programme s'arrête. Pas mal pour une première tentative d'impression à partir d'un programme Visual Basic !

Imprimer du texte à partir d'un objet zone de texte

Après cette rapide introduction au contrôle *PrintDocument* et à l'impression d'images, nous allons employer une technique similaire pour imprimer le contenu d'une zone de texte dans un formulaire Visual Basic. Dans le prochain exercice, vous allez créer un projet simple qui utilise la classe *PrintDocument* pour imprimer. Vous définirez toutefois la classe en vous servant de code, sans ajouter le contrôle *PrintDocument* au formulaire. En outre, vous utiliserez la méthode *Graphics.DrawString* pour envoyer l'ensemble du contenu d'un objet zone de texte à l'imprimante par défaut.

Remarque Le programme suivant est conçu pour imprimer au maximum une page de texte. Pour imprimer plusieurs pages, vous devez ajouter du code, ce que nous verrons plus loin dans ce chapitre. Je préfère ne pas présenter trop de fonctionnalités d'impression à la fois.

Utiliser la méthode *Graphics.DrawString* pour imprimer du texte

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms intitulé **Mon Imprimer du texte**.

   Un formulaire vide s'affiche.

2. Servez-vous du contrôle *Label* pour créer un objet étiquette dans la partie supérieure du formulaire.

   Cette étiquette présentera une ligne d'instructions destinée à l'utilisateur.

3. Servez-vous du contrôle *TextBox* pour créer un objet zone de texte sous l'objet étiquette.

   L'objet zone de texte va contenir le texte à imprimer.

4. Positionnez la propriété *Multiline* de l'objet zone de texte sur *True* et étendez la zone de texte de sorte qu'elle soit suffisamment grande pour contenir plusieurs lignes de texte.

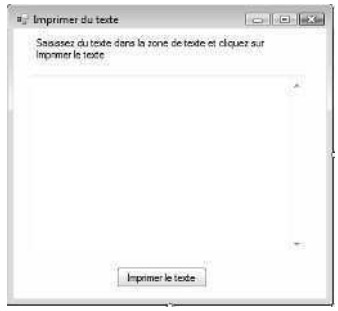
5. Servez-vous du contrôle *Button* pour dessiner un objet bouton sous la zone de texte.

   Cet objet bouton imprimera le fichier texte.

6. Définissez les propriétés suivantes pour les objets du formulaire :

Objet	Propriété	Paramètre
Label1	*Text*	« Saisissez du texte dans la zone de texte et cliquez sur Imprimer le texte »
TextBox1	*ScrollBars*	Vertical
	Multiline	True
Button1	*Text*	« Imprimer le texte »
Form1	*Text*	« Imprimer du texte »

Voici à quoi ressemble votre formulaire :



Ajoutons maintenant le code qui imprimera le contenu de la zone de texte.

7. Double-cliquez sur le bouton Imprimer le texte. La procédure événementielle *Button1_Click* s'affiche dans l'Éditeur de code.

8. Placez le point d'insertion au début du code du formulaire et saisissez l'instruction *Imports* suivante :

```
Imports System.Drawing.Printing
```

Cette instruction facilite les références aux classes de l'espace de noms *System.Drawing.Printing*, dont la classe *PrintDocument* et ses objets obligatoires.

9. Placez maintenant le point d'insertion dans la procédure événementielle *Button1_Click* et saisissez le code suivant :

```
'Imprime en utilisant un gestionnaire d'erreurs pour intercepter les problèmes
Try
    'Déclare la variable ImprimerDoc de type PrintDocument
    Dim ImprimerDoc As New PrintDocument
    AddHandler ImprimerDoc.PrintPage, AddressOf Me.ImprimerTexte
    ImprimerDoc.Print()    'imprime le texte
Catch ex As Exception      'intercepte les exceptions d'impression
    MessageBox.Show("Désolé, il y a un problème d'impression", ex.ToString())
End Try
```

Les nouvelles lignes ou les lignes modifiées par rapport au programme Imprimer une image sont en gras. Au lieu d'ajouter un contrôle *PrintDocument* au formulaire, cette fois vous l'avez simplement créé par programmation en utilisant le mot clé *Dim* et le type *PrintDocument*, défini dans le programme lorsque vous définissez l'espace de noms *System.Drawing.Printing*. À partir de ce point, la variable *ImprimerDoc* représente l'objet *PrintDocument* et sert à déclarer le gestionnaire d'erreurs et à imprimer le document texte. Notez que pour plus de clarté, j'ai renommé la procédure *Sub* qui va gérer l'événement d'impression *ImprimerTexte* (au lieu de *ImprimerImage*).

10. Placez le point d'insertion dans la zone de déclaration générale, au-dessus de la procédure événementielle *Button1_Click*. Saisissez la déclaration de procédure *Sub* suivante :

```
'Sub pour imprimer du texte
Private Sub ImprimerTexte(ByVal sender As Object, _
    ByVal ev As PrintPageEventArgs)
    'Utilise DrawString pour créer le texte dans un objet Graphics
    ev.Graphics.DrawString(TextBox1.Text, New Font("Arial", _
        11, FontStyle.Regular), Brushes.Black, 120, 120)
    'Indique que ceci est la dernière page à imprimer
    ev.HasMorePages = False
End Sub
```

Cette routine gère l'événement d'impression généré par la méthode *ImprimerDoc.Print*. Les changements par rapport à la procédure *ImprimerImage* des précédents exercices sont signalés en gras. Comme vous pouvez le noter, il faut faire appel à une nouvelle méthode pour imprimer du texte.

Au lieu d'utiliser *Graphics.DrawImage*, qui restitue une image graphique, vous employez *Graphics.DrawString* qui imprime une chaîne de texte. Dans la propriété *Text* de l'objet zone de texte à imprimer, nous avons précisé la police et sa mise en forme (Arial, 11 points, style Normal, couleur noire) et les coordonnées *x* et *y* (120, 120) sur la page pour le début du tracé. Ces spécifications donnent à la sortie imprimée un aspect par défaut similaire à celui de la zone de texte à l'écran. Comme la dernière fois, nous avons également attribué la valeur *False* à la propriété *ev.HasMorePages* pour indiquer que la tâche d'impression ne comporte pas plusieurs pages.

11. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap17.

Exécutons à présent le programme pour observer l'impression des objets zone de texte.

Exécuter le programme Imprimer du texte

> **Astuce** Le programme Imprimer du texte complet est disponible dans le dossier c:\vb08epe\chap17\Imprimer du texte.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. Le programme s'exécute dans l'environnement de développement.

2. Vérifiez que l'imprimante est allumée.

3. Saisissez du texte dans la zone de texte. Si vous saisissez plusieurs lignes, veillez à inclure un retour chariot à la fin de chaque ligne.

Le passage à la ligne n'est pas pris en charge dans cet exemple de programme : les lignes trop longues risquent de passer au-delà de la marge de droite (nous allons bientôt résoudre ce problème). Votre formulaire présente un résultat similaire à

4. Cliquez sur le bouton Imprimer le texte. La programme affiche une boîte de dialogue d'impression et imprime le contenu de la zone de texte.
5. Si vous le souhaitez, modifiez le contenu de la zone de texte et imprimez-le.
6. Lorsque vous avez terminé, cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

Vous savez maintenant imprimer du texte et des images à partir d'un programme.

Imprimer des fichiers texte de plusieurs pages

Les techniques d'impression que vous venez d'apprendre sont utiles dans le cadre de documents texte simples, mais elles possèdent quelques limites. Tout d'abord, la méthode employée n'autorise pas les lignes trop longues, autrement dit, le texte qui dépasse la marge de droite. Contrairement à l'objet zone de texte, l'objet *PrintDocument* ne passe pas automatiquement à la ligne lorsque l'on atteint le bord du papier. Si vos fichiers ne contiennent pas de retour chariot à la fin des lignes, vous devez écrire du code qui gère les lignes longues.

Ensuite, le programme Imprimer du texte ne peut pas imprimer plus d'une page de texte. En réalité, il ne sait même pas ce qu'est une page de texte : la procédure d'impression se contente d'envoyer le texte à l'imprimante par défaut. Si le bloc de texte est trop long pour tenir sur une page, le texte supplémentaire n'est pas imprimé. Pour gérer l'impression de plusieurs pages, il faut créer une page de texte virtuelle appelée *PrintPage* puis y ajouter du texte jusqu'à ce qu'elle soit pleine. Quand elle est pleine, elle est envoyée à l'imprimante. Ce processus se poursuit jusqu'à ce qu'il n'y ait plus de texte à imprimer. À ce moment-là, la tâche d'impression se termine.

Si la résolution de ces deux limites semble complexe, pas de panique : il existe plusieurs mécanismes permettant de créer des pages de texte virtuelles dans Visual Basic et d'imprimer des fichiers texte contenant de longues lignes et plusieurs pages de texte. L'événement *PrintPage* est le premier de ces mécanismes. Il se produit à l'impression de la page. *PrintPage* reçoit un argument de type *PrintPageEventArgs* qui fournit les dimensions et les caractéristiques de la page de l'imprimante en cours. La méthode *Graphics.DrawString* constitue l'autre moyen. La méthode *MeasureString* détermine le nombre de caractères et de lignes qui entrent dans une zone rectangulaire de la page. Ces mécanismes et d'autres simplifient la création de procédures traitant les tâches d'impression de plusieurs pages.

Suivez les étapes de la prochaine procédure pour créer un programme intitulé Imprimer un fichier qui ouvre des fichiers texte de n'importe quelle longueur et les imprime. Le programme Imprimer un fichier montre également comment utiliser les contrôles *RichTextBox*, *PrintDialog* et *OpenFileDialog*. Le contrôle *RichTextBox* est une version plus complète

du contrôle *TextBox* que vous venez d'utiliser pour afficher du texte. Le contrôle *PrintDialog* affiche une boîte de dialogue Imprimer standard permettant de spécifier différents paramètres d'impression. Le contrôle *OpenFileDialog* permet de sélectionner un fichier texte à imprimer (vous l'avez utilisé dans le chapitre 4, « Travailler avec les menus, les barres d'outils et les boîtes de dialogue »).

Gérer les requêtes d'impression avec les contrôles *RichTextBox*, *OpenFileDialog* et *PrintDialog*

1. Dans le menu Fichier, cliquez sur la commande Fermer le projet et créez un nouveau projet Application Windows Forms nommé **Mon Imprimer un fichier**.

   Un formulaire vide s'affiche.

2. Servez-vous du contrôle *Button* de la Boîte à outils pour dessiner deux boutons dans l'angle supérieur gauche du formulaire.

   Ce programme propose une interface utilisateur simple, mais les techniques d'impression s'adaptent facilement à des solutions bien plus complexes.

3. Dans la Boîte à outils, cliquez sur le contrôle *RichTextBox* et tracez un objet zone de texte enrichie couvrant la moitié inférieure du formulaire.

4. Double-cliquez sur le contrôle *OpenFileDialog* qui se trouve dans l'onglet Boîtes de dialogue et ajoutez un objet boîte de dialogue Ouvrir dans la zone des composants.

   Vous allez employer l'objet boîte de dialogue d'ouverture de fichier pour parcourir les fichiers texte de votre système.

5. Double-cliquez sur le contrôle *PrintDocument* qui se trouve dans l'onglet Impression pour ajouter un objet d'impression de document dans la zone des composants.

   Vous l'utiliserez pour prendre en charge l'impression dans l'application.

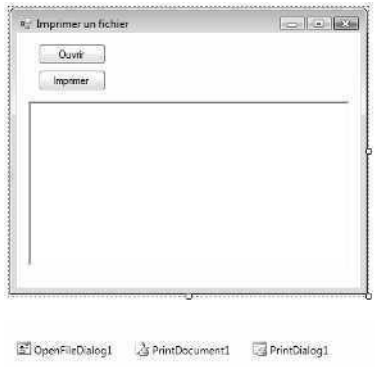
6. Double-cliquez sur le contrôle *PrintDialog* qui se trouve dans l'onglet Impression pour ajouter un objet boîte de dialogue Imprimer dans la zone des composants.

   Vous l'utiliserez pour prendre ouvrir une boîte de dialogue Imprimer dans le programme.

7. Définissez les propriétés suivantes pour les objets du formulaire :

Objet	Propriété	Paramètre
Button1	*Name*	btnOuvrir
	Text	« Ouvrir »
Button2	*Name*	btnImprimer
	Enabled	False
	Text	« Imprimer »
Form1	*Text*	« Imprimer un fichier »

Voici à quoi ressemble votre formulaire :



Ajoutons maintenant le code qui ouvre le fichier texte et l'imprime.

8. Double-cliquez sur le bouton Ouvrir. La procédure événementielle *btnOuvrir_Click* s'affiche dans l'Éditeur de code.

9. Placez le point d'insertion dans la partie supérieure du formulaire et saisissez le code suivant :

```
Imports System.IO      'pour la classe FileStream
Imports System.Drawing.Printing
```

Ces instructions facilitent les références à la classe *FileStream* et aux classes destinées à l'impression.

10. Placez le curseur sous l'instruction *Public Class Form1* et saisissez les déclarations de variables suivantes :

```
Private ParamPageImpr As New PageSettings
Private ChaineAImprimer As String
Private PoliceImpr As New Font("Arial", 10)
```

Ces instructions définissent des informations importantes relatives aux pages qui seront imprimées.

11. Placez le curseur dans la procédure événementielle *btnOuvrir_Click* et saisissez le code suivant :

```
Dim CheminFichier As String
'Affiche la boîte de dialogue Ouvrir et sélectionne le fichier texte
OpenFileDialog1.Filter = "Fichiers texte (*.txt)|*.txt"
OpenFileDialog1.ShowDialog()
'Si le bouton Annuler n'est pas sélectionné, charge la variable CheminFichier
If OpenFileDialog1.FileName <> "" Then
    CheminFichier = OpenFileDialog1.FileName
```

```
    Try
        'Lit le fichier texte et le charge dans RichTextBox1
        Dim MonFileStream As New FileStream(CheminFichier, FileMode.Open)
        RichTextBox1.LoadFile(MonFileStream, _
            RichTextBoxStreamType.PlainText)
        MonFileStream.Close()
       'Initialise la chaîne à imprimer
        ChaineAImprimer = RichTextBox1.Text
        'Active le bouton Imprimer
        btnImprimer.Enabled = True
    Catch ex As Exception
        'Affiche les éventuels messages d'erreur
        MessageBox.Show(ex.Message)
    End Try
End If
```

Lorsque l'utilisateur clique sur le bouton Ouvrir, cette procédure événementielle affiche une boîte de dialogue Ouvrir qui utilise un filtre affichant uniquement les fichiers texte. Quand l'utilisateur sélectionne un fichier, le nom de ce dernier est assigné à une variable de chaîne publique intitulée *CheminFichier*, déclarée dans la partie supérieure de la procédure événementielle. La procédure emploie ensuite un gestionnaire d'erreurs *Try…Catch* pour charger le fichier texte dans l'objet *RichTextBox*. Pour simplifier le processus de chargement, nous avons fait appel à la classe *StreamFile* et au mode de fichier *Open* qui place le contenu du fichier texte dans la variable *MonFileStream*. Pour finir, la procédure événementielle active le bouton Imprimer (*btnImprimer*) permettant à l'utilisateur d'imprimer le fichier. En résumé, cette routine ouvre le fichier et active le bouton Imprimer sur le formulaire mais n'effectue aucune action d'impression.

Vous allez maintenant ajouter le code qui affiche la boîte de dialogue Imprimer et imprime le fichier en vous servant de la logique qui analyse les dimensions de la page de texte en cours.

Ajouter le code pour les objets *btnImprimer* et *PrintDocument1*

1. Affichez à nouveau le formulaire et double-cliquez sur le bouton Imprimer (*btnImprimer*) pour afficher sa procédure événementielle dans l'Éditeur de code.
2. Tapez le code suivant :

```
Try
    'Spécifie les paramètres de la page en cours
    PrintDocument1.DefaultPageSettings = ParamPageImpr
    'Spécifie le document pour la boîte de dialogue Imprimer et l'affiche
    ChaineAImprimer = RichTextBox1.Text
    PrintDialog1.Document = PrintDocument1
    Dim Résultat As DialogResult = PrintDialog1.ShowDialog()
```

```
    'Si l'utilisateur clique sur OK, imprime le document sur l'imprimante
    If Résultat = DialogResult.OK Then
        PrintDocument1.Print()
    End If
Catch ex As Exception
    'Affiche un message d'erreur
    MessageBox.Show(ex.Message)
End Try
```

Cette procédure événementielle définit les paramètres d'impression par défaut du document et assigne le contenu de l'objet *RichTextBox* à la variable de chaîne *ChaineAImprimer* (définie dans la partie supérieure du formulaire) pour le cas où l'utilisateur change le texte dans la zone de texte enrichie. Elle ouvre ensuite une boîte de dialogue Imprimer et autorise l'utilisateur à ajuster les paramètres d'impression (imprimante, nombre de copies, l'option Imprimer dans un fichier, et ainsi de suite). Si l'utilisateur clique sur le bouton OK, la procédure événementielle envoie cette tâche d'impression à l'imprimante en émettant l'instruction suivante :

```
PrintDocument1.Print()
```

3. Affichez à nouveau le formulaire et double-cliquez sur l'objet *PrintDocument1* dans la zone des composants.

Visual Studio ajoute la procédure événementielle *PrintPage* pour l'objet *PrintDocument1*.

4. Saisissez le code suivant dans la procédure événementielle *PrintDocument1_PrintPage* :

```
Dim nbCar As Integer
Dim nbLignes As Integer
Dim strPage As String
Dim strFormat As New StringFormat
'En fonction de la configuration de la page, définit un rectangle dans lequel tracer sur la page
Dim TraceRect As New RectangleF( _
    e.MarginBounds.Left, e.MarginBounds.Top, _
    e.MarginBounds.Width, e.MarginBounds.Height)
    'Définit une zone qui détermine combien de texte entre dans une page
    'Diminue la hauteur d'une ligne pour s'assurer que le texte ne sera pas coupé
    Dim mesureTaille As New SizeF(e.MarginBounds.Width, _
    e.MarginBounds.Height - PoliceImpr.GetHeight(e.Graphics))

'Si les chaînes à imprimer sont longues, coupe entre les mots
strFormat.Trimming = StringTrimming.Word
'Calcule le nombre de caractères et de lignes qui entrent dans mesureTaille
    e.Graphics.MeasureString (ChaineAImprimer, PoliceImpr, _
        mesureTaille, strFormat, nbCar, nbLignes)
'Calcule la chaîne qui entre sur la page
strPage = ChaineAImprimer.Substring(0, nbCar)
'Imprime la chaîne sur la page en cours
e.Graphics.DrawString(strPage, PoliceImpr, _
    Brushes.Black, traceRect, strFormat)
```

```
'S'il reste du texte, indique qu'il reste des pages
If nbCar < ChaineAImprimer.Length Then
    'Soustrait le texte de la chaîne qui a été imprimée
    ChaineAImprimer= ChaineAImprimer.Substring(nbCar)
    e.HasMorePages = True
Else
    e.HasMorePages = False
    'Tout le texte a été imprimé, donc restaure la chaîne
    ChaineAImprimer = RichTextBox1.Text
End If
```

Cette procédure événementielle gère l'impression en soi du document texte. Elle le fait en définissant une zone d'impression (ou rectangle d'impression) en fonction des paramètres de la boîte de dialogue Mise en page. Tout le texte qui tient dans cette zone est imprimé normalement ; le texte qui sort de cette zone est renvoyé à la ligne ou page suivante, comme dans une application Windows standard.

La zone d'impression est définie par la variable *TraceRect*, basée sur la classe *RectangleF*. La variable *strFormat* et la méthode *Trimming* organisent les chaînes qui dépassent le bord de la marge de droite. Les chaînes de texte réelles sont imprimées par la méthode *DrawString*, que vous avez déjà employée dans ce chapitre. La propriété *e.HasMorePages* précise s'il y a d'autres pages à imprimer. S'il ne reste aucune page, la propriété *HasMorePages* prend la valeur *False* et le contenu de la variable *ChaineAImprimer* reprend pour la valeur le contenu de l'objet *RichTextBox1*.

5. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements et choisissez le dossier de destination c:\vb08epe\chap17.

Vous en avez saisi du code ! Vous êtes maintenant prêt à exécuter le programme et à voir comment fonctionne l'impression de fichiers texte comportant plusieurs pages.

Exécuter le programme Imprimer un fichier

Astuce Le programme Imprimer un fichier complet est disponible dans le dossier c:\vb08epe\chap17\Imprimer un fichier.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

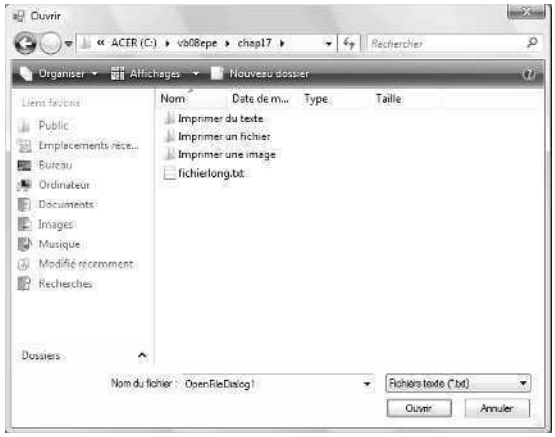
   Le programme s'exécute dans l'environnement de développement. Notez que le bouton Imprimer est désactivé puisque vous n'avez pas encore sélectionné de fichier.

2. Cliquez sur le bouton Ouvrir.

   Ce programme affiche une boîte de dialogue Ouvrir.

3. Localisez le dossier c:\vb08epe\chap17 et cliquez sur le fichier fichierlong.txt.

Voici à quoi ressemble la boîte de dialogue Ouvrir (sous Windows Vista) :

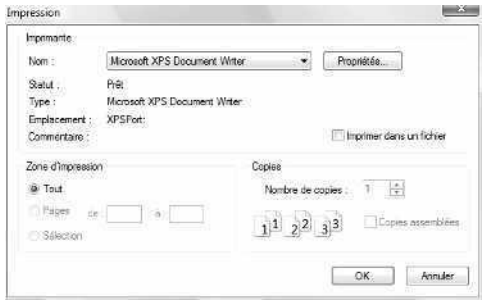


4. Cliquez sur Ouvrir pour sélectionner le fichier.

Votre programme charge le fichier texte dans l'objet zone de texte enrichie sur le formulaire et active le bouton Imprimer. Le fichier est long et contient quelques lignes qui sont renvoyées à la ligne suivante pour vous permettre de tester les options de marge et d'impression de plusieurs pages. Votre formulaire présente un résultat similaire à

5. Vérifiez que l'imprimante est allumée et cliquez sur le bouton Imprimer.

Visual Basic affiche la boîte de dialogue Imprimer, personnalisée avec le nom et les paramètres de votre imprimante, comme le montre la figure suivante :



De nombreuses options de la boîte de dialogue Imprimer sont actives. N'hésitez pas à les tester.

6. Cliquez sur Imprimer pour imprimer le document.

   Votre programme soumet la tâche d'impression de quatre pages à la file d'attente de Windows. Après un moment (et si votre imprimante est prête), l'imprimante démarre l'impression du document. À l'instar des exercices précédents, une boîte de dialogue s'affiche automatiquement pour indiquer l'état de l'impression et le nombre de pages que comporte le document imprimé.

7. Cliquez sur le bouton Fermer sur le formulaire pour arrêter le programme.

Vous venez de créer un ensemble de routines d'impression polyvalentes que vous pouvez ajouter à n'importe quelle application Visual Basic qui doit imprimer plusieurs pages de texte.

Aller plus loin : Ajouter les boîtes de dialogue Aperçu avant impression et Mise en page

L'application Imprimer un fichier est prête à gérer plusieurs tâches d'impression, mais son interface ne répond pas visuellement à celle d'une application Windows. Pour améliorer la souplesse et l'intérêt du programme, vous pouvez ajouter d'autres options qui compléteront la boîte de dialogue Imprimer du précédent exercice.

L'onglet Impression de la Boîte à outils propose deux autres contrôles d'impression, qui fonctionnent de manière similaire aux contrôles *PrintDialog* et *OpenFileDialog* que vous avez déjà utilisés :

- Le contrôle *PrintPreviewDialog* affiche une boîte de dialogue Aperçu avant impression personnalisée ;
- Le contrôle *PageSetupDialog* affiche une boîte de dialogue Mise en page personnalisée.

À l'instar des autres boîtes de dialogue, vous pouvez ajouter ces contrôles d'impression au formulaire en vous servant de la Boîte à outils ou en les créant par programmation.

Dans les prochains exercices, vous allez ajouter les boîtes de dialogue Aperçu avant impression et Mise en page au programme Imprimer un fichier que vous venez de créer. Dans les fichiers d'exercices terminés, ce projet s'intitule Fenêtre impression pour différencier le code des deux projets, mais rien ne vous empêche d'ajouter directement des fonctionnalités de boîte de dialogue au projet Imprimer un fichier.

Ajouter les contrôles *PrintPreviewDialog* et *PageSetupDialog*

1. Si vous n'avez pas réalisé le dernier exercice, ouvrez le projet Imprimer un fichier qui se trouve dans le dossier c:\vb08epe\chap17\Imprimer un fichier.

   Le projet Imprimer un fichier constitue le point de départ de ce projet.
2. Affichez le formulaire et servez-vous du contrôle *Button* pour ajouter deux boutons dans la partie supérieure du formulaire.
3. Sur l'onglet Impression de la Boîte à outils, double-cliquez sur le contrôle *PrintPreviewDialog*.

   Un objet boîte de dialogue Aperçu avant impression s'ajoute à la zone des composants.
4. Sur l'onglet Impression de la Boîte à outils, double-cliquez sur le contrôle *PageSetupDialog*.

   Un objet boîte de dialogue Mise en page s'ajoute à la zone des composants. Si les objets de la zone des composants se chevauchent, faites-les glisser vers un emplacement plus approprié ou cliquez droit dans la zone des composants et choisissez Aligner les icônes.
5. Définissez les propriétés suivantes pour les objets bouton du formulaire :

Objet	Propriété	Paramètre
Button1	*Name*	btnMiseEnPage
	Enabled	False
	Text	« Mise en page »
Button2	*Name*	btnApercu
	Enabled	False
	Text	« Aperçu »

Voici à quoi ressemble votre formulaire :

6. Double-cliquez sur le bouton Mise en page (*btnMiseEnPage*) pour afficher la procédure événementielle *btnMiseEnPage_Click* dans l'Éditeur de code.

7. Tapez le code suivant :

```
Try
    'Charge les paramètres de la page et affiche la boîte de dialogue Mise en page
    PageSetupDialog1.PageSettings = ParamPageImpr
    PageSetupDialog1.ShowDialog()
Catch ex As Exception
    'Affiche un message d'erreur
    MessageBox.Show(ex.Message)
End Try
```

Le code qui crée la boîte de dialogue Mise en page dans ce programme est simple puisque la variable *ParamPageImpr* a déjà été définie dans la partie supérieure du formulaire. Cette variable contient les informations de définition de la page en cours. Lorsque cette variable est assignée à la propriété *PageSettings* de l'objet *PageSetupDialog1*, la méthode *ShowDialog* charge automatiquement une boîte de dialogue qui permet à l'utilisateur de modifier les paramètres définis par défaut par le programme pour l'orientation, les marges et ainsi de suite. Le gestionnaire d'erreurs *Try…Catch* gère toute erreur qui pourrait se produire à l'émission de la méthode *ShowDialog*.

8. Affichez à nouveau le formulaire et double-cliquez sur le bouton Aperçu (*btnApercu*) pour afficher la procédure événementielle *btnApercu_Click* dans l'Éditeur de code.

9. Tapez le code suivant :

```
Try
    'Spécifie les paramètres de la page en cours
    PrintDocument1.DefaultPageSettings = ParamPageImpr
    'Spécifie le document pour la boîte de dialogue Aperçu avant impression et l'affiche
    ChaineAImprimer = RichTextBox1.Text
    PrintPreviewDialog1.Document = PrintDocument1
    PrintPreviewDialog1.ShowDialog()
Catch ex As Exception
    'Affiche un message d'erreur
    MessageBox.Show(ex.Message)
End Try
```

De manière similaire, la procédure événementielle *btnApercu_Click* assigne la variable *ParamPageImpr* à la propriété *DefaultPageSettings* de l'objet *PrintDocument1* puis elle copie le texte qui se trouve dans l'objet zone de texte enrichie dans la variable *ChaineAImprimer* et ouvre la boîte de dialogue Aperçu avant impression. L'Aperçu avant impression utilise automatiquement les données des paramètres de page pour afficher une représentation visuelle du document tel qu'il sera imprimé : inutile d'afficher ces informations manuellement.

Vous allez maintenant modifier légèrement le code de la procédure événementielle *btnOuvrir_Click*.

10. Localisez la procédure événementielle *btnOuvrir_Click* dans l'Éditeur de code.

Cette procédure affiche la boîte de dialogue Ouvrir, ouvre un fichier texte et active les boutons d'impression. Comme nous venons d'ajouter les boutons Mise en page et Aperçu, il nous faut également ajouter le code qui active ces deux boutons.

11. Placez le curseur à la fin de la procédure événementielle, juste avant le dernier bloc de code *Catch* et localisez l'instruction suivante :

```
btnImprimer.Enabled = True
```

12. Sous cette instruction, ajoutez les lignes de code suivantes :

```
btnMiseEnPage.Enabled = True
btnApercu.Enabled = True
```

Le programme active dorénavant les boutons d'impression en présence d'un document à imprimer.

13. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

Tester les fonctionnalités Mise en page et Aperçu avant impression

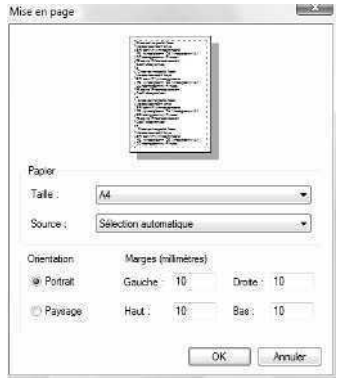
> **Astuce** Le programme Fenêtre d'impression complet est disponible dans le dossier c:\vb08epe\chap17\Fenêtre impression.

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. Le programme s'ouvre : seul le premier bouton est activé.
2. Cliquez sur le bouton Ouvrir et ouvrez le fichier fichierlong.txt qui se trouve dans le dossier c:\vb08epe\chap17.

   Les trois autres objets bouton sont maintenant actifs :

3. Cliquez sur le bouton Mise en page.

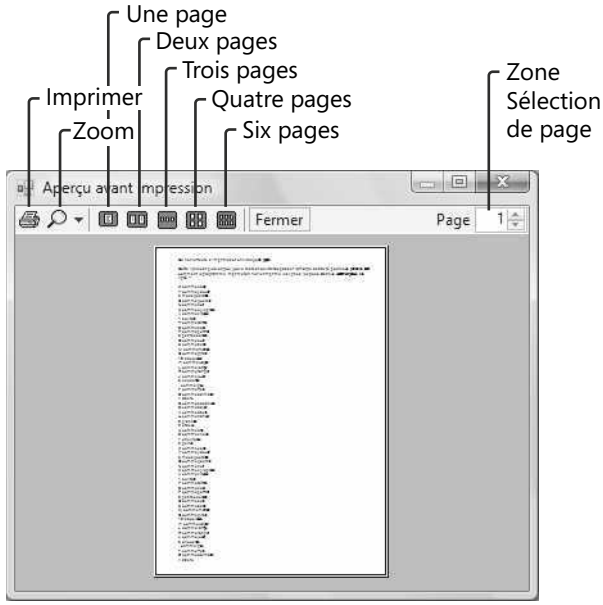
   Le programme affiche la boîte de dialogue Mise en page :



   Cette boîte de dialogue propose de nombreuses options, dont la possibilité de modifier la taille et la source du papier, l'orientation (Portrait ou Paysage) et les marges (Gauche, Droite, Haut, Bas).

4. Remplacez la marge Gauche par 20 et cliquez sur OK. La marge de gauche est à présent de 20 millimètres.
5. Cliquez sur le bouton Aperçu.

   Le programme affiche la boîte de dialogue Aperçu avant impression :



   Si vous avez déjà fait appel à la commande Aperçu avant impression dans Microsoft Office Word ou Microsoft Office Excel, vous reconnaîtrez plusieurs des boutons et fonctionnalités de cette boîte de dialogue Aperçu avant impression. Les contrôles Zoom, Une page, Deux pages, Trois pages, Quatre pages, Six pages et la zone Sélection de la page fonctionnent automatiquement dans cette boîte de dialogue : aucun code n'est nécessaire.
6. Cliquez sur le bouton Quatre pages pour afficher simultanément les quatre pages du document.

7. Cliquez sur le bouton Agrandir dans la barre de titre de la boîte de dialogue pour obtenir un affichage plein écran.
8. Cliquez sur la flèche qui accompagne le bouton Zoom et choisissez 150%.

   Voici à quoi ressemble votre écran :

9. Cliquez sur le bouton Zoom et revenez à Auto.
10. Cliquez sur le bouton Trois pages puis sur la flèche Bas dans la zone Sélection de la page pour afficher les pages 2 à 4.

    Comme vous pouvez le voir, la fenêtre Aperçu avant impression est intéressante et quelques lignes de code suffisent pour l'incorporer aux programmes.
11. Si vous souhaitez à nouveau tester l'impression de l'ensemble du document, cliquez sur le bouton Imprimer.
12. Lorsque vous avez terminé, cliquez sur le bouton Fermer pour fermer la boîte de dialogue Aperçu avant impression puis sur le bouton Fermer du formulaire pour arrêter le programme.

    Nous en avons terminé avec les imprimantes pour l'instant.

Rappel du chapitre 17

Pour	Faites ceci
Faciliter les références aux classes d'impression dans vos projets	Ajoutez l'instruction *Imports* suivante dans la partie supérieure du formulaire : `Imports System.Drawing.Printing`

Créer un gestionnaire d'événements d'impression	Ajoutez l'instruction *AddHandler* et l'opérateur *AddressOf*. Par exemple :
	`AddHandler PrintDocument1.PrintPage, _`
	`  AddressOf Me.PrintGraphic`

Créer un objet *PrintDocument* dans le projet	Sur l'onglet Impression de la Boîte à outils, double-cliquez sur le contrôle *PrintDocument*. *ou* Incluez la déclaration de variable suivante dans le code : `Dim ImprimerDoc As New PrintDocument`

Imprimer des images à partir d'un gestionnaire d'événements d'impression	Servez-vous de la méthode *Graphics.DrawString*. Par exemple :
	`ev.Graphics.DrawImage(Image.FromFile(TextBox1.Text), _`
	`  ev.Graphics.VisibleClipBounds)`

Imprimer du texte à partir d'un gestionnaire d'événements d'impression	Servez-vous de la méthode *Graphics.DrawString* dans un gestionnaire d'événements. Par exemple :
	`ev.Graphics.DrawString(TextBox1.Text), _`
	`  New Font("Arial", 11, FontStyle.Regular), _`
	`  Brushes.Black, 120, 120)`

Appeler un gestionnaire d'événements d'impression	Servez-vous de la méthode *Print* d'un objet de type *PrintDocument*. Par exemple : `ImprimerDoc.Print()`
Imprimer des documents texte de plusieurs pages	Écrivez un gestionnaire pour l'événement *PrintPage* qui reçoit un argument de type *PrintPageEventArgs*. Calculez la zone rectangulaire de la page destinée au texte, utilisez la méthode *MeasureString* pour déterminer la quantité de texte qui entre dans la page en cours et servez-vous de la méthode *DrawString* pour imprimer le texte sur la page. Si d'autres pages sont nécessaires, attribuez la valeur True à la propriété *HasMorePages*. Lorsque tout le texte est imprimé, positionnez la propriété *HasMorePages* sur False.

Ouvrir un fichier texte avec la classe *FileStream* et le charger dans un objet *RichTextBox*	Créez une variable de type *FileStream*, spécifiez le chemin d'accès et le type de fichier, chargez le flux dans un contrôle *RichTextBox* et fermez le flux. Par exemple :
	`Imports System.IO  'dans la partie supérieure du formulaire`
	`...`
	`Dim MonFileStream As New FileStream( _`
	`  FilePath, FileMode.Open)`
	`RichTextBox1.LoadFile(MonFileStream, _`
	`  RichTextBoxStreamType.PlainText)`
	`MonFileStream.Close()`

Afficher des boîtes de dialogue d'impression dans vos programmes	Servez-vous des contrôles *PrintDialog*, *PrintPreviewDialog* et *PageSetupDialog* qui se trouvent sur l'onglet Impression de la Boîte à outils.

Partie IV

Programmer pour les bases de données et le web

Dans cette partie :

Dans la partie IV, vous allez apprendre à exploiter les informations stockées dans des bases de données et sur des sites web. Pour commencer, nous étudierons ADO.NET, un important paradigme de travail avec les informations de base de données. Nous verrons comment afficher, modifier et effectuer des recherches sur le contenu d'une base de données en utilisant une combinaison de code et de contrôles de formulaires Windows. Microsoft Visual Studio 2008 a été spécifiquement conçu pour créer des applications qui donnent accès à une grande variété de sources de données. Ces interfaces personnalisées sont appelées *interfaces d'accès de base de données*, ce qui signifie qu'au sein de votre application Microsoft Visual Basic, l'utilisateur bénéficie d'une interface plus conviviale qu'en manipulant des enregistrements bruts issus de la base de données. Avec Visual Studio 2008, vous créez des applications *centrées sur les données*. Autrement dit, par l'entremise de votre application, l'utilisateur est invité à explorer le potentiel de connexions à des sources de données, locales ou distantes. De surcroît, l'application place ces données au centre de l'expérience informatique de l'utilisateur.

Chapitre 18

Démarrer avec ADO.NET

À la fin de ce chapitre, vous saurez :

- Utiliser l'Assistant Configuration de source de données pour établir une connexion avec une base de données et créer un dataset
- Utiliser le Concepteur de datasets et la fenêtre Sources de données pour examiner les membres du dataset et créer des objets liés sur les formulaires
- Créer des applications centrées sur les données en faisant appel aux objets dataset et navigateur de données
- Utiliser des contrôles *TextBox* et *MaskedTextBox* liés pour afficher les informations de base de données dans un formulaire Windows
- Écrire des instructions SQL pour filtrer et trier les informations d'un dataset avec l'outil Générateur de requêtes Visual Studio

Dans ce chapitre, vous découvrirez ADO.NET et les applications centrées sur les données. Vous utiliserez l'Assistant Configuration de source de données pour établir une connexion avec une base de données Access installée sur votre système, vous créerez un dataset représentant un sous-ensemble de champs et d'enregistrements issus d'une table de la base de données et vous vous servirez du Concepteur de datasets et de la fenêtre Sources de données pour examiner les membres du dataset et créer des objets liés sur vos formulaires. Vous apprendrez également à utiliser les contrôles *TextBox* et *MaskedTextBox* pour présenter les informations de base de données à l'utilisateur et à écrire des instructions SQL SELECT qui filtrent les datasets (et donc ce que l'utilisateur voit et utilise).

Programmation de bases de données avec ADO.NET

Une *base de données* est une collection organisée d'informations stockée dans un fichier. La gamme de produits permettant de créer des bases de données est vaste et comprend notamment Microsoft Access, Microsoft SQL Server et Oracle. Il est également possible de stocker et de transmettre les informations de base de données à l'aide de XML, un format de fichier destiné à l'échange de données structurées *via* l'Internet et avec d'autres paramètres.

La création et la manipulation des bases de données représentent aujourd'hui une tâche essentielle de toute multinationale, institution gouvernementale, organisme à but non lucratif et petite entreprise. Les ressources de données – adresses des clients, inventaires de fabrication, soldes de comptes, enregistrements des employés, listes de donneurs et historiques de commande – sont devenues un élément vital dans le monde de l'entreprise.

Vous pouvez employer Visual Studio 2008 pour créer de nouvelles bases de données, mais il est principalement conçu pour afficher, analyser et manipuler les informations se trouvant dans des bases de données existantes. ADO.NET 2.0, apparu dans Studio .NET 2002, reste le modèle standard de programmation de bases de données dans Visual Studio 2008. ADO.NET a été amélioré au fil dans ans pour fonctionner avec davantage de scénarios d'accès aux données et a été optimisé pour un usage sur l'Internet. Cela signifie qu'il accède à des sources de données locales, client-serveur et fondées sur l'Internet de la même manière. Le format de données interne d'ADO.NET est XML.

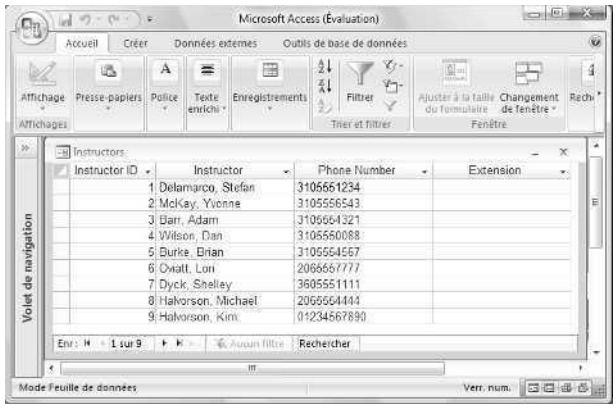
Heureusement, la plus grande partie des applications de base de données créées par des programmeurs à l'aide de Visual Basic 2005 et ADO.NET fonctionnent encore parfaitement. Les techniques de base pour l'accès à une base de données sont fondamentalement identiques en Visual Basic 2008. Il existe cependant deux nouvelles techniques de base de données en Visual Basic 2008 qui seront précieuses aux programmeurs expérimentés de base de données. Ces techniques sont LINQ (*Language_Integrated Query*) et le ADO.NET Entity Framework.

LINQ est incorporé à Visual Studio 2008 et permet d'écrire des requêtes de base de données orientées objet directement dans du code Visual Basic. Peu de temps après la diffusion initiale de Visual Studio 2008, Microsoft s'est engagé à diffuser ADO.NET Entity Framework. Celui-ci apporte un nouveau modèle objet, de puissantes nouvelles fonctionnalités et des outils qui libèrent encore plus les applications de base de données de dépendances codées en dur vis-à-vis d'un modèle logique ou d'un moteur de base de donnée spécifique. Alors que les techniques de bases de données et l'Internet poursuivent leurs avancées, ADO.NET continuera son évolution et les programmeurs Visual Basic devraient rester en bonne position pour en tirer partie.

Terminologie des bases de données

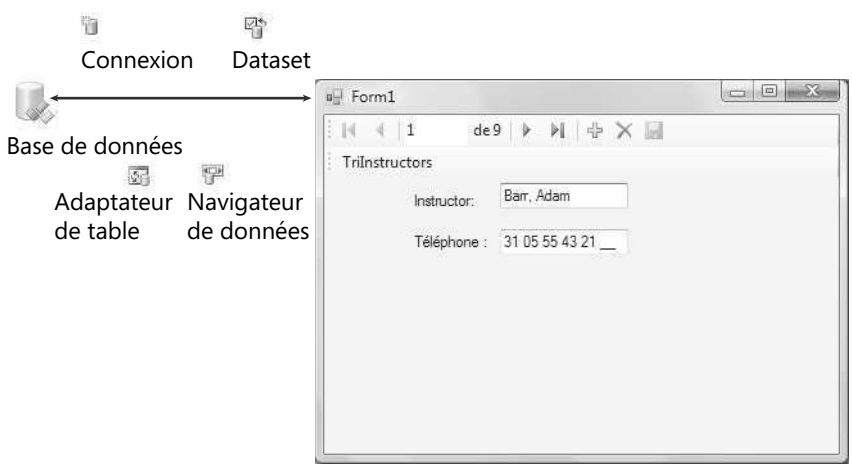
Un thème sous-jacent dans la section précédente est que les programmeurs de bases de données sont souvent confrontés à la nécessité de décoder et de comprendre de nouvelles techniques : une réorientation souvent signalée par les termes « nouveau paradigme » ou « nouveau modèle de bases de données ». Même si apprendre en permanence de nouvelles techniques peut se révéler rapidement frustrant, cette rapidité d'évolution peut s'expliquer par la relative jeunesse de la programmation d'applications Windows de bases de données à multiples étages et distribuées. Interviennent également les innovations techniques, les nécessités sécuritaires et les défis de programmation web qui excèdent le contrôle de l'équipe de développement de Visual Studio. Nous commencerons donc au début dans ce chapitre : en programmation de base de données plus encore qu'avec pratiquement n'importe quel sujet, vous devrez découvrir les thèmes étape par étape. Commençons par un peu de terminologie fondamentale de base de données.

Un *champ* (également appelé *colonne*) représente une catégorie d'informations stockée dans une base de données. Dans une base de données de clients, vous trouvez généralement des champs pour les noms des clients, leurs adresses, leurs numéros de téléphone et des commentaires. Toutes les informations relatives à un client ou une entreprise spécifique constituent un *enregistrement* (plus communément appelé *ligne*). Lors de la création d'une base de données, les informations sont saisies dans une *table* composée de champs et d'enregistrements. Les enregistrements correspondent aux lignes de la table et les champs aux colonnes, comme le montre la figure suivante :



Une *base de données relationnelle* se compose de plusieurs tables liées. La majorité des bases de données auxquelles vous vous connectez depuis Visual Studio sont des bases de données relationnelles qui contiennent plusieurs tables de données organisées autour d'un thème particulier.

Dans ADO.NET, vous faites appel à divers objets pour récupérer et modifier les informations d'une base de données. L'illustration ci-après donne un aperçu de l'approche étudiée dans ce chapitre :



Pour commencer, vous établissez une *connexion* qui spécifie les informations de connexion relatives à la chaîne de connexion et crée un élément auquel les autres contrôles et composants peuvent se lier. Ensuite, l'Assistant Configuration de source de données crée un *dataset* ou ensemble de données : une représentation d'une ou de plusieurs tables de bases de données que l'on prévoit d'exploiter dans le programme, car vous ne manipulez pas les données réelles, mais une copie de celles-ci. L'Assistant Configuration de source de données ajoute également un *fichier de schéma XML* au projet et associe un *adaptateur de table* et un *navigateur de données* au dataset pour gérer la récupération des données dans la base de données, la publication des changements et le déplacement d'un enregistrement au suivant dans le dataset. Vous pouvez ensuite lier les informations du dataset aux contrôles d'un formulaire à l'aide de la fenêtre Sources de données ou des paramètres de la propriété *DataBindings*.

Exploiter une base de données Access

Dans les prochaines sections, vous allez apprendre à utiliser la technologie d'accès aux données ADO.NET 2.0 dans Visual Basic 2008. Nous commencerons par nous servir de l'Assistant Configuration de source de données pour établir une connexion avec une base de données appelée Etudiants.mbd créée au format Microsoft Access 2002/2003 (elle fonctionne bien sûr aussi avec Access 2007, si vous possédez la plus récente version du logiciel de bases de données de Microsoft). La base de données Etudiants.mbd contient diverses tables d'informations académiques, intéressantes pour l'enseignant à la recherche du parcours scolaire d'un étudiant ou à un administrateur qui planifie les classes, les assigne ou crée les emplois du temps. Vous apprendrez à créer un dataset basé sur une table d'informations de la base de données Etudiants.mbd et à afficher ces informations dans un formulaire Windows. Lorsque vous aurez terminé, vous pourrez appliquer ces techniques à vos propres projets de bases de données.

Astuce L'exemple de ce chapitre a recours à une base de données Microsoft Access. Il n'est cependant pas indispensable d'installer Access pour l'utiliser. Visual Studio et ADO.NET prennent en charge le format de fichier Access, ainsi que d'autres formats. Si vous ouvrez la base de données dans Access, vous noterez qu'elle est enregistrée au format Access 2002/2003. J'ai également inclus le fichier au format Access 2000 (Etudiants_format2000.mdb) pour vous permettre de réaliser les tests avec l'exemple de base de données, même si vous disposez d'une version plus ancienne d'Access.

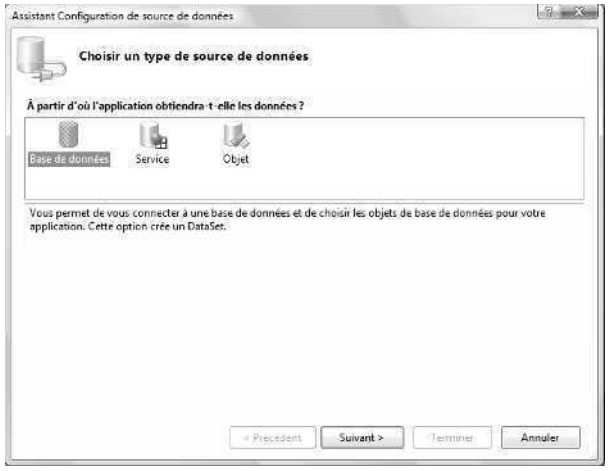
Établir une connexion avec l'Assistant Configuration de source de données

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms intitulé **Mon Formulaire ADO**.

   Un formulaire vierge s'affiche dans l'environnement de développement.

2. Dans le menu Données, cliquez sur la commande Ajouter une nouvelle source de données.

   L'Assistant Configuration de source de données s'affiche dans l'environnement de développement :



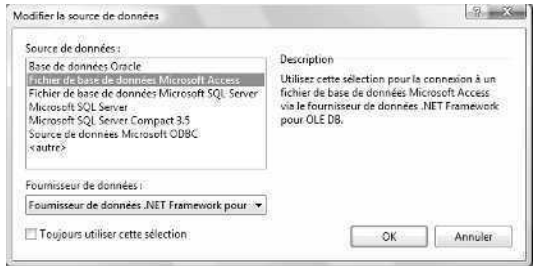
   Cet assistant est une fonctionnalité de l'environnement de développement Visual Studio 2008 qui prépare automatiquement le programme Visual Basic à recevoir les informations de bases de données. L'assistant vous demande le type de base de données auquel vous voulez vous connecter (une base de données locale ou distante, un service web ou un objet de données personnalisé créé par vos soins), établit une connexion avec les données et crée un dataset au sein du programme pour accueillir des tables et des champs spécifiques de la base de données. Au final, l'assistant ouvre la fenêtre Sources de données et la remplit avec une représentation visuelle de chaque objet de base de données que vous utilisez dans le programme.

3. Dans l'Assistant Configuration de source de données, cliquez sur l'icône Base de données puis sur Suivant.

L'écran de l'assistant permet d'établir une connexion avec la base de données en créant une instruction appelée *chaîne de connexion*. La chaîne de connexion contient les informations dont Visual Studio a besoin pour ouvrir et extraire les informations d'un fichier de base de données : nom du chemin d'accès et nom du fichier, voire données sensibles comme un nom d'utilisateur et un mot de passe. En conséquence, la chaîne de connexion est traitée avec la plus grande attention dans l'Assistant de Configuration de source de données. Veillez à la protéger contre tout accès non autorisé pendant la copie des fichiers d'un emplacement à un autre.

4. Cliquez sur le bouton Nouvelle connexion.

La première fois que vous cliquez sur le bouton Nouvelle connexion, la boîte de dialogue Choisir une source de données s'affiche, vous invitant à sélectionner le format de base de données que vous envisagez d'utiliser. Si la boîte de dialogue s'intitule Ajouter une connexion au lieu de Choisir une source de données, cela signifie simplement que votre exemplaire de Visual Studio a déjà été configuré pour favoriser un format de base de données. Aucun problème : cliquez simplement sur le bouton Modifier dans la boîte de dialogue Ajouter une connexion et vous verrez la même boîte de dialogue que les utilisateurs qui se servent de l'assistant pour la première fois, excepté que la barre de titre contient l'intitulé Modifier la source de données, comme le montre l'illustration suivante :



La boîte de dialogue Modifier/Choisir une source de données permet de sélectionner le format de base de données favori, employé comme format par défaut par Visual Studio. Pour ce chapitre, nous sélectionnerons le format Microsoft Access, mais rien ne vous empêche d'en changer à tout moment. Il est également possible d'établir plusieurs connexions, chacune à un type de base de données différent, au sein d'un même projet.

5. Cliquez sur Fichier de base de données Microsoft Access puis sur OK (ou Continuer).

La boîte de dialogue Ajouter une connexion s'affiche, comme le montre l'illustration suivante :

Vous allez maintenant indiquer l'emplacement et les paramètres de connexion de la base de données permettant à Visual Studio de créer une chaîne de connexion valide.

6. Cliquez sur Parcourir.

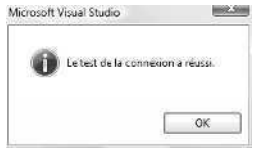
La boîte de dialogue Sélectionnez le fichier de base de données Microsoft Access s'affiche. Elle fonctionne de la même manière que la boîte de dialogue Ouvrir.

7. Localisez le dossier c:\vb08epe\chap18, cliquez sur le fichier Etudiants.mbd et cliquez sur Ouvrir.

Vous avez sélectionné la base de données Access au format 2002/2003 créée pour montrer comment s'affichent les champs et enregistrements d'une base de données dans un programme Visual Basic. La boîte de dialogue Ajouter une connexion s'affiche à nouveau : elle contient à présent le nom du chemin d'accès enregistré. Je n'ai pas restreint l'accès à ce fichier : inutile donc de préciser un nom d'utilisateur ou un mot de passe. Toutefois, si l'utilisation de votre base de données exige un nom d'utilisateur et/ou un mot de passe, vous pouvez le préciser dans cette boîte de dialogue, dans les zones Nom d'utilisateur et Mot de passe. Ces valeurs sont alors incluses dans la chaîne de connexion.

8. Cliquez sur le bouton Tester la connexion.

   Visual Studio tente d'ouvrir le fichier de base de données désigné avec la chaîne de connexion que l'assistant a créée. Si le format de la base de données et éventuellement le nom d'utilisateur et le mot de passe sont corrects, le message suivant s'affiche :



9. Cliquez sur OK pour fermer la boîte de message puis à nouveau sur OK pour fermer la boîte de dialogue Ajouter une connexion.

   Visual Studio affiche à nouveau l'Assistant Configuration de source de données.

10. Cliquez sur le signe plus (+), en regard de l'élément Chaîne de connexion pour afficher la chaîne de connexion complète.

    La page de l'assistant est similaire à la suivante :

La chaîne de connexion identifie un *fournisseur* (également appelé *fournisseur géré*) intitulé Microsoft.Jet.OLEDB.4.0, qui représente le composant de base de données sous-jacent qui sait comment se connecter à une base de données et en extraire les données. Les deux fournisseurs les plus employés proposés par Visual Studio sont Microsoft Jet OLE BD et Microsoft SQL Server. Des fournisseurs tiers sont cependant disponibles pour prendre en charge de nombreux autres formats de base de données.

11. Cliquez sur le bouton Suivant.

L'assistant présente un message d'alerte qui indique qu'une nouvelle base de données locale a été sélectionnée et vous demande si elle doit être copiée dans les dossiers du projet (ce message s'affiche uniquement la première fois que vous établissez la connexion avec un fichier de base de données local. Si vous avez déjà fait cet exercice, le message ne s'affiche pas).

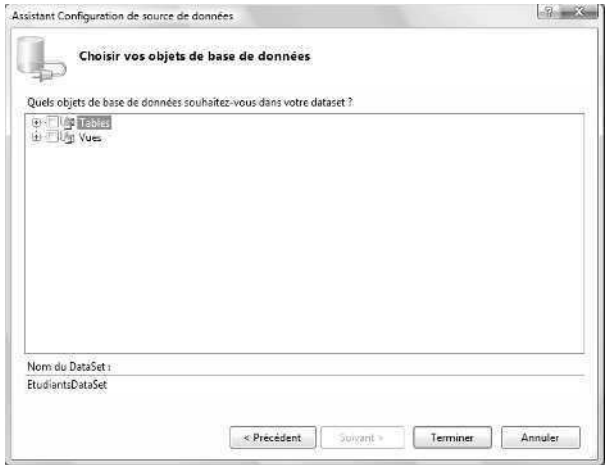
12. Cliquez sur Non pour éviter de créer une copie supplémentaire de la base de données.

Ce projet ne sera pas commercialisé. Il s'agit uniquement d'un exemple de programme : une copie est inutile.

L'Assistant Configuration de source de données vous pose à présent la question suivante : « Voulez-vous enregistrer la chaîne de connexion dans le fichier de configuration de l'application ? ». Cette option est sélectionnée par défaut et, dans cet exemple, le nom recommandé pour la chaîne est « EtudiantsConnectionString ». Il est généralement préférable d'enregistrer cette chaîne au sein du fichier de configuration par défaut de l'application. En effet, si vous modifiez l'emplacement de la base de données, vous pouvez changer la chaîne dans le fichier de configuration (listé dans l'Explorateur de solutions), ce qui évite de rechercher la chaîne de connexion dans le code et de recompiler l'application.

13. Cliquez sur Suivant pour enregistrer la chaîne de connexion par défaut.

Vous êtes ensuite invité à sélectionner le sous-ensemble d'objets de base de données à utiliser pour ce projet, comme le montre la boîte de dialogue suivante :



Remarque Visual Studio permet d'utiliser seulement une partie d'une base de données ou de combiner différentes bases de données : fonctionnalités intéressantes lorsque l'on crée des applications centrées sur les bases de données.

Les éléments sélectionnés dans cette boîte de dialogue sont appelés *objets de base de données*. Ces derniers regroupent les tables de champs et d'enregistrements, les affichages, les procédures stockées, les fonctions ainsi que tout autre élément unique à la base de données. Le terme collectif de tous les objets de base de données que l'on peut sélectionner est *dataset*. Dans ce projet, le nom par défaut du dataset est *EtudiantsDataSet*, que vous pouvez modifier dans la zone Nom du DataSet.

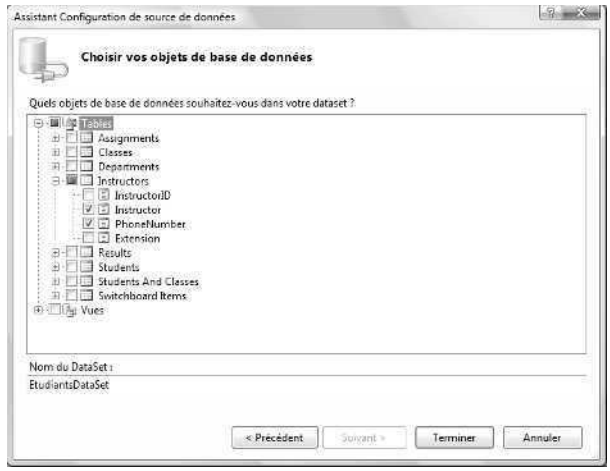
Astuce Notez que le dataset que vous créez maintenant *représente* uniquement les données contenues dans la base de données : si vous ajoutez, supprimez ou modifiez les enregistrements du dataset, les tables de la base de données sous-jacente ne sont pas modifiées tant que vous n'émettez pas une commande qui écrive vos changements dans la base de données d'origine. Les programmeurs de bases de données appellent ce type d'organisation une *source de données déconnectée*, ce qui signifie qu'il existe une couche d'abstraction entre la base de données réelle et le dataset.

14. Cliquez sur le signe plus (+) en regard du nœud Tables pour développer la liste des tables incluses dans la base de données Etudiants.mbd.

Parmi les éléments de la liste des tables qui apparaît dans l'assistant figurent *Assignments*, *Classes*, *Departments* et *Instructors*. Chaque table concerne un aspect de la planification scolaire. Dans cet exemple, nous exploiterons la table *Instructors*.

15. Cliquez sur le signe plus (+) en regard du nœud Instructors puis cochez les cases en regard des champs *Instructor* et *Phone Number*.

Vous allez ajouter ces deux champs au dataset *EtudiantsDataSet*. Voici à quoi ressemble l'assistant :



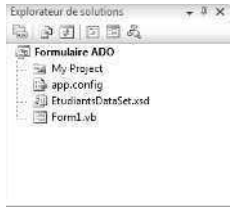
16. Cliquez sur Terminer pour achever l'opération et fermer l'Assistant Configuration de source de données.

Visual Studio ajoute une connexion de base de données au projet et configure le dataset avec les objets de base de données sélectionnés (selon la manière dont l'environnement de développement Visual Studio a été utilisé et configuré, vous voyez ou non un onglet ou une fenêtre Source de données).

17. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements. Désignez le dossier c:\vb08epe\chap18 comme emplacement.

18. Si l'Explorateur de solutions n'est pas visible, ouvrez-le pour afficher les principaux fichiers et composants contenus dans le projet Formulaire ADO.

Voici à quoi ressemble votre écran :



Outre les entrées standards d'un projet, l'Explorateur de solutions contient à présent un nouveau fichier intitulé EtudiantsDataSet.xsd. Il s'agit d'un schéma XML qui décrit les tables, champs, types de données et autres éléments du dataset que vous venez de créer. La présence du fichier de schéma signifie que l'on a ajouté un *dataset typé* au projet (les datasets typés possèdent un fichier de schéma associé que ne possèdent pas les datasets non typés). Les datasets typés présentent l'avantage d'accepter la fonctionnalité d'achèvement (Microsoft Intellisense) des instructions de l'Éditeur de code Visual Studio et de fournir des informations relatives aux champs et tables employés.

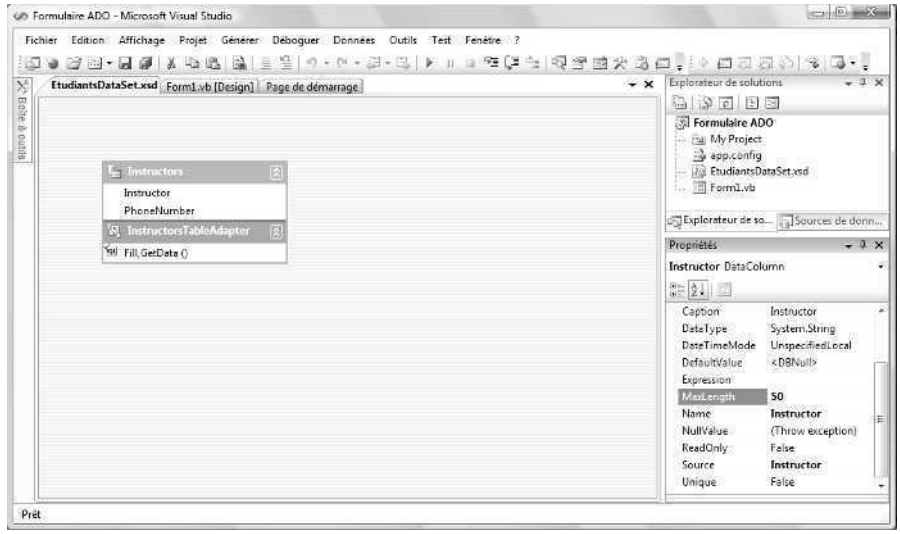
19. Dans l'Explorateur de solutions, cliquez sur le fichier de schéma puis sur le bouton Concepteur de vues.

Une représentation des tables, champs et adaptateur de données relatifs au nouveau dataset s'affiche dans un outil visuel appelé *Concepteur de dataset*. Les outils du Concepteur de dataset permettent de créer des composants qui communiquent entre la base de données et l'application, ce que les programmeurs de bases de données appellent des *composants de la couche d'accès aux données*. Dans cet environnement, vous pouvez créer et modifier les adaptateurs de tables, les requêtes d'adaptateur de table, les tables de données, les colonnes de données et les relations des données. Il sert également à réviser et définir d'importantes propriétés relatives aux objets du dataset, comme la longueur des champs de la base de données et les types de données associés aux champs.

20. Cliquez sur le champ *Instructor* et appuyez sur la touche F4 pour activer la fenêtre Propriétés.

21. Cliquez sur la propriété *MaxLength*.

Voici à quoi ressemble votre écran :



On y voit le Concepteur de dataset hébergeant un dataset actif intitulé *Etudiants-DataSet*. Dans la fenêtre Propriétés, la propriété *MaxLength* autorise un maximum de 50 caractères dans le champ *Instructor*. Bien que cette longueur puisse sembler suffisante, il est possible de l'ajuster (ainsi que les autres) si vous jugez les paramètres de la base de données sous-jacente inadaptés à l'application.

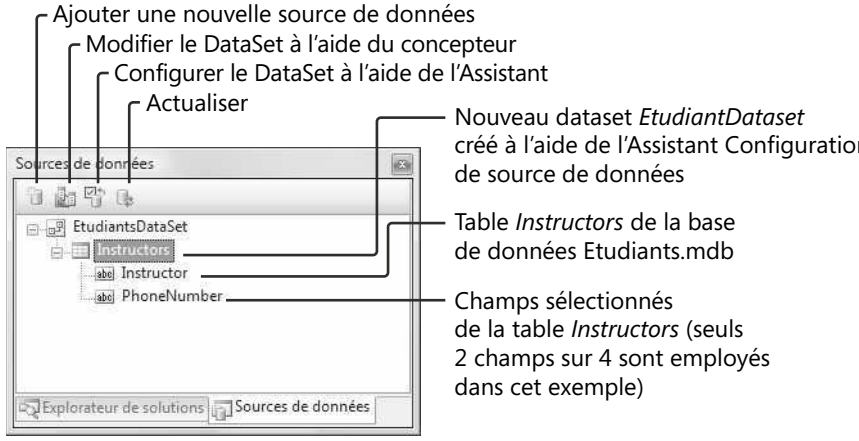
Oublions momentanément le Concepteur de dataset pour nous consacrer à l'exemple d'application de base de données dans la fenêtre Sources de données.

La fenêtre Sources de données

La fenêtre Sources de données est une fonctionnalité utile de l'environnement de développement Visual Studio 2008. Elle a pour objectif d'afficher une représentation visuelle des datasets configurés pour être employés dans un projet et de simplifier la création des liens entre ces datasets et les contrôles du formulaire. Rappelez-vous qu'un dataset n'est autre qu'une représentation temporaire des informations de la base de données dans le programme et que chaque dataset contient uniquement un sous-ensemble des tables et des champs du fichier de la base de données, autrement dit uniquement les éléments que vous avez sélectionnés dans le cadre de l'Assistant Configuration de source de données. Le dataset s'affiche sous forme d'une arborescence hiérarchisée dans la fenêtre Sources de données, avec un nœud racine pour chaque objet sélectionné dans l'assistant. Chaque fois que vous exécutez l'assistant pour créer un nouveau dataset, une nouvelle arborescence est ajoutée à la fenêtre Sources de données, vous offrant un accès à une large palette de sources et d'affichages de données au sein d'un même programme.

Si vous avez suivi les instructions pour sélectionner les champs dans la table *Instructors* de la base de données Etudiants, la fenêtre Sources de données contient à présent des éléments intéressants. Pour préparer les prochains exercices et afficher la fenêtre Sources de données, affichez à nouveau le formulaire (cliquez sur l'onglet Form1.vb) puis sur la commande Afficher les sources de données du menu Données (vous pouvez également cliquer sur l'onglet Sources de données dans l'Explorateur de solutions, s'il est visible).

Dans la fenêtre Sources de données, développez la table *Instructors* pour afficher les deux champs sélectionnés. Voici à quoi ressemble la fenêtre Sources de données :



La manière la plus simple pour afficher les informations d'un dataset sur un formulaire (à l'attention des utilisateurs) consiste à faire glisser les objets à partir de la fenêtre Sources de données vers le Concepteur de formulaires (le Concepteur employé dans les précédents chapitres que j'appelle ici Concepteur de formulaires pour le différencier du Concepteur de datasets).

Le chapitre 19, « Présenter les données avec le contrôle *DataGridView* », décrit comment afficher des tables entières de données dans un formulaire. Pour le reste de ce chapitre, toutefois, nous verrons comment faire glisser des champs individuels de données dans le Concepteur de formulaires pour lier les contrôles aux champs sélectionnés dans la base de données Étudiants.

Utiliser la fenêtre Sources de données pour créer des objets de base de données sur un formulaire

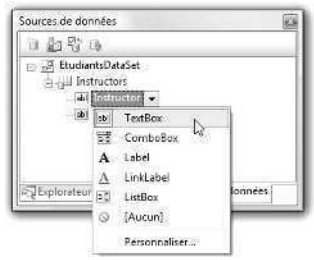
1. Dans la fenêtre Sources de données, cliquez sur le signe plus (+) en regard du nœud Instructors pour afficher les champs disponibles dans EtudiantsDataSet, si ce n'est déjà fait.

   La fenêtre Sources de données ressemble à celle de la précédente illustration. Dans Visual Studio 2008, il est possible d'afficher des champs individuels ou une table entière de données en faisant simplement glisser les objets de base de données sur le formulaire.

2. Cliquez sur le champ *Instructor* qui contient le nom de chaque enseignant de la base de données Étudiants. Une flèche apparaît à droite du champ Instructor dans la fenêtre Sources de données. Si vous cliquez sur la flèche, vous affichez une liste des options relatives à l'affichage d'un champ de base de données sur le formulaire à l'heure de le faire glisser.

3. Cliquez sur la flèche du champ *Instructor*.

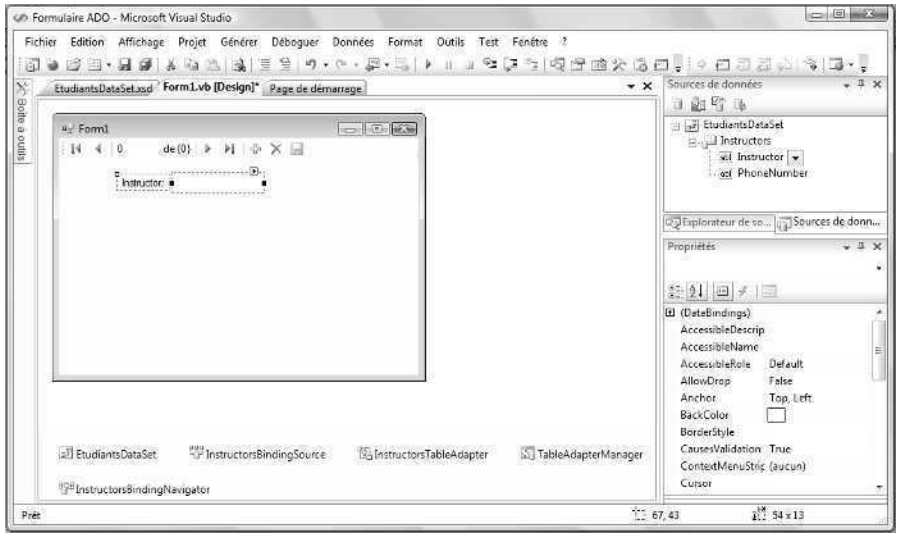
   Un clic sur cette flèche affiche une liste d'options relatives au mode d'affichage d'un champ sur le formulaire lorsque vous le glissez-déposez, comme le montre la figure suivante :



   Même si je ne l'ai pas précisé auparavant, la capacité d'afficher des informations de base de données est intégrée à la plupart des contrôles de l'onglet Contrôles communs de la Boîte à outils. Dans la terminologie Visual Studio, ces contrôles sont appelés *contrôles liés* lorsqu'ils sont connectés aux champs de donnés d'un dataset. La liste des contrôles présents dans le menu contextuel constitue un groupe d'options habituellement employées pour afficher des informations de chaînes issues d'une base de données, mais rien ne vous empêche d'ajouter des contrôles à cette liste (ou d'en supprimer) en cliquant sur la commande Personnaliser. Dans cet exemple, nous allons utiliser le contrôle *TextBox*, le contrôle lié par défaut des données chaîne.

4. Dans la liste, cliquez sur *TextBox* et faites glisser le champ *Instructor* au centre du formulaire dans le Concepteur de formulaires.

   Pendant que vous faites glisser le champ vers le formulaire, un signe plus placé sous le pointeur indique qu'il est possible d'ajouter cet objet de base de données à un formulaire. Lorsque vous relâchez le bouton de la souris, Visual Studio crée un objet zone de texte prêt à accueillir des données et place une barre de navigation à l'aspect professionnel dans la partie supérieure du formulaire. Voici à quoi ressemble le formulaire (votre fenêtre Source de données pouvant se trouver ailleurs) :



Visual Studio a créé deux objets pour ce champ *Instructor* : un objet étiquette descriptif contenant le nom du champ et un objet zone de texte lié qui présentera le contenu du champ à l'exécution du programme. Sous le formulaire, dans la zone des composants, Visual Studio a également créé plusieurs objets pour gérer les aspects internes du processus d'accès aux données :

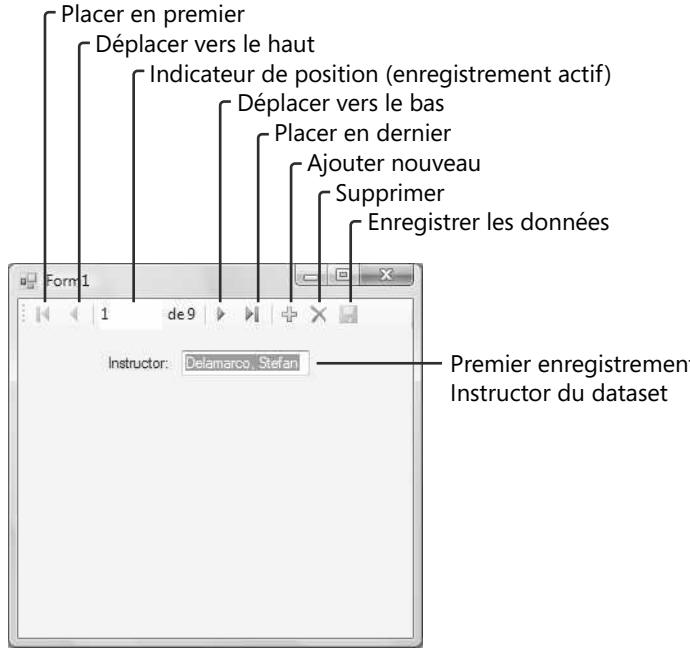
- *EtudiantsDataSet*, le dataset que vous avez créé avec l'Assistant Configuration de source de données pour représenter les champs de la base de données Étudiants ;
- *InstructorsBindingSource*, un composant intermédiaire qui agit comme un conduit entre la table Instructors et les objets liés sur le formulaire ;
- *InstructorsTableAdapter*, un composant intermédiaire qui déplace les données entre *EtudiantsDataSet* et les tables de la base de données Étudiants sous-jacente ;
- *InstructorsBindingNavigator*, qui fournit les services de navigation et les propriétés relatives à la barre de navigation et à la table *Instructors*.

Les lecteurs habitués à Visual Studio .NET 2005 reconnaîtront ces composants comme identiques aux dispositifs de connectivité de base de données de cette version. Visual Studio 2003 exigeait en revanche un objet adaptateur de données pour chaque table ou requête de base de données employée dans un projet.

Exécutons à présent le programme pour observer le fonctionnement de ces objets.

5. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

Le formulaire ADO s'exécute dans l'environnement de développement. L'objet zone de texte est chargé avec le premier enregistrement *Instructor* de la base de données (Delamarco, Stefan) et une barre de navigation apparaît dans la partie supérieure du formulaire, comme le montre l'illustration suivante :



La barre de navigation est un dispositif important des outils de programmation de base de données Visual Studio 2008. Elle contient les boutons Placer en premier, Déplacer vers le haut, Déplacer vers le bas et Placer en dernier, ainsi que l'indicateur de la position actuelle et des boutons qui (s'ils sont correctement configurés) ajoutent de nouveaux enregistrements au dataset, suppriment les enregistrements du dataset et enregistrent le dataset modifié sur le disque. Il est possible de modifier ces boutons de navigation en configurant la propriété *Items* de l'objet *BindingNavigator* dans la fenêtre Propriétés, qui présente un outil visuel appelé Éditeur de collections Items. Vous pouvez également activer ou désactiver les boutons individuellement.

6. Cliquez sur le bouton Déplacer vers le bas pour afficher le nom du deuxième enseignant du dataset.

   L'enregistrement McKay, Yvonne s'affiche.

7. Continuez à parcourir le dataset. À mesure que les noms défilent, notez que l'indicateur de position suit votre position dans la liste.

8. Cliquez sur les boutons Placer en premier et Placer en dernier pour atteindre le premier et le dernier enregistrement du dataset.

9. Supprimez le dernier enregistrement du dataset (Halvorson, Kim) en cliquant sur le bouton Supprimer.

   L'enregistrement est supprimé du dataset et l'indicateur de position montre qu'il contient à présent 8 enregistrements : (Halvorson, Michael est devenu le dernier enregistrement). Voici à quoi ressemble votre formulaire :

   Comme mentionné précédemment, le dataset ne représente qu'un sous-ensemble des tables de la base de données Etudiants employée dans ce projet : le dataset est une image déconnectée de la base de données et non la base de données elle-même. En conséquence, l'enregistrement que vous venez de supprimer l'a uniquement été du dataset chargé en mémoire pendant l'exécution du programme. Pour vérifier que le programme utilise des données déconnectées et qu'il ne modifie pas la base de données d'origine, vous allez arrêter et redémarrer le programme.

10. Cliquez sur le bouton Fermer sur le formulaire pour terminer le programme.

    Le programme s'arrête et vous revenez à l'environnement de développement.

11. Cliquez à nouveau sur Démarrer le débogage pour exécuter le programme.

    Lorsque le programme redémarre et que le formulaire se charge, la barre de navigation montre que le dataset contient bien neuf enregistrements, comme à l'origine. Autrement dit, il fonctionne comme prévu.

12. Cliquez sur le bouton Placer en dernier pour afficher le dernier enregistrement du dataset.

    L'enregistrement de Halvorson, Kim s'affiche à nouveau. Le nom du dernier enseignant a uniquement été supprimé de la mémoire et réapparaît puisque la base de données sous-jacente le contient toujours.

13. Cliquez sur le bouton Fermer sur le formulaire pour arrêter le programme.

Félicitations ! Sans écrire la moindre ligne de code, vous avez construit une application de base de données qui fonctionne et affiche des informations spécifiques issues d'une base de données. La configuration du dataset a exigé quelques étapes, mais le dataset est à présent prêt à être exploité de diverses manières dans le programme. J'ai seulement sélectionné une table et deux champs de la base de données Étudiants pour éviter d'encombrer l'écran et concentrer votre attention. Dans une application de production, vous sélectionnerez sans doute une plage d'objets plus importante dans vos bases de données, au moment de créer les datasets dans l'Assistant Configuration de source de données. Comme vous l'aurez noté, il n'est pas indispensable de créer des objets liés sur un formulaire pour chaque élément du dataset : vous pouvez choisir les enregistrements utilisés et affichés.

Utiliser des contrôles liés pour afficher des informations relatives à une base de données

Comme je l'ai mentionné précédemment, Visual Studio peut faire usage d'une grande variété de contrôles de la Boîte à outils Visual Studio pour afficher des informations relatives à une base de données. Vous pouvez lier des contrôles aux datasets en faisant glisser des champs à partir de la fenêtre Sources de données (la méthode la plus simple) et créer des contrôles indépendamment sur les formulaires puis les lier aux objets du dataset ultérieurement. Cette deuxième option constitue une fonctionnalité importante. En effet, il arrivera sans doute que vous deviez ajouter des sources de données à un projet après la création de l'interface utilisateur de base. La procédure que nous allons étudier dans cette section gère une telle situation, tout en vous permettant de vous entraîner à lier des objets de données aux contrôles d'une application Visual Basic. Vous allez créer un objet zone de texte avec masque sur le formulaire, configurer l'objet pour formater des informations relatives à la base de données, puis lier le champ *PhoneNumber* du dataset EtudiantsDataSet à l'objet.

Lier un contrôle zone de texte avec masque à un objet du dataset

1. Affichez le formulaire dans le Concepteur de formulaires et ouvrez la Boîte à outils si elle n'est pas visible.
2. Dans l'onglet Contrôles communs, cliquez sur le contrôle *MaskedTextBox* et créez un objet zone de texte avec masque sur le formulaire, sous l'étiquette Instructor et la zone de texte.

   Comme nous l'avons étudié au chapitre 6, « Utiliser les structures de décision », le contrôle *MaskedTextBox* est similaire au contrôle *TextBox*, excepté qu'il offre davantage de souplesse pour réglementer ou limiter les informations saisies par l'utilisateur dans un programme. On ajuste le format de saisie du contrôle *MaskedTextBox* en configurant la propriété *Mask*. Dans cet exercice, vous utiliserez la propriété *Mask* pour préparer l'objet zone de texte avec masque à afficher les numéros de téléphone formatés du champ *PhoneNumber* (par défaut, les numéros de télé-

phone de la base de données Etudiants sont stockés sans les espaces, parenthèses ou tirets des numéros de téléphone d'Amérique du Nord).

3. Cliquez sur la flèche de raccourci qui se trouve dans l'angle supérieur droit de l'objet zone de texte avec masque puis choisissez la commande Définir le masque.

   La boîte de dialogue Masque de saisie s'affiche. Elle présente un certain nombre de masques de mise en forme prédéfinis. Visual Studio exploite ces masques pour mettre en forme la saisie faite dans l'objet zone de texte avec masque, ainsi que celle reçue des utilisateurs.

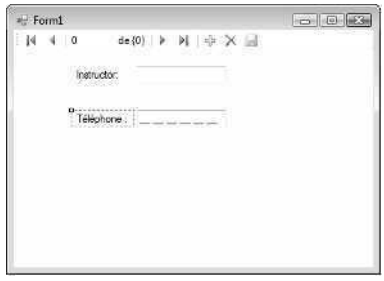
4. Cliquez sur le masque de saisie Numéro de téléphone (français) et cliquez sur OK.

   L'objet zone de texte avec masque contient à présent des lignes de formatage répondant aux paramètres français. Remarquez qu'en réalité, les masques proposés dépendent des paramètres de pays et de langue stockés dans Microsoft Windows, qui varient d'un pays à l'autre.

5. Ajoutez un objet étiquette devant le nouvel objet zone de texte avec masque et attribuez la valeur « Téléphone : » à sa propriété *Text* (sans oublier les deux points).

   La première étiquette descriptive a été automatiquement ajoutée par la fenêtre Sources de données, mais nous devons ajouter celle-ci manuellement.

6. Ajustez l'espacement entre les deux étiquettes et les zones de texte pour les aligner. Lorsque vous avez terminé, votre formulaire est similaire à :



   Nous allons maintenant lier le champ *PhoneNumber* de *EtudiantsDataSet* au nouvel objet zone de texte avec masque. Dans Visual Studio 2005 et 2008, ce processus est plus simple qu'il l'était dans Visual Basic 6 ou Visual Studio .NET 2003 : il suffit de faire glisser le champ *PhoneNumber* depuis la fenêtre Sources de données sur l'objet auquel vous voulez lier les données, dans notre cas l'objet *MaskedTextBox1*.

7. Affichez la fenêtre Sources de données si elle n'est pas visible et faites glisser le champ *PhoneNumber* sur l'objet *MaskedTextBox1*.

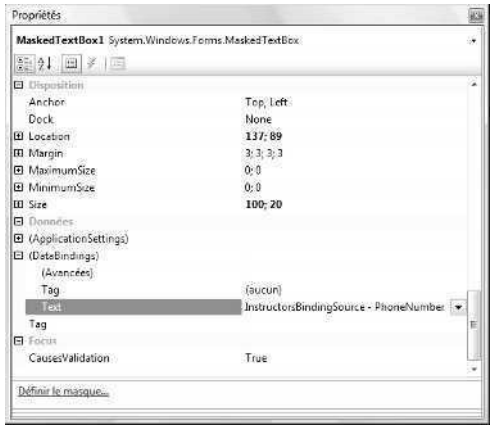
Lorsque vous faites glisser un objet du dataset sur un objet qui existe déjà sur le formulaire (ce que nous appellerons l'*objet cible*), vous ne créez pas de nouvel objet lié. À la place, les propriétés *DataBindings* de l'objet cible sont configurées pour correspondre à l'objet dataset glissé et déposé depuis la fenêtre Sources de données.

Après cette opération de glisser-déposer, l'objet zone de texte avec masque est lié au champ *PhoneNumber* et sa propriété *Text* contient une petite icône de base de données (un signe que l'objet est lié à un dataset).

8. Vérifiez que l'objet *MaskedTextBox1* est sélectionné sur le formulaire et appuyez sur F4 pour mettre en évidence la fenêtre Propriétés.

9. Localisez la catégorie *DataBindings* dans la fenêtre Propriétés et cliquez sur le signe plus (+) en regard pour l'ouvrir.

Visual Studio affiche les propriétés associées à l'accès aux données dans un objet zone de texte avec masque. Votre fenêtre Propriétés est similaire à :

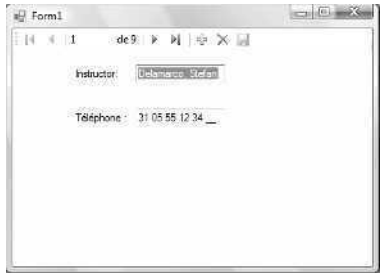


La propriété liée à noter ici est la propriété *Text*, qui a été fixée à « InstructorsBindingSource – PhoneNumber » suite à l'opération de glisser-déposer (remarquez que la petite icône de base de données n'apparaît pas ici, mais uniquement dans la propriété *Text* qui se trouve dans le bas de la liste alphabétique des propriétés). En outre, si vous cliquez maintenant sur la flèche en regard de la propriété *Text*, vous verrez une représentation de l'objet zone de texte avec masque. Cet affichage visuel pratique permet de modifier rapidement la source de données à laquelle le contrôle est lié, mais ne modifiez pas ce paramètre pour l'instant.

10. Cliquez sur le bouton Démarrer le débogage pour démarrer le programme.

Visual Studio exécute le programme dans l'environnement de développement. Après quelques instants, les deux champs de base de données sont chargés dans les objets zone de texte et zone de texte avec masque, comme le montre l'illustration suivante :

> **Remarque** Si vous voyez une boîte de message qui indique « Une valeur de propriété n'est pas valide », cliquez sur OK, cliquez sur Arrêter le débogage, puis cliquez à nouveau sur Démarrer le débogage.



Notez que l'objet zone de texte avec masque formate correctement le numéro de téléphone tel que prévu pour les numéros de téléphone français.

11. Cliquez plusieurs fois sur le bouton Déplacer vers le bas.

Cette action met en évidence une autre fonctionnalité importante : les deux champs du dataset défilent de concert et les noms des enseignants affichés correspondent aux numéros de téléphone enregistrés dans la base de données Étudiants. Cette synchronisation est gérée par l'objet *InstructorsBindingNavigator* qui suit l'enregistrement en cours de chaque objet lié du formulaire.

12. Cliquez sur le bouton Fermer pour arrêter le programme et sur le bouton Enregistrer tout pour enregistrer les changements.

Vous avez appris à afficher plusieurs champs de base de données sur un formulaire, à utiliser la barre de navigation pour parcourir un dataset et à formater des informations de base de données avec un masque. Avant de terminer ce chapitre et de passer au contrôle *DataGridView* que nous étudierons au chapitre 19, prenons un moment pour voir comment personnaliser davantage le dataset en faisant appel à des instructions SQL.

Aller plus loin : instructions SQL, LINQ et filtrage de données

Vous avez utilisé l'Assistant Configuration de source de données pour extraire des tables et des champs de la base de données Étudiants en créant un dataset personnalisé appelé *ÉtudiantsDataSet*. Outre ce filtrage, il est possible d'organiser et d'affiner les données affichées par les contrôles liés en faisant appel à des instructions SQL et le Générateur de requêtes Visual Studio. Cette section présente ces outils.

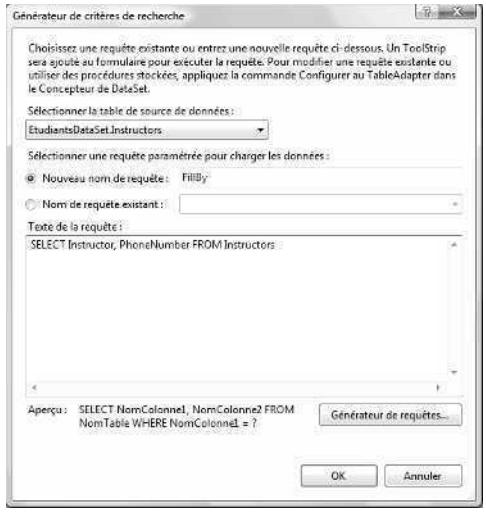
Pour les utilisateurs Visual Basic qui maîtrisent Microsoft Access ou SQL Server, le filtrage des données avec des instructions SQL ne présente rien de nouveau. Pour les autres, sachez que les *instructions SQL* sont des commandes qui extraient, ou *filtrent*, des informations issues d'une ou de plusieurs tables structurées d'une base de données. La raison de ce filtrage est simple : à l'instar des utilisateurs du web qui sont constamment confrontés à une masse incalculable de données sur l'Internet (et emploient des mots clés dans leurs navigateurs pour localiser les informations recherchées), les programmeurs de base de données sont régulièrement confrontés à des tables contenant des dizaines de milliers d'enregistrements qu'il faut épurer et organiser pour accomplir une tâche particulière. L'instruction SQL SELECT constitue l'un des mécanismes classiques de l'organisation des informations de base de données. En enchaînant un groupe de ces instructions, les programmeurs parviennent à créer de complexes directives de recherche, ou *requêtes*, qui extraient uniquement les données nécessaires d'une base de données.

Compte tenu de l'omniprésence des instructions SQL, les précédentes versions de l'environnement de développement Visual Studio et Visual Basic proposaient des mécanismes permettant de les employer. Visual Studio 2008 propose une nouvelle et intéressante technique nommée LINQ (*Language-Integrated Query*) qui permet aux programmeurs expérimentés d'écrire des requêtes de base de données de style SQL directement dans du code Visual Basic. Même si LINQ est nouveau et excitant pour beaucoup, ce n'est pas une technique facile à maîtriser avant de posséder un peu plus d'expérience avec les instructions SQL. Vous allez acquérir un peu de cette expérience dans l'exercice suivant en employant un puissant dispositif de Visual Studio 2008 nommé Générateur de requêtes. C'est un outil visuel qui aide les programmeurs à construire des requêtes de base de données, particulièrement utile à ceux possédant une expérience limitée de la programmation SQL. Dans le prochain exercice, vous allez utiliser le Générateur de requêtes pour organiser davantage votre dataset en le triant par ordre alphabétique.

Créer des instructions SQL avec le Générateur de requêtes

1. Sur le formulaire, cliquez sur la zone de texte *InstructorTextBox* (le premier objet lié que vous avez créé pour afficher les noms des enseignants de la base de données Etudiants).
2. Dans le menu Données, choisissez la commande Ajouter une requête.

   La commande Ajouter une requête est disponible lorsqu'on sélectionne un objet lié, comme *InstructorTextBox*, dans le Concepteur. La boîte de dialogue Générateur de critères de recherche s'affiche, comme le montre l'illustration suivante :



   Cette boîte de dialogue permet d'organiser et d'afficher les requêtes, créées par le Générateur de requêtes et composées d'instructions SQL. La table que la requête filtre et organise par défaut (*EtudiantsDataSet.Instructors*) est sélectionnée dans la zone de liste Sélectionner la table de source de données, dans la partie supérieure de la boîte de dialogue. Vous reconnaissez sans doute le format de la hiérarchie des objets employé par le nom de la table, qui se lit « la table Instructors dans le dataset EtudiantsDataSet ». En présence de plusieurs tables, cliquez sur la flèche de la liste déroulante pour les afficher ou les sélectionner.

3. Dans la zone Nouveau nom de requête, tapez **TriInstructors**.

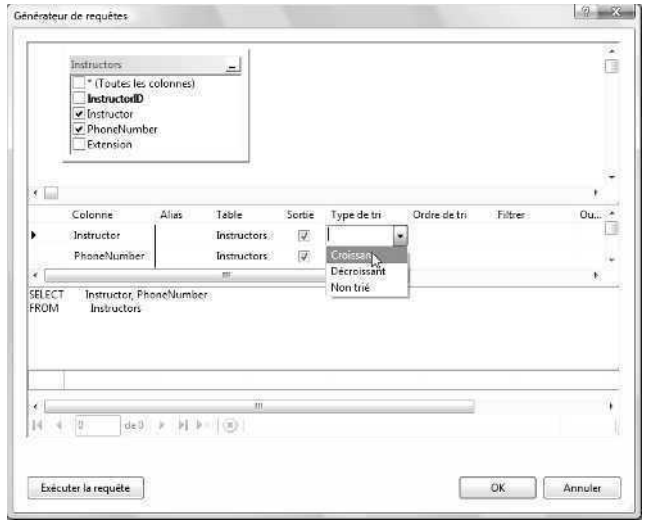
   Cette zone de texte assigne un nom à la requête et forme la base des boutons de la barre d'outils ajoutée au formulaire (ou simplifier l'accès, par défaut, les nouvelles requêtes sont affectées à des boutons de la barre d'outils au sein de l'application en cours de création).

4. Cliquez sur le bouton Générateur de requêtes pour ouvrir l'outil du même nom.

   Le Générateur de requêtes permet de créer des instructions SQL en les saisissant directement dans une grande zone de texte d'instructions SQL ou en cliquant sur les zones de liste et autres outils visuels.

5. Dans la ligne *Instructor*, qui représente le champ *Instructor* du dataset, cliquez sur la cellule qui se trouve en dessous de Type de tri et cliquez sur la flèche pour afficher la zone liste Type de tri.

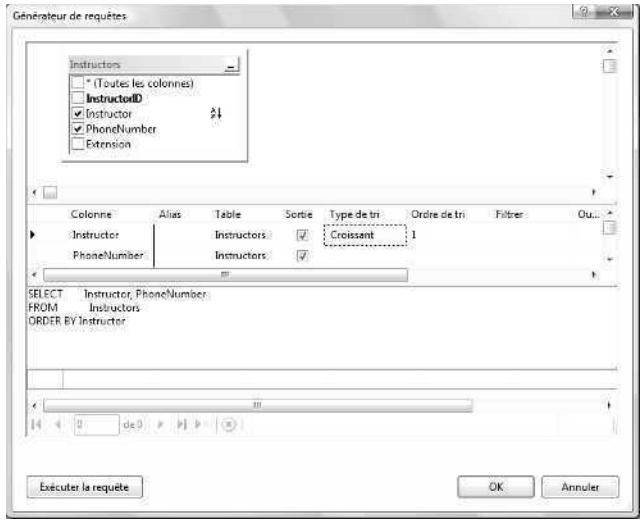
   Voici à quoi ressemble votre écran :



   Vous allez employer l'instruction SQL ORDER BY, qui trie les enregistrements de la base de données en fonction d'un champ clé et d'un numéro d'ordre de tri. Vous allez trier les enregistrements du champ *Instructor* par ordre croissant.

6. Cliquez sur Croissant dans la liste Type de tri.
7. Cliquez sur la zone de texte de l'instruction SQL qui se trouve sous la grille pour actualiser la fenêtre du Générateur de requêtes.

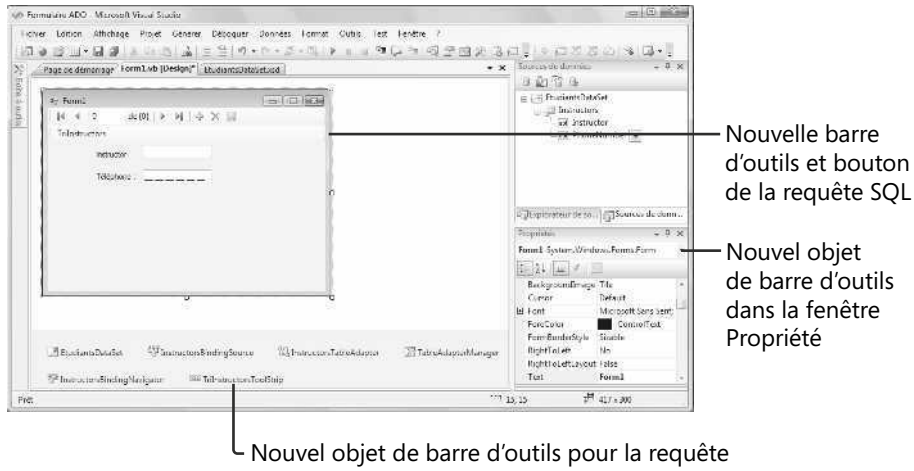
   Une nouvelle clause (ORDER BY Instructor) est ajoutée à la zone de l'instruction SQL et votre écran ressemble à :



8. Cliquez sur OK pour terminer la requête.

   Visual Studio ferme le Générateur de requêtes et affiche la nouvelle requête dans la boîte de dialogue Générateur de critères de recherche. Le nom de la requête (*TriInstructors*) est affiché, ainsi que les instructions SQL qui effectuent le tri.
9. Cliquez sur OK pour fermer la boîte de dialogue Générateur de critères de recherche et configurer l'objet *InstructorTextBox* pour qu'il affiche les noms par ordre alphabétique croissant.

   Cette instruction SQL particulière ne filtre pas les données, mais organise les enregistrements du dataset de manière plus pratique lorsque l'utilisateur clique sur le bouton TriInstructors de la nouvelle barre d'outils, qui se trouve dans la partie supérieure du formulaire. Le processus a également créé un objet *TriInstructorsToolStrip* dans la zone des composants, sous le formulaire. Voici à quoi ressemblent le Concepteur et la zone des composants :

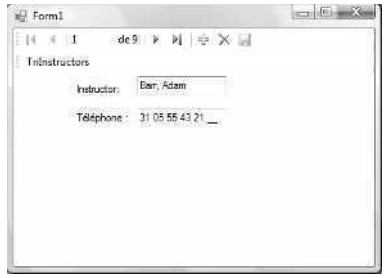


10. Cliquez sur Démarrer le débogage pour démarrer le programme.

Visual Studio charge le formulaire et affiche le premier enregistrement pour deux objets du dataset.

11. Dans la nouvelle barre d'outils, cliquez sur le bouton TriInstructors.

L'instruction SQL trie les enregistrements *Instructor* dans le dataset et affiche les enregistrements selon le nouvel ordre. Le premier enregistrement est à présent Barr, Adam comme le montre l'illustration suivante :



12. Parcourez la liste des enregistrements et constatez qu'elle est bien triée par ordre alphabétique (le dernier enregistrement doit être Wilson, Dan).

13. Cliquez sur le bouton Fermer pour arrêter le programme.

Vous avez fait vos premiers pas dans la création de requêtes avec des instructions SQL et le Générateur de requêtes. La programmation de bases de données est un sujet complexe, mais ce que vous avez déjà appris vous permettra de créer des applications centrées sur les données, autrement sur des collections de données personnalisées qui profitent à l'utilisateur et répondent à ses besoins informatiques. Nous allons poursuivre notre exploration de l'accès aux données dans le chapitre 19. Ensuite, au chapitre 20, « Créer des sites et des pages web avec Microsoft Visual Web Developer et ASP.NET », votre projet final présentera les enregistrements d'une base de données sur un site web.

Rappel du chapitre 18

Pour	Faites ceci
Établir une connexion avec une base de données	Dans le menu Données, choisissez la commande Ajouter une nouvelle source de données et servez-vous de l'Assistant Configuration de source de données pour parcourir la base de données à laquelle vous voulez donner accès en créant une chaîne de connexion.
Créer un dataset	Dans l'Assistant Configuration de source de données, spécifiez le nom du dataset dans la zone Nom du DataSet, développez le nœud Tableaux dans l'arborescence de la base de données présentée par l'assistant et indiquez les tables et champs à inclure dans le dataset (il n'est pas indispensable que le dataset contienne toutes les tables et tous les champs).
Créer des objets liés capables d'afficher les données issues d'un dataset sur un formulaire Windows.	Après l'exécution de l'Assistant Configuration de source de données, ouvrez la fenêtre Sources de données et faites glisser les tables et/ou les champs vers le formulaire Windows. Pour contrôler le type de contrôle lié créé par Visual Studio pour une table ou un champ, cliquez sur sa flèche et sélectionnez un contrôle dans la liste qui s'affiche avant de le faire glisser. Si vous avez placé un contrôle sur le formulaire avant d'ajouter les sources de données au projet, liez l'objet de base de données au contrôle en faisant glisser les objets de la base de données à partir de la fenêtre Sources de données sur le contrôle dans le formulaire. En alternative, positionnez les propriétés *DataBindings* de l'objet sur un champ (ou colonne) valide dans le dataset (la propriété *Text* est l'une des propriétés *DataBindings* les plus intéressantes).
Ajouter des contrôles de navigation à un formulaire Windows	Dans Visual Studio 2005 et 2008, une barre de navigation s'ajoute automatiquement aux formulaires Windows lorsqu'un objet de base de données valide est déposé depuis la fenêtre Sources de données sur le formulaire. Pour personnaliser les boutons de cette barre d'outils, cliquez droit sur l'objet *InstructorsBindingNavigator* dans la zone des composants et choisissez Modifier les éléments.
Formater les informations de la base de données sur un formulaire	Servez-vous du contrôle *MaskedTextBox* pour mettre en forme le contenu des données de chaîne dans le dataset. Le contrôle *MaskedTextBox* propose de nombreux masque de saisie et permet de créer des formats de chaîne personnalisés.
Filtrer ou trier les informations d'une base de données stockée dans un dataset	Servez-vous d'instructions SQL pour créer des requêtes personnalisées dans le Générateur de requêtes Visual Studio et ajoutez ces requêtes à une barre d'outils dans un formulaire Windows.

Chapitre 19

Présenter les données avec le contrôle *DataGridView*

À la fin de ce chapitre, vous saurez :

- Créer un objet grille de données sur un formulaire Windows et l'utiliser pour afficher une table de base de données
- Trier les tables de la base de données par colonne
- Changer le format et la couleur des cellules dans un objet grille de données
- Ajouter et supprimer des colonnes et des en-têtes de colonnes
- Afficher plusieurs tables de données et barres de navigation sur un formulaire et basculer de l'une à l'autre
- Autoriser les changements dans les cellules de la grille et écrire des mises à jour dans la base de données sous-jacente

Dans le chapitre 18, « Démarrer avec ADO.NET », vous avez appris à utiliser les techniques de programmation de bases de données Microsoft ADO.NET pour établir une connexion avec une base de données Microsoft Office Access et afficher les colonnes de la base de données dans un formulaire Windows. Vous avez également vu comment ajouter une barre de navigation à un formulaire et organiser les informations de la base de données avec des instructions SQL et l'outil Générateur de requêtes.

Dans ce chapitre, vous continuerez à exploiter les fonctionnalités de programmation de base de données Microsoft, ainsi que les classes, objets et outils de conception d'ADO.NET. Nous nous concentrerons plus particulièrement sur le contrôle *DataGridView* qui permet de présenter l'intégralité d'une table de base de données à l'utilisateur.

Utiliser le contrôle *DataGridView* pour afficher des enregistrements

Le contrôle *DataGridView* présente les informations en établissant une grille constituée de lignes et de colonnes sur un formulaire pour présenter des données sous une forme similaire à celle rencontrée dans des programmes tels que Microsoft Office Excel ou Access. Le contrôle *DataGridView* peut exposer n'importe quel type de données tabulaire : texte, nombres, dates ou contenu d'un tableau.

Le contrôle *DataGridView* présent dans Visual Basic 2005 et 2008 diffère toutefois du contrôle *DataGrid* de Microsoft Visual Basic .NET 2003 et encore plus sensiblement du contrôle *Datagrid* de Visual Basic 6. Une des améliorations importantes est que le contrôle *DataGridView* de Visual Basic 2008 ne nécessite aucune commande spécifique aux données : les objets sous-jacents dataset et adaptateur de données gèrent toutes les fonctionnalités d'accès aux données.

Dans ce chapitre, nous nous concentrerons sur la capacité du contrôle *DataGridView* à afficher les colonnes (champs) et les lignes (enregistrements) de la base de données Etudiants.mbd, le fichier relatif aux informations structurées sur les étudiants que nous avons employé au chapitre 18. Vous commencerez par remplir un objet grille de données simple avec des enregistrements textuels issus de la base de données, puis vous définirez quelques options de mise en forme. Vous poursuivrez par le tri des enregistrements dans des objets grille et la gestion de plusieurs grilles et barres de navigation sur un formulaire. Pour finir, vous apprendrez à ajuster les propriétés *DataGridView*, dont la propriété *ReadOnly* qui permet ou interdit à l'utilisateur d'enregistrer les changements dans la base de données d'origine.

Le contrôle *DataGridView* est connecté, ou lié, aux composants d'accès aux données sous-jacents par le biais de sa propriété *BindingSource*. Cette propriété contient d'intéressantes informations une fois que le programme a établi une connexion avec une source de données valide *via* l'Assistant Configuration de source de données et la fenêtre Sources de données (les étapes relatives à l'établissement de cette connexion seront rapidement survolées dans ce chapitre. Elles ont été décrites en détail au chapitre 18. Pour plus d'informations, lisez la section intitulée « Exploiter une base de données Access » dans ce chapitre). Une fois l'objet grille de données lié à une source de données valide, Visual Studio remplit, ou peuple, automatiquement la grille *via* la méthode *Fill* lorsque le formulaire est chargé en mémoire.

Établir une connexion avec une table de base de données

1. Démarrez Visual Studio et créez un nouveau projet Visual Basic Application Windows Forms nommé **Mon Exemple DataGridView**.

   Un nouveau projet s'affiche dans l'environnement de développement.

2. Dans le menu Données, choisissez la commande Ajouter une nouvelle source de données.

   L'Assistant Configuration de source de données s'ouvre dans l'environnement de développement. Vous avez utilisé cet outil au chapitre 18 pour lier la base de données Etudiants.mbd au projet et remplir la fenêtre Sources de données avec des tables et des colonnes issues de la base de données. Cette fois, vous allez sélectionner une plage d'informations de l'exemple de base de données Access.

3. Cliquez sur l'icône Base de données et cliquez sur Suivant.

   L'assistant vous demande de créer une chaîne de connexion, mais si vous avez réalisé les exercices du chapitre 18, la base de données Etudiants.mbd vous est automatiquement proposée, comme le montre la figure suivante :

   Si la connexion à la base de données Etudiants n'est pas affichée, cliquez sur le bouton Nouvelle connexion et localisez le fichier Etudiants.mbd dans le dossier c:\vb08\ch18 (la procédure détaillée permettant d'établir cette connexion est décrite dans le chapitre 18).

4. Sélectionnez la chaîne de connexion Etudiants.mbd et cliquez sur le bouton Suivant. L'assistant vous demande si vous souhaitez enregistrer la chaîne de connexion.

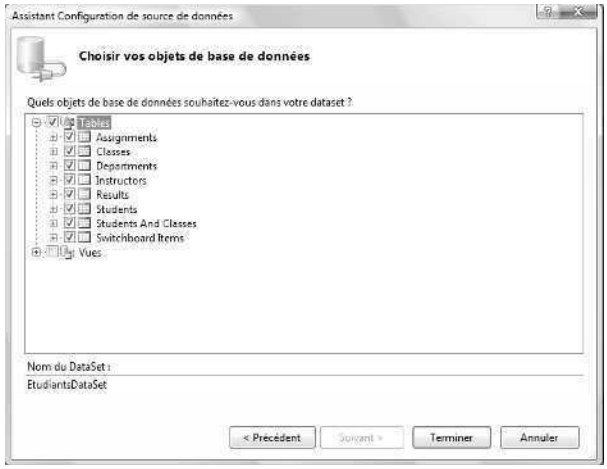
5. Cliquez sur Suivant pour enregistrer la chaîne à l'emplacement par défaut (le fichier de configuration de votre projet).

   Vous êtes maintenant invité à sélectionner les objets de base de données à utiliser pour ce projet. Rappelez-vous qu'à ce niveau, l'Assistant Configuration de source de données permet de choisir les tables et les colonnes : vous pouvez sélectionner tous les objets ou uniquement un sous-ensemble d'objets.

6. Développez le nœud Tables pour afficher les noms des sept tables de la base de données et une entrée supplémentaire appelée SwitchBoard Items.

7. Cochez la case située en regard du nœud Tables pour sélectionner les huit éléments.

L'objectif de ce chapitre étant d'afficher de grandes quantités de données avec le contrôle *DataGridView*, ce projet contiendra une plage plus étendue d'informations. Voici à quoi ressemble l'assistant :



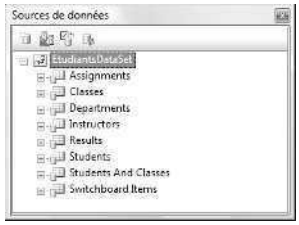
8. Cliquez sur Terminer pour fermer l'Assistant Configuration de source de données.

Visual Studio crée un dataset intitulé *EtudiantsDataSet* pour représenter les huit objets sélectionnés. Il ajoute également un fichier de schéma XML nommé EtudiantsDataSet.xsd au projet et à la fenêtre de l'Explorateur de solutions. Nous utiliserons la connexion que vous venez d'établir avec la base de données Etudiants.mbd tout au long de ce chapitre.

9. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer le projet. Désignez le dossier c:\vb08epe\chap19 comme emplacement.

10. Dans l'Explorateur de solutions, cliquez sur l'onglet Sources de données pour ouvrir la fenêtre Sources de données (si l'onglet Sources de données n'est pas visible, dans le menu Données, choisissez la commande Afficher les sources de données).

La fenêtre Sources de données présente les objets du dataset *EtudiantsDataSet*, comme le montre l'illustration suivante :



Dans le chapitre 18, vous avez fait glisser des champs individuellement de la fenêtre Sources de données vers le formulaire Windows pour lier les objets de données aux contrôles dans l'interface utilisateur. Dans le prochain exercice, vous suivrez une procédure similaire, mais cette fois, vous ferez glisser toute la table sur le formulaire puis la lierez à un contrôle *DataGridView* de façon à pouvoir afficher simultanément tous les champs de la table.

Créer un objet grille de données

1. Redimensionnez le formulaire de sorte qu'il recouvre une grande partie de l'écran.

   Avant la fin de ce chapitre, vous allez placer deux objets grille de données côte à côte dans le formulaire, chacun comportant plusieurs colonnes et environ dix lignes de données. Rappelez-vous que le formulaire peut être plus large que l'espace alloué dans l'environnement de développement. En outre, vous pouvez fermer les outils de programmation ou utiliser les barres de défilement pour voir les parties masquées du formulaire (conservez toutefois la fenêtre Sources de données ouverte pour la prochaine étape).

2. Dans la fenêtre Sources de données, cliquez sur la table *Instructors* puis sur la flèche à sa droite pour afficher la liste des contrôles qui peuvent être liés à la table *Instructors* sur le formulaire.

   Voici à quoi ressemble la fenêtre Sources de données :

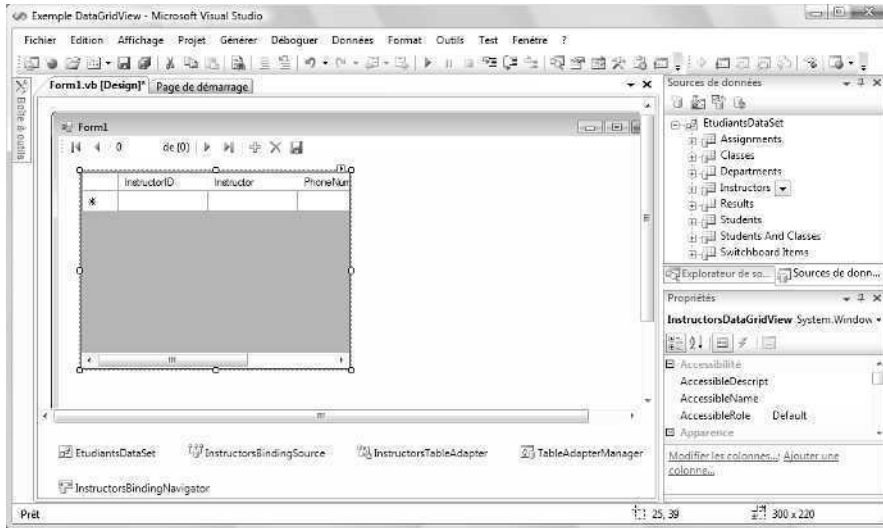
   Comme vous avez sélectionné la table dans son intégralité, cette liste ne contient pas de contrôles liés individuels. Voici les options disponibles :

   - *DataGridView*, la sélection par défaut, qui affiche une grille de colonnes et de lignes représentant les champs et les enregistrements de la table Instructors.
   - *Détails* qui configure Visual Basic pour qu'il crée automatiquement des contrôles individuels (avec leurs étiquettes associées) pour chaque champ que vous faites glisser sur le formulaire. Nous n'étudierons pas cette option maintenant. Sachez cependant qu'elle est intéressante pour présenter des données tabulaires dans un format légèrement plus accessible.

- *Aucun* qui supprime toute association entre la table et un élément ou un contrôle de l'interface utilisateur (si vous sélectionnez *Aucun* pour une table, vous ne pourrez pas faire glisser la table depuis la fenêtre Sources de données vers le formulaire et une icône Null s'affichera en regard du nom de la table).
- *Personnaliser* permet de sélectionner un contrôle différent pour afficher plusieurs champs de la base de données (comme le contrôle *ListBox*).

3. Cliquez sur l'option *DataGridView* et faites glisser la table *Instructors* sur le côté gauche du formulaire.

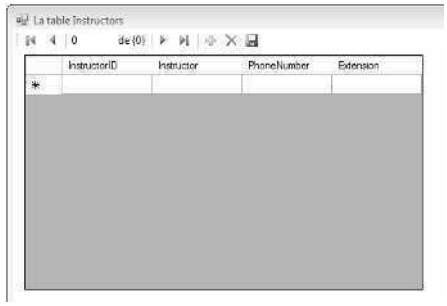
Visual Studio crée une barre de navigation dans la partie supérieure du formulaire, ajoute un dataset, une source de liaison, un adaptateur de table et les composants du navigateur de liaison dans la zone des composants et crée un objet grille de données intitulé *InstructorsDataGridView* sur le formulaire. Votre écran est similaire à :



Vous l'aurez noté, la grille ne contient aucune information pour l'instant et n'est sans doute pas à la bonne taille (mon objet grille de données n'est pas suffisamment large pour présenter les quatre colonnes, par exemple). Vous pouvez toutefois remarquer que Visual Studio a organisé la table *Instructors* sous forme d'une grille dont les colonnes représentent les champs et les lignes correspondent aux enregistrements. Une ligne vide est réservée au premier enregistrement de la table et d'autres lignes seront ajoutées dès que nous exécuterons le programme et que la grille se remplira de données.

4. Déplacez et redimensionnez l'objet grille de données de sorte que ses colonnes soient clairement visibles et que l'espace soit suffisant pour au moins dix lignes de données.
5. Servez-vous de la fenêtre Propriétés pour positionner la propriété *Text* du formulaire sur « La table Instructors ».

   Voici à quoi ressemble votre formulaire :



Vous avez terminé les étapes permettant de créer un objet grille de données sur un formulaire et le dimensionner. Vous allez maintenant prévisualiser les données et personnaliser la table. La prévisualisation des données et le réglage des paramètres de base sont simplifiés par la nouvelle flèche de raccourci Visual Studio.

Prévisualiser les données liées à un objet grille de données

1. Sélectionnez l'objet grille de données sur le formulaire et cliquez sur la flèche raccourci qui se trouve dans l'angle supérieur droit de l'objet.

   Visual Studio affiche les tâches *DataGridView*, une liste de paramètres de propriétés et de commandes classiques relatifs à l'objet grille de données. Voici à quoi ressemble la liste des tâches *DataGridView* :

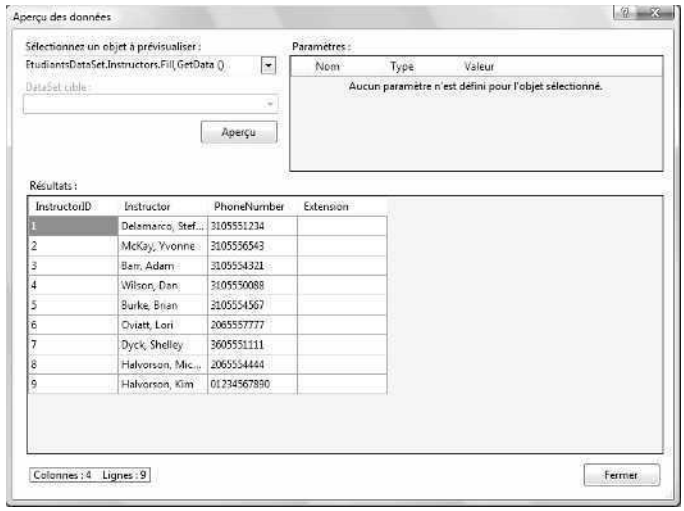
Vous pouvez exploiter les paramètres et les commandes de cette liste pour modifier la table liée à l'objet grille de données et activer ou désactiver l'édition au sein de la grille (le paramètre par défaut accorde des possibilités limitées à l'utilisateur pour modifier les informations de la table, tout en vous permettant de contrôler s'il peut ou non écrire les changements dans la base de données sous-jacente). Il est également possible d'ajuster les colonnes affichées, d'ancrer (attacher) la grille à son conteneur parent (le formulaire, dans ce cas), de filtrer les enregistrements avec une requête (instruction SQL) et de prévisualiser les données dans la table.

2. Cliquez sur la commande Aperçu des données pour ouvrir la boîte de dialogue du même nom.

Cette boîte de dialogue permet d'examiner les données de la table avec d'exécuter le programme : une fonctionnalité intéressante.

3. Cliquez sur le bouton Aperçu.

Visual Studio charge la table *Instructors* à partir du dataset *EtudiantsDataSet*, comme le montre l'illustration suivante :



Vous reconnaissez sans quelques-unes des données utilisées au chapitre 18, mais voyez à présent les quatre colonnes de la table. Cette prévisualisation est intéressante, bien que déconcertante : la colonne *Extension* ne contient pas de donnée, un point qui pourrait gêner ou embrouiller les utilisateurs (cette colonne est destinée à recevoir les extensions des numéros de téléphone, mais aucune donnée n'a été saisie dans cette colonne dans la base de données). Visual Studio simplifie la détection d'un tel défaut et adapte la sortie de la grille de sorte que la colonne inutilisée ne s'affiche pas.

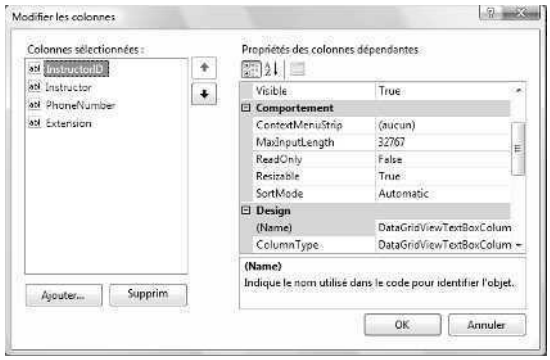
4. Cliquez sur le bouton Fermer pour fermer la boîte de dialogue Aperçu des données.

Vous allez maintenant supprimer la colonne vide de la grille.

Supprimer une colonne d'un objet grille de données

1. Affichez la liste Tâches *DataGridView* et choisissez la commande Modifier les colonnes.

   La boîte de dialogue suivante s'affiche :



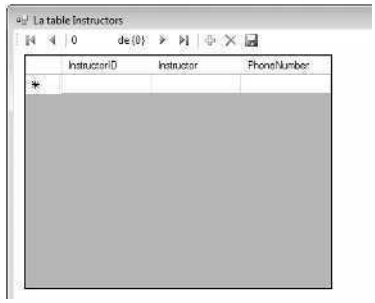
   Cette boîte de dialogue permet d'ajouter et de supprimer des colonnes dans l'objet grille de données (comme nous le verrons plus loin dans ce chapitre, elle sert également à changer les propriétés de l'objet *InstructorsDataGridView*). Pour l'instant, nous voulons supprimer la colonne Extension.

> **Remarque** La suppression de la colonne Extension de la grille de données ne la supprime pas de la base de données Etudiants.mbd sous-jacente.

2. Dans la zone de liste Colonnes sélectionnées, cliquez sur la colonne Extension.
3. Cliquez sur le bouton Supprimer. Visual Studio supprime la colonne de la liste.
4. Cliquez sur OK pour confirmer le changement.

   L'objet *InstructorsDataGridView* s'affiche, sans la colonne Extension. Vous disposez à présent de plus d'espace sur le formulaire pour afficher les informations issues de la base de données.
5. Réduisez la taille de l'objet *InstructorsDataGridView*.

Voici à quoi ressemble votre formulaire :



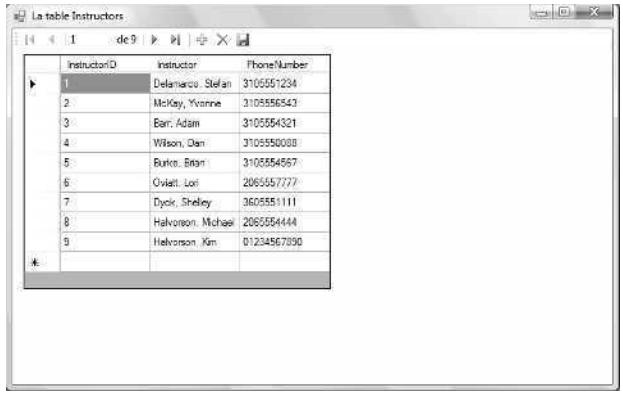
6. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

Vous avez prévisualisé et personnalisé la table avec les outils de base de données. Exécutons maintenant le programme pour examiner la grille pendant l'exécution. Vous allez également apprendre à trier les enregistrements dans un objet grille de données.

Gérer un objet grille de données pendant l'exécution

1. Cliquez sur le bouton Démarrer le débogage.

   Visual Studio exécute le programme dans l'environnement de développement. La table *Instructors* de la base de données s'affiche au sein de l'objet grille de données, telle que vous l'avez configurée. Votre formulaire présente un résultat similaire à



   L'instruction de la procédure événementielle *Form1_Load* qui peuple la grille d'informations issues de la table *Instructors* est la suivante :

```
Me.InstructorsTableAdapter.Fill(Me.EtudiantsDataSet.Instructors)
```

   Cette ligne a été ajoutée au programme par Visual Studio lorsque vous avez fait glisser la table sur le formulaire à partir de la fenêtre Sources de données.

   Chaque ligne de la grille représente un enregistrement de données issu de la table *Instructors* dans la base de données. Les barres de défilement sont prévues : vous

pouvez parcourir tous les enregistrements ou colonne qui ne sont applications immédiatement visibles. Cette fonctionnalité pratique accompagne automatiquement le contrôle *DataGridView*.

2. Faites défiler la liste des enregistrements pour voir les neuf lignes qui représentent les données relatives aux enseignants d'un collège ou d'une université.

3. Réduisez la taille de la colonne InstructorID en plaçant le pointeur entre les en-têtes des colonnes InstructorID et Instructor et en faisant glisser le bord de la colonne vers la gauche.

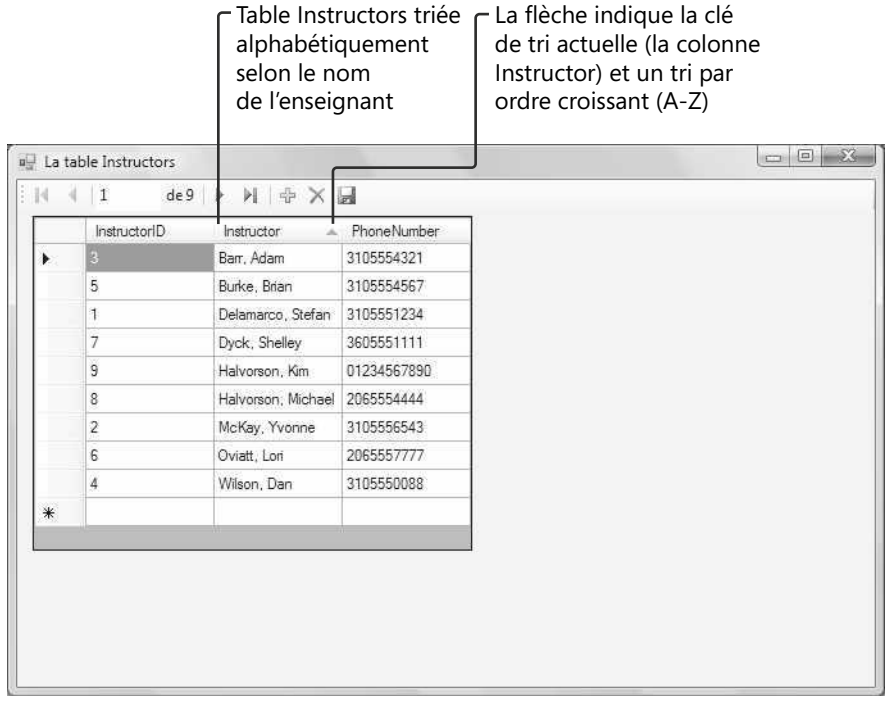
   Lorsque l'on place le pointeur entre des en-têtes de colonnes, il prend la forme d'une poignée de dimensionnement. Il est possible de modifier la taille des colonnes pendant l'exécution puisque la propriété *AllowUserToResizeColumns* de l'objet grille de données prend par défaut la valeur *True*. Pour empêcher le changement de taille, positionnez cette propriété sur *False*.

4. Redonnez à la colonne InstructorID sa largeur d'origine.

   Lorsqu'un objet grille de données contient des données, vous pouvez également profiter de la fonction de tri du contrôle *DataGridView*.

5. Cliquez sur l'en-tête de la colonne Instructor.

   La grille est triée par ordre alphabétique en fonction du nom de l'instructeur (Barr, Adam se trouve en tête de liste). Votre formulaire présente un résultat similaire à



Lorsque du tri des enregistrements de la base de données, vous devez faire appel à une colonne de tri, ou *clé*. Pour définir cette clé, cliquez sur l'en-tête de colonne qui doit servir de base au tri. Le contrôle *DataGridView* dispose d'une identification visuelle de la clé de tri en cours : une petite flèche située à droite de l'en-tête de la colonne. Si l'ordre de tri est alphabétique croissant (A–Z), la flèche pointe vers le haut. Pour inverser l'ordre de tri et obtenir un ordre alphabétique décroissant (Z–A), cliquez sur l'en-tête de colonne. La flèche fait office de bascule : elle permet de commuter entre les sens de tri.

6. Cliquez plusieurs fois sur l'en-tête de la colonne Instructor pour constater l'alternance de l'ordre de tri.
7. Cliquez sur les en-têtes des autres colonnes, InstructorID et PhoneNumber, pour réaliser le tri selon ces clés.
8. Lorsque vous avez terminé de tester les fonctionnalités de défilement, de dimensionnement et de tri du contrôle *DataGridView*, cliquez sur le bouton Fermer du formulaire pour arrêter le programme.

   Le programme se ferme et vous revenez à l'environnement de développement.

Formater les cellules du contrôle *DataGridView*

Pour personnaliser l'apparence du dataset sur un formulaire, il est possible de personnaliser quelques-unes des caractéristiques du contrôle *DataGridView* en modifiant ces propriétés pendant la conception. Vous pouvez, par exemple, changer la largeur par défaut des cellules de la grille, ajouter ou supprimer les en-têtes de colonnes, modifier les couleurs d'arrière-plan de la grille ou des en-têtes et remplacer la couleur des lignes de la grille. Le prochain exercice présente quelques-uns de ces paramètres.

Définir les propriétés de l'objet grille de données pendant la conception

1. Affichez le formulaire, cliquez sur l'objet grille de données (s'il n'est pas déjà sélectionné) et activez la fenêtre Propriétés.
2. Cliquez sur la propriété *Columns* puis sur l'ellipse pour ouvrir la collection *Columns* dans la boîte de dialogue Modifier les colonnes.

   Vous avez utilisé cette boîte de dialogue précédemment pour supprimer la colonne *Extension* de la table *Instructors* (elle sert à définir les paramètres des propriétés de chaque colonne). Cette boîte de dialogue va maintenant servir à changer la largeur par défaut de la colonne *InstructorID*.

Remarque Cette colonne étant actuellement sélectionnée (elle est en surbrillance dans la zone de liste Colonnes sélectionnées), inutile de cliquer dessus à nouveau. Pensez cependant à toujours sélectionner la colonne de votre choix avant d'en modifier les propriétés.

3. Positionnez la propriété *Width* sur 70.

   Une largeur de 70 (mesurée en pixels) laisse suffisamment d'espace pour les valeurs entières de la colonne *InstructorID*.

4. Cliquez sur OK pour fermer la boîte de dialogue Modifier les colonnes.

   Définissons à présent les propriétés qui contrôlent l'apparence de toutes les colonnes de la table.

Remarque La boîte de dialogue Modifier les colonnes permet de configurer individuellement les colonnes. Pour modifier des propriétés qui s'appliquent à toutes les colonnes de la table, ajustez les paramètres des propriétés de l'objet grille de données dans la fenêtre Propriétés.

5. Dans la fenêtre Propriétés, attribuez la valeur *False* à la propriété *ColumnHeadersVisible*.

   Bien que les noms des colonnes possèdent leur utilité dans une base de données, ils n'identifient pas toujours clairement leur contenu ou sont constitués d'abréviations ou de mots qu'il est préférable de masquer. En définissant cette propriété, vous avez supprimé les noms des colonnes de la table.

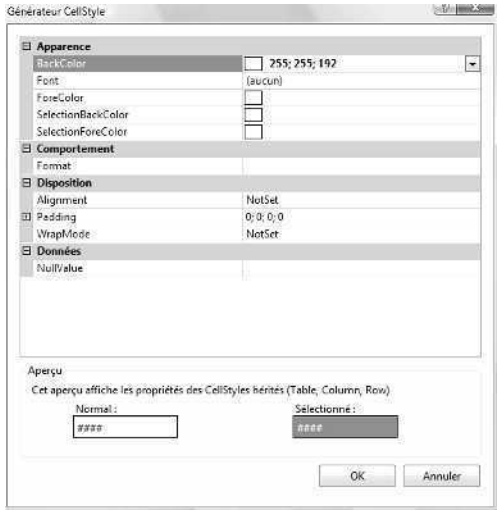
6. Cliquez sur la propriété *AlternatingRowsDefaultCellStyle* puis sur l'ellipse.

   Cette propriété contrôle la couleur qui s'affiche à l'arrière-plan des cellules de la grille d'une ligne sur deux. En modifiant ce paramètre, vous obtenez une alternance des couleurs (blanc et la couleur sélectionnée) d'une ligne à l'autre de la grille. Cet effet simplifie la lecture des enregistrements dans les tables de grande taille.

   Visual Studio affiche la boîte de dialogue Générateur CellStyle, un outil qui permet de définir les propriétés des cellules de colonnes dans les tables de l'objet grille de données.

7. Cliquez sur la propriété *BackColor*, puis sur sa flèche, sur l'onglet Personnaliser et enfin sur la couleur jaune pâle.

Voici à quoi ressemble la boîte de dialogue :



8. Cliquez sur OK pour fermer la boîte de dialogue.

Lorsque vous exécutez le programme, les couleurs des lignes de la grille alternent entre le blanc et le jaune pâle.

> **Remarque** La couleur qui entoure des bords des cellules est contrôlée par la propriété *BackgroundColor*. Pour modifier la couleur de toutes les cellules de la grille, modifiez la propriété *DefaultCellStyle*. Pour modifier la couleur de l'arrière-plan utilisée dans les cellules des en-têtes (s'ils sont affichés), servez-vous de la propriété *ColumnHeadersDefaultCellStyle*.

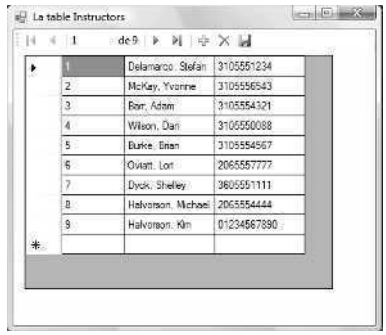
9. Cliquez sur la propriété *GridColor*, puis sur sa flèche, sur l'onglet Personnaliser et enfin sur la couleur bleue la plus foncée.

Ce paramètre contrôle la couleur des lignes de la grille. Si vous modifiez la couleur d'arrière-plan des cellules, il peut être intéressant de changer également celle des lignes de la grille.

Exécutons à présent le programme pour observer l'effet de ces changements de mise en forme.

10. Cliquez sur le bouton Démarrer le débogage.

Après quelques secondes, la grille s'affiche avec les informations de la table *Instructors*. Voici à quoi ressemble votre écran :



Notez l'absence des en-têtes de colonnes et la largeur réduite de la première colonne. Remarquez également l'alternance des lignes blanches et jaunes, ainsi que les lignes de la grille bleu foncé (difficiles à distinguer sur ce livre, mais visibles à l'écran).

11. Cliquez sur le bouton Fermer sur le formulaire pour arrêter le programme.

N'hésitez pas à parcourir la fenêtre Propriétés à la recherche d'autres paramètres à personnaliser. Si vous examinez la liste des options de mise en forme, vous constaterez qu'il existe plusieurs possibilités. Rappelez-vous que ces paramètres affectent toutes les colonnes de la table et pas uniquement les colonnes individuelles.

Ajouter une deuxième grille et un deuxième contrôle de navigation

Pour proposer à vos utilisateurs une interface plus riche en données, contenant plusieurs tables extraites de bases de données, il peut être intéressant d'ajouter un deuxième objet grille de données au formulaire. Une fois que vous avez établi un dataset dans la fenêtre Sources de données, il est relativement simple d'ajouter un autre contrôle *DataGridView* lié à une deuxième table du dataset. La seule partie plus complexe concerne l'ajout d'une deuxième barre de navigation et le paramétrage de la propriété *BindingSource* à l'objet source lié sous-jacent. Voyons comment procéder avec le prochain exercice.

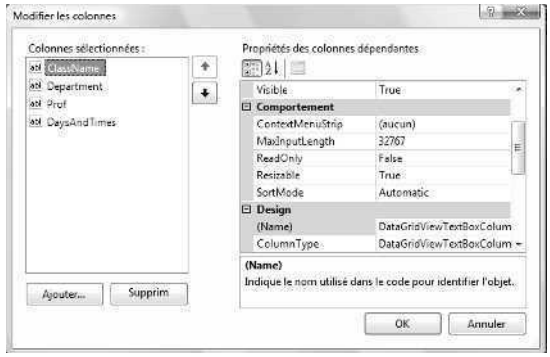
Lier un deuxième contrôle *DataGridView* à la table Classes

1. Ouvrez la fenêtre Sources de données si elle n'est pas affichée.
2. Faites glisser la table *Classes* depuis la fenêtre Sources de données sur la droite du formulaire.

   Visual Studio crée un deuxième objet grille de données intitulé *ClassesDataGridView* sur le formulaire.
3. Cliquez droit sur l'objet *ClassesDataGridView* puis sur la commande Modifier les colonnes.

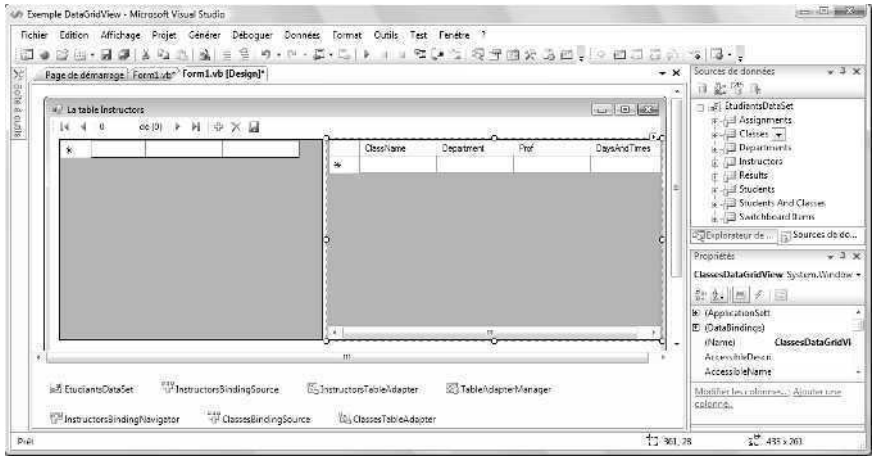
   La boîte de dialogue Modifier les colonnes s'ouvre.
4. Sélectionnez et supprimez les colonnes *ClassID*, *SectionNumber*, *Term*, *Units*, *Year*, *Location* et *Notes*.

   Le formulaire n'est pas suffisamment large pour accueillir toutes les colonnes de la table, c'est pourquoi nous écourtons la liste. Lorsque vous avez terminé, il reste les colonnes *ClassName*, *Department*, *Prof* et *DaysAndTimes*, comme le montre l'illustration suivante :



5. Cliquez sur OK pour fermer la boîte de dialogue Modifier les colonnes.
6. Déplacez et redimensionnez l'objet *ClassesDataGridView* sur le formulaire de sorte que les quatre colonnes s'affichent correctement. Il sera peut-être nécessaire de déplacer et de redimensionner le formulaire, voire l'objet *InstructorsDataGridView*, pour afficher les quatre colonnes.

Voici comment se présente à présent le formulaire :



Notez que Visual Studio a ajouté deux nouveaux objets dans la zone des composants : un objet *ClassesBindingSource* et un objet *ClassesTableAdapter*. Ces deux éléments sont des composants intermédiaires qui déplacent les données entre l'objet *ClassesDataGridView*, le dataset *EtudiantsDataSet* et la table *Classes* dans la base de données Etudiants sous-jacente.

Vous allez ajouter une deuxième barre de navigation au formulaire pour offrir les services de navigation et les propriétés relatives à la table *Classes* et à la deuxième grille. La deuxième barre est indispensable. En effet, la première (*InstructorsBindingNavigator*) est uniquement liée à l'objet *InstructorsDataGridView*.

Lier un contrôle *BindingNavigator* à l'objet *ClassesDataGridView*

1. Sur l'onglet Données de la Boîte à outils, double-cliquez sur le contrôle *BindingNavigator*.

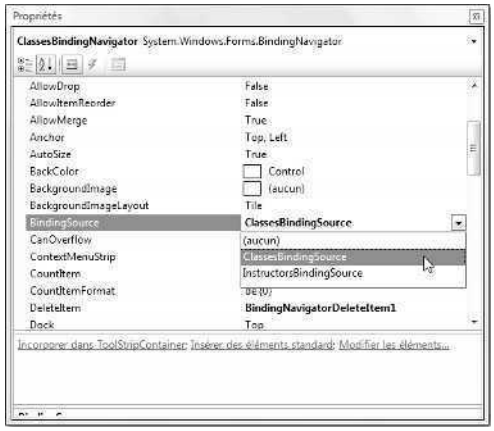
   Visual Studio ajoute un objet navigateur de liaison intitulé *BindingNavigator1* dans la zone des composants et une deuxième barre de navigation dans la partie supérieure du formulaire. Il pourrait être nécessaire de déplacer légèrement vers le bas vos objets grille si la nouvelle barre de navigation les recouvre.

2. Dans la fenêtre Propriétés, remplacez la propriété *Name* de l'objet *BindingNavigator1* par **ClassesBindingNavigator**.

   L'utilisation de ce nom clarifie la source de liaison et l'adaptateur de table auxquels la nouvelle barre de navigation est liée.

3. Remplacez la propriété *BindingSource* de l'objet *ClassesBindingNavigator* par *ClassesBindingSource*.

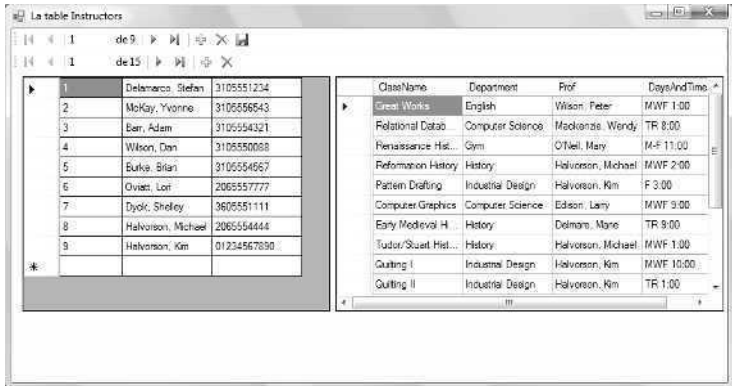
Si vous cliquez sur la flèche de la propriété *BindingSource*, la fenêtre Propriétés présente les noms des deux sources de liaison valides dans le programme, comme le montre l'illustration suivante :



Maintenant que nous avons établi une liaison entre la deuxième barre de navigation et l'objet source de liaison qui représente la table *Classes*, nous pouvons exécuter le programme.

4. Cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.
5. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Visual Studio exécute le programme Exemple DataGridView dans l'environnement de développement. Deux grilles et deux barres de navigation s'affichent, comme dans l'illustration suivante :



6. Cliquez sur le bouton Placer en dernier de la barre de navigation qui contient 15 enregistrements.

   Le dernier nom de classe de la table *Classes*, Deviant Behavior (Psychology), est sélectionné. La nouvelle barre de navigation fonctionne et vous disposez d'un accès intégral aux colonnes sélectionnées dans la table *Classes*.

7. Servez-vous des deux barres de navigation pour mettre en évidence différents enregistrements des deux tables.

   Les deux objets grille de données fonctionnent indépendamment, vous donnant accès aux enregistrements exacts que vous souhaitez afficher. Vous pouvez apprécier l'utilité de cet accès pour les utilisateurs qui doivent comparer des tables plus longues, contenant des informations étroitement liées. Si on filtre les données avec des instructions SQL, l'application devient rapidement puissante.

8. Lorsque vous avez terminé vos essais avec les deux tables et barres de navigation, cliquez sur le bouton Fermer pour fermer l'application Exemple DataGridView.

Aller plus loin : Actualiser la base de données d'origine

Comme je l'ai mentionné précédemment, l'objet dataset de votre programme n'est autre qu'une représentation des données de la base de données d'origine. Ceci est également vrai pour les informations stockées dans les grilles du formulaire : si l'utilisateur apporte des changements aux données, les modifications ne sont pas écrites dans la base de données, sauf si vous avez positionné la propriété *ReadOnly* de l'objet grille de données sur *False* et que l'utilisateur clique sur le bouton Enregistrer dans la barre de navigation. Les concepteurs d'ADO.NET et de Visual Studio ont créé cette relation pour protéger la base de données d'origine et pour permettre aux utilisateurs de manipuler librement les données dans les programmes, que vous envisagiez ou non d'enregistrer les changements.

Dans le prochain exercice, vous allez examiner la propriété *ReadOnly* du premier objet grille de données qui active ou désactive les changements dans l'objet *InstructorsDataGridView*. Vous verrez également comment utiliser le bouton Enregistrer les données, qui écrit les changements dans les tables de la base de données d'origine sur le disque.

Autoriser les mises à jour de la base de données

1. Sur le formulaire, cliquez sur le premier objet grille de données (*InstructorsDataGridView*) et activez la fenêtre Propriétés.
2. Localisez la propriété *ReadOnly* et étudiez ses paramètres.

   Si la valeur de la propriété *ReadOnly* est *False*, l'utilisateur est libre de modifier les informations dans ales cellules de la grille. Pour autoriser les utilisateurs à modifier les informations et à les écrire dans la base de données à laquelle le programme est connecté, conservez ce paramètre par défaut. Pour empêcher l'édition, positionnez la propriété *ReadOnly* sur *True*.

Nous conserverons le paramètre par défaut, *False*, pour tester la mise à jour de la base de données Etudiants.mbd sous-jacente.

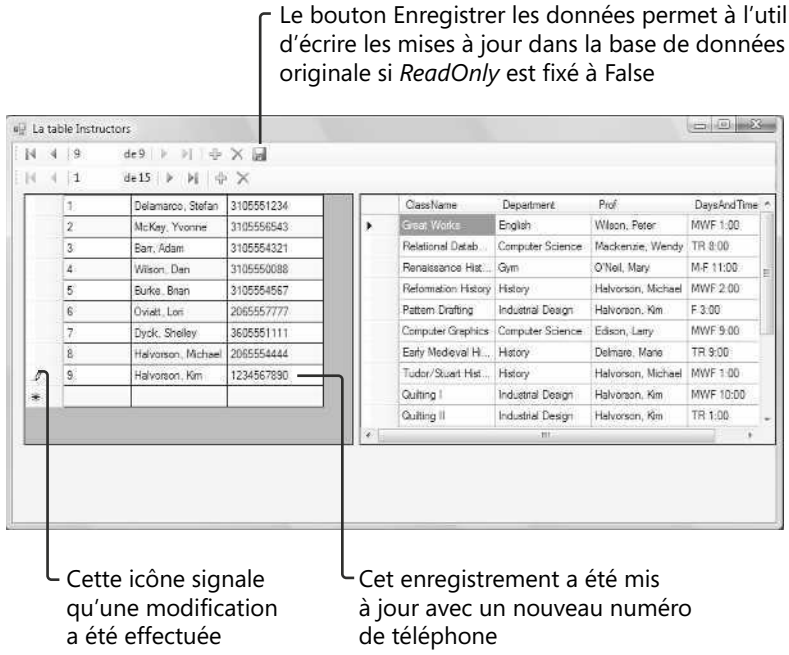
> **Astuce** Le programme Exemple DatagridView complet est disponible dans le dossier c:\vb08epe\chap19\Exemple DatagridView.

3. Cliquez sur le bouton Démarrer le débogage pour tester la propriété *ReadOnly* de la première grille.

   Les deux grilles contiennent les données des tables *Instructors* et *Classes* issues de la base de données Etudiants.mbd.

4. Dans la première grille, cliquez sur la cellule qui contient le numéro de téléphone de Kim Halvorson, tapez **1234567890** et appuyez sur ENTRÉE.

   Lorsque vous modifiez le contenu de la cellule, une icône en forme de crayon apparaît dans l'en-tête de la ligne, à gauche, pour indiquer qu'un changement est en cours. Votre écran est similaire à



Lorsque vous appuyez sur ENTRÉE ou cliquez sur une autre cellule de la grille, le changement est stocké dans le dataset *EtudiantsDataSet*.

5. Dans la barre de navigation, cliquez sur le bouton Enregistrer les données.

Visual Studio se sert de la méthode *UpdateAll* de l'objet adaptateur de table de la grille pour écrire le dataset modifié dans la base de données sous-jacente. Voici l'instruction employée pour réaliser cette opération d'enregistrement dans la procédure événementielle *BindingNavigatorSaveItem_Click* :

```
Me.TableAdapterManager.UpdateAll(Me.EtudiantsDataSet)
```

TableAdapterManager est un nouveau composant de Visual Studio 2008 qui facilite le contrôle et la manipulation de plusieurs tables de base de données dans un programme. La méthode *UpdateAll* enregistre toutes les tables ouvertes d'un programme, ce qui signifie qu'elle enregistre les modifications apportées tant à la table *Instructors* qu'à la table *Classes*. Vous n'avez pas à accepter ici le comportement par défaut. Si vous préférez n'enregistrer que les modifications apportées à la table *Instructors* lorsque l'utilisateur clique sur le bouton Enregistrer les données, remplacez l'instruction ci-dessus par l'instruction suivante, qui était le comportement par défaut avec Visual Basic 2005 :

```
Me.InstructorsTableAdapter.Update(Me. EtudiantsDataSet.Instructors)
```

Si vous employez la méthode *Update* pour un objet adaptateur de table nommé, seules les données associées à cette table seront enregistrées. Rappelez-vous aussi que vous pouvez contrôler les modifications de l'utilisateur au sein des tables à l'aide de la propriété *ReadOnly*.

Après l'opération d'enregistrement, la table *Instructors* est définitivement actualisée.

6. Cliquez sur le bouton Fermer pour arrêter le programme.

Le programme se ferme et vous revenez à l'environnement de développement. Exécutons à nouveau le programme pour vérifier que la table *Instructors* de la base de données Etudiants.mbd a bien été modifiée Lorsque vous redémarrez le programme, il charge les données à partir du fichier de la base de données.

7. Cliquez sur le bouton Démarrer le débogage.

Après quelques instants, les objets grille de données sont chargés avec leurs données. Comme vous le constatez, la ligne de la table *Instructors* qui contient le nom Kim Halvorson a été mise à jour et présente le numéro de téléphone modifié. Le programme fonctionne !

8. Cliquez sur le bouton Fermer pour arrêter le programme.

Pour poursuivre vos expériences avec la propriété *ReadOnly* sur l'une ou l'autre grille, attribuez-lui maintenant la valeur True et voyez ce qui se produit lorsque vous tentez de modifier la base de données : vous ne pourrez pas enregistrer les changements. Testez également l'ajout de nouvelles lignes de données dans la base de données en vous servant des fonctionnalités d'édition intégrées et des boutons de la barre d'outils associés aux contrôles *DataGridView* et *BindingNavigator* (avant d'ajouter de nouvelles lignes, refixez la propriété *ReadOnly* à False).

Faisons l'inventaire de vos réalisations. Vous avez appris à afficher plusieurs tables et enregistrements *via* les contrôles *DataGridView* et *BindingNavigator*, à employer les paramètres des propriétés pour personnaliser la grille et à écrire des mises à jour de tables de la grille vers la base de données d'origine. Comme vous commencez à le pressentir, la programmation des bases de données avec ADO.NET et Visual Studio est directe, quoiqu'un peu complexe. Il existe de nombreux outils, composants et techniques de programmation relatifs à l'affichage, la manipulation et la mise à jour des enregistrements d'une base de données et nous n'avons pas encore sérieusement abordé les problèmes importants comme la sécurité et les conséquences de l'exploitation de bases de données importantes, simultanément utilisées par de nombreux utilisateurs. Même si vous avez obtenu d'importants résultats avec très peu voire pas de code, il vous reste beaucoup à apprendre avant d'exploiter intensivement les bases de données dans les applications Visual Basic. L'annexe A, « Où trouver d'autres informations », présente une liste des livres que je vous conseille d'étudier.

Accès aux données dans un environnement de formulaires web

Les techniques d'accès aux données examinées dans le chapitre 18 et celui-ci sont conçus pour être exploités dans le Concepteur de formulaires Windows : l'environnement Visual Studio que vous avez utilisé pour créer la plupart des programmes de ce livre. Il est cependant possible d'employer les techniques de programmation ADO.NET dans un environnement de formulaires web, ce qui permet de partager des ressources *via* l'Internet et des applications centrées sur les données accessibles par l'intermédiaire d'un Navigateur Web comme Internet Explorer. Nous évoquerons cela vers la fin du prochain chapitre. Nous verrons également comment utiliser quelques nouveaux outils dont le contrôle *GridView*, une version du contrôle *DataGridView* conçu pour afficher des tables de bases de données sur les sites web.

Rappel du chapitre 19

Pour	Faites ceci
Établir une connexion avec des tables de base de données dans un projet	Servez-vous de l'Assistant Configuration de source de données pour lier le projet à une base de données, créez un dataset et remplissez la fenêtre Sources de données par une représentation des tables sélectionnées.
Créer un objet grille de données sur un formulaire pour afficher l'intégralité d'une table de base de données	Faites glisser l'icône de la table de la fenêtre Sources de données vers le formulaire. Redimensionnez ensuite l'objet grille de données de sorte que chaque colonne soit visible.
Prévisualiser des données liées à un objet grille de données	Cliquez sur la flèche raccourci de l'objet grille de données pour afficher la liste des Tâches DataGridView. Cliquez sur la commande Aperçu des données puis sur le bouton Aperçu dans la boîte de dialogue Aperçu des données.
Supprimer une colonne d'un objet grille de données	Cliquez sur la flèche raccourci de l'objet grille de données pour afficher la liste des Tâches DataGridView. Cliquez sur la commande Modifier les colonnes, puis sur la colonne à supprimer dans la zone de liste Colonnes sélectionnées et enfin sur le bouton Supprimer.
Trier les enregistrements d'une grille pendant l'exécution	Cliquez sur l'en-tête de colonne qui doit servir de base au tri. Visual Studio trie la grille par ordre alphabétique en fonction de cette colonne.
Inverser l'ordre de tri des enregistrements d'une grille pendant l'exécution	Cliquez une deuxième fois sur l'en-tête de la colonne pour inverser l'ordre de tri (de A–Z à Z–A).
Changer la largeur par défaut d'une colonne dans un objet grille de données	Dans la fenêtre Propriétés, cliquez sur la propriété *Columns* puis sur l'ellipse. Dans la boîte de dialogue Modifier les colonnes, ajustez la propriété *Width*.
Masquer les en-têtes de colonnes dans un objet grille de données	Positionnez la propriété *ColumnHeaderVisible* sur *False*.
Créer un autre schéma de couleur pour les lignes d'un objet grille de données	Servez-vous de la propriété *AlternatingRowsDefaultCellStyle* pour choisir une couleur d'une ligne sur deux. Dans la boîte de dialogue Générateur CellStyle, ajustez la propriété *BackColor*. La couleur sélectionnée alternera avec le blanc.
Modifier la couleur des lignes de la grille	Modifiez la propriété *GridColor*.
Ajouter un deuxième objet grille de données à un formulaire	Faites glisser une deuxième table de la fenêtre Sources de données vers le formulaire. Redimensionnez et personnalisez la table en veillant à prévoir un formulaire suffisamment large pour afficher toutes les colonnes et tous les enregistrements que l'utilisateur doit voir. Pour ajouter une deuxième barre de navigation permettant au formulaire d'accéder à la table, créez un deuxième contrôle *BindingNavigator* sur le formulaire et attribuez à sa propriété *BindingSource* la valeur de la source de liaison qui représente la nouvelle table créée.

Pour	Faites ceci
Empêcher l'utilisateur de modifier les données d'un objet grille de données	Positionnez la propriété *ReadOnly* de la grille sur *True*.
Écrire les changements apportés à la grille dans la base de données sous-jacente	Vérifiez que la propriété *ReadOnly* de l'objet grille de données est positionnée sur *False*. Pendant l'exécution, servez-vous du bouton Enregistrer les données, qui se trouve dans la barre de navigation, pour enregistrer les changements et mettre la base de données à jour. Alternativement, utilisez la méthode *Update* de l'adaptateur de la table ou la méthode *Me.TableAdapterManager.updateAll* au sein du code.

Chapitre 20

Créer des sites et des pages web avec Microsoft Visual Web Developer et ASP.NET

À la fin de ce chapitre, vous saurez :

- Démarrer Visual Web Developer et créer un nouveau site web
- Utiliser les outils et les fenêtres Visual Web Developer, y compris le Concepteur de pages web
- Utiliser la Boîte à outils Visual Web Developer pour ajouter des contrôles serveur aux pages web
- Ajouter du texte, des effets de mise en forme et du code Visual Basic pour créer une page web qui calcule les mensualités d'un prêt automobile
- Créer une page HTML qui présente des informations d'aide
- Utiliser le contrôle *HyperLink* pour lier une page web à une autre sur un site web
- Utiliser le contrôle *GridView* pour afficher une table d'informations issues d'une base de données sur une page web
- Définir la propriété *Title* de l'objet *DOCUMENT* et assigner un nom à une page web

Dans ce chapitre, nous verrons comment créer des sites web et des pages web avec le nouvel outil Visual Web Developer intégré à Microsoft Visual Studio 2008. Visual Web Developer possède l'apparence et le comportement de l'environnement de développement Visual Studio, mais il a été adapté à la programmation web et à Microsoft ASP.NET 3.5, le composant Microsoft .NET Framework proposant une fonctionnalité Internet moderne. ASP.NET a fait son apparition dans Microsoft Visual Studio .NET 2002 et remplace *WebClasses* et le Concepteur de pages DHTML de Microsoft Visual Basic 6. Bien qu'il ne soit pas possible de détailler ici la programmation web et ASP.NET, la programmation web et la programmation Windows possède suffisamment de similitudes pour permettre quelques expérimentations intéressantes, même si vous ne possédez que peu d'expérience dans le domaine de HTML. Consacrez quelques heures à ce chapitre et vous saurez comment créer rapidement un site web qui calcule les loyers de prêts automobiles, créer une page HTML avec des informations d'aide et afficher les candidats à l'emprunt issus d'une base de données Microsoft Office Access *via* le contrôle *GridView*.

ASP.NET

ASP.NET 3.5 est la dernière plate-forme de développement web de Microsoft. Il a été amélioré dans cette version grâce à l'ajout du modèle de programmation AJAX (*Asynchronous JavaScript and XML*), de nouveaux contrôles serveur, de services d'authentification et de profils ainsi que du contrôle *LinqDataSource*, qui autorise l'emploi du langage LINQ (*Language-Integrated Query*) dans des contextes de développement web. Même s'il possède des similitudes avec une technologie de programmation web antérieure appelée ASP (*Active Server Pages*), ASP.NET a été fortement amélioré depuis sa première apparition dans Visual Studio .NET 2002. Il poursuit son évolution au fur et à mesure de l'ajout de nouvelles fonctionnalités aux logiciels .NET Framework et Visual Studio. Visual Web Developer est l'outil qui permet de créer et gérer les interfaces utilisateur ASP.NET, communément appelées *pages web* ou, dans au sens plus large, *sites web*.

Astuce Dans les ouvrages consacrés à la programmation, les pages web sont parfois appelées *formulaires web* et les sites web *applications web*, mais ces termes sont moins répandus dans la documentation Visual Studio 2008.

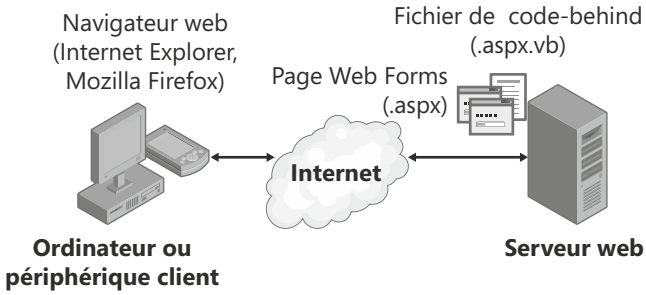
Visual Web Developer permet de créer un site web qui affiche une interface utilisateur, traite des données et propose la plupart des commandes et fonctionnalités présentes dans une application Microsoft Windows standard. Le site web créé est toutefois conçu pour être présenté dans un Navigateur Web comme Internet Explorer, Mozilla Firefox, Apple Safari ou Netscape Navigator. Il est généralement stocké sur un ou plusieurs *serveurs web* qui ont recours à IIS (*Microsoft Internet Information Services*) pour afficher les pages web appropriées et gérer l'essentiel des tâches informatiques impliquées par le site web (dans Visual Studio 2005 et 2008, les sites web peuvent également se trouver et s'exécuter sur un ordinateur local qui ne requiert pas IIS, offrant davantage d'options de développement et de déploiement). Cette stratégie distribuée permet d'exécuter les sites web sur une vaste gamme d'ordinateurs autonomes ou basés sur l'Internet, quelque que soit l'endroit où les utilisateurs et leurs riches sources de données se trouvent.

Pour créer un site web dans Visual Studio 2008, il suffit de cliquer sur la commande Nouveau Site Web du menu Fichier et de se servir du Visual Web Developer pour créer une ou plusieurs pages web représentant collectivement un site web. Chaque page se compose de deux parties :

- Une page Web Forms qui contient le code HTML et les contrôles qui créent l'interface utilisateur ;
- Un fichier contenant le code d'arrière-plan (fichier *code-behind*), un module de code qui contient le code qui se trouve « derrière » la page Web Forms.

Cette division est conceptuellement similaire à celle des formulaires Windows que vous avez créés dans Visual Basic et qui se composent d'un composant interface et d'un com-

posant module de code. Le code de ces deux composants peut être stocké dans un fichier .aspx unique, mais le code de la page Web Forms est généralement stocké dans un fichier .aspx et le fichier code-behind dans un fichier .aspx.vb. L'illustration suivante montre une vue conceptuelle d'un site web ASP.NET stocké sur un serveur et affiché dans un Navigateur Web.



Outre les pages web, les sites web peuvent héberger des modules de code (fichiers .vb), des pages HTML (fichiers .html), des informations de configuration (fichier Web.config), des informations globales relatives à l'application web (fichier Global.asax) ainsi que d'autres composants. Le Concepteur de pages web et l'Explorateur de solutions permettent de passer rapidement d'un composant à l'autre.

Pages web et formulaires Windows

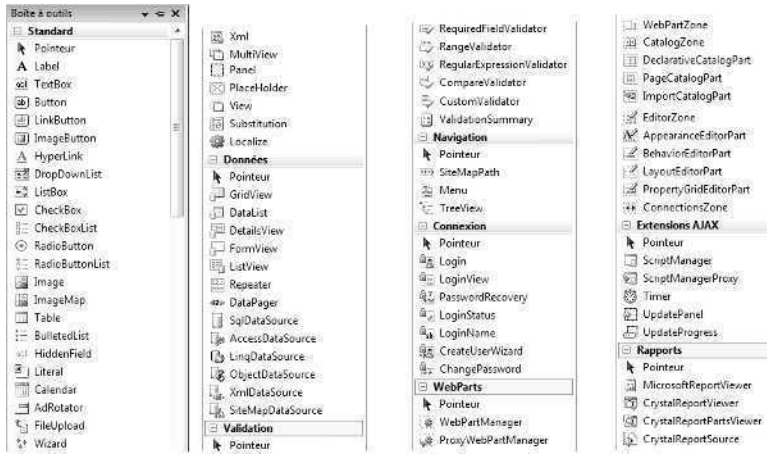
Quelles sont les principales différences entre les pages web et les formulaires Windows ? Pour commencer, les pages web proposent un modèle de programmation légèrement différent de celui des formulaires Windows. Alors que ces derniers utilisent une fenêtre d'application Windows comme interface utilisateur principale d'un programme, le site web présente les informations à l'utilisateur *via* une ou plusieurs pages web supportées par du code. Ces pages s'affichent d'un Navigateur Web et sont créées avec le Concepteur de pages web.

À l'instar d'un formulaire Windows, une page web peut contenir du texte, des graphismes, des boutons, des zones de liste et autres objets employés pour fournir des informations, traiter les entrées ou afficher le résultat. En revanche, le jeu de contrôles de base qui sert à créer une page web ne se trouve pas dans l'onglet Contrôles communs de la Boîte à outils. Les sites web ASP.NET exploitent les contrôles des onglets de la Boîte à outils Visual Web Developer : Standard, Data, HTML et bien d'autres. Chaque contrôle Visual Web Developer possède ses propres méthodes, propriétés et événements et, même s'il existe des similitudes entre ces contrôles et ceux des formulaires Windows, on note cependant quelques différences importantes. Par exemple, le contrôle *DataGridView* de Visual Studio s'appelle *GridView* dans Visual Web Developer et possède des propriétés et des méthodes différentes.

De nombreux contrôles de page web sont des contrôles serveur, ce qui signifie qu'ils s'exécutent sur le serveur web. Les *contrôles serveur* possèdent dans leur balise un préfixe "asp". Les contrôles HTML (situés sur l'onglet HTML de la Boîte à outils Visual Web Developer) sont des *contrôles clients* par défaut, autrement dit, ils s'exécutent uniquement au sein du navigateur de l'utilisateur final. Pour l'instant, il vous suffit de savoir qu'il est possible d'utiliser des contrôles serveur, des contrôles HTML ou une combinaison des deux dans vos projets de sites web. Au fil de l'amélioration de votre expérience en programmation web, vous pourriez souhaiter examiner la programmation AJAX avec Visual Studio : celle-ci est capable d'améliorer l'efficacité de vos applications Web et d'ajouter des éléments d'interface utilisateur avancés pour les utilisateurs.

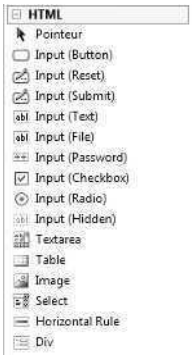
Contrôles serveur

Les contrôles serveur bénéficient de meilleures fonctionnalités que les contrôles HTML et fonctionnent de manière similaire à ceux des formulaires Windows. En fait, la majorité des contrôles serveur possèdent des noms identiques à ceux des formulaires Windows et disposent pour l'essentiel des mêmes propriétés, méthodes et événements. Outre les contrôles simples comme *Button*, *TextBox* et *Label*, les onglets de la Boîte à outils proposent des contrôles plus élaborés comme *FileUpload*, *LoginView* et *RequiredFieldValidator* (cette liste de contrôles a été significativement augmentée dans Visual Studio 2008). L'illustration suivante montre la plupart des contrôles serveur de la Boîte à outils Visual Web Developer :



Contrôles HTML

Les contrôles HTML constituent un ensemble d'anciens contrôles d'interface utilisateur pris en charge par tous les Navigateurs Web et qui se conforment aux premiers standards HTML développés pour la gestion des éléments de l'interface utilisateur sur une page web classique. On y trouve, par exemple, les contrôles *Button*, *Text Field* et *Checkbox*, pratiques pour gérer les informations d'une page web et qui peuvent intégralement être représentés par du code HTML. Vous les reconnaîtrez sans doute si vous avez déjà écrit du code HTML ou si vous avez expérimenté le Concepteur de page DHTML Visual Basic 6. Toutefois, bien qu'ils soient simples d'utilisation et possèdent l'avantage d'être un « dénominateur commun » à la plupart des Navigateurs Web, ils restent limités par le fait qu'ils ne peuvent pas conserver leur état (autrement dit, les données qu'ils contiennent se perdent entre les affichages d'une page web). L'illustration suivante montre les contrôles HTML de l'onglet HTML de la Boîte à outils Visual Web Developer :



Créer un site web avec Visual Web Developer

Les exercices pratiques restent la meilleure manière de comprendre Visual Web Developer et ASP.NET. Dans les exercices de ce chapitre, vous allez créer un calculateur de prêt automobile simple qui détermine les mensualités de remboursement et affiche une page HTML contenant un texte d'aide. Plus loin dans le chapitre, nous verrons comment utiliser le contrôle *GridView* pour afficher une table de données sur une page web du même site web. Nous commencerons par vérifier que Visual Studio est correctement configuré pour la programmation ASP.NET, puis vous créerez un nouveau projet de site web. Vous utiliserez ensuite le Concepteur de pages web pour créer une page web contenant du texte et des liens et vous servirez des contrôles de la Boîte à outils Visual Web Developer pour ajouter des contrôles à la page web.

Exigences logicielles pour la programmation ASP.NET

Avant de créer votre premier site web ASP.NET, vous devez vous assurer de la bonne configuration de votre ordinateur. Pour effectuer une programmation ASP.NET, Visual Web Developper doit être installé. C'est un composant des éditions Standard Edition, Professionnal Edition et Team System de Visual Studio 2008. Visual Studio 2008 propose son propre serveur web local, si bien que mettre en place et configurer un serveur web à l'aide d'IIS et du .NET Framework est superflu. Posséder un serveur web local facilite la création et le test de vos sites web ASP.NET.

Visual Studio 2008 bénéficie d'une amélioration intéressante : il n'est plus nécessaire de développer le site web sur un ordinateur entièrement configuré pour agir comme un serveur web. Dans Visual Studio .NET 2002 et 2003, le système de développement devait héberger ou avoir accès à un serveur web exécutant Windows 2000, Windows XP Professionnel ou Windows Server 2003 et contenant également une installation d'IIS, des Extensions serveur Microsoft FrontPage 2000 et le .NET Framework. Autrement dit, si vous aviez installé la version Microsoft Windows XP Edition Familiale, vous jouiez de malchance puisque cette version ne contient ni ne prend en charge IIS ou les extensions serveurs FrontPage 2000 : la seule possibilité était d'accéder à un serveur web distant correctement configuré.

Avec Visual Studio 2005 et 2008, vous pouvez créer et exécuter un site web à l'un des trois emplacements suivants :

- Votre propre ordinateur (le système de fichiers local) ;
- Un serveur HTTP qui contient IIS et les composants associés ;
- Un site FTP (un serveur de fichiers distant).

Dans ce livre, nous choisirons la première option, autrement dit votre ordinateur, puisqu'elle n'exige ni logiciel ni matériel supplémentaire. De surcroît, lors du développement d'un site web sur le système de fichiers local, tous les fichiers du site web sont stockés à un seul emplacement. Après avoir réalisé les tests de l'application, vous pouvez déployer les fichiers sur le serveur web de votre choix.

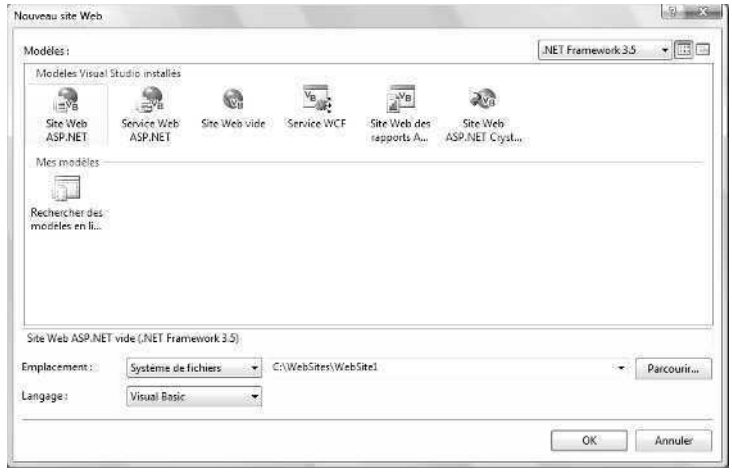
Remarque Avant de déployer le site web sur un serveur web, servez-vous du Panneau de configuration pour vérifier qu'IIS et les extensions serveur FrontPage 2000 sont installés sur votre système (Avec Windows XP, dans le Panneau de configuration, cliquez sur Ajouter ou supprimer des programmes, puis sur Ajouter ou supprimer des composants Windows et recherchez IIS. Avec Windows Vista, ouvrez dans le Panneau de configuration Programmes et fonctionnalités, cliquez sur Activer ou désactiver des fonctionnalités Windows, puis recherchez IIS et ASP.NET). Microsoft demande d'installer ces composants du serveur web avant le .NET Framework et Visual Studio. En effet, le .NET Framework enregistre ses extensions avec IIS. Si vous avez installé IIS après Visual Studio et que vous rencontrez des problèmes, cela pourrait en être la raison.

Créer un nouveau site web

1. Démarrez Visual Studio et cliquez sur la commande Nouveau Site Web du menu Fichier.

Remarque Si vous ne voyez pas la commande Nouveau site web dans le menu Fichier, Visual Web Developer n'est pas installé.

Vous avez sans doute remarqué cette commande auparavant, mais nous ne l'avons pas encore utilisée. Elle démarre Visual Web Developer et prépare Visual Studio à la création d'un site web. La boîte de dialogue Nouveau site web s'affiche :

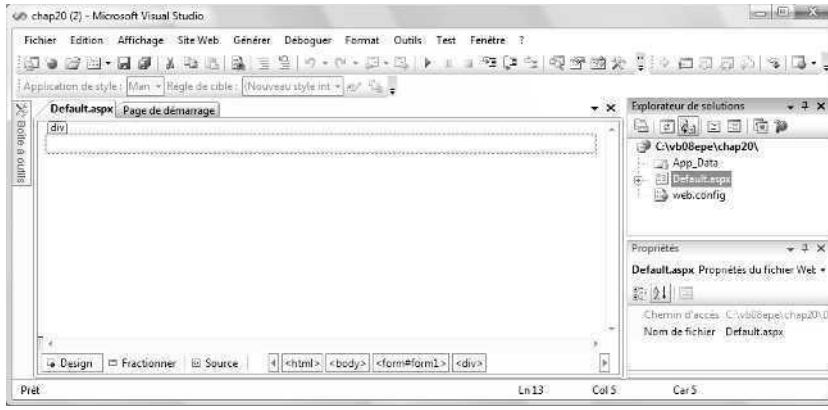


Dans cette boîte de dialogue, vous pouvez sélectionner un modèle d'application ou de site web, l'emplacement du site web (système de fichiers local, serveur HTTP ou site FTP) et le langage de programmation à employer (Visual Basic ou Visual C#). Vous pouvez également préciser la version du .NET Framework associée à votre application web. Si la version 3.5 offre plus de fonctionnalités, il sera parfois nécessaire de concevoir une application pour des plates-formes qui possèdent une version antérieure du .NET Framework.

2. Vérifiez que le modèle Site web ASP.NET est sélectionné et que le langage choisi est Visual Basic.
3. Cliquez dans la zone de texte Emplacement sur Système de fichiers.
4. Cliquez sur Parcourir, créez un nouveau dossier nommé "monchap20", vérifiez que le dossier sélectionné est c:\vb08epe\monchap20, puis cliquez sur Ouvrir.

   La boîte de dialogue Choisir un emplacement est légèrement différente de la boîte de dialogue Emplacement du projet utilisée jusque-là. En outre, vous avez toujours désigné l'emplacement du dossier des projets après avoir créé ces derniers. Les projets Visual Web Developer sont, quant à eux, enregistrés en aval. Le préfixe « mon » dans le chemin d'accès évite tout conflit avec l'exemple de site web des exercices (c:\vb08\chap20).
5. Cliquez sur OK pour finaliser les changements.

   Visual Studio charge Visual Web Developer et crée une page web (Default.aspx) qui contient l'interface utilisateur et le fichier code-behind (Default.aspx.vb) qui accueillera le code de la page web. Le code HTML source de la page web s'affiche sur l'onglet Source du Concepteur de pages web (si le Concepteur de pages web ne s'affiche pas, double-cliquez sur Default.aspx dans l'Explorateur de solutions). Voici à quoi ressemble votre écran :



Contrairement au Concepteur de formulaires Windows, le Concepteur de pages web présente la page web selon trois modes dans l'EDI. Les trois onglets situés dans la partie inférieure du concepteur (Design, Fractionner et Source) permettent de basculer d'une vue à l'autre. Selon la configuration et l'utilisation de votre système, vous affichez l'un ou l'autre de ces onglets (dans la figure, l'onglet présenté est l'onglet Design).

L'onglet Source permet de voir et de modifier le code HTML utilisé pour afficher la page web dans un Navigateur Web. Si vous avez déjà utilisé Microsoft Visual InterDev ou Microsoft FrontPage, vous reconnaîtrez ces deux méthodes d'affichage d'une page web et éventuellement certaines balises de mise en forme HTML qui contrôlent l'affichage des pages web.

L'onglet Design indique approximativement l'aspect de votre page web dans un Navigateur Web. Lorsque cet onglet est sélectionné, une page blanche s'affiche dans le Concepteur et présente le résultat de la mise en forme du code source. Vous pouvez y ajouter des contrôles et définir la disposition des objets sur la page. L'onglet Fractionner propose un affichage combiné des onglets Source et Design.

Visual Web Developer a subi d'autres modifications qu'il est inutile de préciser. La Boîte à outils contient plusieurs collections de contrôles exclusivement réservés à la programmation web (ce qui change par rapport à Visual Studio .NET 2003, où les contrôles de programmation web se trouvent sur les onglets Web Forms et HTML de la Boîte à outils Visual Studio classique). L'Explorateur de solutions propose également une liste différente de fichiers pour le site web en construction. Remarquez tout particulièrement le fichier Default.aspx dans l'Explorateur de solutions. Ce fichier contient le code de l'interface utilisateur de la page web active. Imbriqué sous le fichier Default.aspx figure également un fichier nommé Default.aspx.vb. Un fichier de configuration, intitulé Web.config, est également présent.

Ajoutons maintenant un peu de texte à la page web avec le Concepteur de pages web.

Utiliser le Concepteur de pages web

Contrairement à un formulaire Windows, il est possible d'ajouter du texte directement sur une page web lorsqu'elle se trouve dans le Concepteur de pages web. En mode Source, le texte apparaît entre des balises HTML, un peu comme dans l'Éditeur de code Visual Studio. En mode Design, le texte apparaît de manière linéaire de haut en bas, comme dans un traitement de texte, tel que Microsoft Word, et sans code HTML. Dans cette section, vous allez saisir le texte en mode Design, le modifier puis apporter des changements de mise en forme par l'intermédiaire de la barre d'outils Mise en forme. Cette méthode est beaucoup plus rapide que l'ajout d'un contrôle *Label* à la page web. Dans le prochain exercice, vous allez saisir le texte du calculateur de prêt automobile.

Ajouter du texte en mode Design

1. Cliquez sur l'onglet Design, s'il n'est pas sélectionné, pour afficher le Concepteur de pages web en mode Design.

   La silhouette d'un rectangle apparaît en haut de la page web.

2. Placez le point d'insertion dans le rectangle.

   Un pointeur en I clignotant apparaît dans la partie supérieure de la page web.

3. Saisissez **Calculateur de prêt automobile** et appuyez sur ENTRÉE.

   Visual Studio affiche le titre de la page web exactement tel qu'il se présentera lorsque le site web s'ouvrira dans un navigateur.

4. Sous le titre, tapez la phrase suivante :

   Saisissez les informations nécessaires et cliquez sur Calculer !

   Servons-nous maintenant de la barre d'outils Mise en forme pour formater le titre en gras et dans une police plus grande.

5. Effectuez un clic droit sur la barre d'outils Standard de Visual Web Developer pour afficher la liste des barres d'outils disponibles dans l'environnement de développement.

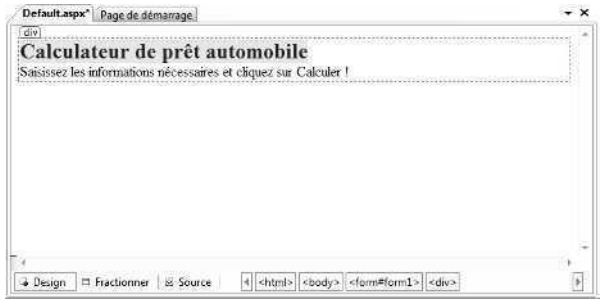
6. Si vous ne voyez pas de coche à côté de Mise en forme dans la liste, cliquez sur Mise en forme pour ajouter la barre d'outils Mise en forme.

   La barre d'outils Mise en forme s'affiche dans l'environnement de développement. Remarquez qu'elle contient quelques fonctionnalités habituellement absentes d'une barre d'outils de mise en forme.

7. Sélectionnez le texte « Calculateur de prêt automobile ». Avant de formater le texte dans Visual Web Developer, vous devez le sélectionner.

8. Cliquez sur le bouton Gras de la barre d'outils Mise en forme et choisissez une police de 24 points.

   Voici à quoi ressemble votre écran :

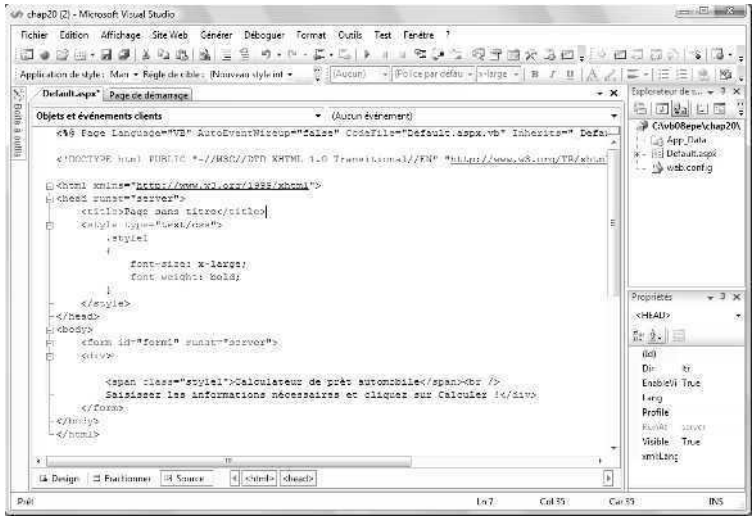


Examinons maintenant le code HTML du texte saisi et sa mise en forme.

Afficher le code HTML d'une page web

1. Cliquez sur l'onglet Source dans la partie inférieure du Concepteur.

   L'onglet Source présente le code HTML de la page web. Pour afficher davantage de code, fermez temporairement quelques outils. Voici à quoi ressemble le code HTML (le vôtre pourrait présenter quelques différences mineures) :



   Une page web se compose d'informations relatives au fichier et ou document, des codes de mise en forme appelés balises HTML placées entre crochets, ainsi que du texte et des objets à afficher sur la page web. Cette dernière est encore courte pour l'instant : elle contient un en-tête avec les informations relatives au langage sélectionné au moment de la création de l'application web, le nom de tout fichier code-behind et les formulaires hérités.

   Les balises HTML se présentent par paires, ce qui identifie clairement le début et la fin de la section. Par exemple, la balise `<body>` identifie le début du document et la balise `<\body>` désigne la fin. Notez que le texte « Calculateur de prêt automobile » est placé au sein d'un élément HTML `span` situé sous un bloc de style qui lui applique un style gras (`font-weight : bold`) avec une taille de police de grande taille (`font-size :X-large`). Sous ce texte, la deuxième ligne de texte saisie est présente.

> **Astuce** Rappelez-vous que l'onglet Source est un éditeur. Il est en conséquence possible de modifier le texte saisi en se servant des techniques d'édition standards. Si vous possédez des connaissances en HTML, vous pouvez ajouter des balises de mise en forme et du contenu.

2. Cliquez sur l'onglet Design pour afficher la page web en mode Design et ouvrez la Boîte à outils si elle n'est pas visible.

Ajouter des contrôles serveur à un site web

Vous allez à présent ajouter les contrôles *TextBox*, *Label* et *Button* au calculateur de prêt automobile. Ces contrôles, qui se trouvent dans la Boîte à outils Visual Web Developer, sont très similaires à ceux des formulaires Windows du même nom étudiés jusqu'ici. Nous étudierons toutefois certaines de leurs différences importantes à mesure qu'elles se présenteront. Le plus important est de se rappeler que dans le Concepteur de pages web, vous insérez les contrôles au point d'insertion suite à un double-clic sur le nom d'un contrôle dans la Boîte à outils. Après avoir ajouté les contrôles à la page web, vous définirez les paramètres de leurs propriétés.

Utiliser les contrôles *TextBox*, *Label* et *Button*

1. Affichez l'onglet Standard de la Boîte à outils, s'il n'est pas visible.
2. Placez le point d'insertion à la fin de la deuxième ligne de texte sur la page web et appuyez trois fois sur la touche ENTRÉE pour créer sous le texte un petit espace vide destiné aux contrôles.

   Les contrôles étant placés au point d'insertion, vous devez utiliser les touches d'édition de texte pour positionner correctement le point d'insertion avant de double-cliquer sur un contrôle dans la Boîte à outils (c'est une différence importante entre le Concepteur de pages web et le Concepteur de formulaires Windows). Ce dernier permet de créer des contrôles à l'emplacement de votre choix sur le formulaire.

> **Remarque** Par défaut, le Concepteur de page web place les contrôles relativement aux autres contrôles. C'est une différence importante entre le Concepteur de page web et le Concepteur de formulaire Windows. Ce dernier permet de placer un contrôle là où vous le voulez sur un formulaire. Vous pouvez configurer le Concepteur de page web pour placer les contrôles là où vous le voulez sur la page web (ce qui porte le nom de positionnement absolu), mais cela peut entraîner un comportement différent selon le Navigateur Web employé par l'utilisateur final.

3. Dans l'onglet Standard de la Boîte à outils, double-cliquez sur le contrôle *TextBox* pour créer un objet zone de texte au point d'insertion sur la page web.

   Remarquez le texte *asp:textbox#TestBox1* qui apparaît au-dessus de l'objet zone de texte. Le préfixe « asp » indique que ce contrôle est un contrôle serveur. Ce texte disparaît lors de l'exécution du programme.
4. Cliquez à droite de l'objet zone de texte pour placer le point d'insertion à l'extérieur du bord et appuyez deux fois sur ENTRÉE.

5. Double-cliquez à nouveau sur le contrôle *TextBox* pour ajouter un deuxième objet zone de texte à la page web.
6. Répétez les étapes 4 et 5 pour créer un troisième objet zone de texte sous le deuxième.

   Vous allez maintenant placer le contrôle *Label* pour insérer des étiquettes identifiant les zones de texte.
7. Cliquez à droite du premier objet zone de texte pour placer le point d'insertion à l'extérieur de son bord droit.
8. Appuyez deux fois sur la barre d'espace pour insérer deux espaces vides et double-cliquez sur le contrôle *Label* dans la Boîte à outils pour ajouter un objet étiquette à la page web.
9. Répétez les étapes 7 et 8 pour ajouter les objets étiquette à droite de la deuxième et de la troisième zone de texte.
10. Cliquez à droite du troisième objet étiquette pour placer le point d'insertion à l'extérieur de son bord droit et appuyez deux fois sur ENTRÉE.
11. Double-cliquez sur le contrôle *Button* pour créer un objet bouton dans la partie inférieure de la page web.

    Le contrôle *Button*, à l'instar des contrôles *TextBox* et *Label*, est similaire à son homonyme des formulaires Windows. Voici à quoi ressemble votre écran :

Définissons maintenant quelques propriétés pour les sept nouveaux contrôles que vous venez de créer sur la page web. Ouvrez la fenêtre Propriétés si elle n'est pas visible (appuyez sur F4). À mesure que vous définirez les propriétés, vous remarquerez d'importantes différences entre les pages web et les formulaires Windows : la classique propriété *Name* a été remplacée par *ID* dans Visual Web Developer. Malgré ces différences de noms, les deux propriétés remplissent la même fonction.

12. Définissez les propriétés suivantes pour les objets du formulaire :

Objet	Propriété	Paramètre
TextBox1	*ID*	txtMontant
TextBox2	*ID*	txtIntérêt
TextBox3	*ID*	txtPaiement
Label1	*ID*	lblMontant
	Text	« Montant de l'emprunt »
Label2	*ID*	LblIntérêt
	Text	« Taux d'intérêt (par exemple : 0,09) »
Label3	*ID*	LblPaiement
	Text	« Mensualité »
Button1	*ID*	BtnCalculer
	Text	« Calculer »

Voici à quoi ressemble votre page web :

Écrire des procédures événementielles pour les contrôles de la page web

Pour écrire les procédures événementielles (ou gestionnaires d'événements) par défaut pour des contrôles d'une page web, il suffit de double-cliquer sur les objets dans la page web et de saisir le code nécessaire dans l'Éditeur de code. L'utilisateur voit les contrôles de la page web dans son navigateur, mais le code qui s'exécute se trouve sur l'ordinateur test local ou un serveur web, selon la manière dont vous configurez le développement du projet et le déployez. Par exemple, lorsque l'utilisateur clique sur un bouton d'une page web hébergée par un serveur web, le navigateur envoie l'événement *Click* du bouton au serveur, qui traite l'événement et envoie la page web au navigateur. Si ce processus semble similaire à celui des formulaires Windows, le processus d'arrière-plan est conséquent lorsqu'on utilise un contrôle sur une page web ASP.NET !

Dans le prochain exercice, vous allez créer la procédure événementielle par défaut de l'objet *btnCalculer* de la page web.

Créer la procédure événementielle *btnCalculer_Click*

1. Sur la page web, double-cliquez sur le bouton *Calculer*.

   Le fichier code-behind (Default.aspx.vb) s'ouvre dans l'Éditeur de code et la procédure événementielle *btnCalculer_Click* s'affiche.

2. Tapez le code suivant :

   ```
   Dim PaiementPrêt As Double
   'Utilise la fonction Pmt pour déterminer le paiement d'un prêt sur 36 mois
   PaiementPrêt = Pmt(CDbl(txtIntérêt.Text) / 12, 36, CDbl(txtMontant.Text))
   txtPaiement.Text = Format(Abs(PaiementPrêt), "0.00 €")
   ```

   La procédure événementielle fait appel à la fonction *Pmt*, une fonction financière qui fait partie du langage Visual Basic, pour déterminer le loyer mensuel d'un prêt automobile en fonction d'un taux d'intérêt spécifique (*txtIntérêt.Text*), une période de prêt de 3 ans (36 mois) et le montant principal indiqué (*txtMontant.Text*). Le résultat est stocké dans la variable en double précision *PaiementPrêt* puis est mis en forme avec le format monétaire approprié et affiché sur la page web *via* l'objet zone de texte *txtPaiement*.

Les deux propriétés *Text* sont converties du format chaîne au format double précision *via* la fonction *CDbl*. La fonction *Abs* (valeur absolue) sert à faire du remboursement du prêt un nombre positif (*Abs* est souligné dans l'Éditeur de code car elle repose sur la classe *System.Math*, que nous importerons plus loin). Pourquoi fait apparaître le remboursement du prêt sous forme d'un nombre positif ? La fonction *Pmt* retourne un nombre négatif par défaut (reflétant le montant dû), mais un format négatif peut prêter à confusion s'il ne fait pas partie d'un bilan.

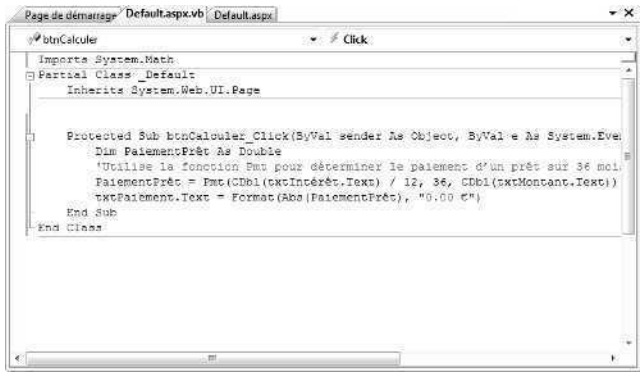
Remarquez que les instructions dans le fichier code-behind ne sont autres que du code Visual Basic, identique à celui employé jusqu'ici. Le processus est fondamentalement similaire à celui qui consiste à créer une application Windows.

3. Placez le point d'insertion au début de l'Éditeur de code et saisissez l'instruction suivante comme première du fichier :

```
Imports System.Math
```

Comme vous l'avez appris au chapitre 5, « Variables et formules Visual Basic et l'environnement .NET Framework », la fonction *Abs* n'est pas comprise dans Visual Basic par défaut. Elle fait partie de la classe *System.Math* du .NET Framework et peut être plus facilement référencée dans un projet *via* l'instruction *Imports*. Les applications web, à l'instar des applications Windows, peuvent faire usage des bibliothèques de classes du .NET Framework.

L'Éditeur de code présente l'aspect suivant :



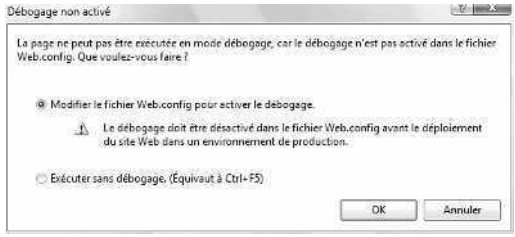
4. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout.

Et voilà ! Vous avez saisi le code nécessaire à l'exécution du calculateur de prêt automobile et rendu votre page web interactive. Vous allez maintenant générer et exécuter le programme. Vous en apprendrez également un peu sur les réglages de sécurité d'Internet Explorer, un sujet étroitement apparenté au développement web.

Générer et afficher le site web

1. Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage.

   Visual Studio affiche la boîte de message suivante relative au débogage :



   Cette boîte de dialogue ne présente guère d'intérêt. Elle indique simplement que le fichier Web.config du projet n'autorise actuellement pas le débogage (une fonction de sécurité standard). Bien qu'il soit possible de contourner cette boîte de dialogue chaque fois que vous testez une application dans Visual Studio en sélectionnant l'option Exécuter sans débogage, je vous recommande de modifier le fichier Web.config maintenant.

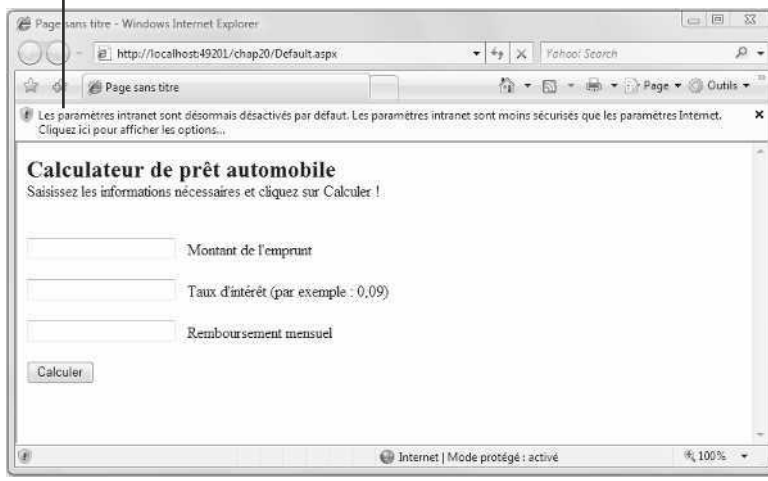
Astuce de sécurité Avant de distribuer ou déployer largement un vrai site web, veillez à désactiver le débogage dans Web.config pour garder votre application à l'abri de toute modification non autorisée.

2. Cliquez sur OK pour modifier le fichier Web.config.

   Visual Studio modifie le fichier, génère le site web et affiche la page web d'ouverture dans Internet Explorer. Voici à quoi ressemble la fenêtre Sources de données :

Astuce de sécurité Si Internet Explorer affiche la boîte de dialogue « Le débogage de script est désactivé », cliquez sur Oui pour poursuivre. Vous pouvez ajuster un paramètre de sécurité d'Internet Explorer de sorte que ce message n'apparaisse plus à l'avenir (nous ne réaliserons pas le débogage maintenant). Dans Internet Explorer, cliquez sur le menu Outils, choisissez la commande Options Internet, cliquez sur l'onglet Avancé puis supprimez la coche de l'option Désactiver le débogage des scripts.

Lors de la première exécution de votre application Web dans Internet Explorer, vous pourriez voir un avertissement de sécurité.



Astuce de sécurité La Barre d'information située en haut d'Internet Explorer pourrait signaler que les paramètres intranet sont désactivés par défaut (cette Barre d'information est montrée dans la figure précédente). Un avertissement intranet est encore dû au désir d'Internet Explorer de vous protéger contre tout programme sauvage ou accès non autorisé. Un intranet est un réseau local (généralement un réseau familial ou de petit groupe de travail). Comme Visual Studio se sert d'un adressage de type intranet lors du test de sites web construits sur votre propre ordinateur, le message d'avertissement risque fort de vous être affiché. Pour le supprimer temporairement, cliquez sur la Barre d'information et cliquez sur Ne plus afficher ce message. Pour supprimer les avertissements intranet de façon plus définitive, cliquez sur la commande Options Internet du menu Outils d'Internet Explorer. Dans l'onglet Sécurité, cliquez sur Intranet local. Cliquez sur le bouton Sites, puis dans la boîte de dialogue Intranet local éliminez la coche de l'option Détecter automatiquement le réseau Intranet. Restez toutefois prudent lorsque vous désactivez tout avertissement de sécurité, car leur but est de vous protéger.

Astuce Au lancement de ce site web, vous avez peut-être remarqué une infobulle dans la zone d'avertissement de la Barre des tâches Windows. Elle signale que le serveur web local a démarré et exécute le site web. Un clic droit sur la zone du Serveur de développement ASP.NET dans la zone d'avertissement vous procure plus d'informations sur le serveur web.

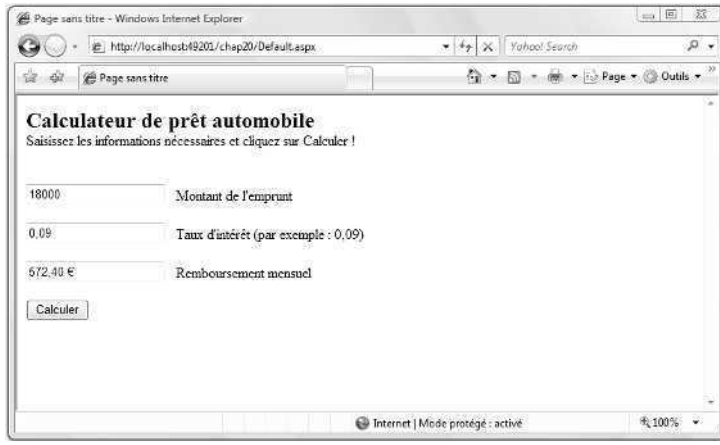
Revenez maintenant au test de la page web.

3. Tapez **18000** dans la zone de texte Montant de l'emprunt et **0,09** dans la zone de texte Taux d'intérêt.

   Vous allez calculer le remboursement mensuel d'un prêt de 18 000 € avec un taux d'intérêt de 9 % sur 36 mois.

4. Cliquez sur le bouton Calculer.

   Visual Basic calcule le remboursement et affiche 572,40 € dans la zone de texte Remboursement mensuel. Voici à quoi ressemble votre écran :



5. Fermez Internet Explorer.

Vous avez terminé de tester le site web pour l'instant. Lorsque vous fermez Internet Explorer, le programme s'arrête. Comme vous l'aurez remarqué, la génération et l'affichage d'un site web ne sont guère différents de ceux d'une application Windows, excepté que le site web s'exécute dans le navigateur. Il est même possible de placer des points d'arrêt dans l'application.

Vous vous demandez comment installer un tel site web sur un vrai serveur web ? La procédure fondamentale pour déployer un site web consiste à copier les fichiers .aspx et tous autres fichiers d'accompagnements nécessaires du projet dans un répertoire virtuel correctement configuré sur un serveur web qui exécute IIS et le .NET Framework. Vous pouvez effectuer le déploiement de différentes façons avec Visual Web Developer. Pour commencer, cliquez sur Copier le site web dans le menu Site Web ou cliquez sur Publier le site Web dans le menu Générer. Pour plus d'informations, recherchez « Déploiement ASP.NET » dans la documentation de Visual Studio.

Valider les champs d'entrée d'une page web

Bien que cette page web présente un intérêt, elle engendre des problèmes si l'utilisateur ne saisit pas le montant principal ou le taux d'intérêt ou qu'il saisit les données dans un format erroné. Pour consolider ce site web, il serait intéressant d'ajouter des contrôles de validation qui obligent l'utilisateur à saisir une entrée au format approprié. Les contrôles de validation se trouvent dans l'onglet Validation de la Boîte à outils Visual Web Developer. On y trouve des contrôles qui exigent une entrée de données dans un champ (*RequiredFieldValidator*), exigent une entrée dans une plage donnée (*RangeValidator*), et ainsi de suite. Pour plus d'informations sur les contrôles de validation, consultez la documentation de Visual Studio. Ces contrôles sont d'emploi simple.

Ajouter des pages web et des ressources à un site web

C'est maintenant que tout cela devient intéressant ! Seuls les sites web extrêmement simples se composent d'une seule page web. Avec Visual Web Developer, il est simple de développer rapidement un site web pour y inclure d'autres informations et ressources : pages HTML, pages XML, fichiers texte, enregistrements de base de données, services web, plans de site, etc. Pour ajouter une page HTML (une page web standard qui contient du texte et des contrôles HTML côté client), vous disposez de deux options :

- Créer une nouvelle page HTML en vous servant de la commande Ajouter un nouvel élément du menu Site Web. Après avoir créé la page HTML, vous y ajoutez du texte et des objets HTML *via* le Concepteur de pages web.
- Ajouter une page HTML déjà créée *via* la commande Ajouter un élément existant du menu Site Web puis personnaliser la page dans le Concepteur de pages web. Cette méthode est employée pour ajouter une ou plusieurs pages web déjà créées avec un outil tel que Microsoft Expression Web (si possible, ajoutez des pages qui ne se fondent pas sur des feuilles de style et des ressources externes, sans quoi vous devrez ajouter ces éléments au projet).

Pour lier les pages entre elles, Visual Web Developer propose le contrôle *HyperLink* qui crée un objet étiquette lien hypertexte sur lequel l'utilisateur clique pour passer de la page à cours à une autre page. Pour définir le texte affiché sur la page dans le contrôle *HyperLink*, vous faites appel à sa propriété *Text* et à sa propriété *NavigateUrl* qui désigne la ressource vers laquelle se rendre (URL ou chemin d'accès local).

Dans le prochain exercice, vous allez créer une deuxième page web en vous servant de la commande Ajouter un nouvel élément et l'enregistrer au format HTML avec les autres fichiers du projet. La nouvelle page sera un fichier d'aide auquel les utilisateurs du site web peuvent accéder pour obtenir des instructions sur le fonctionnement du calculateur de prêt. Après avoir créé la nouvelle page, vous ajouterez un contrôle *HyperLink* à la première page et attribuerez à sa propriété *NavigateUrl* la valeur de la nouvelle page HTML.

Créer une page HTML

1. Dans le menu Site Web, choisissez la commande Ajouter un nouvel élément.

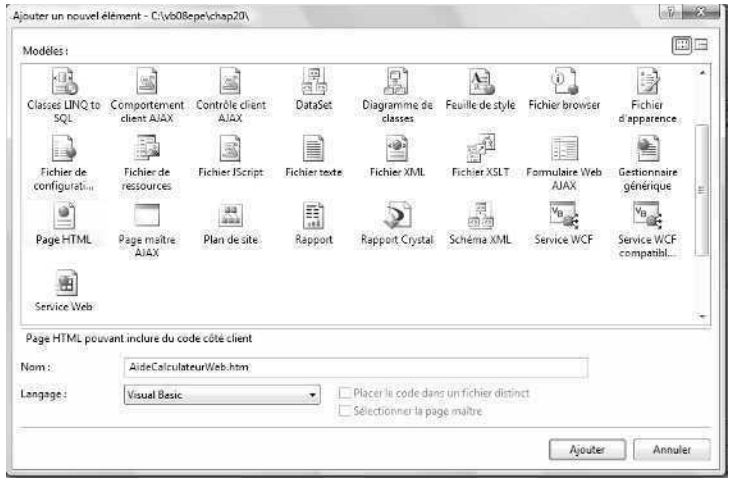
   La boîte de dialogue Ajouter un nouvel élément s'affiche, vous permettant d'ajouter un certain nombre de ressources Internet au site web.

2. Cliquez sur le modèle Page HTML.

   Vous allez insérer une page HTML vierge dans le projet, que vous pourrez utiliser pour afficher du texte formaté et des contrôles HTML (vous ne pouvez pas ajouter des contrôles serveur à cette page, puisque les pages HTML simples sont contrôlées par le navigateur du client et non par le serveur web).

3. Dans la zone de texte Nom, tapez **AideCalculateurWeb.htm**.

Voici à quoi ressemble votre écran :



4. Cliquez sur Ajouter.

   Le fichier AideCalculateurWeb.htm s'ajoute à l'Explorateur de solutions et s'ouvre dans le Concepteur de pages web en mode Design.

   Notez que la Boîte à outils ne contient que des contrôles HTML. En effet, ceci étant une page HTML, les contrôles serveur ne sont pas pris en charge.

5. Si nécessaire, cliquez sur l'onglet Design pour afficher la page HTML en mode Design. Le pointeur en I clignote sur la page, attendant votre saisie.

6. Saisissez le texte suivant :

   Calculateur de prêt automobile

   Le site web Calculateur de prêt automobile a été développé pour le livre *Microsoft Visual Basic 2008 Étape par étape* par Michael Halvorson (Microsoft Press, 2008). Visualisez de préférence ce site web avec Microsoft Internet Explorer version 6.0 ou ultérieure. Pour en savoir plus sur la manière dont cette application ADO.NET a été créée, lisez le chapitre 20 du livre.

   Instructions :

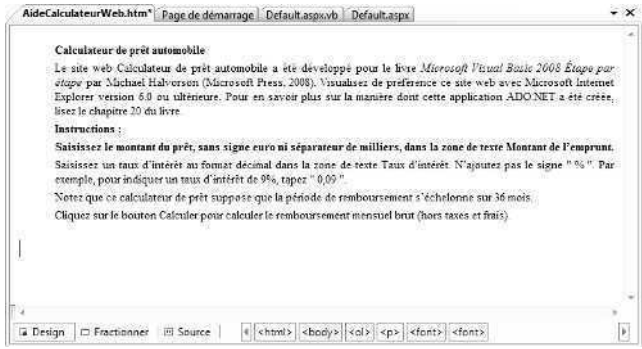
   Saisissez le montant du prêt, sans signe euro ni séparateur de milliers, dans la zone de texte Montant de l'emprunt.

   Saisissez un taux d'intérêt au format décimal dans la zone de texte Taux d'intérêt. N'ajoutez pas le signe « % ». Par exemple, pour indiquer un taux d'intérêt de 9 %, tapez « 0,09 ».

   Notez que ce calculateur de prêt suppose que la période de remboursement s'échelonne sur 36 mois.

   Cliquez sur le bouton Calculer pour calculer le remboursement mensuel brut (hors taxes et frais).

7. Servez-vous des boutons de la barre d'outils Mise en forme pour ajouter une mise en forme gras et italique, comme dans la figure suivante :



8. Dans la barre d'outils Standard, cliquez sur le bouton Enregistrer tout pour enregistrer vos changements.

Nous allons maintenant faire appel au contrôle *HyperLink* pour créer un lien hypertexte sur la première page web qui ouvrira le fichier AideCalculateurWeb.htm.

Utiliser le contrôle *HyperLink*

1. Affichez la page web (Default.aspx) en mode Design.
2. Placez le pointeur en I à droite de l'objet bouton et appuyez deux fois sur ENTRÉE.
3. Dans l'onglet Standard de la Boîte à outils, double-cliquez sur le contrôle *HyperLink* pour créer un objet lien hypertexte au niveau du point d'insertion.
4. Attribuez la valeur « Aide » à sa propriété *Text*.

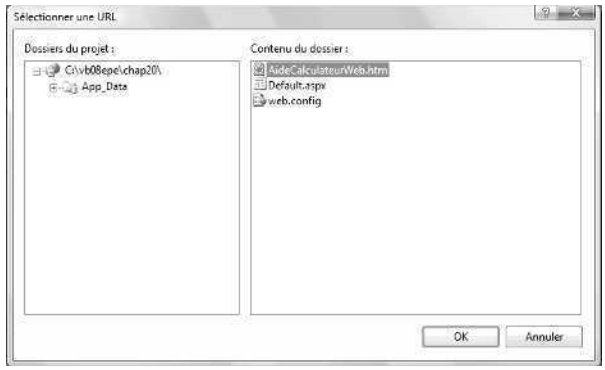
   La propriété *Text* contient le texte qui s'affiche en tant que lien hypertexte souligné sur la page web. Utilisez ici des mots qui indiquent clairement qu'il existe une page web contenant un texte d'aide.
5. Attribuez la valeur « lnkAide » à sa propriété *ID*.

   Cet objet est nommé pour maintenir la cohérence avec les autres objets du site web.
6. Cliquez sur la propriété *NavigateUrl* puis sur l'ellipse dans la deuxième colonne.

   La boîte de dialogue Sélectionner une URL s'affiche et vous invite à indiquer l'emplacement de la page web vers laquelle créer le lien.

7. Dans la zone de liste Contenu du dossier, cliquez sur le fichier AideCalculateurWeb.htm.

   Voici à quoi ressemble la boîte de dialogue :



8. Cliquez sur OK pour définir la propriété *NavigateUrl*.

   Voici à quoi ressemble votre page web :

Nouvel objet lien hypertexte

Le lien est terminé et vous pouvez à nouveau visualiser le site web dans le navigateur.

9. Cliquez sur le bouton Enregistrer tout.

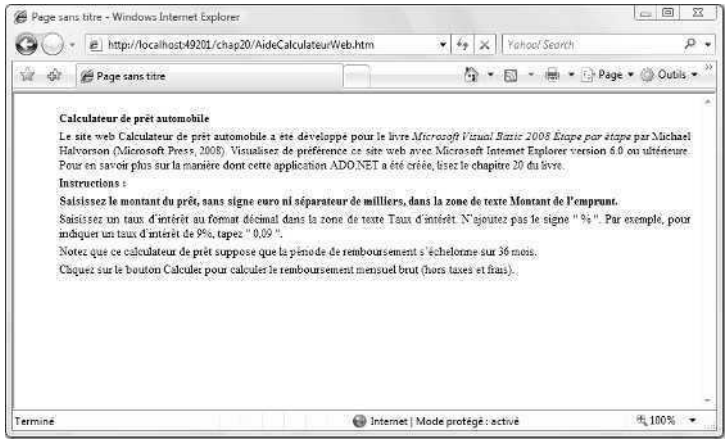
10. Cliquez sur le bouton Démarrer le débogage. Visual Studio génère le site web et l'affiche dans Internet Explorer.

11. Calculez un autre emprunt pour tester votre calculateur.

Essayez, par exemple, un emprunt de 20000 avec un taux d'intérêt de 0,075. Le résultat est 622,12 €.

12. Cliquez maintenant sur le lien hypertexte Aide pour voir fonctionner le contrôle *HyperLink*.

Internet Explorer affiche la nouvelle page HTML à l'écran (redimensionnez la fenêtre si nécessaire pour voir tout le texte). Voici à quoi ressemble votre page HTML :



13. Lisez le texte et cliquez sur le bouton Précédent d'Internet Explorer.

À l'instar de n'importe quel site web, vous pouvez cliquer sur le bouton Précédent et le bouton Suivant pour passer d'une page web à l'autre.

14. Fermez Internet Explorer pour fermer le site web.

Vous venez d'ajouter une page HTML simple au site web et de tester l'utilisation du contrôle *HyperLink* pour lier des pages web. Essayons maintenant quelque chose de plus élaboré qui montre jusqu'où vous pouvez aller dans un site web en choisissant d'inclure des informations issues d'une base de données.

Afficher les enregistrements d'une base de données sur une page web

Pour nombre d'utilisateurs, l'un des aspects les plus intéressants du web est la possibilité d'accéder rapidement à de grandes quantités d'informations *via* un Navigateur Web. La quantité d'informations à afficher sur un site web commercial dépasse souvent de beaucoup ce qu'un développeur peut de façon réaliste préparer avec de simples documents texte. Dans ce cas, il ajoute des objets de base de données aux sites web pour afficher des tables, champs et enregistrements issus d'une base de données, puis connecte les objets à une base de données protégée, résidant sur un serveur web ou un autre emplacement.

Visual Studio 2008 simplifie l'affichage des tables de base de données simples sur un site. Ainsi, à mesure que vos besoins s'accroissent, vous pouvez faire appel à Visual Studio pour traiter des commandes, gérer la sécurité, gérer des informations complexes relatives aux clients et créer de nouveaux enregistrements de base de données, tout cela à partir du web. Vous pouvez, par exemple, employer le contrôle *GridView* et Visual Web Developer pour afficher une table de base de données contenant des dizaines ou de milliers d'enregistrements sur un page web sans aucun code. Nous verrons comment cela fonctionne au cours du prochain exercice, dans lequel nous ajouterons une page web contenant les données des contacts au projet Calculateur de prêt automobile. Si vous avez réalisé les exercices de programmation de base de données du chapitre 18, « Démarrer avec ADO.NET » et du chapitre 19, « Présenter les données avec le contrôle *DataGridView* », vous noterez les similitudes (mais quelques différences aussi) entre la programmation d'une base de données dans un environnement Windows et la programmation de base de données sur le web.

Ajouter une nouvelle page web pour des informations relatives à une base de données

1. Dans le menu Site Web, choisissez la commande Ajouter un nouvel élément. Visual Web Developer affiche la liste des composants que vous pouvez ajouter au site web.
2. Cliquez sur le modèle Web Form. Dans la zone de texte Nom, tapez **PrêtsEnseignants.aspx** et cliquez sur Ajouter.

   Visual Web Developer ajoute une nouvelle page web au site web. Contrairement à la page HTML que vous avez ajoutée précédemment, le composant Web Form peut contenir des contrôles serveur.
3. Si nécessaire, cliquez sur l'onglet Design pour basculer en mode Design.

4. Saisissez le texte suivant dans la partie supérieure de la page web :

   La grille suivante indique les enseignants qui souhaitent un prêt ainsi que leur numéro de téléphone :

5. Appuyez deux fois sur la touche ENTRÉE pour ajouter une ligne vide sous le texte.

   Rappelez-vous que les contrôles de page web s'insèrent au niveau du point d'insertion. Il est important de toujours créer quelques lignes vides au moment d'ajouter un contrôle.

Vous allez ensuite ajouter deux champs de la table *Instructors* qui se trouve dans la base de données Etudiants.mbd, en ajoutant un contrôle *GridView* à la page web. Le contrôle *GridView* est similaire au contrôle *DataGridView* que vous avez utilisé au chapitre 19. Il a toutefois été optimisé pour un usage sur le web (il existe d'autres différences, que vous découvrirez en examinant la fenêtre Propriétés et la documentation de Visual Studio). Nous nous servirons de la même base de données Access que dans les chapitres 18 et 19. Vous verrez ainsi les similitudes de la programmation de base de données dans Visual Web Developer. De nombreux programmeurs se servent également de bases de données SQL sur leurs sites web : Visual Web Developer gère parfaitement ce format.

Ajouter un contrôle *GridView*

1. Avec la nouvelle page ouverte et le point d'insertion à l'emplacement approprié, double-cliquez sur le contrôle *GridView* qui se trouve sur l'onglet Données de la Boîte à outils Visual Web Developer.

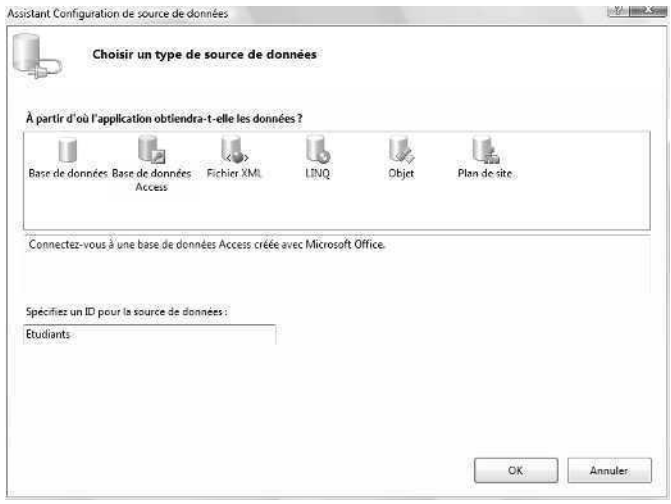
   Visual Web Developer ajoute un objet affichage de grille intitulé *GridView1* à la page web. L'objet affichage de grille contient pour l'instant des informations fictives.

2. Si la zone de liste Tâches GridView n'est pas affichée, cliquez sur la flèche de raccourci de l'objet *GridView1*.

3. Cliquez sur la flèche de la liste Choisissez une source de données, puis sur l'option Nouvelle source de données.

   Visual Web Developer affiche l'Assistant Configuration de source de données, un outil que vous avez employé aux chapitres 18 et 19 pour établir une connexion avec une base de données et sélectionner les tables et champs qui constituent un dataset.

Voici à quoi ressemble votre écran :



4. Cliquez sur l'icône Base de données Access. Dans la zone Spécifiez un ID pour la source de données, tapez **Etudiants** et cliquez sur OK.

   Vous êtes ensuite invité à indiquer l'emplacement de la base de données Access sur le système (cette boîte de dialogue est légèrement différente de celle rencontrée au chapitre 18).

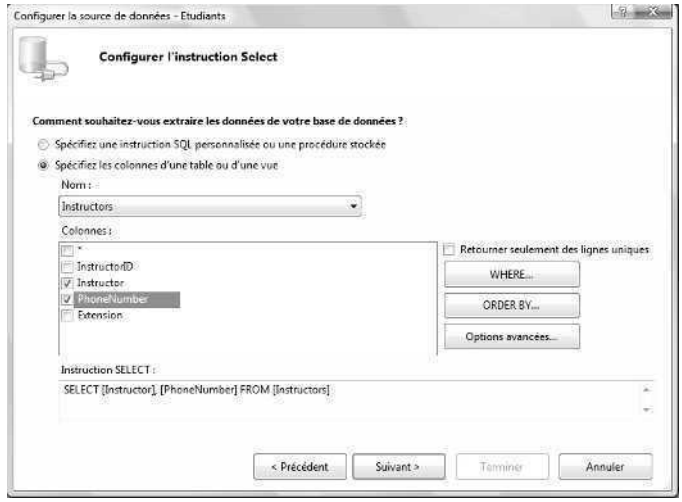
5. Tapez **c:\vb08epe\chap18\Etudiants.mdb** et cliquez sur Suivant.

   Vous devez maintenant configurer la source de données, autrement dit, sélectionner la table et les champs à afficher sur la page web. Nous allons faire appel à deux champs de la table *Instructors* (rappelez-vous que dans Visual Studio, le terme colonne désigne souvent un champ de la base de données ; vous voyez donc le mot « Colonnes » employé dans l'environnement de développement et dans les instructions suivantes).

6. Cliquez sur la flèche de la liste déroulante Nom et cliquez sur Instructors.

7. Dans la zone de liste Colonnes, cochez les cases en regard de Instructor et PhoneNumber.

Voici à quoi ressemble votre écran :

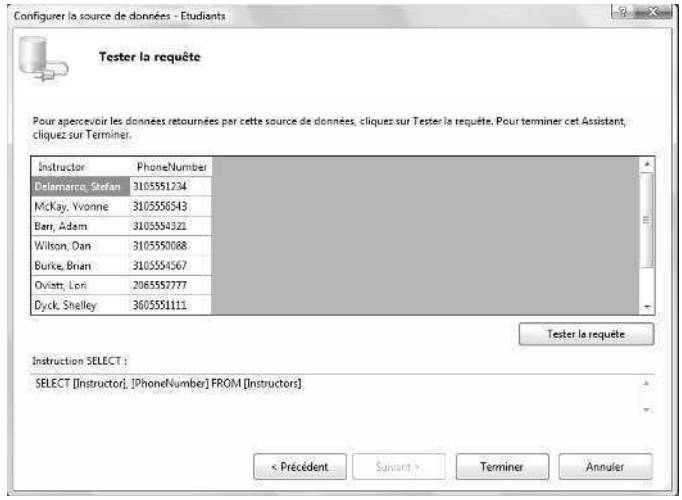


Ces actions créent une instruction SQL SELECT qui configure un dataset représentant une partie de la base de données Etudiants.mbd, visible dans la partie inférieure de la boîte de dialogue.

8. Cliquez sur Suivant pour afficher la page Tester la requête.

9. Cliquez sur le bouton Tester la requête pour prévisualiser les données.

Vous voyez un aperçu des champs réels Instructor et PhoneNumber de la base de données. Ces données se présentent tel que prévu. Toutefois, si nous étions en train de préparer ce site web pour une plus large distribution, nous aurions ajouté une étape pour formater la colonne PhoneNumber de sorte qu'elle contienne les espaces et mises en forme standards des numéros de téléphone.

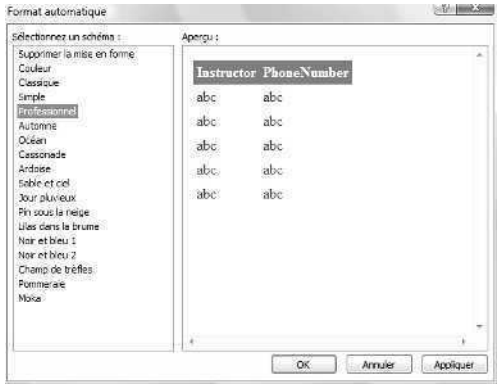


10. Cliquez sur Terminer.

Visual Web Developer ferme l'assistant et ajuste le nombre de colonnes et d'entêtes dans l'objet affichage de grille pour répondre aux sélections effectuées. Il continue cependant d'afficher des informations fictives (« abc ») dans les cellules de la grille.

11. Dans la liste Tâches *GridView*, choisissez la commande Mise en forme automatique.

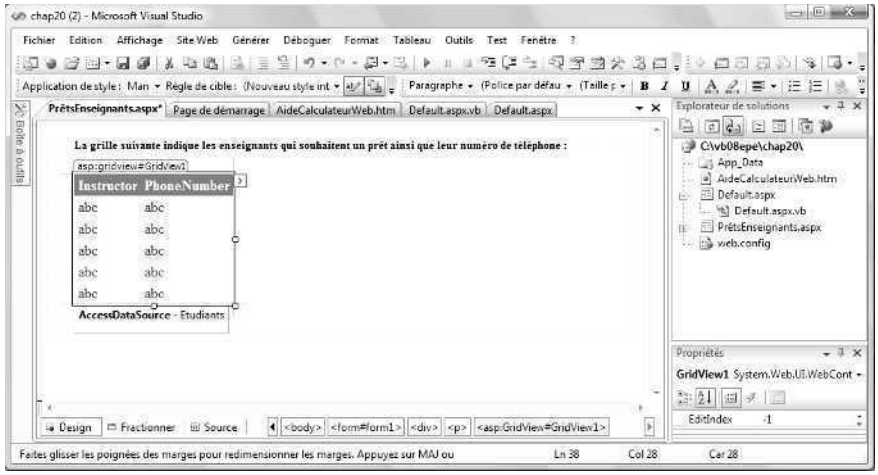
12. Cliquez sur le modèle Professionnel. Voici à quoi ressemble la boîte de dialogue Mise en forme automatique :



La possibilité de formater, d'ajuster et de prévisualiser rapidement les options de mise en forme est une intéressante fonctionnalité du contrôle *GridView*.

13. Cliquez sur OK et fermez la liste Tâches GridView.

La page web PrêtsEnseignants.aspx est à présent terminée et se présente comme suit (mon contrôle *GridView* est situé dans un élément `p`, mais le vôtre pourrait être placé dans un élément `div`) :



Ajoutons maintenant un lien hypertexte sur la première page (ou page d'accueil) qui affiche cette page web lorsque l'utilisateur veut voir la table. Nous allons nous servir du contrôle *HyperLink* pour créer le lien hypertexte.

Ajouter un lien hypertexte à la page d'accueil

1. Dans la partie supérieure du Concepteur, cliquez sur l'onglet Default.aspx.

   La page d'accueil du site web s'affiche dans le Concepteur.
2. Cliquez à droite de l'objet Aide (*lnkAide*) pour placer le point d'insertion après l'objet.
3. Appuyez deux fois sur ENTRÉE pour créer un espace pour un deuxième lien hypertexte.
4. Dans l'onglet Standard de la Boîte à outils, double-cliquez sur le contrôle *HyperLink* pour créer un objet lien hypertexte au niveau du point d'insertion.
5. Attribuez la valeur « Afficher les contacts » à sa propriété *Text*.

   Nous allons supposer que les utilisateurs sont les responsables des prêts bancaires (ou des vendeurs de voitures bien informés) cherchant à faire contracter des prêts automobiles aux professeurs d'université. Ils devront cliquer sur le lien Afficher les contacts pour visualiser les enregistrements de la base de données.
6. Attribuez la valeur « lnkContacts » à la propriété *ID* de l'objet lien hypertexte.
7. Cliquez sur la propriété *NavigateUrl* puis sur l'ellipse.

   Visual Studio ouvre la boîte de dialogue Sélectionner une URL.
8. Dans la zone de liste Contenu du dossier, cliquez sur le fichier PrêtsEnseignants.aspx puis cliquez sur OK.

Le lien est terminé et vous pouvez tester le site web et le contrôle *GridView* dans le navigateur.

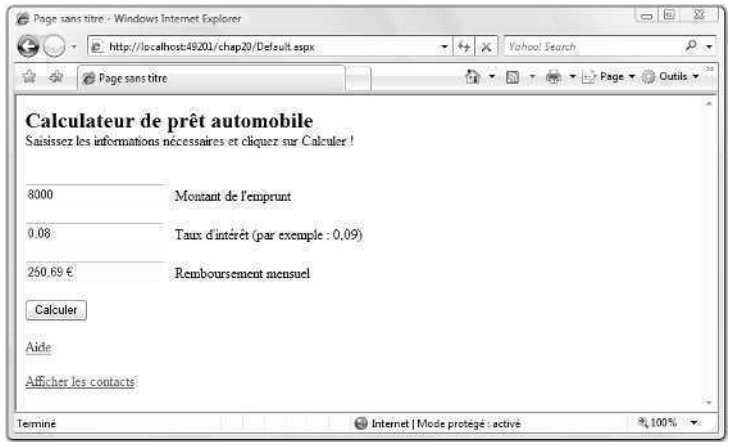
Tester le site web Calculateur de prêt automobile final

Astuce Le programme Calculateur de prêt automobile complet est disponible dans le dossier c:\vb08epe\chap20. Servez-vous de la commande Ouvrir un site web du menu Fichier pour ouvrir un site web existant.

1. Cliquez sur le bouton Démarrer le débogage. Visual Studio génère le site web et l'affiche dans Internet Explorer.
2. Tapez **8000** pour le montant du prêt et **0,08** pour le taux d'intérêt, puis cliquez sur Calculer.

   Le résultat est 250,69 €. Lorsque vous ajoutez des éléments à un projet, il est conseillé de tester à nouveau les fonctionnalités d'origine pour vérifier qu'elles n'ont pas été modifiées par inadvertance.

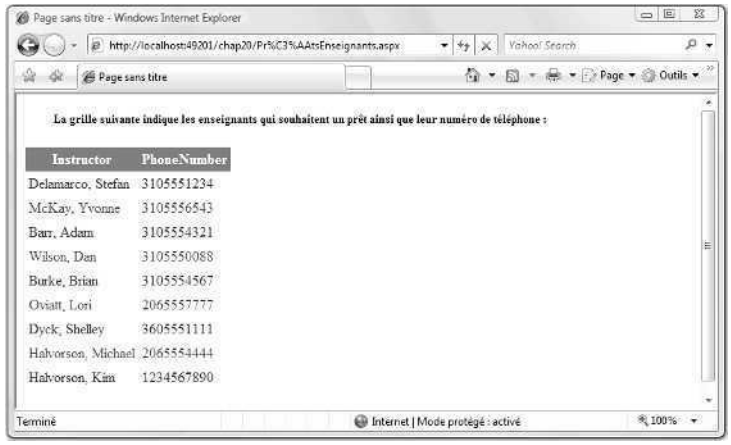
Voici à quoi ressemble votre écran :



Le nouveau lien hypertexte (Afficher les contacts) est présent dans la partie inférieure de la page web.

3. Cliquez sur Afficher les contacts pour charger la table de la base de données.

Internet Explorer charge les champs *Instructor* et *PhoneNumber* de la base de données Etudiants.mbd dans l'objet affichage de grille. Voici à quoi ressemble votre page web :



Les informations sont formatées de manière claire et utile. Par défaut, il est impossible de trier les données de la table, mais vous pouvez modifier cette option en cochant la case Activer le tri dans la liste Tâches GridView. Si la base de données contient un nombre important de lignes (enregistrements), cochez la case Activer la pagination de la liste Tâches GridView pour afficher une liste de numéros de pages dans la partie inférieure de la page web (à l'instar d'une liste présentée dans l'Explorateur de documents Microsoft ou un moteur de recherche qui affiche plusieurs pages de résultats de recherche).

4. Dans Internet Explorer, cliquez sur les boutons Précédent et Suivant.

   Comme nous l'avons étudié précédemment et à l'instar de n'importe quel site web professionnel, il est possible d'avancer et de reculer au sein des pages web du site web.

5. Lorsque vous avez terminé le test, fermez Internet Explorer pour fermer le site web.

Vous avez ajouté une table d'informations personnalisées, issues d'une base de données, sans ajouter de code !

Aller plus loin : Définir le titre du site web dans Internet Explorer

Pas encore repu ? Voici une dernière astuce de programmation web qui met le site web en valeur et élargit vos horizons.

Pendant que vous testiez le site web Calculateur de prêt automobile, vous aurez remarqué qu'Internet Explorer indique « Page sans titre » dans la barre de titre lorsqu'il affiche le site web. Autrement dit, votre écran ressemble à cela :

Il est possible de configurer ce que contient la barre de titre d'Internet Explorer ou de tout autre navigateur en définissant la propriété *Title* de l'objet *DOCUMENT* d'une page web. Testons-la.

Définir la propriété *Title*

1. Ouvrez la page web Default.aspx en mode Design. Dans la liste des objets de la fenêtre Propriétés, cliquez sur l'objet *DOCUMENT*.

Chaque page web d'un site web contient un objet *DOCUMENT* qui regroupe les paramètres généraux importants de la page web. Cet objet n'est cependant pas sélectionné par défaut dans le Concepteur, aussi ne l'aurez-vous sans pas remarqué. L'une des propriétés importantes de l'objet *DOCUMENT* est *Title*, qui définit le titre de la page web en cours dans le navigateur.

2. Attribuez la valeur « Calculateur de prêt automobile » à la propriété *Title*.

   Le changement n'apparaît pas à l'écran, mais Visual Web Developer l'enregistre en interne.

3. Cliquez sur le bouton Démarrer le débogage.

   Visual Studio ouvre Internet Explorer et charge le site web. Une intéressante barre de titre s'affiche à présent, comme le montre l'illustration suivante :

   C'est mieux !

4. Fermez Internet Explorer et modifiez la propriété *Title* des autres pages web du site web.
5. Lorsque vous avez terminé vos tests, enregistrez vos changements et fermez Visual Studio.

Félicitations pour avoir terminé l'intégralité du cours de programmation Microsoft Visual Basic 2008 Étape par étape ! Prenez le temps de parcourir à nouveau ce livre pour revoir tout ce que vous avez appris. Vous êtes prêt à relever des défis et apprendre des techniques de programmation Visual Basic plus élaborés. Dans l'Annexe, « Où trouver d'autres informations », vous trouverez une liste de ressources pour élargir vos connaissances. Mais avant, faites une pause : vous l'avez méritée !

Rappel du chapitre 20

Pour	Faites ceci
Créer un nouveau site web ASP.NET	Dans le menu Fichier, cliquez sur la commande Nouveau Site Web, cliquez sur le modèle Site web ASP.NET, désignez un dossier pour y placer le site dans la zone de liste Emplacement et cliquez sur OK.
Basculer entre le mode Design et le mode Source dans le Concepteur de pages web	Cliquez sur l'onglet Source ou l'onglet Design dans le Concepteur de pages web. Pour un affichage mixte, cliquez sur l'onglet Fractionner.
Saisir du texte sur une page web	Cliquez sur l'onglet Design et saisissez le texte à ajouter.
Formater le texte d'une page web	Sur la page, sélectionnez le texte à mettre en forme et cliquez sur un bouton ou un contrôle de la barre d'outils Mise en forme.
Afficher le code HTML d'une page web	Cliquez sur l'onglet Source dans le Concepteur.
Ajouter des contrôles à une page web	Affichez la page web en mode Design, ouvrez la Boîte à outils (qui contient automatiquement les contrôles Visual Web Developer), positionnez le point d'insertion à l'endroit où vous voulez insérer le contrôle sur la page et double-cliquez sur le contrôle dans la Boîte à outils.
Changer le nom d'un objet sur une page web	Servez-vous de la fenêtre Propriétés pour positionner la propriété *ID* de l'objet sur un nouveau nom.
Écrire la procédure événementielle par défaut pour un objet d'une page web	Double-cliquez sur l'objet pour afficher le fichier code-behind et écrivez le code de la procédure événementielle de l'objet dans l'Éditeur de code.
Vérifier le format des données saisies par l'utilisateur dans un contrôle d'une page web	Servez-vous d'un ou de plusieurs contrôles de validation, qui se trouvent dans l'onglet Validation de la Boîte à outils, pour tester les données saisies dans un contrôle d'entrée.
Exécuter et tester un site web dans Visual Studio	Dans la barre d'outils Standard, cliquez sur le bouton Démarrer le débogage. Visual Studio génère le projet et charge le site web dans Internet Explorer.
Créer une page HTML pour un projet	Dans le menu Site Web, choisissez la commande Ajouter un nouvel élément et ajoutez une nouvelle page HTML au projet. Créez et formatez la page HTML en vous servant du Concepteur de pages web.
Créer un lien vers d'autres pages web du site web	Ajoutez le contrôle *HyperLink* à la page web et positionnez la propriété *NavigateUrl* du contrôle sur l'adresse de la page web liée.
Afficher les enregistrements d'une base de données sur une page web	Ajoutez un contrôle *GridView* sur une page web dans le Concepteur de pages web. Établissez une connexion avec la base de données et formatez les données en vous servant des commandes de la liste Tâches GridView (la commande Choisir une source de données lance l'Assistant Configuration de source de données).
Définir le titre affiché dans la barre de titre d'Internet Explorer pour les pages web	Pour chaque page web, servez-vous de la fenêtre Propriétés pour définir la propriété *Title* de l'objet *DOCUMENT*.

Annexe

Où trouver d'autres informations

Ce livre présente des techniques de programmation Visual Basic 2005 pour utilisateur débutant, intermédiaire et avancé avec pour objectif d'en faire un développeur logiciel confiant et un programmeur Windows. Vous avez expérimenté de nombreux outils et fonctionnalités de Visual Basic 2008 et pouvez à présent vous orienter vers des sujets plus avancés et prendre la mesure de la suite de développement Microsoft Visual Studio 2008.

Si vous envisagez une carrière de programmeur Visual Basic, il peut également être intéressant de tester vos aptitudes en préparant un examen de certification en développement Visual Basic 2008. Dans cette annexe, vous trouverez une liste de ressources complémentaires pour la programmation Visual Basic, parmi lesquelles des sites web, une source d'informations sur la certification et des ouvrages que vous pouvez utiliser pour développer vos techniques en matière de programmation Visual Basic.

Sites web Visual Basic

Le web fait le bonheur des programmeurs et constitue définitivement le mécanisme le plus rapide pour collecter des informations sur Visual Basic 2008 et les technologies apparentées. Dans cette section, vous trouverez une liste de sites web que j'ai exploités pour découvrir les nouveaux produits et services relatifs à Visual Basic. Les adresses Internet et le contenu des sites de cette liste sont susceptibles d'évoluer au fil du temps : les sites peuvent donc ne pas se présenter tels que décrits. Si l'on tient compte du perpétuel flux et reflux de l'Internet, il est également intéressant d'effectuer occasionnellement une recherche sur « Visual Basic », « Visual Studio 2008 » et « VB.NET » pour vérifier la présence de nouvelles informations (pour obtenir des résultats plus spécifiques, placez l'élément recherché entre guillemets, tel que présenté ici). Des informations utiles peuvent également être obtenues en employant les noms de code originaux des produits, ici « Orcas » et « Visual Basic 9 ».

www.microsoft.com/france/vbasic/default.mspx

La page d'accueil du Centre de ressources pour les développeurs Visual Basic constitue le meilleur site pour la documentation, les dernières nouvelles, les informations relatives aux conférences et le support produit pour Visual Basic 2008. Ce site propose des informations à jour sur l'intégralité de la ligne de produits Visual Basic et explique comment les nouveaux systèmes d'exploitation, applications et outils de programmation affectent le développement de Visual Basic. Nouvelles fonctionnalités intéressantes, les blogs des membres de l'équipe Visual Basic et l'accès aux diffusions web (webcast) les plus récents. Adresse du site américain : *www.msdn.microsoft.com/vbasic/*.

Astuce Rappelez-vous que vous pouvez également accéder rapidement aux ressources MSDN du site américain à partir de la page de démarrage Visual Studio dans l'environnement de développement. La page de démarrage charge des articles et des informations à jour chaque fois que vous démarrez Visual Studio. Son contenu est en constante évolution.

www.devx.com/

Ce site en anglais est un site commercial dédié à de nombreux thèmes de développement Windows, dont la programmation Visual Studio et Visual Basic. Les groupes de discussion de programmeurs Visual Basic professionnels vous font bénéficier d'une interaction de collègue à collègue et d'un retour d'informations sur de nombreux problèmes de développement. En outre, les partenaires commerciaux du site DevX vendent des livres, des contrôles et des logiciels tiers. Depuis plusieurs années, un sondage rassemble des opinions sur les informations et outils du marché, ainsi que des discussions animées sur les produits concurrents comme Java et .NET.

www.microsoft.com/France/mspress/

Le site web de Microsoft Press propose les plus récents ouvrages sur la programmation Visual Basic écrits par les auteurs Microsoft Press. Consultez également la liste des nouveaux ouvrages relatifs à Microsoft Visual C#, Microsoft Visual C++ et les technologies de prise en charge de programmation web et de bases de données. Ce site permet également d'envoyer des courriels à Microsoft Press. Adresse du site américain : *www.microsoft.com/learning/books/*.

www.microsoft.com/france/formation/

Il s'agit du site web Formation et Certification Microsoft sur lequel se trouvent la liste des formations et des services relatifs aux formations et aux certifications. Au cours de ces dernières années, nombre de programmeurs Visual Basic ont découvert qu'ils étaient plus à même de démontrer leurs compétences en matière de développement à d'éventuels employeurs après avoir passé l'un des examens de certification et reçu une certification Microsoft, comme Microsoft Certified Professional (MCP), Microsoft Certified Systems Engineer (MCSE) ou Microsoft Certified Systems Administrator (MCSA). Visitez le site web pour en savoir plus sur les options de certification.

www.microsoft.com/France/communautes/

Ce site des communautés techniques pour de nombreux produits et technologies Microsoft propose des opportunités d'interagir avec les employés de Microsoft et vos collègues de développement logiciel. Par le biais de ce site web, vous pouvez accéder à des blogs, des groupes de discussion, des émissions web, des discussions techniques, des groupes d'utilisateurs et d'autres ressources relatives au développement Visual Studio. Adresse du site américain : *www.microsoft.com/communities/*.

Livres sur la programmation Visual Basic et Visual Studio

Les ouvrages relatifs à la programmation Visual Basic et Visual Studio fournissent des sources d'informations exhaustives et des formations autonomes que les sites web peuvent compléter, sans les remplacer. Dans le cadre de votre recherche à améliorer vos compétences en matière de programmation Visual Basic et Visual Studio, je vous recommande de consulter les sources suivantes (réparties par catégories). Cette bibliographie relative à Visual Studio n'est pas exhaustive, mais elle est représentative des ouvrages (en anglais et en français) disponibles au moment de la mise sur le marché de Visual Studio 2008. Vous trouverez également dans cette liste, des ouvrages relatifs à la programmation de base de données, de programmation web, de programmation VBA (*Visual Basic pour Applications*) et d'ouvrages plus généraux sur le développement logiciel et l'informatique.

Programmation Visual Basic

- *Microsoft Visual Basic 2008 Express Edition : Build a Program Now !,* de Patrice Pelland (Microsoft Press, ISBN 978-0-7356-2541-9).
- *Programming Microsoft Visual Basic 2005 : The Language*, de Francesco Balena (Microsoft Press, ISBN 978-0-7356-2183-1). Cet ouvrage traite de Visual Basic 2005 mais demeure très utile car la plupart des caractéristiques du langage restent identiques entre versions.
- *Practical Guidelines and Best Practices for Microsoft Visual Basic and Visual C# Developers* par Francesco Balena et Guiseppe Dimauro (Microsoft Press, ISBN 978-0-7356-2172-5).
- *Programming Windows Services with Microsoft Visual Basic 2008*, de Michael Gernaey (Microsoft Press, ISBN 978-0-7356-2433-7).
- *Visual Basic 2005, Les outils du développeur, Manuel de référence*, de Francesco Balena, *adaptation française* Emmanuelle Burr, Véronique Campillo et Véronique Warion (Microsoft Press, 2006, ISBN 978-2-1004-9941-0). D'après *Microsoft Visual Basic .NET Programming Microsoft Visual Basic 2005 Core Reference*.

Microsoft .NET Framework

- *Microsoft Windows Presentation Foundation : A Scenario-Based Approach,* by Billy Hollis (Microsoft Press, ISBN 978-0-7356-2418-4).
- *Développer avec Windows Communication Fundation*, de John Sharp, adaptation française Fabrice Lemainque (Microsoft Press, 2007, ISBN 972-2-1005-1211-9). D'après *Microsoft Windows Communication Foundation Step by Step*.

- *Développer avec Microsoft Windows Worklow Foundation*, de Kenn Scribner, adaptation française James Guérin (Microsoft Press, ISBN 972-2-1005-1212-6). D'après *Microsoft Windows Worklow Foundation Step by Step*
- *Debugging Microsoft .NET Framework 2.0 Applications*, par John Robbins (Microsoft Press, ISBN 978-0-7356-2202-9).
- *Working with Microsoft Visual Studio 2005 Team System*, par Richard Hundhausen (Microsoft Press, ISBN 978-0-7356-2185-3).

Programmation de bases de données avec ADO.NET

- *Microsoft ADO.NET 2.0 Step by Step*, par Rebecca Riordan (Microsoft Press, ISBN 978-0-7356-2164-0).
- *Manuel de référence Microsoft ADO.NET*, par David Sceppa, adaptation française Marc Israël (Microsoft Press, ISBN 978-2-1000-6523-3), paru en France sous le titre. *Programming Microsoft ADO.NET 2.0 Core Reference*.
- *Programming Microsoft ADO.NET 2.0 Applications : Advanced Topics*, par Glenn Johnson (Microsoft Press, ISBN 978-0-7356-2141-1).

Programmation web avec ASP.NET

- *Microsoft Visual Web Developer 2005 Express Edition : Build a Web Page Now !* par Jim Buyens (Microsoft Press, ISBN 978-0-7356-2212-4). Encore utile, malgré les améliorations apportées à Visual Web Developer dans Visual Studio 2008.
- *Microsoft ASP.NET 3.5 Programming Step by Step*, par George Shepherd (Microsoft Press, ISBN 978-0-7356-2426-9). ASP.NET 3.5 est la version présente dans Visual Studio 2008.
- *Programming Microsoft ASP.NET 3.5 Core Reference*, par Dino Esposito (Microsoft Press, ISBN 978-0-7356-2527-3).
- *Programming ASP.NET 2.0 Applications : Advanced Topics*, par Dino Esposito (Microsoft Press, ISBN 978-0-7356-2177-2). ASP.NET 2.0 est la version présente dans Visual Studio 2005.
- *Programming Microsoft LINQ*, par Paolo Pialorsi et Marco Russo (Microsoft Press, ISBN 978-0-7356-2400-9). C'est une étude en profondeur de la nouvelle technique LINQ présente dans Visual Studio 2008.

Programmation VBA (*Visual Basic for Applications*)

- *Microsoft Office Excel 2003 Programming Inside Out*, par Curtis Frye, Wayne S. Freeze et Felicia K. Buckingham (Microsoft Press, ISBN 978-0-7356-1985-9).

- *Programming Microsoft Office Access 2003 (Core Reference)*, par Rick Dobson (Microsoft Press, ISBN 978-0-7356-1942-5).

Ces deux livres restent utiles pour l'écriture de macros VBA dans des applications Microsoft Office. Microsoft se dirige toutefois vers un nouveau paradigme de programmation pour Office 2007 (*Visual Studio Tools for Office*), qui devrait gagner en popularité au fil du temps. Au moment de l'adaptation de ce livre, début 2008, il n'existe pratiquement pas d'ouvrage traitant de cette technique.

Ouvrages généraux sur la programmation et l'informatique

- *Code Complete, Second Edition*, par Steve McConnell (Microsoft Press, ISBN 978-0-7356-1967-8). Ce livre est probablement l'ouvrage le plus important pour des programmeurs autodidactes.
- *Code*, par Charles Petzold (Microsoft Press, ISBN 978-0-7356-1131-3), paru en France sous le même titre.
- *Writing Secure Code, Second Edition*, par Michael Howard, David LeBlanc (Microsoft Press, ISBN 978-0-7356-1722-3).
- *Software Project Survival Guide*, par Steve McConnell (Microsoft Press, ISBN 978-1-5723-1621-8).
- *Data Structures and Algorithms Using Visual Basic .NET*, par Michael McMillan (Cambridge University Press, ISBN 978-0-5215-4765-9).
- *The Art of Computer Programming, Volumes 1-3*, par Donald Knuth (Addison-Wesley Professional, 1997-98, ISBN 978-0-2014-8541-7). J'ai reçu ces trois volumes en cadeau et cela a fait mon bonheur ! Si vous ne devez n'en acheter qu'un, choisissez le premier volume.
- *Data Structures and Algorithms*, par Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft (Addison-Wesley, ISBN 978-0-2010-0023-8).

Il est particulièrement important que les programmeurs autodidactes se constituent au fil du temps une bibliothèque d'ouvrages généraux de programmation susceptibles de les aider sur des sujets plus théoriques et indépendants du langage, comme les algorithmes, les structures de données, le tri, les recherches, la compression, les nombres aléatoires, les mathématiques avancées, le travail en réseau, les compilateurs, etc. La liste ci-dessus ne constitue qu'un exemple, et nombre de ces ouvrages devraient pouvoir être trouvés en occasion.

Index

Symboles

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Z